Department of Computer Science
OST – University of Applied Sciences
Campus Rapperswil-Jona

Spring Term 2022

SR-App FlexAlgo

Term Project

Authors: Yael Schärer
Myriam Assunção

Advisors: Prof. Laurent Metzger
Urs Baumann

Version: June 3, 2022

Part I.

Abstract

Objective Segment Routing is used to engineer traffic on a specific path in the network. The characteristics of the path are manually written in the configuration of the routers and each path has to be encoded specifically. This approach doesn't scale. FlexAlgo configures Algos, which can be seen as a subset of the network. This gives flexibility to engineer network traffic based on less general constraints and uses a simplified way of configuration. Flexible algorithms are therefore much more dynamic and manage the traffic on a network as granular as necessary. Segments and packet routes become infinitely customizable and independent from each other. But, the maintenance of flexible algorithms in a network can be complicated, time consuming and needs intimate knowledge of network configuration. Here the application of this project comes into play. The SR-App FlexAlgo will make configurations fast and easy and allow not only a graphical view of the configurations but also show possible inconsistencies in the network. With the proof of concept built in this project, the application's feasibility can be determined and any mistakes in the architecture corrected.

Approach The application will work closely with the external system Jalapeño API Gateway built by the INS institute. The gateway allows a real time view of the connected network, the currently configured FlexAlgos and the surrounding networks. The application will have to be capable of handling large network workloads, as it will be used by service providers that can have networks with up of thousands of elements. To be able to handle these workloads and fulfill scaling and availability needs, the architecture was planned with a cloud native approach. A microservice architecture with serverless computing whenever reasonable and very lightweight frameworks further support the application's technical requirements.

Conclusion Studying the FlexAlgo technology in the duration of this project revealed all necessary data that is needed and where to find it. With this knowledge a prototype was built that reads and displays all relevant network and FlexAlgo configurations. Thanks to the Cisco proprietary application Jalapeño the project's application can retrieve the required configuration data from the underlaying network devices. The SR-App FlexAlgo prototype provides multiple views of the relevant data according to different groupings to help understand and work with the running network. Additionally, it allows a graphical view of the topology and its algos on a simplified website to show how the finished product may display this data.

This proof of concept shows that an application based on the flex algo technology is a fitting addition to the SR-App series.

Part II.

Management Summary

Authors	Myriam Assunção, Yael Schärer
Advisors	Prof. Laurent Metzger, Urs Baumann
Topic	Software Engineering, Segment Routing
Project Partner	Institute for Netowrk Solutions

Initial Situation

Flexible Algorithm (FlexAlgo) is a Segment Routing technology that improves the static nature of its parent protocol. FlexAlgo allows a dynamic configuration of network paths that bring high individuality to package traffic routing. This freedom brings much complexity and configuring FlexAlgo is a time consuming task that needs to be manually performed on each node of a network. Additionally it needs intimate knowldege of network configurations and is prone to human error, as configuration can be highly inconsistent between nodes.

Flexible Algorithm (FlexAlgo) is a Segment Routing technology that improves the static nature of its parent protocol. FlexAlgo allows a dynamic configuration of network paths and adds individuality to package routing. This freedom brings complexity. Configuring FlexAlgos is a time consuming task that needs to be manually performed on each node of a network. It needs intimate knowldege of network configurations and is prone to human error, as configuration can be highly inconsistent between nodes.

Here the SR-App FlexAlgo will find its place in todays market for network solutions. It will handle the FlexAlgo technology on a given network with an easy to use graphical interface that displays all the relevant data in a neat, understandable format. To change a FlexAlgo the user will merely have to update the information on any given FlexAlgo displayed in the application and the application will handle the task of updating and synchronizing all relevant network routers to ensure a correct and globally consistent configuration.

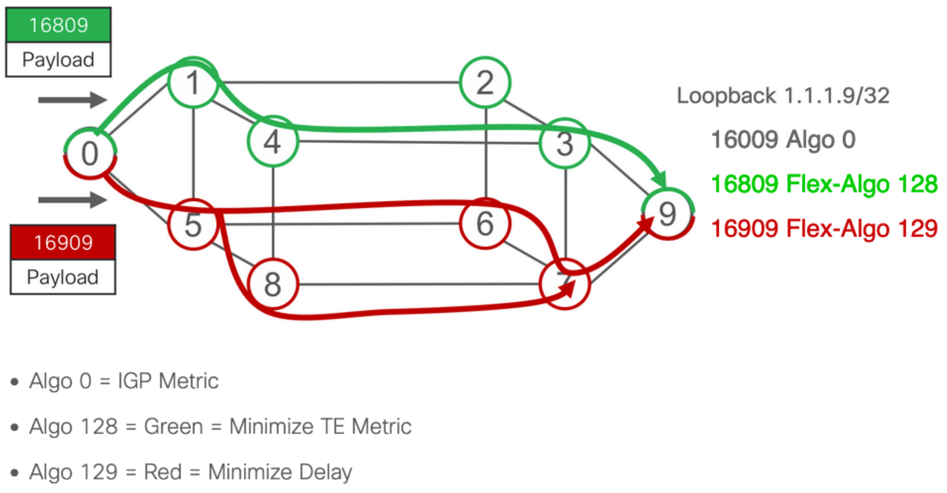


Figure 0.1.: Flexible Algorithm Topology [30]

Procedure and Technology

In this project a prototype for the SR-App FlexAlgo was built to ensure the SR-App will fit into and bring additional value to the SR-App series of the Institute for Network Solutions at the OST University of Applied Science.

To built the prototype all functional and non-functional requirements were defined and reviewed with the project advisors. This ensured all use cases and production demands of the software would be met. Additionally, research was conducted on Segment Routing and flexible algorithm to ensure that the SR-App FlexAlgo would cover all necessary information. This is ensures the prototype can be used a basis of a future project where configuring FlexAlgos will be the focus.

Based on the research and requirements an architecture was developed. High attention was paid to the performance of the application. This because the demands of current networks, that can grow quite large, have to be met to provide a satisfying user experience. Concurrently the functionalities of the prototype were chosen to give an overview of how the application would handle the collection and aggregation of network data.

With requirements, functionalities and architecture defined, the construction phase started. In the first week of this phase a stripped down draft of the prototype architecture was implemented to check its feasibility. Upon meeting success the prototype itself was built and extensively tested in the following weeks.
At first the network read service was constructed that accesses all network data and performs the necessary business logic to make the data easily consumable by a client. With the transformed data a server side rendered view was implemented to better determine if all necessary data was accessed. As time allowed a stripped down graphical frontend was adapted to give a basic view of how a finished frontend could be constructed.

Results

During this project a functional prototype of a SR-App FlexAlgo application was developed. With this product a comprehensive review of the software is possible and a decision can be reached about the feasability of the product SR-App FlexAlgo.

The backend service allows a user to access all relevant FlexAlgo data and provides an Application Programming Interface (API) to return the information in an easily consumable format for any given frontend. Additionally server side rendered web pages show an simple overview of the current configurations.

The additional graphical frontend shows a way to display a subset of the gathered data and how algorithms can be shown on a network as easily understandable graphs.

The prototype was tested and optimized for the fulfillment of the functional requirements and shows that handling the large amount of data in a reasonable amount of time is possible. As the prototype implements the performance heaviest part of the application, a good user experience for the final software will be achieveable.

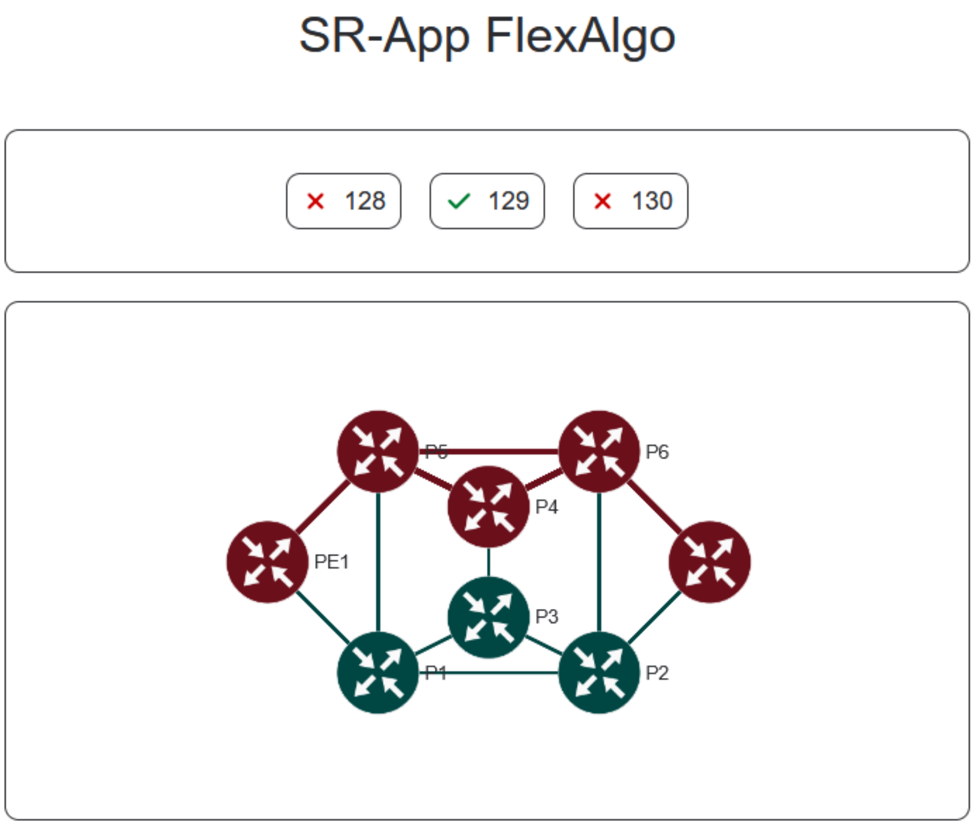


Figure 0.2.: Basic User Interface

Outlook
The prototype functions as a proof of concept and as such shows the SR-App FlexAlgo will be a worthwile addition to the SR-App line. The architecture planned for the project can be utilized for the final application with minimal changes to it. Now the production of the final software can be started.

Acknowledgements

We offer thanks to the following people that have helped to shape and support this project.

Prof. Laurent Metzger We thank as the main supervisor and initiant of the SR-App project. He helped us understand the domain and gave patient reviews during the weekly meetings.

Urs Baumann We thank for the support and supervision of this project as the co-advisor. His input at the weekly meetings improved our project greatly.

Yannick Zwicker We thank for the infrastructure he helped provide and the patient help with the Kubernetes technologies.

Severin Dellsperger We thank for the kind help with the FlexAlgo protocol and for lending us the books.

Michel Bongard We thank for the introduction to the Jalapeño API Gateway and the Jalapeño technologies. We would have struggled without his kind help. We also thank him for the use of this React demo application.

Michael Zollinger We thank for the support with the development of the architecture of this project and his valuable input.

Manuel Bauer We thank for his valuable input on the architecture of this project.

Florian Bruhin We thank for his patient help with the python programming language.

Davor Gajic and Leonard Obernhuber We thank for the usage of their mocking tool.

Contents

Glossary

API Application Programming Interface. Endpoint for requests to an application exposed to the outside.. 24, 25, 28, 29, 31–35, 69, 72, 73, 79, 80, 84, 86, 89–94, 96, 97, 100, 104, 108, 109

ArangoDB Multi-model, no-sql database with additional functionality for graphs.. 19, 34–36, 38, 63, 64, 66, 69, 72, 73, 77–79, 87, 88, 103, 104, 112, 136, 142, 175

BA Bachelor Thesis. 33, 34, 66, 75, 120, 128, 136

backend Python with Flask will be used.. 28, 35, 121, 122, 124, 125

bash Command line tool.. 112

BGP Border Gateway Protocol. A routing protocol that can be used inside an autonomous systems as internal protocol. Or it can be used as external routing protocol between multiple autonomous systems.. 15, 20, 60

Black A Python code formatter. 35, 127, 134, 138

cache A technology that stores data so future requests can be made faster.. 35, 38, 87

CI/CD Continous integration and deployment. 62, 122, 127

Cisco Company that develops software-defined networking, cloud, and security solutions.. 21, 22, 56, 60

Clockify A web tool to track the time team members spent on the poject.(https://app.clockify.me/timesheet). 125

cloud native A way to design, construct and operate architectures designed to utilize the cloud computing model.. 33, 57, 62, 121

Cobertura Java tool to calculate code coverage.. 138

CRUD Create, Read, Update, Delete. 40, 45, 50, 52, 56, 58, 75, 85, 97

database Data persistence technology. For this project PostgreSQL. 121, 122

decorator function Function object that wraps around an input function. It executes the input functions code with additions around it.. 99

Django A web framework that enables rapid development of secure and maintainable websites in Python.. 82, 83

Docker A set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. (by Wikipedia). 35–37, 107, 138

DTO Data Transfer Object. 76

Duckly An IDE plugin or web tool that makes it possible to code remotely together on a single file (pair programming).. 125

EoE End of Elaboration. 124

etcd Kubernetes Database for cluster information.. 106

FastAPI High performance Python Framework.. 83

Flask Lightweight web framework for Python.. 38, 83–86, 99, 100, 125

framework Premade software that provides generic functionality and can be extended with user-written, more specific code.. 38, 47, 64, 82–86, 121, 125

frontend A web application with React that extends an already existing application from INS that illustrates the devices and their connections. 11, 24, 25, 28, 29, 32–36, 74, 75, 110, 121, 122, 124, 128, 138

Git A free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.. 125

GitLab An open source end-to-end software development platform with built-in version control, issue tracking, code review, CI/CD, and more. OST version will be utilized in this project.. 35, 36, 62, 100, 106–108, 121–127, 138

golang Programming language, fast and easy to understand.. 69, 82, 83, 112

gRPC Google Remote Procedure Call. An open source remote communication technology that uses HTTPS. It supports the usage of TLS and token-based authentication and is consistent across platforms and implementation.. 22, 24, 28, 34, 35, 37–39, 62–64, 67, 69, 72, 76–78, 80, 83, 84, 86–91, 97, 121

html Hypertext Markup Language. 24, 28, 69, 73, 80, 91, 93, 94, 100, 137

HTTP Hyptertext Transfer Protocol. 28, 34, 38, 80, 104, 138

IETF Internet Engineering Task Force. 10

IGP Interior Gateway Protocol. This type of protocols let the packets be exchanged inside an autonomous system. Consider a more detailed site here.. 12, 13, 15, 18, 42, 61, 101

ingress A Kubernetes element that handles access from outside to the elements of a cluster.. 13, 63, 106, 107, 112

INS Institute of Networked Solutions in Rapperswil St.Gallen. 11, 12, 21, 22, 64, 66, 69, 100, 108, 112, 113, 121, 125

interface Physical or virtual points that a device owns over which the data packets get sent or received.. 121

IOS XR Virtual devices can be built according to the IOS images. This one is called XR and is used for router devices.. 20, 21, 33, 56, 60, 101

IS-IS Intermediate System to Intermediate System. A routing protocol that can be used in the control plane.. 13, 15, 18, 20, 61, 141

issue The project will be separated into different issues that have to be worked on. These can be seen on the repository on GitLab as boards (https://gitlab.ost.ch/ins-stud/flexalgo/sr-flexalgo/-/boards) or as a list (https://gitlab.ost.ch/ins-stud/flexalgo/sr-flexalgo/-/issues). 122

Jalapeño Cisco data collection framework.. 11, 19–22, 40, 88, 122

Jalapeño API Gateway A cloud-native infrastructure platform for network services to be used as an API Gateway between network devices and applications. INS proprietary software. The documentation can be read here.. 11, 21, 22, 24, 34, 35, 37, 39, 40, 47–50, 52, 54–58, 62–67, 69, 76, 79, 80, 84, 87–90, 97, 112, 120–122, 124, 125, 132

JavaScript Programming language for frontend solutions.. 104

Jinja Template engine that compiles text-based formats like html out of Jinja templates, which are text files.. 100

json JavaScript Object Notation. 28, 42, 80, 90, 94–98, 112

K8s Kubernetes. 62–64

Knative A way for Kubernetes based application to be build serverless and event driven.. 64, 67, 85, 86

Kubernetes System to manage container applicactions.. 30, 35, 38, 39, 57, 62–64, 67, 72, 84–86, 89, 90, 99, 106, 107, 122, 125, 126, 135

LaTeX A type setting system that can build pdf files out of tex files. 35–37, 121, 125, 138

load balancing Process of balancing workloads on cloud-based application systems.. 63

LTB Lab Topology Builder. A web application of INS to provide virutal networks.. 21, 141

message queue Communication technology that handles and stores messages in a system. They use a queue for receiving and distributing messages.. 38, 64, 85, 89, 121

MPLS Multiprotocol Label Switching. A network technology to route traffic according to labels. Usable as data plane protocol.. 12–15, 18, 56, 87

MyPy Static type checker for Python.. 35, 138

namespace A mechanism for isolating groups of resources within a single Kubernetes cluster.. 30, 64

orm Object Relation Mapper. 83, 86

OSPF Open Shortest Path First. A routing protocol that can be used in the control plane.. 13

pod Smallest deployable unit of computing in Kubernetes.. 30, 64, 125

Poetry A Dependency management and packag tool written in Python. It installs and updates the specified libraries of the project. (https://python-poetry.org/). 62, 64, 108, 126

PostgreSQL A relational database management system (RDBMS). 85, 86, 121

Pre-forking A master process creates forked sub processes to handle overwhelming workloads. These processes do not share variable data.. 38

Prefix-SID Adjacency Segment Identifier. Used to identify a devices interface per defined Flex-Algo in the segment domain.. 15–17, 141

Prefix-SID Prefix Segment Identifier. Used to identify a device per defined FlexAlgo in the segment domain.. 15, 17, 19, 103

protobuf Protocol Buffers are a mechanism for serializing and deserializing data structures. It is platform and language neutral, developed by Google.. 76, 77

Pylance An extension for the IDE to provide performant language support.. 127

Pytest The pytest framework makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries, as it is written on the official website.. 35

Python An object-oriented programming language for small- and large-scale projects. The used version for this project is 3.9.. 35, 38, 82–86, 121, 125–127, 134

React.js JavaScript library for building modular single page applications.. 25, 28, 29, 31, 37, 40, 47, 63, 69, 73, 80, 82, 83, 94–97, 104, 108, 121, 136–138, 141

Redis It is a in-memory data structure store, used as a database, cache, streaming engine, and message broker. Consider official website.. 39

repository Remote repository for versioning. For this project hosted on GitLab.. 35, 37, 62, 100, 106

rolling update A way for Kubernetes to update pod code without losing up time by incrementally updating pod instances with new ones.. 64

SA Term Thesis. 11, 28, 64, 120, 121, 123, 134, 142

SCRUM+ Agile project management system that will be used as a base for this project. The method of time estimation was not done in this project.. 121, 122

Segment Routing A source-based routing technique that simplifies traffic engineering and management across network domains. (by Juniper). 121

SerPro Bachelor Thesis made by Severin Dellsperger and Julian Klaiber.. 40, 75, 110

Sigma.js JavaScript Library for rendering graphs using WebGL.. 83, 104

SonarQube Code review tool that can be integrated into pipelines.. 35–37, 135, 136, 138

SPA Single Page Application. 69, 80, 81, 84, 85, 104, 105, 108, 137

SPF Shortest Path First. It is also called Dijkstra algorithm and is a algorithm to route along the shortest calculated path according to the metric and avoiding loops in a network.. 18, 101

SPRING Source Packet Routing in Networking. 10

sprint An iteration of 2 weeks duration and is part of the agile project management system.. 122

SR-Apps Segment Routing Applications by INS. 10, 100, 104

SRLG Shared Link Risk Group. 19, 42, 102

SRTE Segment Routing Traffic Engineering. 10

static typing A way to include typing into the untyped python language. This imporves code readability and maintainability.. 83

stederr The standard stream for error logging. Prints output message to the console, can be redirected to a logging service.. 64, 81, 99

stedstr The standard output stream. Prints output message to the console, can be redirected to a logging service.. 99

Swagger API documentation technology. Used in form of a library in this project.. 31, 69, 91, 92, 125

TE Traffic Engineering. 18, 42, 61, 103

Teams MS Teams or Microsoft Teams, a program for remote communication.. 122, 125

twelve-factor methodology A set of rules to plan and build software-as-a-service that ensures optimal utilization of cloud resources.. 33, 34, 62, 138

UI User Interface. 128

UNIX Uniplexed Information and Computing Service. 38

url Userfriendly adress to reach a website. (Uniform Resource Identifier). 33, 35, 62, 69, 79, 108, 109

Use Case Intended actions of the user with the application.. 79

Visual Studio Code An IDE from Microsoft that can be extended with plugins, like Duckly.. 125, 134

VPN Virtual Private Network. 69, 108

websocket Network protocol based on TCP that allows bidirectional communication.. 63, 89, 90

wireframe Early, conceptual functual design of a website.. 110, 134

WSGI Web Server Gateway Interface. 38

YAML Human-readable data-serialization language. Often used for network configuration and operation files.. 106, 107

YANG Yet Another Next Generation. 19, 20, 25, 42, 66, 87, 89, 90, 124, 131, 132

Bibliography

[1] *apache - What exactly is a pre-fork web server model? - Stack Overflow. What exactly is a pre-fork web server model?* 2022-05-28. URL: https://stackoverflow.com/questions/25834333/what-exactly-is-a-pre-fork-web-server-model.

[2] *ArangoDB-Community/python-arango: Python Driver for ArangoDB*. Mar. 15, 2022. URL: https://github.com/ArangoDB-Community/python-arango.

[3] *Axios. AXIOS*. May 14, 2022. URL: https://axios-http.com/.

[4] *Basics tutorial | Python | gRPC. Basics tutorial*. Mar. 20, 2022. URL: https://medium.com/@biplav.nep/grpc-using-flask-restful-code-2ed5607ae9a.

[5] *Border Gateway Protocol - Link State (BGP-LS) Parameters. Border Gateway Protocol - Link State (BGP-LS) Parameters*. May 31, 2022. URL: https://www.iana.org/assignments/bgp-ls-parameters/bgp-ls-parameters.xhtml.

[6] *Checklist End of Elaboration | Thomas Wiki. Checklist End of Elaboration*. Mar. 28, 2022. URL: https://elsensohn.ch/en/docs/projectmanagement/checklist-end-of-elaboration/.

[7] *cisco-open / jalapeno*. May 30, 2022. URL: https://github.com/cisco-open/jalapeno.

[8] *Creating a Web Application using Python Flask with Server Side Rendering | by Simranjit Kamboj | Medium. Creating a Web Application using Python Flask with Server Side Rendering*. Apr. 4, 2022. URL: https://medium.com/@simranjitkamboj/creating-a-web-application-using-python-flask-with-server-side-rendering-9ebea8204193.

[9] *Creating Beautiful REST APIs using Python Flask and Swagger UI - YouTube. Creating Beautiful REST APIs using Python Flask and Swagger UI*. Apr. 19, 2022. URL: https://www.youtube.com/watch?v=k10ILjUyWuQ.

[10] Alberto Donzelli. *Introduction to Segment Routing*. On slide 19. June 1, 2022. URL: https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2019/pdf/BRKRST-2124.pdf.

[11] Clarence Filsfils, Kris Michielsen, and Ketan Talaulikar. *Segment Routing, Part I*. Cisco Systems, Inc., Jan. 2017. Chap. 2.1 What is Segment Routing.

[12] Clarence Filsfils, Kris Michielsen, and Ketan Talaulikar. *Segment Routing, Part I*. Cisco Systems, Inc., Jan. 2017. Chap. 3.5 MPLS TTL and TC (or EXP) Treatment.

[13] Clarence Filsfils et al. *Segment Routing Part II - Traffic Engineering*. Cisco Systems, Inc., 2019. Chap. 7.2.2.

[14] *Flask-Caching – Flask-Caching 1.0.0 documentation. Flask-Caching*. Apr. 15, 2022. URL: https://flask-caching.readthedocs.io/en/latest/index.html.

[15] *Flask-Caching – Flask-Caching 1.0.0 documentation. Flask-Caching*. May 5, 2022. URL: https://flask-caching.readthedocs.io/en/latest/index.html.

[16] *flask-swagger · PyPI. flask-swagger 0.2.14*. Apr. 19, 2022. URL: https://pypi.org/project/flask-swagger/.

[17] *flask-swagger-ui - PyPI. Python Arango ORM Package Documentation*. May 31, 2022. URL: https://pypi.org/project/flask-swagger-ui/.

[18] *Flask-WTF – Flask-WTF Documentation (1.0.x). FlaskWTF*. Apr. 15, 2022. URL: https://flask-wtf.readthedocs.io/en/1.0.x/.

[19] *Flexible Algorithms: Bandwidth, Delay, Metrics and Constraints. draft-ietf-lsr-flex-algo-bw-con-02*. June 1, 2022. URL: https://datatracker.ietf.org/doc/html/draft-ietf-lsr-flex-algo-bw-con.

[20] *GitHub repository of React SigmaJS Demo of Michel Bongard.* May 30, 2022. URL: https://github.com/mbongard/react-sigmajs-demo.

[21] *gRPC using Flask restful code. Last week I was playing with gPRC, so I... | by Biplab Pokhrel | Medium. gRPC using Flask restful code.* Mar. 20, 2022. URL: https://medium.com/@biplav.nep/grpc-using-flask-restful-code-2ed5607ae9a.

[22] *Gunicorn - Python WSGI HTTP Server for UNIX. Gunicorn.* May 10, 2022. URL: https://axios-http.com/.

[23] *Home - Knative. Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications.* Apr. 22, 2022. URL: https://knative.dev/docs/.

[24] *How to create a Backend with Ptyhon and Flask - YoutTube.* Mar. 12, 2022. URL: https://www.youtube.com/playlist?list=PLab_if3UBk98jBTmyxShFVirMbgfFYu8W.

[25] *Introduction | Documentation | Poetry - Python dependency management and packaging made easy. Poetry Documentation.* Mar. 10, 2022. URL: https://python-poetry.org/docs/.

[26] *ISO - ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. ISO/IEC 25010:2011.* Feb. 26, 2022. URL: https://www.iso.org/standard/35733.html.

[27] *Jalapeño API Gateway.* May 30, 2022. URL: https://jalapeno-api-gateway.github.io/jagw-docs/.

[28] Craig Larman. *Applying UML and Patterns.* Addison Wesley Professional, Oct. 2004.

[29] *List of Fastest Frameworks for Python App Development? Fastest Frameworks for Python App development- Dev Technosys.* Mar. 5, 2022. URL: https://devtechnosys.com/insights/fastest-frameworks-for-python-app-development/.

[30] *Modern IX Fabric Design.* June 2, 2022. URL: https://xrdocs.io/design/blogs/2019-02-02-modernizing-ixp-design/.

[31] *python - Flask + gRPC trouble - Stack Overflow. Flask + gRPC trouble.* Mar. 5, 2022. URL: https://stackoverflow.com/questions/61387844/flask-grpc-trouble.

[32] *python-arango - PyPI. Python Arango Package Documentation.* Mar. 15, 2022. URL: https://pypi.org/project/python-arango/.

[33] *python-orm - PyPI. Python Arango ORM Package Documentation.* Mar. 15, 2022. URL: https://pypi.org/project/arango-orm/.

[34] *Queues — RabbitMQ. Queues.* Mar. 17, 2022. URL: https://www.rabbitmq.com/queues.html.

[35] *RFC 1142 - OSI IS-IS Intra-domain Routing Protocol.* Obsoleted by RFC 7142. May 31, 2022. URL: https://datatracker.ietf.org/doc/html/rfc1142.

[36] *RFC 5305 - IS-IS Extensions for Traffic Engineering.* May 30, 2022. URL: https://datatracker.ietf.org/doc/html/rfc5305.

[37] *RFC 7752 - North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP.* May 30, 2022. URL: https://datatracker.ietf.org/doc/html/rfc7752.

[38] *RFC 7752 - North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP. 3.3.2.3 TE Default Metric TLV.* May 30, 2022. URL: https://datatracker.ietf.org/doc/html/rfc7752#section-3.3.2.3.

[39] *RFC 7752 - North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP. 3.3.2.5 Shared Risk Link Group TLV.* May 30, 2022. URL: https://datatracker.ietf.org/doc/html/rfc7752#section-3.3.2.5.

[40] *RFC 8570 - IS-IS Traffic Engineering (TE) Metric Extensions.* May 30, 2022. URL: https://datatracker.ietf.org/doc/html/rfc8570.

[41] *RFC 8660 - Segment Routing with the MPLS Data Plane.* June 1, 2022. URL: https://datatracker.ietf.org/doc/html/rfc8660.

[42] *RFC 9104 - Distribution of Traffic Engineering Extended Administrative Groups using the Border Gateway Protocol - Link State (BGP-LS). 2. Advertising Extended Administrative*

Groups in BGP-LS. May 30, 2022. URL: https://datatracker.ietf.org/doc/html/rfc9104#section-2.

[43] *Segment Routing - SR IGP Flexible Algorithm. SR IGP Flexible Algorithm*. May 24, 2022. URL: https://www.segment-routing.net/tutorials/2018-03-06-segment-routing-igp-flex-algo/.

[44] *Segment routing - Wikipedia. Segment Routing*. May 24, 2022. URL: https://en.wikipedia.org/wiki/Segment_routing.

[45] *Sigma.js. sigma.js*. May 31, 2022. URL: https://www.sigmajs.org/.

[46] *SR IGP Flexible Algorithm. SR IGP Flex Algo*. On slide 3. Feb. 26, 2022. URL: https://www.segment-routing.net/tutorials/2018-03-06-segment-routing-igp-flex-algo/.

[47] *Static Typing in Python | Engineering Education (EngEd) Program | Section. Static Typing in Python*. May 25, 2022. URL: hhttps://www.section.io/engineering-education/python-static-typing/.

[48] *Template Designer Documentation – Jinja Documentation (3.1.x). Template Designer Documentation*. May 31, 2022. URL: https://jinja.palletsprojects.com/en/3.1.x/templates/.

[49] *Templates – Flask Documentation (1.1.x). Templates*. May 31, 2022. URL: https://flask.palletsprojects.com/en/1.1.x/tutorial/templates/.

[50] *The C4 model for visualising software architecture*. Mar. 1, 2022. URL: https://c4model.com/.

[51] *The Twelve-Factor App. The Twelve-Factor App*. Mar. 23, 2022. URL: https://12factor.net/.

[52] *Unit Testing in Python with pytest - YouTube. Unit Testing in Python with pytest*. Apr. 2, 2022. URL: https://www.youtube.com/playlist?list=PLyb_C2HpOQSBWGekd7PfhHnb9GnqDgrxS.

[53] *WebSocket - Wikipedia. WebSocket*. Mar. 12, 2022. URL: https://de.wikipedia.org/wiki/WebSocket.

[54] *Welcome to Flask – Flask Documentation (2.1.x). Flask*. Mar. 30, 2022. URL: https://flask.palletsprojects.com/en/2.1.x/.

[55] *What Is a Flex-Algo? How Is a Flex-Algo Defined and How Does It Work? - Huawei. What Is a Flex-Algo?* Mar. 2, 2022. URL: https://info.support.huawei.com/info-finder/encyclopedia/en/Flex-Algo.html.

[56] *What is segment routing? What are its benefits and applications?* Author is Sreejith Gs of the blog. June 2, 2022. URL: https://www.quora.com/What-is-segment-routing-What-are-its-benefits-and-applications.

[57] *What is the control plane? | Control plane vs. data plane*. May 31, 2022. URL: https://www.cloudflare.com/learning/network-layer/what-is-the-control-plane/.

[58] *What's Segment Routing*. Prefix: clns-isis-cfg. June 2, 2022. URL: https://www.juniper.net/us/en/research-topics/what-is-segment-routing.html.

[59] *Yang Catalog*. June 2, 2022. URL: https://datatracker.ietf.org/doc/draft-ietf-lsr-flex-algo/.

Part III.

Technical Report

1. Introduction

This thesis is written for engineers in the field of computer science. Therefore basic understanding of the subject matter of software engineering and networking is expected.

1.1. Thesis Composition

This thesis is written in accordance with the regulations of the department of computer science at OST university of applied science. The following documents are provided in this report.

Technical Report

The technical report gives a description of the outcome of this research project. It provides an overview of the project, techniques used and the goals and objectives of this research project. Basic information about the networking technique this project is based on is provided. Finally, it gives a comprehensive review of the achieved results with a critical eye on problems in the solution and an outlook on the future of this project.

Project Documentation

The document provides a comprehensive documentation of the software with the requirements engineering, domain analysis and all information about the architectural design of the software. Additional notes are given about special software cornerstones of the project.

Project Management

How the project was managed is documented in this document. It provides the project plan and all further management information like a risk analysis, summary of the quality measures and test protocols. Additionally the administrative documents of the research project like the taks formulation and signed documents can be found here.

1.2. Motivation

Segment Routing and it's child protocol flexible algorithm, are network routing technologies. Segment Routing is a source-routing network protocol by SPRING and IETF, that manages traffic by segments, a set of instructions defined on a packet [44]. Flexible algorithm refines this process by adding flexible, user-defined segments to the SRTE toolbox [43]. As such it allows for highly customizable network trafficking and covers today's specialized demands of networks to a much better degree. As such it is a technology that shows great potential and should be added to the SR-Apps line. The application prototyped in this research project, SR-App FlexAlgo, slots neatly into this app series.

For the following documentation the term `algorithm` will be used as FlexAlgo algorihtm.

1.3. Aims and Objectives

Problem

FlexAlgo, in contrast to its parent technology, allows for customizable traffic routing on a packet level. This high level of individualism adds a lot of complexity to the configuration process of networks. A network engineer has to define and maintain the algorithms on each routing device on the network manually, which is error prone and requires a significant amount of time and network knowledge.

A solution to make FlexAlgo configurations easier and uniform over the whole network is to be found. To achieve this the topology with all FlexAlgos defined on the network and their property values has to be displayed in a comprehensive way to give an optimal user experience. This information has to be real-time data to ensure engineers are always up to date with the current network. It must therefore be constantly updated without user intervention. Further, updating FlexAlgos and their properties, adding new ones and deleting unnecessary ones has to be possible with minimal input from the engineer. In other words configurations are to be defined once by the user and then automatically deployed through the whole network without the user having to manually do so.
While a configuration deployment is underway the functionality needs to be locked for all users of the application. No simultanous deployments from multiple users are to be permitted. This to remove the danger of lost updates and an inconsistent system.

The solution must be able to handle large networks with up to 1'000 objects. The application must be planned to allow the additional feature of persisting network data for a historical overview of the FlexAlgo configurations in a future project.

Solution

To solve the problems presented in the last paragraph a SR-App is planned that slots into the existing application series. As a first step this SA project will implement a prototype of this software. With this prototype the planned functionality and architecture can be checked.
To achieve this solution, research into the technology of Segment Routing and flexible algorithm is to be done. The functionality of these protocols needs to be understood and the current structure of the data on the nodes and in the Jalapeño software is to be determined. Additionally necessary changes to the INS software Jalapeño API Gateway need to be defined to ensure the Gateway can provide all FlexAlgo data.
Based on the research an architecture for the SR-App FlexAlgo is to be developed. A first draft for the whole architecture and a detailed development plan for the prototype is to be developed.
A prototype for the software is to be implemented to check feasibility of the product. The prototype should read all necessary network data for configuring FlexAlgos and present them first in a very simple text based frontend as a list. In a further step a graphical frontend may be implemented to show the algorithms running on the current network.
The prototype is to be tested extensively for performance to ensure the service is able to handle the performance demands of large networks.

2. Segment Routing Fundamentals

This project is built on the basis of a network provided by the INS and implements Segment Routing, Intermediate System to Intermediate System (IS-IS) 5 and Multiprotocol Label Switching (MPLS) 5.

For a better understanding of the business domain of the application built in this project, information about the fundamental structures of networks and the usage of Segment Routing are provided in this chapter. Specific terminology used in this chapter may be explained in more detail in chapter Terminology 5.
As this project merely reads FlexAlgo data and not yet manages configurations of such a more basic overview is given. Detailed information about the protocols handled in the SR-App FlexAlgo will be provided in a further project.

2.1. Basics

A network, or also called network domain, can range from two devices that communicate with each other to thousands of devices with a complicated net of links between them.

To let devices of a network communicate properly with each other and for messages to successfully navigate the often times quite complex net of connections, protocols were created to handle the flow of traffic. There are different kinds of protocols, each with different benefits and drawbacks. Some protocols are suited for the communication inside a network (intradomain or IGP) while others handle connections between netowrk domains (interdomain protocols). Each protocol uses different criterias and configurations to define a network and its traffic.
These protocols can become quite complex and performance intensive, especially on large networks. If there is a destination based routing protocol configured, a network device needs to calculate the best route according to the protocols properties through the network and find alternatives should there be problems with other devices or links.

As the network increases in size, the construction of the domain with the configurations of the devices, their redundancies and their maintainance get more challenging. To manage this, techniques were developed which handle traffic in a combination of other protocols that work on the control and data plane.

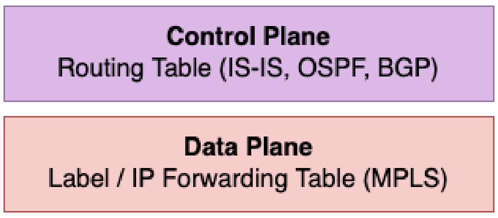

Control Plane
Routing Table (IS-IS, OSPF, BGP)
Data Plane
Label / IP Forwarding Table (MPLS)

Figure 2.1.: Control and Data Planes

While this solves performance issues on the network, the two planes also add complexity to network engineering. The routing and forwading technologies of the planes require expertise and make the maintenance and supervision of all configurations harder. Additionally, adding new devices and protocol specific configurations to the domain is time consuming and error prone.

2.1.1. Segment Routing

Attempts at making configurations and maintainace of networks easier has already been made. Segment Routing, a source-based technique, is one such protocol that has been developed for some years (RFC [41]). It works thanks to a combination of multiple protocols which are explained more in the next section 2.1.1.
Segment Routing simplifies traffic engineering and maintainance of network domains. It places the path information in packet headers, done by ingress nodes, and so removes the network state infomration from transit routers. This reduces performance loads and complexity from a lot of the network devices. [56]

Segments

In Segment Routing network devices work with semgents, instructions. Each packet has a list of these segments in the packet header, which is used for steering the packet. The node that receives the packet can read the top most entry of the segment list and decides how to handle the packet accordingly.
The process therefore works with the principle of source routing instead of the more usual destination routing. Additionally, only the source node decides the traffic steering of the packets, not as before all traversed nodes. Meaning these nodes don't have to maintain and store the per-flow state anymore. This provides more flexible behaviours in the network and increases scalability [11].

Protocols

A big difference to other protocols is that Segment Routing uses source routing instead of destination routing. This process changes how routers treat packets on the network.Segment Routing works with the combination of control and data plane. The IGP, which builds the control plane, will distribute the required segment information, the link attributes, to all devices that want to participate in the Segment Routing domain. It will use either the protocol IS-IS or OSPF. The difference between those two protocols lies in the configurations of the network devices and their behaviour.
OSPF declares a central backbone area which acts as a controlling sector of the network. Neighbouring network areas have to send their traffic through this area to reach other networks. The resulting architecture mimiks a star, with multiple OSPF areas spaced around one transfer centre. IS-IS on the other hand works on OSI layer 2. This can bring problems to the protocol, like configuration incompatibility between different networks. This has to be solved by the network engineer. It works with different messages to orchestrate network traffic.

Both IS-IS and OSPF work well with the data plane protocols MPLS and IPv6. The difference between those two lies in the handling of the segments in the packets. Although IPv6 lies on the OSI Layer3 it provides the possiblity to be used as the data plane. In contrast to MPLS, IPv6 won't remove the top segment information from the packets's extension header after traversing the corresponding device. To work with this the extension header contains the field `Segments Left`, which contains the number of the next segment to send the packet to. This segment information gets copied into the `Destination Address` of the IPv6 header.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Last Entry   |     Flags     |              Tag              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.2.: IPv6 Extension Heade with List of Segments [10]

MPLS, on the other hand, saves the segment into its MPLS header field `Label`, viewable in the following graphic 2.3. The `Exp`erimental field is used as `traffic class` indicating the used class of service. The `S`-flag is just 1 bit long and indicates if the MPLS tag stack is at the end or if another tag is following 2.4. The `TTL`, time to live, functions in the same way as the one in an usual IP packet. It tells the devices how many hops the packet is still allowed to do until it's considered invalid.

Label [20]	Exp [3]	S-Bit [1]	TTL [8]

Figure 2.3.: MPLS Header with 20Bit long Label Field [12]

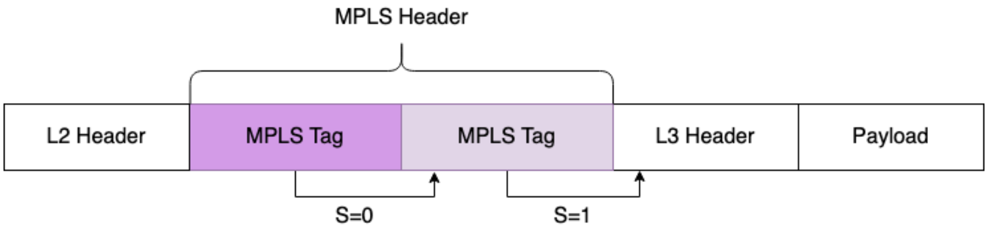


Figure 2.4.: MPLS Tags in a single Packet

After reaching the indicated device with the corresponding Segment ID, derived from the segment in the list, the label gets popped, making the following segment the new active one.

For the following chapters the protocol IS-IS in combination with MPLS is used.

Global and Local Segments

Two types of SIDs are used to complete the functionality of Segment Routing:

- `16001 - 23999`: Global Segments - Prefix-SID. Defined on Loopback interface.
- `24000 - 1048575`: Local Segments - Prefix-SID. Defined per non-Loopback interface.

Each device that belongs to the Segment Routing Domain will have at least one Prefix-SID configured. A Prefix-SID is significant per algorithm, and therefore a device can have multiple Prefix-SIDs, one per configured algorithm. These prefixes are not necessary globally unique, they only have to be unique in the scope of one algorithm.

Segment Routing Example with MPLS

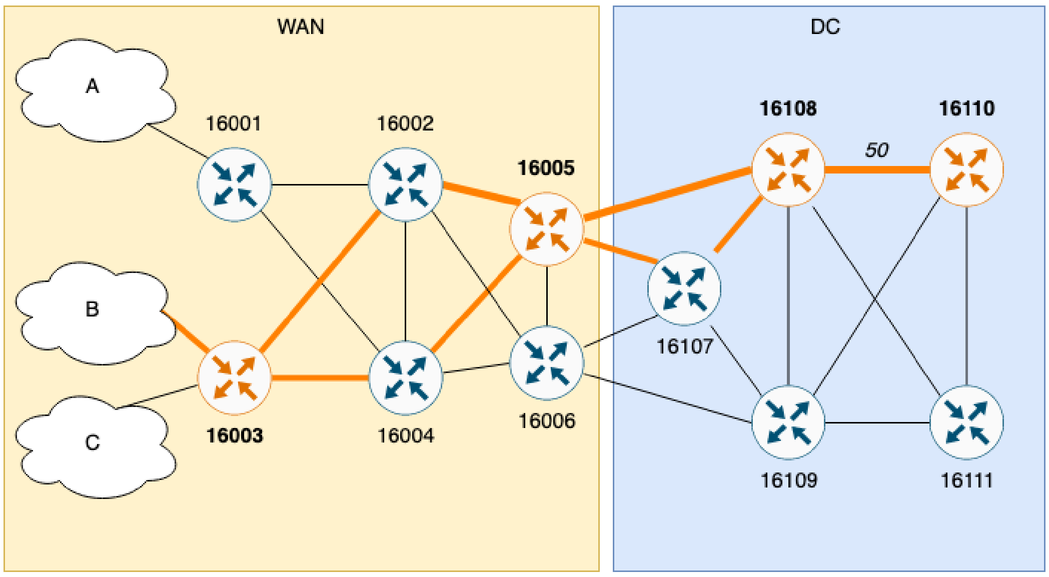


Figure 2.5.: SR Topology Example

For a better understanding of Segment Routing, an example is provided. The graphic 2.5 depicts two networks, a WAN and a DC (5), which are connected to each other. Both domains use Segment Routing. The WAN network uses an IGP-SR protocol, such as IS-IS. The DC domain uses BGP-SR for its paths. These are two different protocols that are combinable with Segment Routing. This illustrates that Segment Routing can work over different neighbouring network domains, which is a great advantage.

The network engineer now wants to lead the traffic flow from the neighbouring network `B` over certain routers to the device `16110` in the DC domain. This may be the path with lowest cost in the network, for example. Engineers can choose many different properties and combinations therefor to plan such a route.
In our example the engineer wants to hit the routers with the Prefix-SIDs `16003, 16005, 16108,` and `16110` for this traffic flow. It does not matter how the traffic reaches these routers as longs as the flow does through those devices. Such a specific path can be achieved thanks to Segment Routing and its Prefix-SIDs 2.1.1.

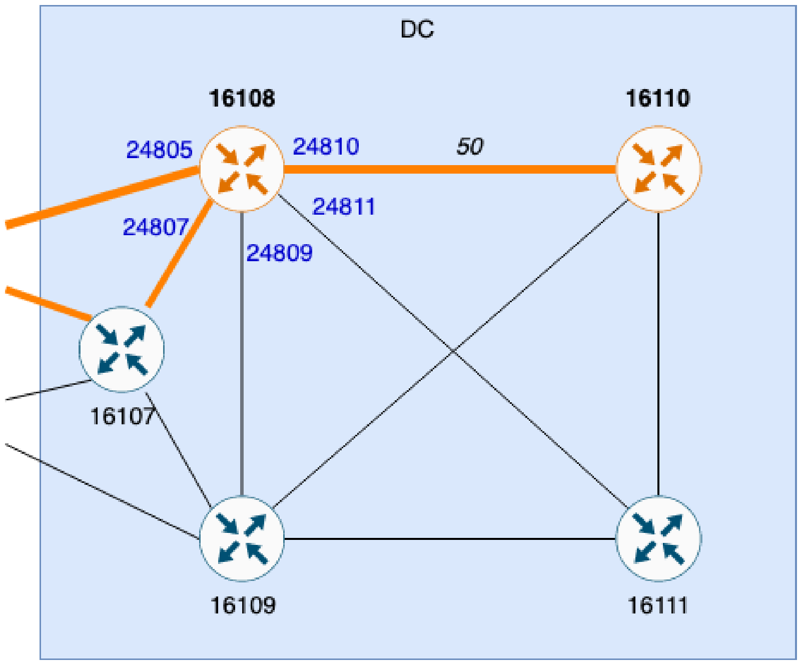


Figure 2.6.: SR Topology Example - Prefix-SIDs

Although the default cost metric is 10 on each link, the last link between `16108` and `16110` has the value `50`, see graphic 2.6. This does not support the concept of lowest cost path, which would actually calculate the path to go over the device `16111`, as this would result in a cost of 20 to reach `16110`. But nevertheless, the engineer wants to include this direct link to the router `16110` over the interface with Prefix-SID `24810` and not another one.

Thanks to Segment Routing's Prefix-SID 2.1.1 this decision to take a certain interface of a router can be implemented.

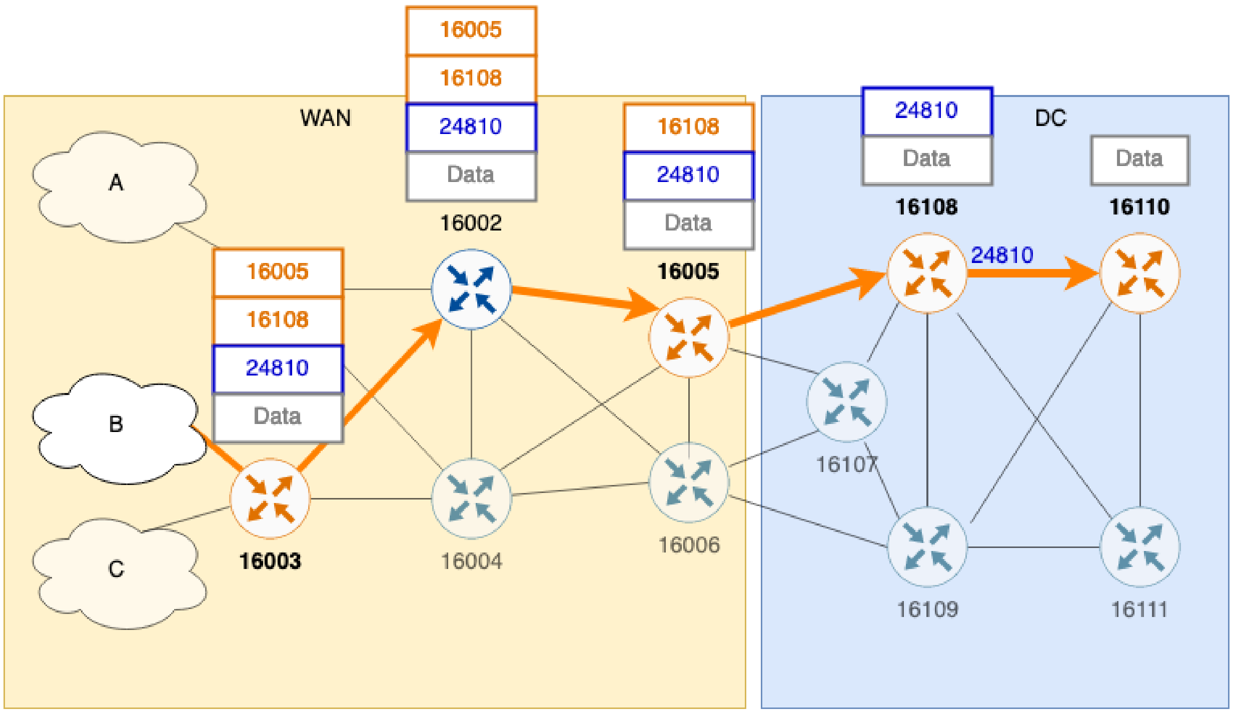


Figure 2.7.: SR Topology Example - Packets

The packets that get sent through the network to reach the destination 16110 include all the relevant information to follow the desired flow of the network engineer. Each router inspects the packet header to get the information to which device it should send the packet to.

In the graphic 2.7 the orange parts symbolise the Prefix-SIDs of the packet and the blue parts show the Adjacency-SIDs.
The first router of the segment domain that will process the packet from the neighbouring network B is the router 16003, the source. Although the device receives all relevant segments for the whole route, it will look only at the top one to decide where to send the packet. The router knows of two ways to send traffic on, over 16002 or 16004. It has a basic idea of where these routes can lead. The Prefix-SID in the top instruction says 16005, telling the router that the packet needs to be sent to that device over the most suitable route it knows. Router 16003 now checks which exit works best to reach 16005, which is via 16002 according to its routing table.
The next router, 16002, inspects the packet, compares the top segment with its own Prefix-SID, which is not the same, and so sends it to 16005 according to its routing table.
16005 checks the packet and sees the top most segment defines itself as a part of the route. It pops this instruction and reads the next one for further routing. Thanks to this process, the router sends the packet further to the device with the Prefix-SID 16108 fulfilling the constraint of the lowest cost path.
The router 16108 that lies in the network DC pops the top segment, its own id, as well and reads as next instruction the Prefix-SID. The router inspect its own interface configurations and look for the one that is configured with the Prefix-SID 24810. This interface is now used as the outgoing route, leading to the destinated router 16110.

Flexible Algorithm

The Flexible Algorithm (FlexAlgo) is a technology to make Segment Routing even more flexible and steer traffic individually according to packet needs. Thanks to the properties of FlexAlgo, Segment Routing can be used to differentiate traffic paths according to diverse use cases such as customer groups, latency needs, etc. While doing so it still uses the same underlaying infrastructure. This is handy for larger networks that have to deliver to different customer networks at the edges. This mechanism is called network slicing, or flexible algorithm (Internet-Draft [19]).

The advantages of Segment Routing with FlexAlgo include smaller configurations on the source router to allow a more flexible engineering of the network, while retaining the advantages source-routing brings to network trafficking. Additionally, load balancing of different routes will be possible, as MPLS remains supported by the Segment Routing. In case a link goes down, the devices will be able to reroute wihtin 50 milliseconds because of pre-configured backup paths. This can be calculated so fast because each device of a FlexAlgo already knows the path to all other devices that participate in this algorithm. Thanks to how Segment Routing is constructed the FlexAlgo can easily be added to an already existing network domain. [**sr-benefits**]

Algorithms

The network engineer can define an algorithm by entering the required configuration lines into the routing table of the device. The device can then participate in this algorithm. Multiple algorithms per device can be configured. The information for this is transported in the Sub-TLV part of IS-IS packet.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |      Type     |    Length     |Flex-Algorithm |  Metric-Type  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Calc-Type   |    Priority   |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          Sub-TLVs                             |
 +                                                               +
 |                            ...                                |

 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.8.: FlexAlgo Sub-TLV [59] in IS-IS Packet

- `Type`: 26
- `Length`: Length of the header, can be variable depending on the `Sub-TLVs`.
- `Flex-Algorithm`: Algorithm number.
  - `0 to 127`: Predefined algorithms. Algorithm `0` uses the Djikstra SPF.
  - `128 to 255`: Customizable algorithms
- `Metric-Type`: Calculation metric for path finding. 0 is IGP, 1 is link delay, 2 is TE default metric.
- `Calc-Type`:

- `Priority`: Specifies the priority of the FlexAlgo advertisement. Value can range between 0 and 255.
- `Sub-TLVs`: optional sub-TLVs.

Affinities

In Segment Routing the property affinities helps to direct the traffic flow per algorithm even more granularly. Affinity values, usually color names, are configured on the interfaces of a router. The FlexAlgos then defines how the different values are to be handled.
There are four different options to define the constraints for the algorithm:

- `includ-any` <color> - the FlexAlgo can traverse all links with this color
- `includ-all` <color> - the FlexAlgo must traverse all links with this color
- `exclude-any` <color> - the FlexAlgo must avoid all links with this color
- `exclude-SRLG` <risk-group> - the FlexAlgo must avoid all SRLG links with this risk-group

FlexAlgo Example

As an example the graphic 2.9 illustrates two configured algorithms in this network. The source router is A and the destination of both algorithms is router D. As D participates in both algorithms it has corresponding Prefix-SIDs configured. For algorithm `128` it has `16001` and for algorithm `129` it is known as `16004`.
The FlexAlgo `128` consists of router A, B and D (The yellow colored routers).
The FlexAlgo `129` consists of router A, B, C and D (The blue colored routers). Additionally, this algorithm is configured to avoid any links that are marked as `RED`, which in this example is the link between router C and D.

Now, if a packet is sent from A to D by FlexAlgo 128, it will traverse the path A, B, D. Alternatively, if the packet travels in FlexAlgo 129 it will take the path A, C, B, D.

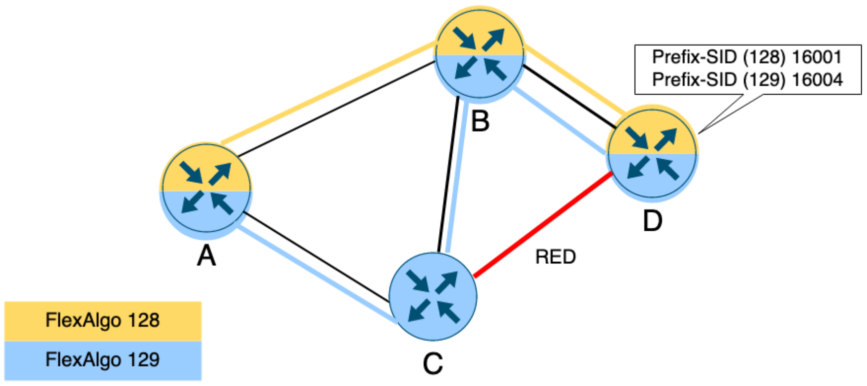


Figure 2.9.: Algorithms Example

2.2. YANG Model

The network devices send their configuration and traffic data in the format of YANG models to the Jalapeño'ss topology database (ArangoDB) and telemetry database (Influx). While telemetry data gets updated approximately every 10 seconds, topology data is event driven and is only persisted if changes have happpened.

After some comparisons of already used YANG models from other SR-App projects, the YANG model `Cisco-IOS-XR-clns-isis-cfg` with the latest revision 2021-04-02 [58] was used. This model contains all necessary information for FlexAlgo on IS-IS and the router image IOS XR can use it. The relevant parts of the model and a link to the full model can be found in the appendix 6.

Decoding

The YANG model sends most data in human-readable format that could easily be utilized in the application. Others, such as the affinity assignment to the interfaces are not only encoded in Base64, but are also stored in a certain way on the routers that has to be handled in the application.

Severin Dellsperger managed to reconstruct the Base64 encoding and how routers persit this data by analysing single packets in bit and byte format. In the appendix of this document hiss notes on decoding Affinities can be found 8.

His research notes 8 show that if the network engineer configures an Affinity with the number 10, the value AAAEAA== gets saved into the Jalapeno's database.
To understand this encoding, the binary value in the network packet itself had to be analysed with Wireshark. In it the bit of the 10. place (starting by first position as 0) is set. This binary string will then be encoded with Base64 which results in the final value of AAAEAA==.

Overview of encoding process, example 1:

1. Affinity "red" with bit-position `10` is defined on an interface
2. Router saves bit-position into binary value of
   `00000000 00000000 00000100 00000000`
   (*the 1 is at the 10. position counted from the right*)
3. Jalapeño encodes received binary with Base64 and stores the result into database as
   `AAAEAA==`

Bit-position values that are bigger than 31 are a bit more complicated, as the new block for the binaries will be set at the end of the previous blocks of zeroes according to the Little Endian principle.

Overview of encoding process, example 2:

1. Affinity purple with bit-position `66` is defined on an interface
2. Router saves bit-position into binary value of
   `00000000 00000000 00000000 00000000`
   `00000000 00000000 00000000 00000000`
   `00000000 00000000 00000000 00000100`
3. Jalapeño encodes received binary with Base64 and stores the result into database as
   `AAAAAAAAAAAAAAAE`

The properties Sub-TlVs of the FlexAlgos are be constructed with these bit-maps. In the transmitted BGP-LS packets it's viewable that for each of these 4 Sub-TlVs a block of zeroes and ones get send, which represent the different bit-maps. For example, if the FlexAlgo gets configured to have two Affinities ino the Sub-TlV `Include-Any`, like 10 and 13, the glsbgp-LS will look as follows in the packet's specific section.

`00000000 00000000 00100100 00000000` which is a combination of:

```
00000000 00000000 00000100 00000000 = 10

00000000 00000000 00100000 00000000 = 13
```

As one can see there are two 1s set, which build the information that the Affinity 10 (set 1 on 10.place from left side) and Affinity 13 will be used for the corresponding Sub-TlV.

2.3. Project Related Network Technologies

2.3.1. Underlaying Network

The network that was used during this project is a virtual one provided by INS. It is built in the application LTB and uses the Cisco IOS XR image to represent the network devices. The configurations on the P- and PE-routers can be seen in the Appendix 9. This was the basis for the team to build the SR-App FlexAlgo application.
For this project the network is configured with the protocols IS-IS as control plane protocol and for the data plane it uses the protocol MPLS.

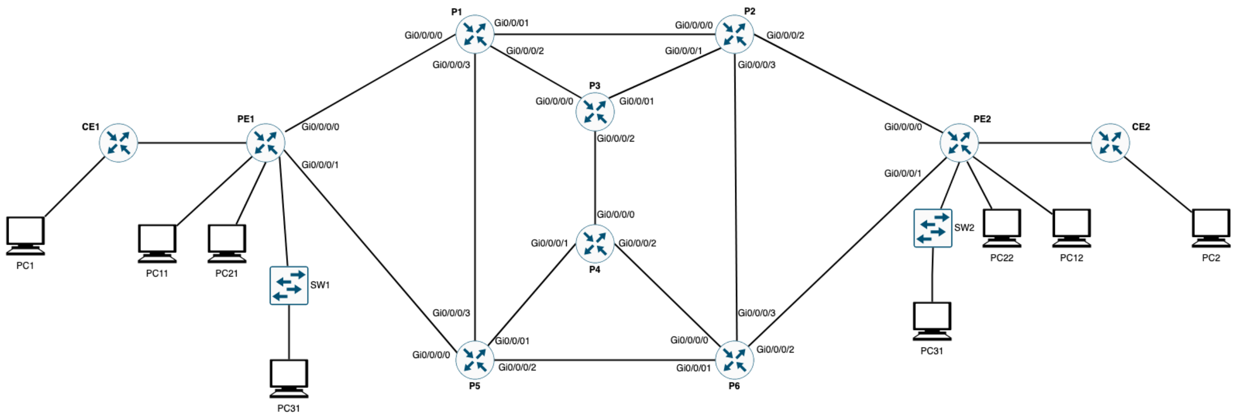


Figure 2.10.: Network Topology in LTB

2.3.2. Jalapeño

"A cloud-native infrastructure platform to enable development of network services." [7]

Jalapeño is a Cisco proprietary open source software that collects network data into a streaming event handling platform. From this platform the data is processed and persisted in two databases, depending on the data type. Jalapeño differs between two data types, telegraf and topology. Telegraf data is operational data from the network and is saved in a time series database. Telegraf data is renamed to telemetry data in the Jalapeño API Gateway. Topology data is saved into a Graph database and represents the static data of the network, ie how the topology of the network looks like and the configurations. [7]

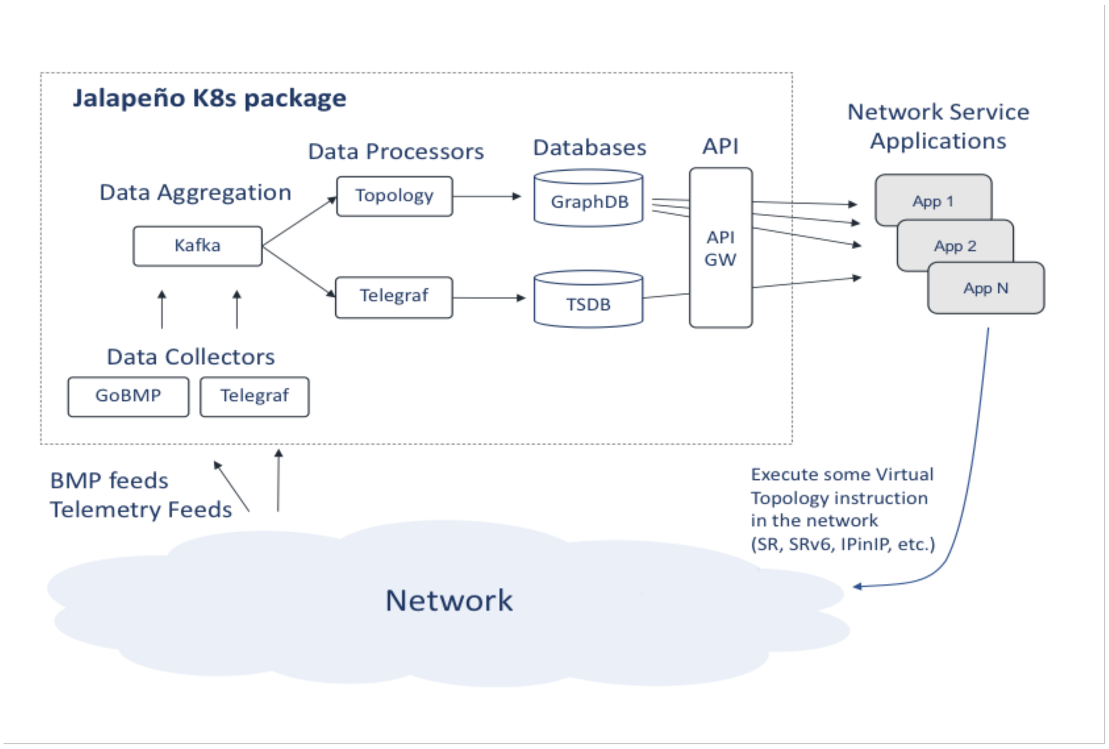


Figure 2.11.: Jalapeño architecture

2.3.3. Jalapeño API Gateway

The Jalapeño API Gateway offers an interface for applications in the SR-App series to connect to via gRPC client. It provides filtered and aggregated data from the Cisco Jalapeño application that collects network information for Segment Routing. It exposes a request based service that returns one time responses of selected data. Alternative apps can connect to the subscription service that builds a bidirectional connection to stream data updates through. The software is developed and maintained by the INS, which provided a running instance for this project. [27]

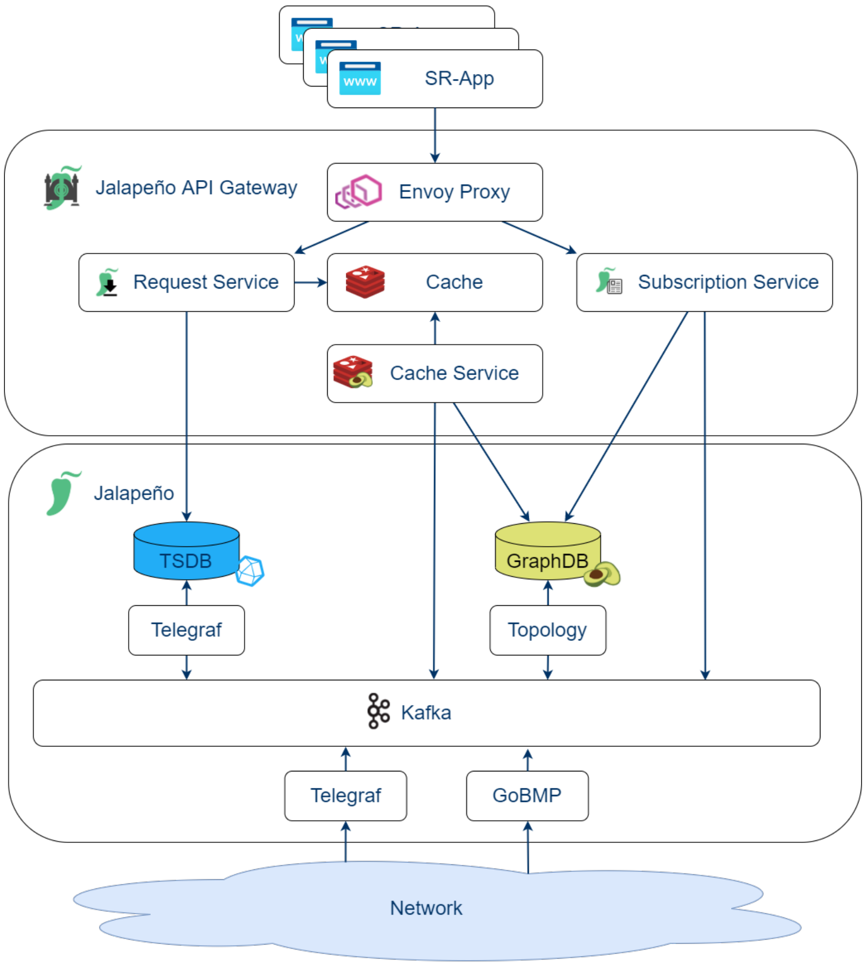


Figure 2.12.: Jalapeño API Gateway architecture

3. Results

This chapter will show what was achieved during this research project. The Distinction section 3.1 will provide a short summary of what was expected and which limitations were met during the project. In Achievements 3.2 an overview of the planned functionality is given and the status of implementation that were reached during this project. Finally, in the section Implementation 3.3 a deeper explanation of interesting technical aspects of this work is given.

3.1. Distinction

The main core of this project was a prototype for the SR-App FlexAlgo. It was decided by the team mebers in agreement with the project advisors that this prototype would be a mvp of the `network read` microservice. This included a request based call to the Jalapeño API Gateway to access all FlexAlgo specific data. Should time allow the gRPC streaming service would also be implemented. As frontend an API with at least one path returning a server side rendered html template would suffice. Should the time allow the implementation of a graphical interface to show the network and its containing FlexAlgos would be added. Additionally research of the FlexAlgo technology and its functionality was expected in order to correctly implement the transition of all relevant data through the gateway.

During the construction phase it became clear that the Jalapeño API Gateway will need further adjustments to make it useable for the FlexAlgo technology. As such it was decided together with the advisors that a direct connection to the gateway's database will be implemented instead of a gRPC connection. This has no impact on the functionality of the software but will need to be kept in mind for further considerations of the project.

As this project was originally planned to continue in a bachelor thesis, where the complete application would be developed, the complete planning of requirements, the rough architecture, etc. was expected. This would ensure a fast elaboration phase and longer construction phase in the bachelor thesis project. The team members reached the decision not to continue with this project during the construction phase and thus had no great impact on this.

3.2. Achievements

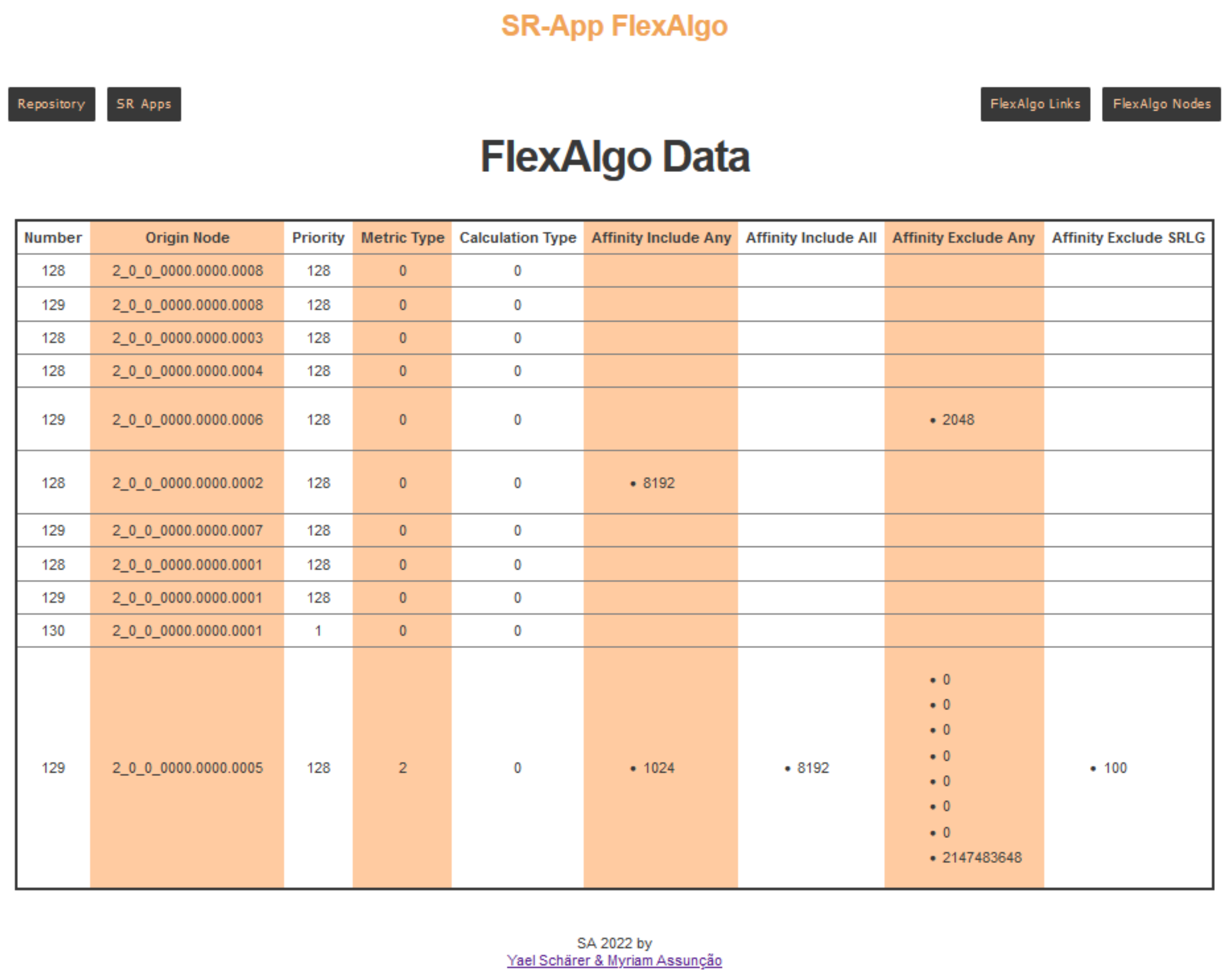
3.2.1. Functional Requirements

In this section the final product of this project will be discussed. To better illustrate them the use cases of section 1.1 are briefly addressed.

UC00 See a list of FlexAlgo information

This use case was planned as a simple frontend with server side rendered html to ensure the finished prototype could be measured for suitability. With this a check could be made that all necessary FlexAlgo information was accessed during the process.

During the study of the YANG models and how FlexAlgo definitions are distributed in the network, it was made clear that collecting all relevant information for FlexAlgos into one place would be counter productive. Generally FlexAlgos themselves are defined on nodes as a list of which algorithms are hosted on said object. Affinities on the other hand are configured on interfaces but are found in the links collection of the gateway's database.
As such the decision to split this use cases into three API calls was made, to reduce unnecessary code overhead and ensure an easier transition to the graphical React.js frontend.



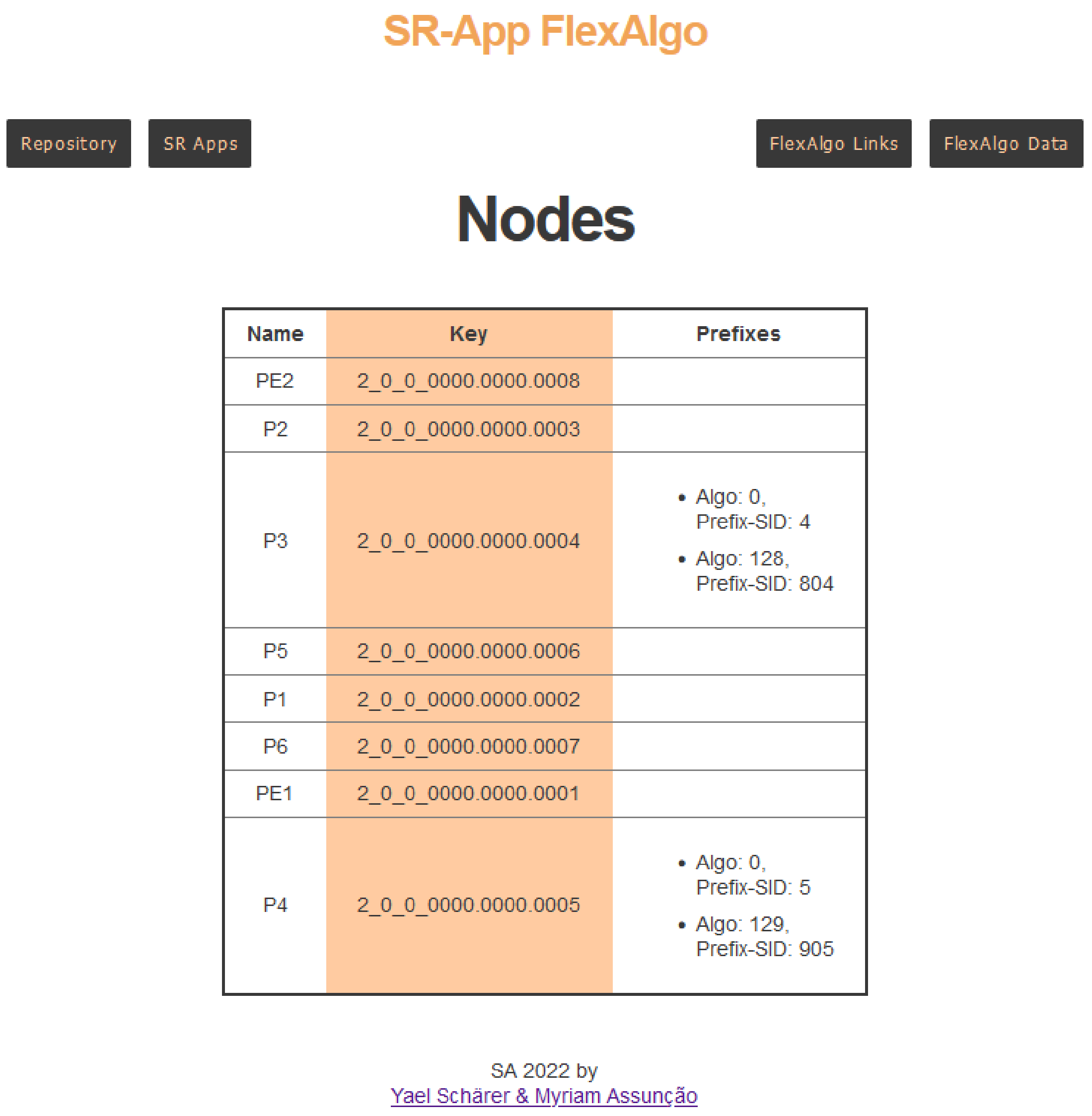
FlexAlgo Data

Number	Origin Node	Priority	Metric Type	Calculation Type	Affinity Include Any	Affinity Include All	Affinity Exclude Any	Affinity Exclude SRLG
128	2_0_0_0000.0000.0008	128	0	0				
129	2_0_0_0000.0000.0008	128	0	0				
128	2_0_0_0000.0000.0003	128	0	0				
128	2_0_0_0000.0000.0004	128	0	0				
129	2_0_0_0000.0000.0006	128	0	0			• 2048	
128	2_0_0_0000.0000.0002	128	0	0	• 8192			
129	2_0_0_0000.0000.0007	128	0	0				
128	2_0_0_0000.0000.0001	128	0	0				
129	2_0_0_0000.0000.0001	128	0	0				
130	2_0_0_0000.0000.0001	1	0	0				
129	2_0_0_0000.0000.0005	128	2	0	• 1024	• 8192	• 0 • 0 • 0 • 0 • 0 • 0 • 0 • 2147483648	• 100

SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.1.: HTML View with FlexAlgo Data

The prototype now provides the following API calls for FlexAlgo information:
`FlexAlgo Definitions`. Here all global FlexAlgo configurations with their origin nodes can be found.



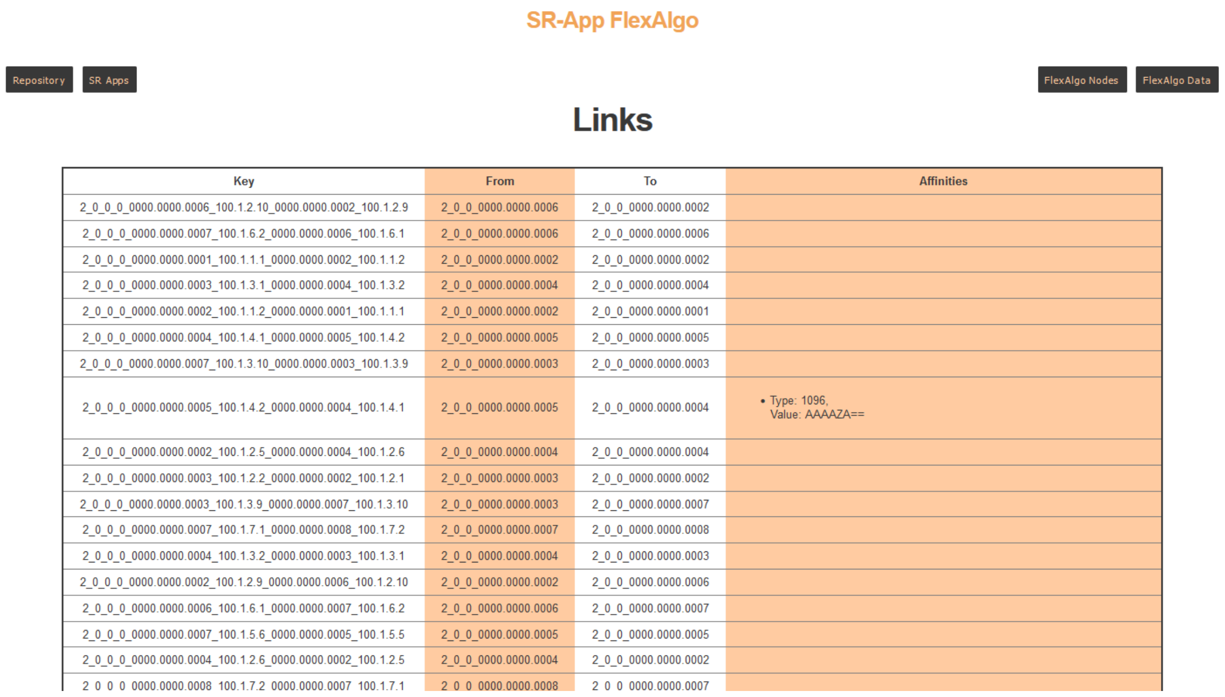
Nodes

Name	Key	Prefixes
PE2	2_0_0_0000.0000.0008	
P2	2_0_0_0000.0000.0003	
P3	2_0_0_0000.0000.0004	• Algo: 0, Prefix-SID: 4 • Algo: 128, Prefix-SID: 804
P5	2_0_0_0000.0000.0006	
P1	2_0_0_0000.0000.0002	
P6	2_0_0_0000.0000.0007	
PE1	2_0_0_0000.0000.0001	
P4	2_0_0_0000.0000.0005	• Algo: 0, Prefix-SID: 5 • Algo: 129, Prefix-SID: 905

SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.2.: HTML View with Node Data

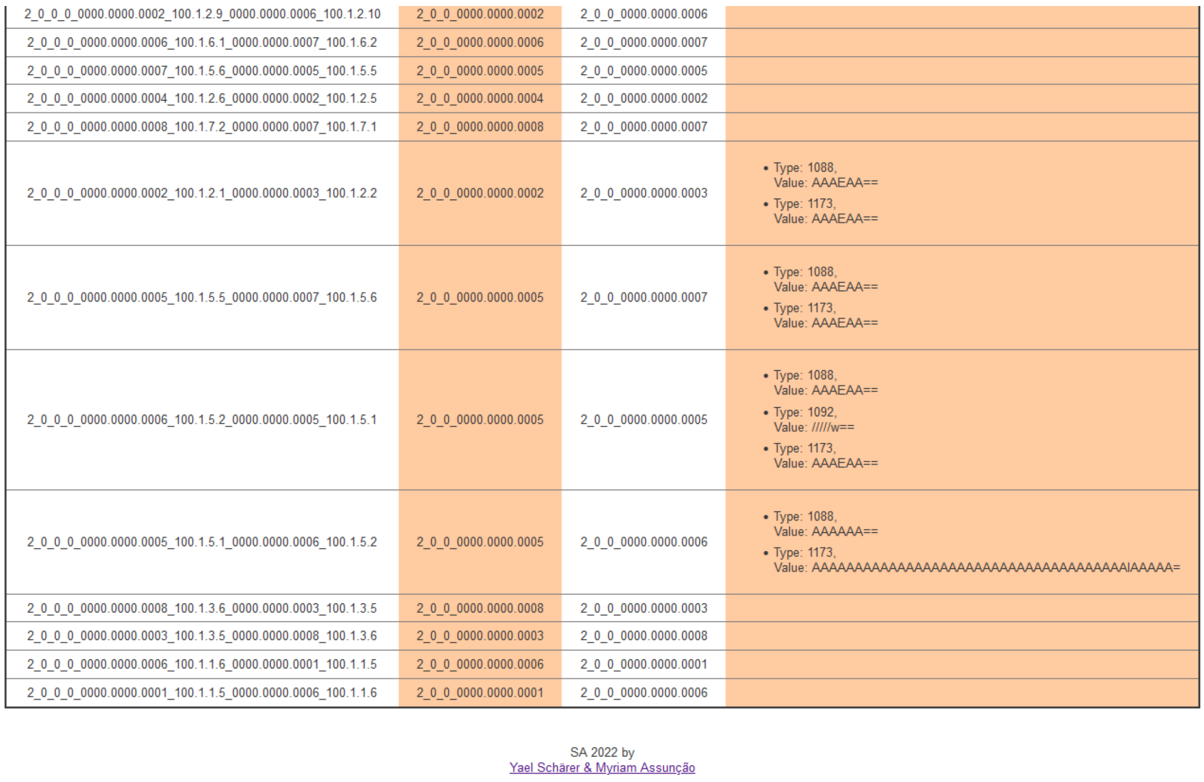
Here the nodes and their FlexAlgo specific data like prefix values can be found. Additionally general node data like longitude and latitude is displayed. In a further project more data can be accessed from the network, for example IP addresses for the use case "UC14 Show Router Information".



Links

Key	From	To	Affinities
2_0_0_0_0000.0000.0006_100.1.2.10_0000.0000.0002_100.1.2.9	2_0_0_0000.0000.0006	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0007_100.1.6.2_0000.0000.0006_100.1.6.1	2_0_0_0000.0000.0006	2_0_0_0000.0000.0006	
2_0_0_0_0000.0000.0001_100.1.1.1_0000.0000.0002_100.1.1.2	2_0_0_0000.0000.0002	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0003_100.1.3.1_0000.0000.0004_100.1.3.2	2_0_0_0000.0000.0004	2_0_0_0000.0000.0004	
2_0_0_0_0000.0000.0002_100.1.1.2_0000.0000.0001_100.1.1.1	2_0_0_0000.0000.0002	2_0_0_0000.0000.0001	
2_0_0_0_0000.0000.0004_100.1.4.1_0000.0000.0005_100.1.4.2	2_0_0_0000.0000.0005	2_0_0_0000.0000.0005	
2_0_0_0_0000.0000.0007_100.1.3.10_0000.0000.0003_100.1.3.9	2_0_0_0000.0000.0003	2_0_0_0000.0000.0003	
2_0_0_0_0000.0000.0005_100.1.4.2_0000.0000.0004_100.1.4.1	2_0_0_0000.0000.0005	2_0_0_0000.0000.0004	• Type: 1096, Value: AAAAZA==
2_0_0_0_0000.0000.0002_100.1.2.5_0000.0000.0004_100.1.2.6	2_0_0_0000.0000.0004	2_0_0_0000.0000.0004	
2_0_0_0_0000.0000.0003_100.1.2.2_0000.0000.0002_100.1.2.1	2_0_0_0000.0000.0003	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0003_100.1.3.9_0000.0000.0007_100.1.3.10	2_0_0_0000.0000.0003	2_0_0_0000.0000.0007	
2_0_0_0_0000.0000.0007_100.1.7.1_0000.0000.0008_100.1.7.2	2_0_0_0000.0000.0007	2_0_0_0000.0000.0008	
2_0_0_0_0000.0000.0004_100.1.3.2_0000.0000.0003_100.1.3.1	2_0_0_0000.0000.0004	2_0_0_0000.0000.0003	
2_0_0_0_0000.0000.0002_100.1.2.9_0000.0000.0006_100.1.2.10	2_0_0_0000.0000.0002	2_0_0_0000.0000.0006	
2_0_0_0_0000.0000.0006_100.1.6.1_0000.0000.0007_100.1.6.2	2_0_0_0000.0000.0006	2_0_0_0000.0000.0007	
2_0_0_0_0000.0000.0007_100.1.5.6_0000.0000.0005_100.1.5.5	2_0_0_0000.0000.0005	2_0_0_0000.0000.0005	
2_0_0_0_0000.0000.0004_100.1.2.6_0000.0000.0002_100.1.2.5	2_0_0_0000.0000.0004	2_0_0_0000.0000.0002	
2 0 0 0 0000.0000.0008 100.1.7.2 0000.0000.0007 100.1.7.1	2 0 0 0000.0000.0008	2 0 0 0000.0000.0007	

Figure 3.3.: HTML View with Link Data - Part 1



2_0_0_0_0000.0000.0002_100.1.2.9_0000.0000.0006_100.1.2.10	2_0_0_0000.0000.0002	2_0_0_0000.0000.0006	
2_0_0_0_0000.0000.0006_100.1.6.1_0000.0000.0007_100.1.6.2	2_0_0_0000.0000.0006	2_0_0_0000.0000.0007	
2_0_0_0_0000.0000.0007_100.1.5.6_0000.0000.0005_100.1.5.5	2_0_0_0000.0000.0005	2_0_0_0000.0000.0005	
2_0_0_0_0000.0000.0004_100.1.2.6_0000.0000.0002_100.1.2.5	2_0_0_0000.0000.0004	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0008_100.1.7.2_0000.0000.0007_100.1.7.1	2_0_0_0000.0000.0008	2_0_0_0000.0000.0007	
2_0_0_0_0000.0000.0002_100.1.2.1_0000.0000.0003_100.1.2.2	2_0_0_0000.0000.0002	2_0_0_0000.0000.0003	• Type: 1088, Value: AAAEAA== • Type: 1173, Value: AAAEAA==
2_0_0_0_0000.0000.0005_100.1.5.5_0000.0000.0007_100.1.5.6	2_0_0_0000.0000.0005	2_0_0_0000.0000.0007	• Type: 1088, Value: AAAEAA== • Type: 1173, Value: AAAEAA==
2_0_0_0_0000.0000.0006_100.1.5.2_0000.0000.0005_100.1.5.1	2_0_0_0000.0000.0005	2_0_0_0000.0000.0005	• Type: 1088, Value: AAAEAA== • Type: 1092, Value: /////w== • Type: 1173, Value: AAAEAA==
2_0_0_0_0000.0000.0005_100.1.5.1_0000.0000.0006_100.1.5.2	2_0_0_0000.0000.0005	2_0_0_0000.0000.0006	• Type: 1088, Value: AAAAAA== • Type: 1173, Value: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAA=
2_0_0_0_0000.0000.0008_100.1.3.6_0000.0000.0003_100.1.3.5	2_0_0_0000.0000.0008	2_0_0_0000.0000.0003	
2_0_0_0_0000.0000.0003_100.1.3.5_0000.0000.0008_100.1.3.6	2_0_0_0000.0000.0003	2_0_0_0000.0000.0008	
2_0_0_0_0000.0000.0006_100.1.1.6_0000.0000.0001_100.1.1.5	2_0_0_0000.0000.0006	2_0_0_0000.0000.0001	
2_0_0_0_0000.0000.0001_100.1.1.5_0000.0000.0006_100.1.1.6	2_0_0_0000.0000.0001	2_0_0_0000.0000.0006	

SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.4.: HTML View with Link Data - Part 2

Here all links with their conected nodes and their FlexAlgo data, i.e. the affinities assigned, can be found.

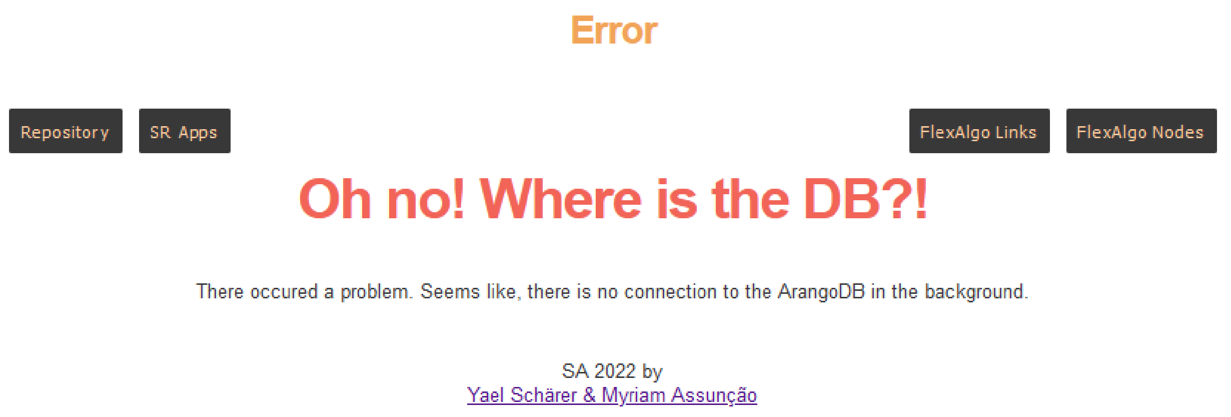
Conclusion: IMPLEMENTED with adjustments to the definition

UC01 Live Updates

Before the construction phase it was agreed with the advisors that live updates in the SA prototype would be implemented as page refreshes. Currently each API call will perform a complete refresh of the live data in the backend. As this will not be the case in the final version of the app, as it will work with gRPC subscription functionality, additional API calls were implemented. These update calls will simulate an updated object being sent to the application. When such an update arrives the application checks if it has initialized data, checks the global cache 4.2.0.5 and then performs an update on the current data. Is no data in either cache or application then a full refresh is performed. In a further step this can be built upon to implement a push notification functionality that will handle live updates automatically instead of having to manually trigger a refresh.
Conclusion: IMPLEMENTED with additional functionality simulated.

UC02 Failure to Connect to Network

Multiple failure message systems are implemented in the prototype. As was shown before one set of API calls returns server side rendered html. This will return a specific html template to show error messages. Different types of error, like connection to database or gRPC connection loss versus internal server error are specified in the template. As such a user will have a better view of what exactly has happened and if further actions are needed.


Error

Repository SR Apps FlexAlgo Links FlexAlgo Nodes

Oh no! Where is the DB?!

There occured a problem. Seems like, there is no connection to the ArangoDB in the background.

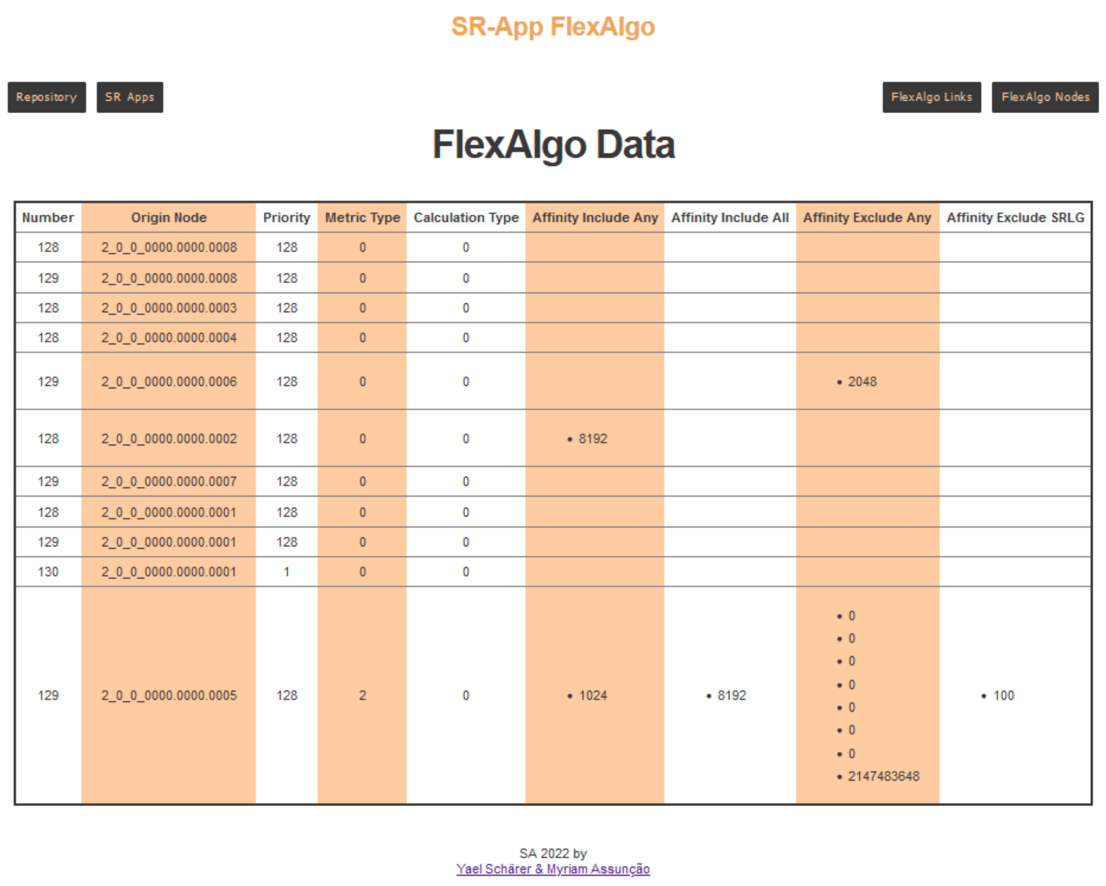
SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.5.: HTML view Error Message

Additionally to the html, API calls for the React.js frontend were implemented that return json objects with more standard HTTP codes. If an error occurs here the specific HTTP code is set and an error message returned specifying the problem to be handled in a more intelligent frontend. For further information consult the API documentation 3.6
Conclusion: IMPLEMENTED

UC03 Show List of All Running FlexAlgos

As previous shown the API call `/flexGetAlgoDefinitions` returns a comprehensiv list of the FlexAlgos and their configurations.



FlexAlgo Data

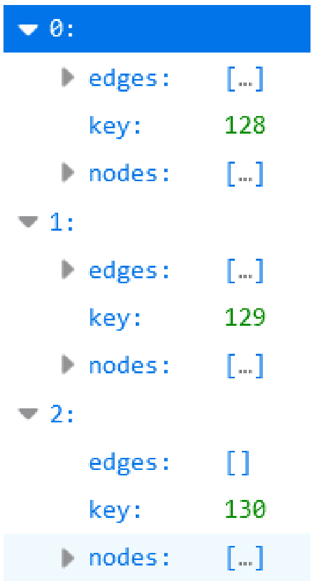
Number	Origin Node	Priority	Metric Type	Calculation Type	Affinity Include Any	Affinity Include All	Affinity Exclude Any	Affinity Exclude SRLG
128	2_0_0_0000.0000.0008	128	0	0				
129	2_0_0_0000.0000.0008	128	0	0				
128	2_0_0_0000.0000.0003	128	0	0				
128	2_0_0_0000.0000.0004	128	0	0				
129	2_0_0_0000.0000.0006	128	0	0			• 2048	
128	2_0_0_0000.0000.0002	128	0	0	• 8192			
129	2_0_0_0000.0000.0007	128	0	0				
128	2_0_0_0000.0000.0001	128	0	0				
129	2_0_0_0000.0000.0001	128	0	0				
130	2_0_0_0000.0000.0001	1	0	0				
129	2_0_0_0000.0000.0005	128	2	0	• 1024	• 8192	• 0 • 0 • 0 • 0 • 0 • 0 • 0 • 2147483648	• 100

SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.6.: HTML view FlexAlgo Configurations

To see a simpler list of all algorithm keys, the `/getAlgoNumbers` call returns a list of algo numbers currently configured on the network. ["128", "129", "130"]

Additionally, in consideration of the React.js frontend, an API call with all algorithms with their nodes and links was implemented.



```
▼ 0:
    ▶ edges:   […]
      key:     128
    ▶ nodes:   […]
▼ 1:
    ▶ edges:   […]
      key:     129
    ▶ nodes:   […]
▼ 2:
      edges:   []
      key:     130
    ▶ nodes:   […]
```

Figure 3.7.: JSON return of FlexAlgo Graphs

Conclusion: IMPLEMENTED

UC04 Logging Application
As the result of this project is a prototype, logging of application errors are merely written to the standard error stream and can therefore be found in the console if run locally or the Kubernetes pod if run in the namespace. This can easily be built upon in a further project to properly log and persist these streams to a wide variety of logging solutions, depending on customer needs. The current solution also optimally covers integration into Kubernetes and scaling. For further information see chapter Logging 3.7
Conclusion: IMPLEMENTED

UC05 View Topology

The time management of this project was sufficient enough that optional use cases could be implemented. As such overview of the topology in a graphical representation is possible. This was done utilizing the demo application of Michel Bongard. This project merely adjusted the existing solution to call our API and optimally presents our data. An added functionality of seeing a list of currently running FlexAlgos and being able to select one to see its graph highlighted was also implemented.

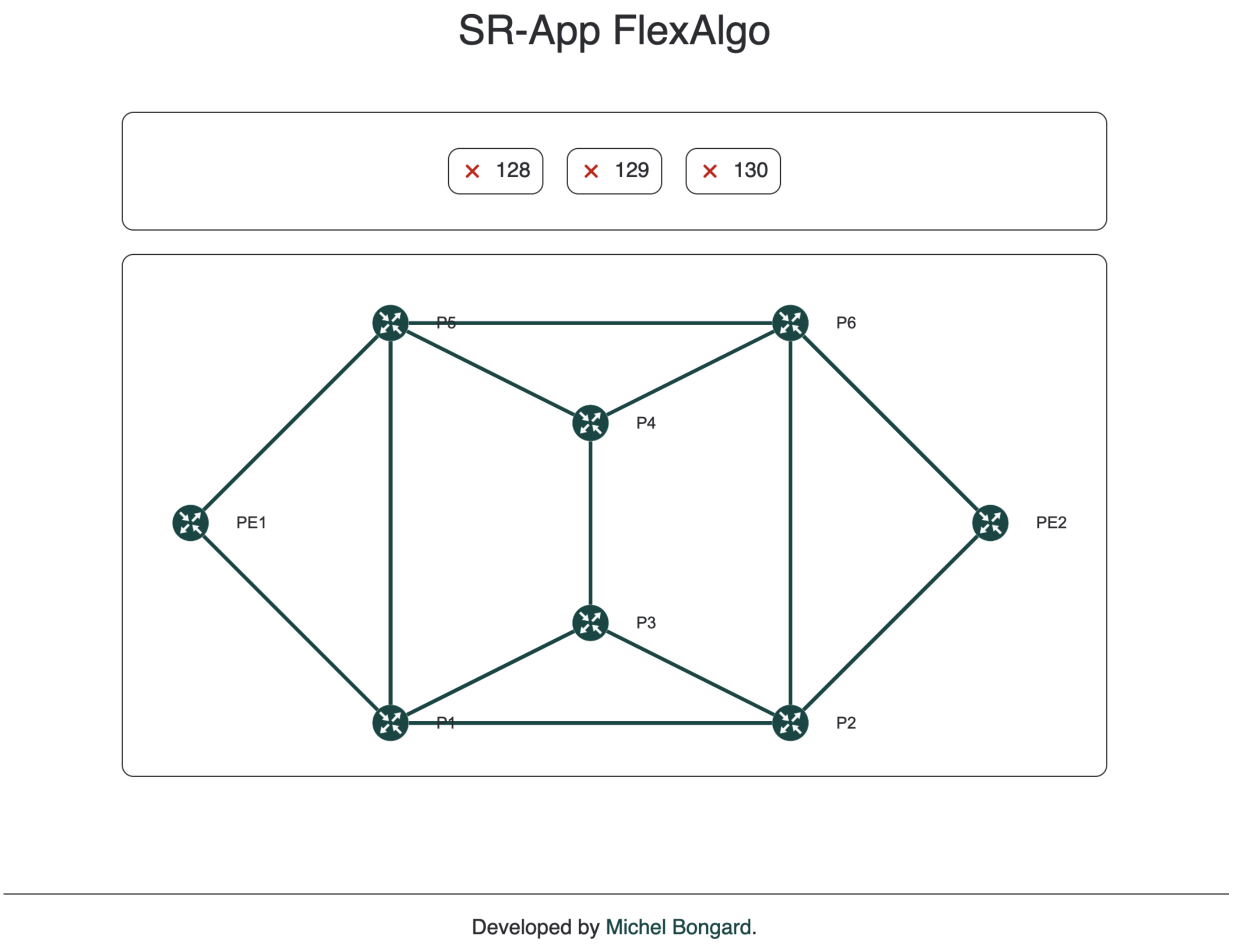


Figure 3.8.: React.js Graph Topology

To be able to utilize this demo multiple API calls were implemented that can be found in the Swagger documentation under the tag `API Paths` or in the API documentation 3.6.
Conclusion: IMPLEMENTED

UC06 Graphical Representation of One FlexAlgo

With the functionality of the demo frontend already discussed beforehand a representation of one algorithm is indeed possible. Clicking on the corresponding button will highlight all nodes that have a particular algorithm definition and their connecting links. Algorithm numbers list and topology are updated live with a page refresh.

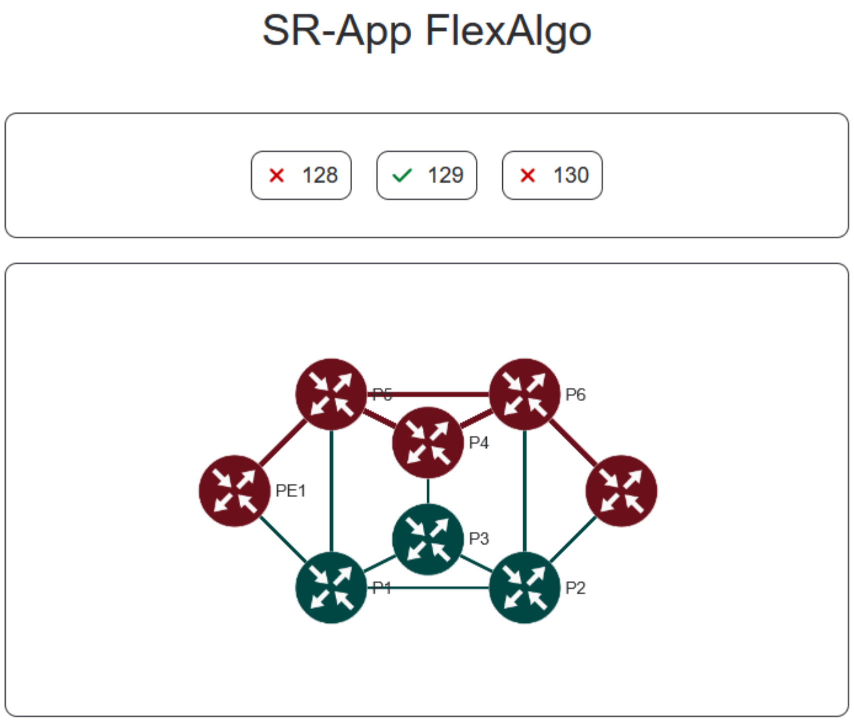


Figure 3.9.: React Graph Topology - FlexAlgo 129 Highlighted

Conclusion: IMPLEMENTED

UC07 Graphical Representation of All FlexAlgos

This use case proved to be troublesome not in the implementation, as this would indeed be doable with the existing functionality of highlighting paths. But to make this not only visually appealing but also understandable and bringing additional value to the existing functionalities of UC06 and UC05 proved to be challenging. It was decided by the team members not to implement this use case in the short time remaining for the project. Especially on large networks with a lot of FlexAlgos this could prove very confusing for the user. First developing UC15 could prove to be illuminating in this regard, as this would allow the user to select only a small subset of all paths depending on filter value. A corresponding API call with query parameters was prepared and experimented with.
Conclusion: NOT IMPLEMENTED, FROZEN

UC14 Show Router Information

In a happy coincidence, steps to implement this use case are already under way with the prototype. Functionality for getting information of a router is implemented, though at the moment constrained to names, algorithms, prefixes and geological data. Any aditional data in the objects nodes and node_coordinates can easily be added and displayed with minimal overhead. Merely

the implementation of this functionality in the graphical frontend is not yet present.
Conclusion: PARTLY IMPLEMENTED

UC15 Filter Topology

In the study of the feasibility of UC07 filter functionality was added to the `/getAlgoNumbers` API call. With query parameters added to the url a subset of the algorithms can be called. At the moment functionality is limited to algorithms defined on specified nodes. How such a call should look like can be found as comment in the code of the `flexalgo_route` class in the `get_algo_numbers` method.
Conclusion: NOT IMPLEMENTED, FIRST STEPS MADE

UC17 Historic Analysis

With the addition of caching to the application, a rudimentary functionality of sending data to other systems is implemented and proved to be minimally impactful on the performance. This may be helpful should this use case be implemented.
Conclusion: NOT IMPLEMENTED

3.2.2. Non-Functional Requirements

Functionality

`Security:` BA specific requirements, not implemented in prototype.

`Accuracy:` Connection to the gateway is currently not feasible, see Distinction 3.1. All the same, the application sends the newest status of network back and allows the user to see the most up-to-date information.
Conclusion: FULFILLED

`Interoperability:` Connection to the gateway is currently not feasible, see chapter Distinction 3.1. Application works with the gateway's database directly. Network data is generated from virtual IOS XR devices. Configuration is a BA requirement.
Conclusion: FULFILLED

Reliability

`Fault Tolerance:` The software was made fault tolerant by implementing error handling. A differentiation was made between connection failure, data set problems, missing fields handling and internal server handling should the application itself have a problem.
Conclusion: FULFILLED

`Maturity:` The software is developed to run stable and error handling to prevent unexpected shutdown is implemented.
Conclusion: FULFILLED

`Recoverability:` The cloud native twelve-factor methodology was used. How it was implemented can be seen in chapter 3.1.
Conclusion: FULFILLED

Usability

These requirements were predefined for the BA project and were not implemented in the prototype.

Performance
`Data Integrity:` The newest data from the network is shown with a page refresh, as such users operate on up to date information.
Conclusion: FULFILLED

`Scalability:` Twelve-factor methodology was used and as such scalability is guaranteed. A further analysis for scaling needs in relation to large networks may be advisable, to ensure the application scales the computation of the graph properly. This will have to be done in conjunction with the Jalapeño API Gateway. This was agreed upon with the advisors.
Conclusion: FULFILLED AS AGREED

`Resource Behaviour:` BA specific requirements.

`Time Behaviour:` The computation of the algorithm paths (`/getAlgos` path) has the highest demand on performance and time management. As such there was intensive testing done on this path and the others. Currently the graphs are calculated in under 3 seconds with a 1'000 nodes heavy network, with 1'000 links and 100 FlexAlgos defined. See testing protocols for further information 5. The requirement for telemetry data is no longer needed, as the FlexAlgorithm data istopology data only, so no telemetry data is required.
Conclusion: FULFILLED

Maintainability
`Analysability:` Error handling is implemented in the application. If errors are caught in the application a message is logged to the standard error stream and a descriptive message is sent to the frontend. It is possible to send HTTP specific codes instead of messages to the frontend too. For further information about the logging process see section Logging 3.7.
Conclusion: FULFILLED

`Testability:` All functionality of the software is sufficiently tested in unit and integration tests. System and acceptance tests were performed in each sprint review and can be found in here 5. Continuous integration with pipeline with testing functionality is implemented and allows only succesfully tested code to be merged.
Conclusion: FULFILLED

3.3. Implementation

This section will discuss especially interesting aspects of the prototype.

3.3.1. Update Functionality

As previously discussed the Jalapeño API Gateway could not be implemented in this project. The request based architecture currently implemented connects to the ArangoDB and calls the whole dataset upon each `/get` API call. This does not represent the gRPC functionality of subscription the gateway would provide. These subscriptions means that a gRPC client can "observe" certain data structures via the gateway and always get notified if changes to them happen. As such changed objects get sent to the client application that then can apply its business logic to them. To still ensure the prototype will provide a significant simulation of the final product, update functionality was built into the system. With `/update` API calls a user can simulate updated data, mocked in the application itself, being fed into the system and receives a full set of the topology data with the updated object back. Generally the node with the `_key` "2_0_0_0000.0000.0008" was mocked. Therefore node data, FlexAlgo configurations and links with that node as either starting or end point can be observed to change certain properties. Which values change can be seen in the service `UpdateData` where the function `randint` was used to simulate changes. Further information about the API can be seen in the API documentation 3.6.

3.3.2. Live Model

In the future the SR-App will receive updates from the Jalapeño API Gateway and push these changes to the frontend. The architecture of the protoype was chosen to make migration to this push architecture easy to implement. To not only accommodate this architectural change but also ensure performance that will be under an acceptable limit, an instantiated topology was implemented. This live model of the data on the system is currently initialized at the first request to the application. In the future this may be initialized at start up. This will ensure that updates to the topology can be computed fast and will not have to reload the whole network data. Additionally, in the future an `Observer Pattern` can be implemented on the live model, that pushes the updates to the frontend after the gRPC updates are written in the model. This isolates the backend communication to the gateway completely from the frontend communication and ensures easily replacable communication functionality.

3.3.3. Caching

As it is not possible to ensure the production application will have global variables syncronized over the different scaled applications, caching was built into the project. Each production application of `network read` accesses the cache upon receiving an update request and imports the cached data. As such every application instance will work on the same data set and remain consistent. Further information about this can be found in the conclusion chapter Caching 4.2.0.5.

3.3.4. Pipeline

There is a single pipeline script which handles all required jobs for this project. To be able to run the tests of glspytest on the project it was required to deposit the url, username and password to the ArangoDB into the GitLab repository to follow the basic security aspect of publishing those credentials into a repository .

The pipeline consists of those stages in this order:

- `documentation:` builds the LaTeX documentation
- `formatting:` runs Black to format Python code
- `check-static-types:` runs MyPy to check for static types in Python code
- `test:` runs Pytest and check the unit and integration tests
- `sonar-scanner:` uploads output from previous steps and own checks to SonarQube frontend
- `build-image:` generates the Docker image file out of the providen application code
- `deployment:` updates Kubernetes Deployment with the new Docker image

For further insights into the pipeline, one can consider the file in the Appendix 10.1 or directly the original file `.gitlab-ci.yml` from the GitLab repository.

Credentials

To be able to run the tests of glspytest on the project it was required to deposit the URL, username and password to the ArangoDB into the GitLab repository. This was done to follow the basic security guidelines of public accessability of those credentials.

Workflow

If someone wants to create a pdf out of the LaTeX files, one can create branch with the name that starts with doc_*. This will kick off the pipeline script part with the single job of running the Makefile in which the required rubber, pdflatex and makeglossaries commands are listed. The two generated documents thesis.pdf and management_document.pdf can then be retrieved under the artifacts in GitLab's pipeline.

In case there is an open merge request and the branch starts with dev_*, which are sub-branches of the main dev-branch for the final production, the jobs of `formatting, static-test, pytest, sonarqube scanner` will be ran before a merge can be completed. If all those jobs succeed, a team member can visit the SonarQube frontend and check for the code analysis. After inspecting this, one can merge the desired branch into another and the last part of the pipeline will be kicked off to build the Docker file, which is based on the merged code. The output file can then be downloaded under the artifacts of GitLab.

4. Conclusion

This section critically discusses the achieved results, the decisions made and illuminates what could have been handled better. As such it also comments on what will have to be changed should the project be continued on the basis of this prototype.

4.1. Retrospective

As discussed in the Results section of this paper, while the planned use cases are sufficiently implemented, there is a heavy limitation on the prototype. A gRPC connection to the Jalapeño API Gateway could not be established and as such the prototype had to directly access the database of the Gateway. This means the prototype will have to be refactored to accompany a gRPC service. While it was build to make the migration as smooth as possible there is still overhead in a future migration.

4.2. Discussion

4.2.0.1. Pipeline

A larger microservice project usually would be distributed over multiple repositorys with separate pipelines. This would provide greater felxibility and adjust better to different software needs.

In this research project the team started with an approach of multiple pipelines per branch, that would fulfill different tasks. There was for example a pipeline script to build the LaTeX based documentation and another for the production, which would format, run tests, build input for the SonarQube and deliver a Docker file for the release at the end. While this method worked well in the beginning, it became soon obvious that this was not a sustainable solution with the feature branch system the project worked with. The many branches created, updated and merged made it impossible to manage the pipelines in an efficient way. To solve the problem the team merged the different pipeline scripts into one with branch specific rules to run different jobs according to the branch it currently ran on.

With the implemented React.js frontend the team would have liked to have a pipeline for the deployment of this code into its own pod, too. Sadly, the time did not allow for this to happen.

4.2.0.2. Architecture

The class diagram for the `network read` service was created with a more general request service approach in mind, contraty to the streaming functionality the final service would implement. This as both team members did not have much experience in either class diagrams or indeed different approaches to architecture than the classical request based one. Upon construction of the prototype it became obvious that this would not be an optimal solution to the problem and changes had to be implemented where needed and doable. The streaming based approach was not possible to experiment with in the construction. This means that the architecture may need slight updates when implementing push architecture. But due to the isolation of the packages this should not be a large undertaking.

In retrospect a stream based push architecture for the prototype with a basic message queue or something similar as provisional "frontend" (observing and analysing the messages received by the queue) may have been a better approach to prototype to allow for a more similar architecture to the final product. Even with no connection possible to the gRPC, with a makeshift polling mechanism this would have been possible and given a better starting point for a future project.

4.2.0.3. Live Model

The live instance of the topology, on which observer can be build to push notifications to the frontend, proved to be a stable system in the request based application. As mentioned before it may be needed further experimentation for the final architecture. The team still believes it will be a stable solution for the stream based architecture, even if it makes the backend marginally less stateless. As the state, i.e. the model, should be globally the same and does not have differences between scaled instances, i.e. will be updated in the exact same way by each application, this should not pose a problem.

4.2.0.4. Gunicorn - Global Variables

[**Gunicorn**][1] A limitation to the previouesly discussed live model proves to be the production server. The Flask application, when not run locally in development mode, is run on a Gunicorn server in Kubernetes. Gunicorn is a Python WSGI HTTP server that runs on UNIX. It provides basic scaling functionality together with Kubernetes and allows easy, save and stable deployment of the Flask framework.

The scaling is done via Pre-forking. The drawback of this is that previously global variables are not guaranteed to be global anymore. The scaling of Gunicorn is implemented with a pre-fork worker model, i.e. the application can be run as multiple instances that handle great workloads of requests simultaneously. This means the live model could hold different information between the instances and so leads to an inconsistent system. This proves to be somewhat troublesome for the live model, as that is supposed to be a globally valid variable.

Flask does not provide a build in mechanic to solve this problem, as all functionalities like context, etc. are constrained to whithin one Gunicorn Pre-forking at best. To avoid this problem a possible solution was implemented in the prototype, namely caching.

4.2.0.5. Caching

As discussed in the last two sections, the supposed global live model is not stable in a production environment due to the mechanics of the Gunicorn server. To mitigate this problem a cache was implemented. The application now checks the cache if there is an instance of the model and if yes, loads this model. Then it checks for newest updates and changes the cached model accordingly. The application runs a normal request to the ArangoDB and gets all topology data it needs, should there not be a cached value. The application again caches the value of the live model after completing all necessary calculations. This saves performance time and may be useful especially on the computation heavy operations like the algo path calculations.

In combination with the Guncicorn server it may be recommended to use the file based caching service. If Kubernetes will add additional scaling an external caching service like Redis may even be needed. Replacing caching versions is an easy change of the configuration and is easy to implement.

One last question is if this will even be needed. As all workers, no matter the scaling, will receive the same model data and updates, there should in theory never be much differences between the "global" variables of the workers. This will need extensive testing though, as the behaviour can not be predicted, especially on heavy workloads in the production. Live tests on a bigger network may be recommendable.

4.3. Outlook

In this chapter we will discuss improvements and insights gleaned from building the prototype that will help in a further project to improve the application.

4.3.1. Improvements

gRPC Connection

First and foremost the gRPC connection has to be built up. An integration of the functionality already exists on branch feat_add-grpc that succesfully sets a request to the Jalapeño API Gateway and receives data back that it writes on the console. At the moment only the request service is utilized as the connection to the gRPC is unstable. As such, especially the streaming functionality should be implemented and tested once the Gateway is updated.

Caching

Depending on the findings of the application's behaviour with global handling as discussed previously, caching may be built upon. In an extreme case a stipped down persistence of the aggregated data may be the best solution to ensure performance. At the moment this does not seem necessary, as even test data with 1'000 routers, links and up to 10 algorithms is performing well above the minimal requirements, if only updating functionality is looked at. Complete loads of the data are also still in the set time. Further testing and experimenting is needed here.

Decoding Affinities

Affinities are predefined in a bit-map on each router to map the string values the engineer configures to numbers. These values are then used in the affinity handling of FlexAlgos and configured on interfaces. Sadly, the mapping is not sent out of the router with the rest of the FlexAlgo data, merely the number values. Not only that, but the values are saved as binary strings that are base64 encoded. How exactly the values are saved can be seen in section 2.2.

The decoding and matching of these values poses a larger problem. As it was only found out in the last week of the construction phase of this project time, for finding a solution to this could not be spared. Writing the thesis had to take priority at this time. A beginning of a decoding can be found on branch `fix_decoding-affinity`, though, and can be built upon in a future project.

4.3.2. Innovations

Coordinates

The application uses coordinate data on the nodes to place them in a meaningful way on the graphical representation of the network. This uses Jalapeño functionality that has not been implemented yet but thought up in other SR-App projects. Is this functionality developed the application will be able to use it seamlessly.

Frontend

Currently a bachelor thesis for a general SR-App frontend is being done by the students Davor Gajic and Leonard Obernhuber. They have already developed a prototype research project "Central Frontend for Segment Routing Applications" and are now in the process of finalising a frontend for SR-Apps that can be implemented and adapted as needed for the applications of this series. SR-App FlexAlgo will also be able to implement this project upon completion. As such our basic React.js demo will be replaced and a further project can concentrate on the more specialized backend services.

4.3.3. Further Thoughts

CRUD Operations

The Jalapeño API Gateway is currently only able to read data.

As the SR-Apps are planned to be build upon and encompass a wide variety of functionality, a premade gateway to handle deployments of different configurations may be advisable. At the moment the team is aware of at least one other project, SerPro from Julain Klaiber and Severin Dellsperger, that deploys configurations. Instead of duplication of the architecture for the different applications, which may only have minimal differences, a single gateway be of great use for the bigger scope of the SR-App project.

Mocking Tool and Frontend - Connected Services

Similar to the CRUD operations, the mocking tool used in this project holds very similar requirements between applications. A more general mocking mechanic, perhaps even built into an application, may be of use for the future. Adapting and expanding the minimal mocking tool of the project "Central Frontend for Segment Routing Applications" to seamlessly work with different needs of SR-Apps may be a worthwhile endeavor.

5. Terminology

WAN

Wide Area Network consists of multiple networks that can communicate with each other. For a more detailed explanation one can consider Ciscos definition.

DC

In a Data Center there is a hierarchy of multiple levels built with different kinds of devices to ensure a high performance while delivering reliability in parallel. This structure is chosen because the DC usually acts as a distribution center of data that has to be transferred from one (large) network to another. As a good overview, one can consider the TechTargets explanation.

Source Routing

This network routing principle allows the source device, the starting point of the routing process, to direct the complete or part of the path that the packet will take over the network to the destination. For this, the routing steps will be inserted into the packet directly. A short explanation can be considered on Junipers section or the entry of Wikipedia.

Destination Routing

In contrast to the above described source routing network devices send packets according to the destination IP address that is written into the packet header. In other words, each router on the way calculates the path to the destination, instead of merely following a preset routing path to the next step.

Control Plane

The plane is an abstract separation of processes network devices run to be able to route packets. The Control Plane describes the processes that provide information about the topology of the network and all information needed how to forward packets [57]. See Cloudflares site for further information.

Data Plane or Forwarding Plane

The processes that actually forward the packets in the network according to the Control Plane [57]. For a fast overview consider Cloudflares site.

Node

A network device capable of participating in a network communication with other devices as an endpoint or as a transit point.

Link
A physical connection between two active Nodes on a network. Usually represented by a cable or something similar. Links are not intelligent on their own but may be handled as such in this project.

Interface
Physical or virtual points that a device owns over which the data packets get sent or received.

IS-IS
With this routing protocol, the `Intermediate System to Intermediate System` (RFC 1142 [35]), different kinds of networks can be built. As it is an IGP it is usable as control plane. A fast overview can be read on Juniper's website.

MPLS
The network technology `Multiprotocol Label Switching` lies on OSI layer 2.5. It is independent of the upper layer's protocol and can encapsulate their packets. It belongs to the data plane. MPLS works with labels to send the packets to its next destination instead of IP addresses (see source routing parapragh). A good definition can be read in Palo Alto Network's docuement.

YANG model
YANG is a data modeling language derived of json. It is popular to configure and control network devices. The FlexAlgo information for this project is stored in this language on the network devices. Please consult the YANG documentation for further information.

Segment Routing
Segment Routing is a source routing protocol that steers packets according to segments, instruction written into the packet header. The Main infomration is therefore stored not in the devices, as with other protocols, but in the packets themselves. As such traffic can be adjusted to the desires of the network enigneer. A good overview of the technique and its different terms can be read on Juniper's website.

Segment
Segments are instructions placed in a packet header, called segments. The receiving device will consult these segments on how to send the packet on to the next Segment Routing device. Please consult hrefhttps://www.juniper.net/us/en/research-topics/what-is-segment-routing.htmlJuniper's website for further information.

Flexible Algorithm (FlexAlgo)
Complemets the Segment Routing solution to traffic engineering by adding new, customizable prefix-segments with specific optimization objective and constraints. For example, it can minimize IGP-metric, delay or TE-metric or avoid SRLGs. Where it shines most is avoiding or including routes based on manually configured values on links, such as SRLGs or affinities [46]. A complete presentation of this topic can be read Cisco's page.

Affinity
A value given to an interface on a node to customize path handling. Used to calculate routing paths. Custom is to use color names. Packet paths in FlexAlgos will then be calculated to include or exclude these values depending on the FlexAlgo configuration.

Prefix-SID
An ID for a node in a flexible algorithm, unique per algorithm.

Part IV.

Project Documentation

1. Requirements

1.1. Use Cases

🟧 Minimal Viable Product SA
Optional Features SA
🟦 Minimal Viable Product BA
Optional Features BA

Table 1.1.: Use Case Color Code

1.1.1. Actors

Actor	**Description**
Unauthorized	User is not authorized in the system. User has to provide identity to get access to functionality of system.
User	User is authenticated and can use basic features of the system. Basic features include viewing flex algo related information of the network.
Admin	Admin is authenticated and has additional permissions for suplementary features of the application. These features include performing CRUD operations on the network configuration and viewing logged information.

Table 1.2.: Actor Description

1.1.2. Diagram

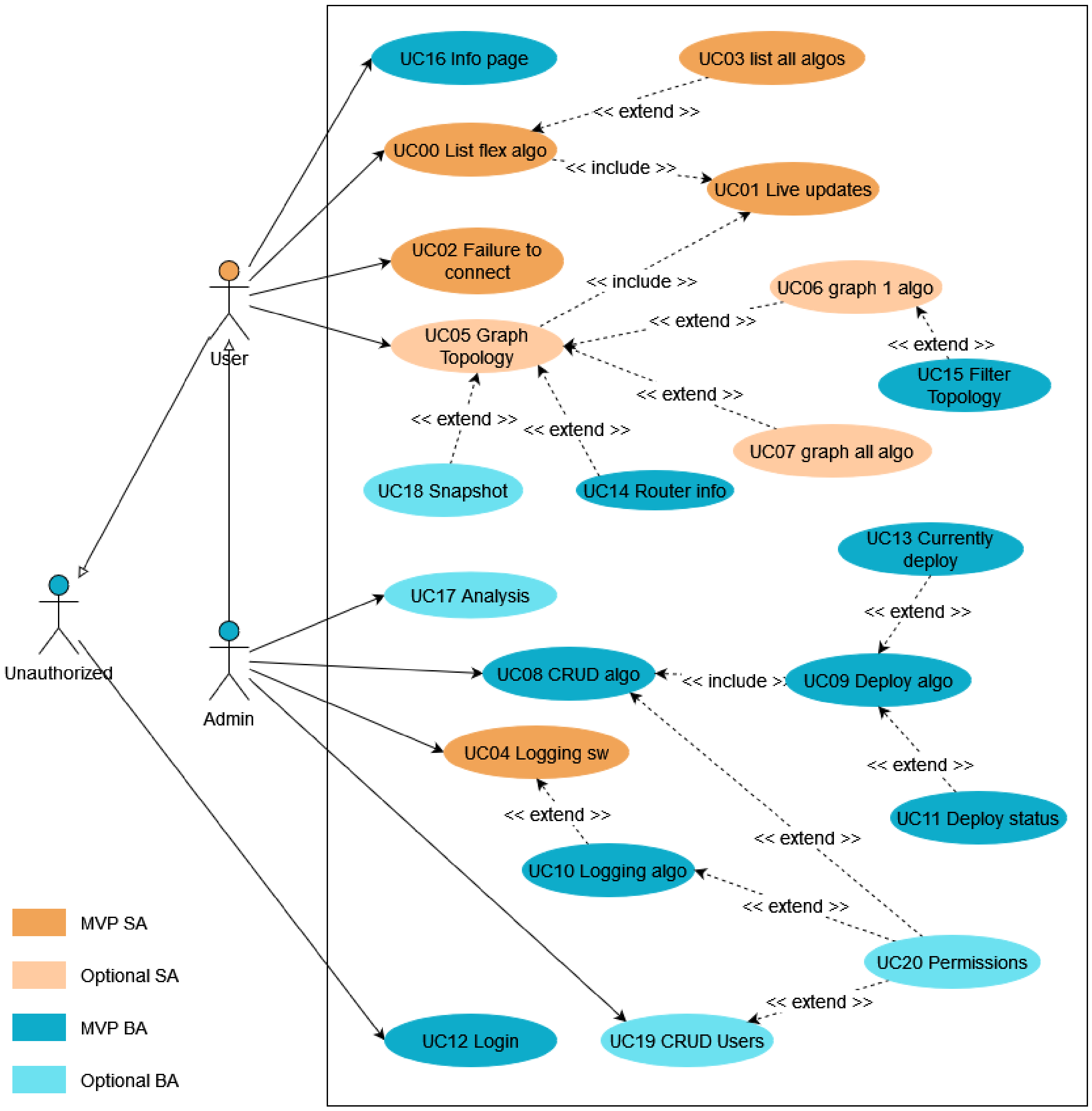


Figure 1.1.: Use Case Diagram

1.1.3. SA Use Cases

UC00 See a list of flex algo information

As User I see a list of all the information of the flex algos currently running on the network connected to the application.

Actor	User
Overview	This UC will be replaced with UC05 as soon as it is implemented. It will only be used for prototype development. As a User I look for the current network topology of the to the Gateway connected network. I see a list of all relevant information of the flex algo.
Preconditions	Jalapeño API Gateway is connected to a network and is sending data to the system.
Main Success Scenario	1. User wants to get an overview over the network and it's running flex algos. 2. System provides a text list of the network upon refresh of web page.
Failure Outcome	The User sees an error message in place of the list. In it is defined if the error occured in the application or before the application.
Frequency of Occurence	As often as required.

Table 1.3.: UC00: List of Flex Algo - Fully Dressed Description

UC01 Live Updates

As User I see live changes to the network.

Actor	User
Overview	As User I see updates to the flex algo in real time.
Postconditions	Has UC03 and UC05 been implemented, then the flex algo information in those will be updated. Is UC17 implemented then all network changes will be persisted in database.
Main Success Scenario	SA: If a the flex slgo configuration on the network changes the web frontend reflects these changes with a page refresh. BA: The React.js framework will allow the User to see all changes without page refresh via event handling and data binding.
Failure Outcome	User will experience UC02 should application loose connection to Jalapeño API Gateway.
Necessary UC	UC00

Table 1.4.: UC01: Live Updates - Fully Dressed Description

UC02 Failure to Connect to Network

As User I see a pop-up message if the applicationhas failed to connect to the Jalapeño API Gateway or the network and live updates are no longerpossible.

UC03 Show List of All Running Flex Algos

As User I see a list of all flex algorithms running on the network connected to the application.

Actor	User
Overview	As a User I can see all currently configured flex algos on the connected network.
Assumptions	User is logged into the application.
Preconditions	Jalapeño API Gateway is connected to a network and is sending data to the system.
Main Success Scenario	1. User wants to see an overview over all flex algos running on the connected network. 2. System provides a list of algos.
Failure Outcome	The User sees an error message displayed in place of the list.
Frequency of Occurence	As often as required.
Necessary UC	U00, U01

Table 1.5.: UC03: Show List of All Running Flex Algos - Fully Dressed Description

UC04 Logging Application

As Admin I have an error log for the backend server of the application.

UC05 View Topology

As User I see a graphical representation of the topology of the network connected to the application that is updated live.

Actor	User
Overview	As a User I look at the current flex algo topology of the connected network. I see this topology as a graphical representation.
Assumptions	User is logged into the application via UC12.
Preconditions	Jalapeño API Gateway is connected to a network and is sending data to the system.
Main Success Scenario	1. User wants to get an overview over of the network. 2. System provides a graphic of the network with all to the flex algo relevant objects.
Failure Outcome	The User sees an error message in place of the graphic. In it is defined if the error occured in or before the application.
Frequency of Occurence	As often as required.
Necessary UC	UC00, UC01

Table 1.6.: UC05: View Topology - Fully Dressed Description

UC06 See Graphical Representation of One Flex Algo

As User I can choose a flex algo and see the graphical representation with all contained nodes.

Actor	User
Overview	When in graphical view of the network the User can click on a specific algorithm and filter the visual to show only the relevant nodes and links to this flex algo.
Necessary UC	UC01, UC05

Table 1.7.: UC06: See Graphical Representation of One Flex Algo - Casual Description

UC07 See Graphical Representation of All Flex Algos

As User I see a global map of the network with all running flex algos graphically represented.

Actor	User
Overview	When in the graphical overview of the whole network, User can see all running flex algos in the graphical interfaces. Meaning links are colored according to the algorithms running on them.
Necessary UC	UC01, UC05

Table 1.8.: UC07: See Graphical Representation of All Flex Algos - Casual Description

1.1.4. BA Use Cases

UC08 CRUD Flex Algos

As Admin I am able to create, update and delete flex algorithms currently running on the connected network in the web interface. The read part of CRUD is implemented in UC.

Actor	Admin
Overview	As an Admin I want to configure the currently running flex algos on the network. Create: As Admin I create all necessary parameters for a new flex algo. Update: As Admin I view an existing flex algo in the graphical User interface. Locating the relevant properties I change configuration settings on a node or algorithm according to the new policy. Delete: As Admin I delete an existing flex algo to revoke the existing configuration on the network in the next UC.
Assumptions	Admin is logged into the application.
Preconditions	Admin has required permission to alter network configurations. Jalapeño API Gateway is connected to a network and is sending data to the system.
Postconditions	Viable flex algo configuration is prepared for UC09.

Main Success Scenario	Create: 1.1 Admin wants to create a new flex algo. 1.2 Admin clicks on 'new algorithm' button. 1.3 System opens flex algo detail view. 1.4 Admin inputs all relevant information about new flex algo. 1.5 System makes a check if all necessary input for a new configuration file is present and valid (input validation, like no letter in int, etc). 1.6 Admin clicks on 'deploy' button. 1.7 System provides a new configuration to be deployed in UC09. 1.8 System automatically starts UC09. Update: 2.1 Admin wants to update an existing flex algo. 2.2 Admin finds desired flex algo in existing algo list. 2.3 Admin clicks on 'edit' button of flex algo. 2.4 System opens flex algo detail view. 2.5 Admin changes relevant information. 2.6 Items 1.5-1.8 will be stepped through. Delete: 3.1 Admin wants to delete an existing flex algo. 3.2 Admin finds desired flex algo in existing algo list. 3.3 Admin clicks on 'edit' button of flex algo. 3.4 System opens flex algo detail view. 3.5 Admin clicks on 'delete algorithm' button. 3.6 Items 1.5-1.8 will be stepped through.
Other Success stories	Cancel: 4.1 Admin wants to cancel algo configuration without applying changes. 4.2 In algo detail view Admin clicks on 'cancel' button. 4.3 System reverts to topology view without providing new config file.
Failure Outcome	• Invalid configuration: User cannot click 'deploy' button to stop invalid network changes. • Config file creation failed: User gets notified about failure to create configuration file and returns to algo detail view.
Frequency of Occurence	User defined frequency
Necessary UC	UC01, UC04, UC05

Table 1.9.: UC08: CRUD Flex Algos - Fully Dressed Description

UC09 Deploy Flex Algo Changes

As Admin I deploy flex algo changes to the connected network.

Actor	Admin
Overview	As Admin I want to deploy the new configuration made in UC08.
Assumptions	Admin is logged into the application.
Preconditions	Admin has required permission to alter network configurations. Network is connected to the Jalapeño API Gateway. Network data is live.
Main Success Scenario	1. System changes to deployment view. 2. Admin sees old and new version of configuration. 3. Admin clicks on ’deploy’ button. 4. System starts UC10 and UC11 if implemented. 5. System returns to topology view with deploying status displayed.
Other Success Scenario	Cancel CRUD operation: 1. Admin does not want to deploy changes. 2. Admin clicks on cancel. 3. System reverts to algo detail view with previous changes still logged. 4. Admin clicks on ’cancel’ button to remove all changes to existing configuration. 5. System reverts to topology view. Change CRUD operation: 1. Admin finds mistake in new configuration. 2. Admin clicks on ’cancel’ button. 3. System reverts to algo detail view with previous changes still logged in. 4. Admin repeats UC08.
Failure Outcome	New configuration failes to deploy. Admin receives error message with detailed info about deployment failure.
Frequency of Occurence	User defined frequency
Necessary UC	UC01, UC04, UC08

Table 1.10.: UC09: Deploy Flex Algo Changes - Fully Dressed Description

UC10 Logging Flex Algo

As Admin I want to have a log of all network changes done by the application.

Actor	Admin
Overview	All User changes to the flex algo configurations on the network will be logged for the Admin. Information to be logged will be User login, time of change, type of change, difference to preexisting config file. Logfile will be viewable in the web application if necessary permissions are given.
Necessary UC	UC09

Table 1.11.: UC10: Logging Flex Algo - Casual Description

UC11 Status of Deplyoment

As Admin I see the status of the deployed flex algos.

Actor	Admin
Overview	Status of deployment from use case UC09 will be shown in web application. Possible values will be "currently deploying", "successful deployment", "failed deployment".
Necessary UC	UC09

Table 1.12.: UC11: Status of Deplyoment - Casual Description

UC12 Login

As Unauthorized I am able to register and login to the application.

UC13 Currently Deploying

As User I get notified if a deployment of an Admin is running.

Actor	User
Overview	When an Admin is deploying a change to the network all Users will get notified of this change via pop up.
Necessary UC	UC11

Table 1.13.: UC13: Currently Deploying - Casual Description

UC14 Show Router Information

As User I see relevant information for a link or node.

Actor	User
Overview	As User I can click on a node and see detailed information of this object. Information includes metrics, algos and algo specific properties, ssh connection info, etc.
Assumptions	User is logged into the application.
Preconditions	Jalapeño API Gateway is connected to a network and is sending data to the system.
Main Success Scenario	1. User wants to get specific information about a node. 2. User clicks on object. 3. System provides all information of chosen object in a separate panel.
Failure Outcome	The User sees an error message displayed in place of the information.
Frequency of Occurence	User defines frequency.
Necessary UC	U01, U04, UC05

Table 1.14.: UC14: Show Router Information - Fully Dressed Description

UC15 Filter Topology

As User I can filter the currently displayed network.

Actor	User
Overview	As User I can filter the shown topology by node name and properties. Filtering is done only in the frontend of the application.
Assumptions	User is logged into the application.
Preconditions	Jalapeño API Gateway is connected to a network and is sending data to the system.
Main Success Scenario	1. User wants to only see objects with a certain affinity. 2. User inputs the desired values into GUI. 3. System provides a graphic of the filtered network.
Other Success Stories	Empty filter: 1. User wants to only see objects with a certain property. 2. User inputs the desired values into GUI. 3. System provides a message ("no objects in this filter") if filter has no valid entry.
Failure Outcome	The User sees an error message in place of the graphic.
Frequency of Occurence	User defines frequency.
Necessary UC	U01, U04, UC05

Table 1.15.: UC15: Filter Topology - Fully Dressed Description

UC16 Info Pages

As User I see an info page about the application and general information about flex algorithms.

UC17 Historic Analysis

As Admin I see and analyse historic data of flex algorithms of the connected network.

UC18 Snapshots

As User I can make snapshots of the network with currently chosen visual filters.

UC19 CRUD Roles and Users

As Admin I create, read, update and delete roles and Users on the application.

UC20 Permissions

As User I have access only to operations I am authorized to.

1.2. Non-Functional Requirements

[26]

1.2.1. Functionality

Security (BA)

- Access to application will be secured via authentification.
- Network changes are not allowed via the application by unauthorized users.
- Network changes are logged with user information for non-repudiation.

Accuracy

- Application will work with latest data from the Jalapeño API Gateway. All updates to flex algo configurations will be done with up-to-date information.

Interoperability

- The application will work with the by INS developed Jalapeño API Gateway. The underlying network data will be generated using Cisco IOS XR devices on which MPLS is configured. Configurations are written to the same devices in the same topology.

1.2.2. Reliability

Fault Tolerance

- The backend server will remain running, even if the communication with the Jalapeño API Gateway is down.

Maturity

- In production usage with sufficient network connection 95% of all requests will be completed successfully.
- The annual uptime of the app will be 95%.

Recoverability

- The application will be developed according to cloud native standard. As such redeploying on Kubernetes cluster after failure will be possible without further manual intervention.

1.2.3. Usability

Understandability (BA)

- The design is clear, minimalistic and shows the complex relationship of a flex algorithm as easy as possible.
- A configurations status is available at a glance.
- Application may provide an info page for app usage if project time allows.

Operability (BA)

- Via info page and icons a user is able to change a flex algo without further assistance.
- All inputs and views are reachable within 5 clicks.

User Error Protection (BA)

- Changes to the network by the user need to be separately confirmed.
- User input will be validated in frontend.

Accessibility (BA)

- The design is clear, minimalistic and shows the complex relationship of a flex algorithm as easy as possible.
- The app is usable with color weakness.
- The font size is changable by the user.

1.2.4. Performance

Data Integrity

- (BA) Multiple users will be able to work on the same network at the same time.
- The network configurations will be updated real time.

Scalability

- The application will be scalable through the usage of cloud native development.

Resource Behavior

- (BA) Each service provider receives an own instance of the application environment and can give access to its customers. More detailled analysis for user capacity will be done in the BA. (The app will support up to 6'000 registered users.)

Time Behavior

- Once Jalapeño API Gateway has processed a topology change the application will take no longer than 1 minute to fetch and display the new data. (Initial loading is not included in this estimation.)
- Telemetry data from the Jalapeño API Gateway will be shown in real time in the application.

- (BA) Time behaviour by CRUD operations on flex algorithmn will be defined in BA.

1.2.5. Maintainability

Analysability

- When an error occurs the user will receive a popup-message that asks them to repeat the action.
- Errors in the backend and the interface to Jalapeño API Gateway will be logged.
- (BA) Log the actions of each user for forensical reasons.

Testability

- All use cases will be tested automatically with unit and integration tests.
- Performance of the non-functional requirements will be checked with system tests.
- All test cases have to succeed before the new code will be integrated in the productive software.

2. Domain Analysis

A domain analysis was done in order to understand the flexible algorithm Segment Routing technology. As both participants had little knowledge of FlexAlgo itself, this was an especially important step. In the following section a visual representation of the domain is shown, in the section Administrative Concepts the concepts are further explained in detail. A base understanding of the FlexAlgo technology is assumed, as can be read in chapter 2.

2.1. Domain Model

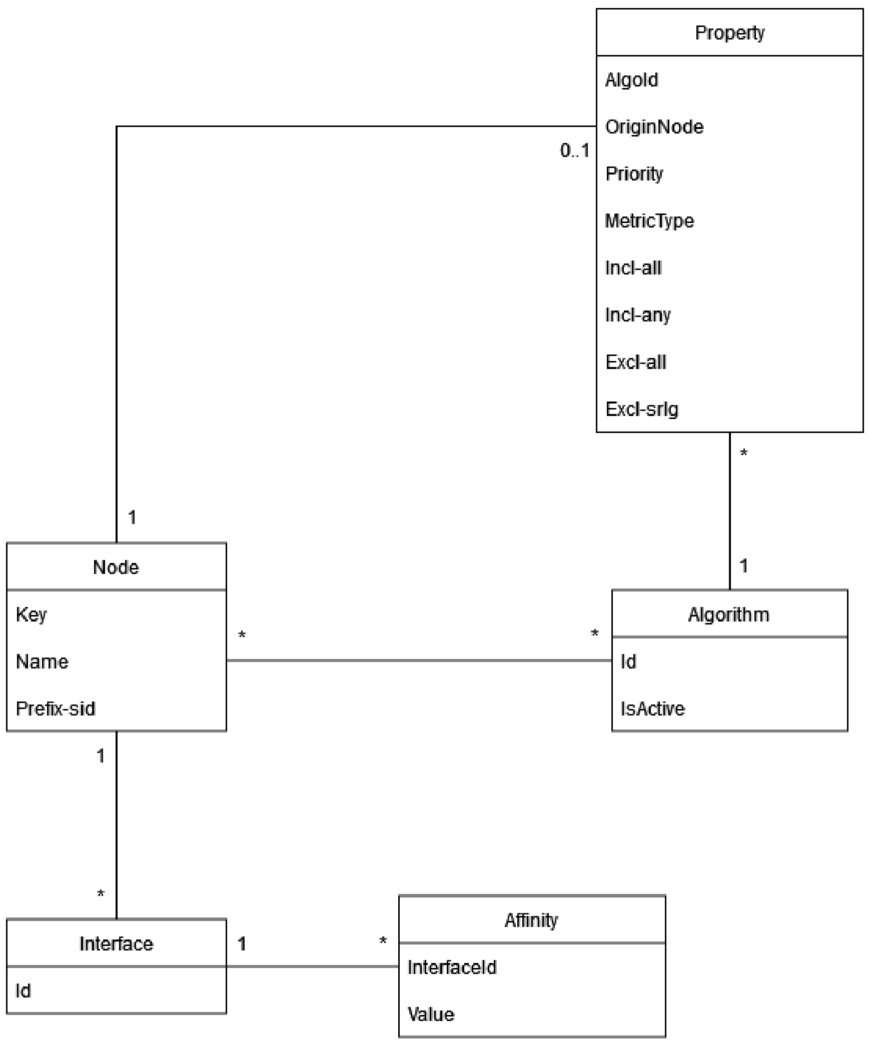


Figure 2.1.: Domain Model [28]

2.2. Administrative Concepts

[55]

Algorithm

The Algorithm domain represents the flexible algorithm of Segment Routing. It is the central part of the domain. For a clearer explanation of what a flexible algorithm is and how it works consider the thesis section on Segment Routing 2. An `Algorithm` can be defined on a `Node` and the configuration is then advertised through the `Network`. Each `Node` can have multiple `Algorithms` with different property values configured on it. Property values for the same algorithm can differ between the `Nodes` that have this algorithm configured. As such the domain needs to be able to handle different, even contradictory, definitions for one algorithm. The network handles these inconsistencies with the `priority` property. The propagation of the FlexAlgo information through the network BGP-LS is used, which distributes sub-tlv packets for this [5].

For this project properties of an `Algorithm` are moved from the `Algorithms` domain object and outsourced in `Property` to be able to work with priorities and origin node properties.

Node

A `Node` refers to a device capable of routing network traffic and therefore calculating FlexAlgo information. Most often this refers to routers, but other devices like servers are capable of handling these too. In the scope of this project `Nodes` refer to Cisco IOS XR routers. Each router has a network wide unique key assigned to differentiate between the `Nodes` in the domain. Additionally, it has a name and other properties, which are not yet interesting for this project and therefore ignored for now. Each Node has at least one `Interface` connected to it, which is the loopback interface that is represented and configured with a prefix-sid. This id is a number that marks the router as unique not only network wide but also whithin each FlexAlgo it is assigned to. Most routers will have more than one `Interface` configured.

Interface

This is the point of interconnection between a `Node` and other `Nodes` and allows connectivity to the network. An `Interface` has an identification unique to it's `Node` and the `Node` it is connected to.

Each `Interface` can have `interface` assigned to it that are valid for all `Algorithms` defined on the `Node`, but doesn't have to. This provides more granular configuration posibilities for the algorithms in the segment domain and are where the FlexAlgo technology shines best. Affinities are meant to be a characteristic of a link, but as a link is not a logical object in a network, they are defined on the `Interface` domain element. Is there a mismatch between the `Affinities` on both `Interfaces` connected by a link, the outdoing definition is used.

The Affinity information is saved and porpagated via BGP-LS and saved in sub-tlv lists of type 1088 (Administrative Group - color) and 1173 (Extended Administrative Group).

Property

Properties are values that are assigned on `Node` level, in contrast with Affinities, which are assigned on `Interface` level. These values are assigned per FlexAlgo, meaning a `Node` can have very different properties assigned depending on which Algorithm is observed. There are different kind of Properties:

- `AlgoId` algorithm identification number, connects to the `Algorithm` domain object.
- `OriginNode` defines which Node this configuration comes from.
- `Priority` is implemented to handle configuration mismatches of algorithms between nodes. Has node 1 a different property than node 2 the one with the higher priority will be used to calculate the routes. Have both nodes the same priority, then the prefix-sid will decide.
- `MetricType` defines which value is used to calculate paths through the network. Per default this metric is IGP (Interior Gateway Protocol) which is per default 10 for the protocol IS-IS and therefore calculates the shortest path. Other metrics are TE, i.e. traffic engineering which represents a user settable value, or delay, which calculates time spend in the network. The time tracking functionality needs to be enabled in all nodes first, before this metric can be used.
- `Incl-*` represent parts of the `Affinity` handling for the algorithm. In the incl-* lists values are defined, that *can* (incl-any) or *need* (incl-all) to be traversed. As such, links with these `Affinities` are preferred over other links for the path calculation.
- `Excl-all` these paths are to be strictly *avoided.* As such traffic can be steared individually per algorithm. Bottlenecks and overload of links can be avoided.
- `Excl-srlg` performs similar to the Incl-* and Excl-all lists, but is a more global concept. Here links are bundled into risk groups that can be avoided by defining it in this list.

Affinity

An `Affinity` is a value that can be assigned to an `Interface` for easier algorithm configuration through the whole segment domain. How the `Affinities` are handled can be read in section Property 2.2. `Affinities` are a key-value pair. Keys are identifying numbers between 0 and 255 and the value is a user defined string. These categories are saved as color names by convention. The mapping between name and number is scoped on a `Node`. Generally it is best practice for these mappings to be globally the same on the whole network, but similar to the `Prefix-sid` this is not enforced. `Affinity` values are defined on `Interfaces` for each `Algorithm` on the parent `Node`. They are globally assigned, meaning they are valid for all algorithm on configured ont he `Node`.

3. Architecture

3.1. Twelve Factor Methodology

To ensure the application will be cloud native, the twelve-factor methodology was used to analyse the application and it's surroundings. From this analysis the following measures to ensure cloud nativity were planned and implemented in the application.

I. Codebase

"One codebase tracked in revision control, many deploys."[51]
The INS institute GitLab will be used to for versioning of the project. The prototype will be saved in one Repository and all additional data like deployment files and documentation. As the SR-App is planned as a microservices architecture it may be advisable to separate the different services into their own repositorys, when the project will be implemented. Using the CI/CD mechanics of GitLab the deployment will be strictly separated into a test and production deployment pipeline.
Evaluation: Fulfilled.

II. Dependencies

"Explicitly declare and isolate dependencies."[51]
All dependencies will be handled with Poetry to ensure the best isolation and dependency management. Poetry allows depenedency handling via files and provides a virtual environment in which the defined dependencies are installed. This ensures development, testing and production remain the same over all environments.

Evaluation: Fulfilled.

III. Config

"Store config in the environment."[51]
Configurations for the application are handled in the isolated `.Config` class. The class accesses environment files for the specific values that are used in Kubernetes by utilizing the K8s secrets. With this the application code does not have to be changed in different environments and sensitive information is hidden and encrypted. The application loads the files on startup and can then consume the values specific for the current environment.

Evaluation: Fulfilled.

IV. Backing services

"Treat backing services as attached resources."[51]
At the time of evaluation the planned solution for the prototype has slightly changed from what is described here, as can be seen under the paragraph evaluation. But the basic functionality of Backing service is still very much the same. In the prorotype for the application the only backing service the application will consume is the Jalapeño API Gateway. The prototype is connected via a gRPC Client to the API Gateway, which merely needs a connection string. The url itself

will be stored in a configuration file and loaded upon startup of the application. The connecting client class is itself a derivative of a connection interface and as such isolated from the consuming service of the application. Therefore, should the technology be replaced, for example with a websocket instead of gRPC, merely the implementation of the interface needs to be changed. The rest of the application remains the same.

Evaluation: Fulfilled. As gRPC connection could not be implemented yet, the backing service was the direct connection to the ArangoDB instead. This connection is isolated in the `utils` class and can easily be replaced.

V. Build, Release, Run

"Strictly separate build and run stages."[51]
We have a build and deployment stage in the pipeline. The building stage requires previous stages, like check format, testing, etc., to build the Docker image file. After completing this part of the pipeline successfully the deployment stage will be kicked off to update the Kubernetes pod with the previously created Docker imgage.

Evaluation: Fulfilled.

VI. Processes

"Execute the app as one or more stateless processes."[51]
The app will be run as six stateless processes that operate separately from each other. Data will only be stored as a whole for perfromance optimizing. It can be discarded without any data loss. For long term persistence and future requests certain data may be stored in connected databases, such as login, FlexAlgo configurations and historic data. This will not be implemented in the scope of this prototype. The prototype itself holds two processes, a striped down React.js frontend and the `Network Read` service.

Evaluation: Fulfilled.

VII. Port Binding

"Export services via port binding."[51]
Port Binding will be provided by Kubernetes's methodology.
The application can be reached over the url `sa-sr-flexalgo.stu.network.garden/` which will bind to the exposed port 80 of the ingress object that is the first entry point to the Kubernetes cluster and its elements. Over the Service in the cluster the application that exposes the Port 5000 can then be accessed.

Evaluation: Fulfilled.

VIII. Concurrency

"Scale out via the process model."[51]
Kubernetes will handle any needed scaling for high user traffic and object heavy network traffic. All services except `Network Read` can be handled easily with the K8s scaling and availability functionality. The scaling needs of the `Network Read` service will be tested with the prototype of the application made in this project. Should there be a need to add load balancing for router heavy networks a solution will be reviewed in combination with the Jalapeño API Gateway in a further step.

Evaluation: Fulfilled. Tests of networks of up to 1000 routers and 1000 links with 100 algorithms configured were tested. Performance was not significantly impacted and still in acceptable range. See acceptance tests in appendix 5.

IX. Disposability

"Maximize robustness with fast startup and graceful shutdown."[51]
Like with point IV the gRPC connection could not be implemented. See results in evaluation paragraph. The porotype will be stateless with only the gRPC connection to the Jalapeño API Gateway needing further handling upon shutdown, for wich the gRPC technology already brings it's own functionality. Data loss from network data is minimally damaging, as it is completely replenished upon next refresh of page. All further microservices in the application beside `Network Read` can be build with a serverless architecture via Knative and may be designed to be fast and lightweight. As such they will provide fast start and shutdown times. Data loss is additionally mitigated via message queue usage, that provides temporary storage for unprocessed messages where necessary.

Evaluation: Fulfilled. The ArangoDB connection that replaces the gRPC connection to the Jalapeño API Gateway handles startup and shutdown without further input needed, as it a request based connection and not a bidirectional live one. The application starts and shuts down satisfactory.

X. Dev/prod parity

"Keep development, staging, and production as similar as possible."[51]
The production environment will be the K8s namespaces provided by the INS institute. The development of the SA prototype will be done locally, which will look as similar to the K8s namespace as possible. As Poetry is used for all dependencies, it is guaranteed that frameworks, libraries, etc. are the same over all environments. For a further project the separation of a development, test and production environment may be recommendable and can be handled via namespaces in K8s.

Evaluation: Fulfilled.

XI. Logs

"Treat logs as event streams."[51]
The application currently logs error events in the application and sends them to the stederr. Kubernetes features will be utilized for logging of the stream to the pod console. This will be sufficient for the prototype architecture of this project. In further construction of the application logs may be pesisted with a separate logging solution.

Evaluation: Fulfilled.

XII. Admin Processes

"Run admin/management tasks as one-off processes."[51]
Administrative tasks will mostly be handled by Kubernetes and are easily done by redeploying pods in a rolling update. Should additional administrative tasks be needed, like for example migration scripts, they will be tested first locally in development, then in the test namespace which is as similar to production as possible, and only then run in production.

Evaluation: Fulfilled. Not yet needed.

3.2. Design Goals

The two main design goals of the SR-App FlexAlgo, especially the `network-read` service prototyped in this project, are scalability and performance. Depending on the connected network there is a large amount of data that the application will have to compute. The project has to support networks with up of 1'000 routers. To make sure real live updates with this data load are performant is therefore the main design goal of the application.

- `Highly Scalable`: The application will have to support not only the computation of networks with thousands of routers in it, but also a sizeable user base. As such all components will have to be built to be reasonably scalable and therefore be as stateless as possible.
- `Performance`: As mentioned before, configurations of large networks do not only have to be read via the Jalapeño API Gateway but also written back to the routers in the network in a reasonable amount of time.
- `Availability`: As this application will allow for monitoring of the configurations of a network, availability has to be prioritized to ensure a complete and comprehensive experience.
- `Convenience`: This design goal will mainly come into play past the prototype implemented in this project. All the same it still needs to be considered for its impact on the underlying architecture of the application. The process of deploying and maintaining FlexAlgo configurations on a given network, especially large ones, is time consuming and complicated, as each algorithm on each node has to be manually set and configured. While a certain amount of synchronization is handled by the progapation of FlexAlgo definitions and especially discrepancies are handled with priorisation, this still leaves a large amount of liability and tedious work. The SR-App FlexAlgo needs to make this process simple and globally uniform. The complicated structures of the existing FlexAlgo configuration and possible altercations therefore need to be simplified and made into a good customer experience.

3.3. C4 Diagrams

[50]

3.3.1. Context Diagram

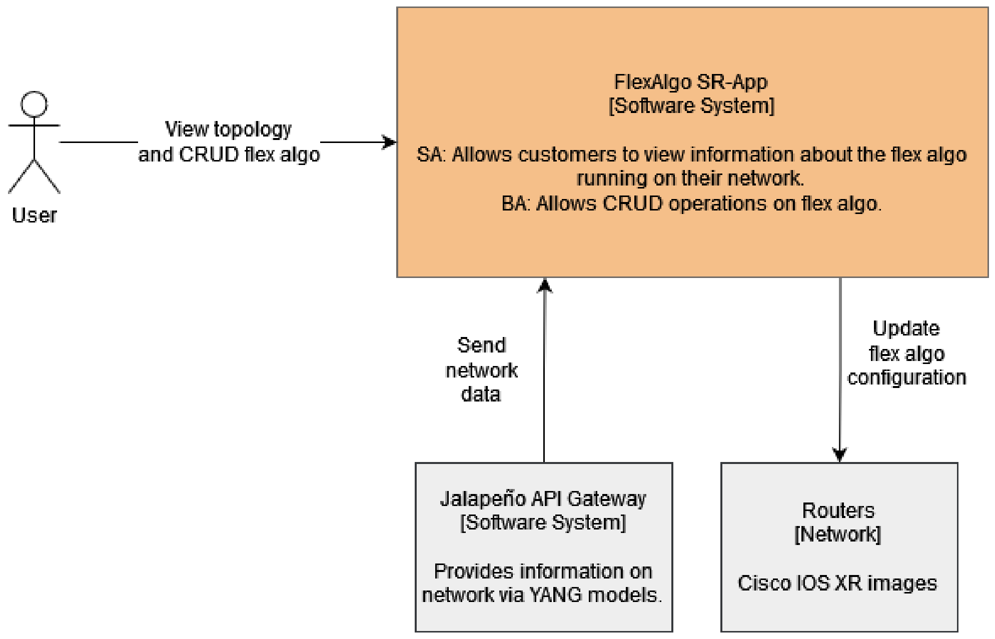

FlexAlgo SR-App
[Software System]
SA: Allows customers to view information about the flex algo running on their network.
BA: Allows CRUD operations on flex algo.
User
View topology and CRUD flex algo
Send network data
Update flex algo configuration
Jalapeño API Gateway
[Software System]
Provides information on network via YANG models.
Routers
[Network]
Cisco IOS XR images

Figure 3.1.: Context Diagram

As shown in figure 3.1 the `User` can interact with the SR application to see information of the network connected via the `Jalapeño API Gateway`.
In this research paper a prototype of this application is implemented. This prototype will interact with the Jalapeño API Gateway software system provided by INS to access and display network data. The final product, originally planned to be created in the BA of the team members, would implement the functionality of configuring the network itself by accessing the nodes directly.

The application built in this project will provide a view of all currently running network configurations. For this, the application was planned to retrieve data over the connection interface of the Jalapeño API Gateway. Unfortunately, the gateway wasn't in a stable state and could not be utilized yet. To circumvent this problem a direct connection to the gateway's ArangoDB was implemented, which provides the proper data in YANG model format.

3.3.2. Container Diagrams

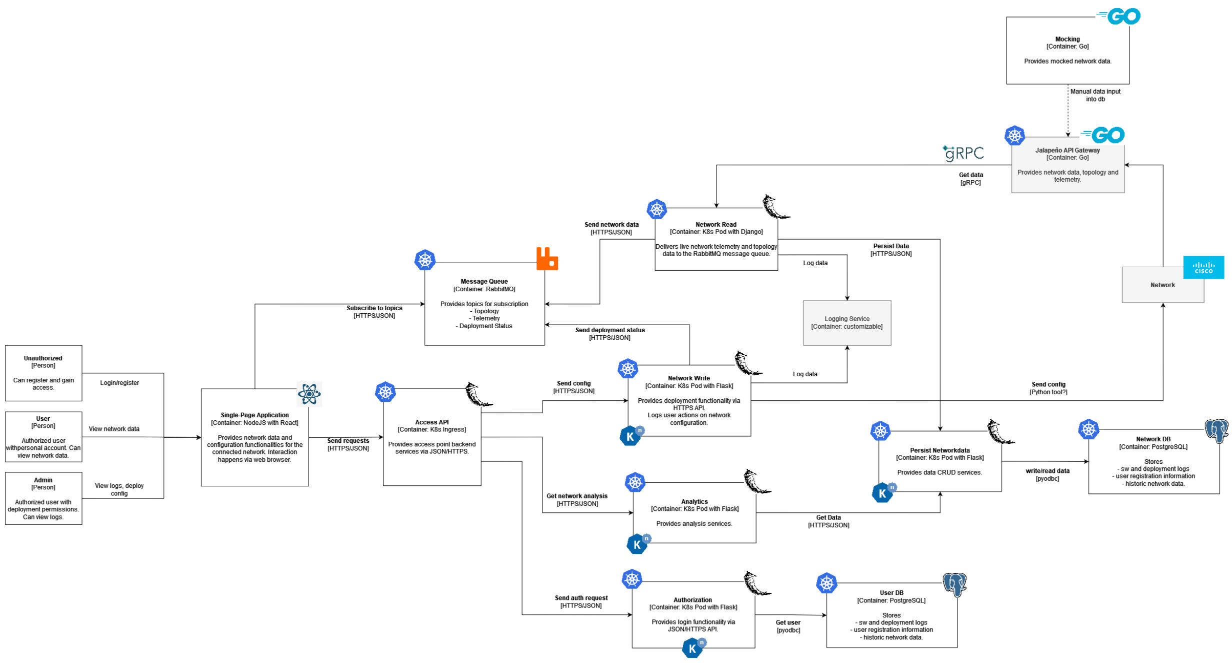

Mocking
[Container: Go]
Provides mocked network data.
Manual data input into db
gRPC
Jalapeño API Gateway
[Container: Go]
Provides network data, topology and telemetry.
Get data
[gRPC]
Send network data
[HTTPS/JSON]
Network Read
[Container: K8s Pod with Django]
Delivers live network telemetry and topology data to the RabbitMQ message queue.
Persist Data
[HTTPS/JSON]
Log data
Network
Message Queue
[Container: RabbitMQ]
Provides topics for subscription
- Topology
- Telemetry
- Deployment Status
Subscribe to topics
[HTTPS/JSON]
Send deployment status
[HTTPS/JSON]
Logging Service
[Container: customizable]
Unauthorized
[Person]
Can register and gain access.
Login/register
User
[Person]
Authorized user withpersonal account. Can view network data.
View network data
Admin
[Person]
Authorized user with deployment permissions. Can view logs.
View logs, deploy config
Single-Page Application
[Container: NodeJS with React]
Provides network data and configuration functionalities for the connected network. Interaction happens via web browser.
Send requests
[HTTPS/JSON]
Access API
[Container: K8s Ingress]
Provides access point backend services via JSON/HTTPS.
Send config
[HTTPS/JSON]
Network Write
[Container: K8s Pod with Flask]
Provides deployment functionality via HTTPS API.
Logs user actions on network configuration.
Log data
Send config
[Python tool?]
Persist Networkdata
[Container: K8s Pod with Flask]
Provides data CRUD services.
write/read data
[pyodbc]
Network DB
[Container: PostgreSQL]
Stores
- sw and deployment logs
- user registration information
- historic network data.
Get network analysis
[HTTPS/JSON]
Analytics
[Container: K8s Pod with Flask]
Provides analysis services.
Get Data
[HTTPS/JSON]
Send auth request
[HTTPS/JSON]
Authorization
[Container: K8s Pod with Flask]
Provides login functionality via JSON/HTTPS API.
Get user
[pyodbc]
User DB
[Container: PostgreSQL]
Stores
- sw and deployment logs
- user registration information
- historic network data.

Figure 3.2.: BA - Container Diagram

Container Diagram for BA

The Container Diagram 3.2 shows the architecture planned for the SR-App FlexAlgo with all necessary components and their relations.

A microservice architecture that runs on separate Kubernetes pods is planned. For better scalability and performance Knative will be used to make certain services serverless.
The spearation into microservices was chosen as it adjusts better to individual perfromance and scaling needs of use cases and functionalities.

The appplication will differentiate between an unauthorized user, a registered user and an authorized administrator. This provides better security due to separated functionalities and ensures only those with the right privileges can make network changes. A user that is registered and logged in can view the provided network data, while an admin can additionly deploy network configurations and see the logs of all user actions. This ensures traceability of network changes and non-repudiation of actions.

For further information about the services please check chapter microservices 3.5.3. A larger diagram can be seen in the Appendix 6.

`Access API`:
Handles all request based traffic from frontend to the backend services.

`Network Read`:
Connects to the Jalapeño API Gateway via gRPC. Handles reading and preparing network data and pushing it into the message queue.

`Network Write`:
Receives updated network configurations from the frontend via `Access API` and handles the coordination of the update on the network nodes.

`Log Service:`
Centraized logging system for the SR-App FlexAlgo. Receives and persists logs if neccessary and is requested by the customer. Third party software solution.

`Authorization:`
Handles all authorization and authentication functionalities for the whole system. Sensitive user data is persisted in the `UserDB` that is only accessible from this service.

`Persist Networkdata:`
Persists and retrieves information about the network and its configurations for further analysis purposes. Data is stored in `Network DB` that only this service accesses.

`Analytics:`
Retrieves data from `Persist Networkdata` and aggregates them for historic analysis of network behaviour.

Only a subset of the functionalities of the planned final product is implemented for this research project, as shown in the following section.

Container Diagram for SA

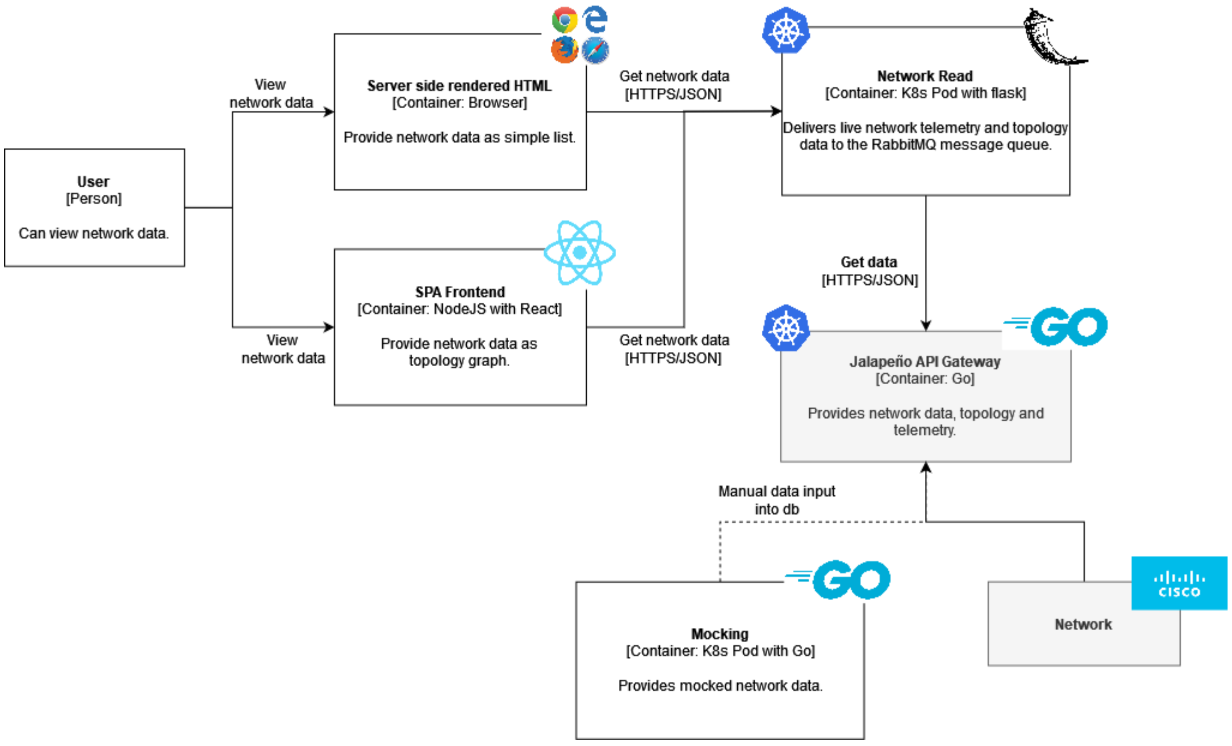

User
[Person]
Can view network data.
View network data
Server side rendered HTML
[Container: Browser]
Provide network data as simple list.
View network data
SPA Frontend
[Container: NodeJS with React]
Provide network data as topology graph.
Get network data
[HTTPS/JSON]
Get network data
[HTTPS/JSON]
Network Read
[Container: K8s Pod with flask]
Delivers live network telemetry and topology data to the RabbitMQ message queue.
Get data
[HTTPS/JSON]
Jalapeño API Gateway
[Container: Go]
Provides network data, topology and telemetry.
Manual data input into db
Mocking
[Container: K8s Pod with Go]
Provides mocked network data.
Network
cisco

Figure 3.3.: SA - Container Diagram

The Container Diagram for the prototype 3.3 shows the elements chosen for a working Proof of Concept. This includes the service to read the network configurations and a simple frontend. For further details consider the chapter of Use Cases 1.1.

Rather than constructing a bidirectional communication with a message queue already, the prototype communicates to the frontend over an API that is implemented in the `Network Read`

service.
The initial idea was to use gRPC between the `Network Read` service and the `Jalapeño API Gateway`. As the gateway is not ready to be implemented yet, there is a direct connection between the `Network Read` service and the gateways ArangoDB without gRPC.

The users can interact with the `Network Read` either by directly accessing server side rendered html templates via the API or by a demo SPA application.
To access the html pages the user needs to be in the INS VPN and access the
url `sa-sr-flexalgo.stu.network.garden/` with the desired postfix. List of API calls can be found in the Swagger documentation under the `/api` postfix or chapter API-Documentation 3.6.
The React.js demo can be run locally and connected either to a locally run instance of `Network Read` or the deployed one under the before mentioned url.

The `Mocking` functionality is a locally run golang script that generates mocked objects which are manually inserted into the ArangoDB of the gateway. For further information see chapter mocking 4.

3.3.3. Package Diagram

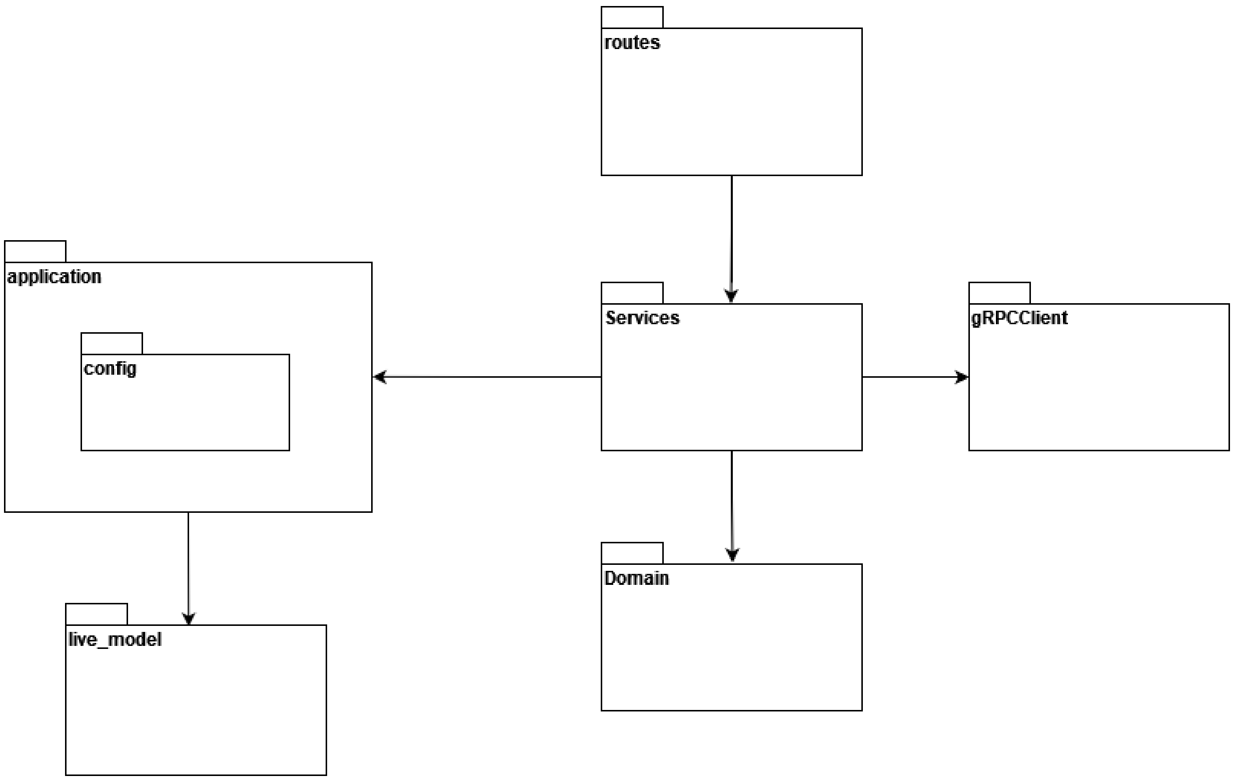


Figure 3.4.: Package Diagram of Microservice Network Read

As illustrated in the graphic 3.4 above, the application's code is divided into packages for better maintainability. Detailed information about the content of the packages can be found in the next section called Class Diagram 3.3.4.

As mentioned before the gRPC connection could not be implemented in the prototype. As such the package `gRPCClient` was only partly implemented in the feature branch feat_add-grpcc.

3.3.4. Class Diagram

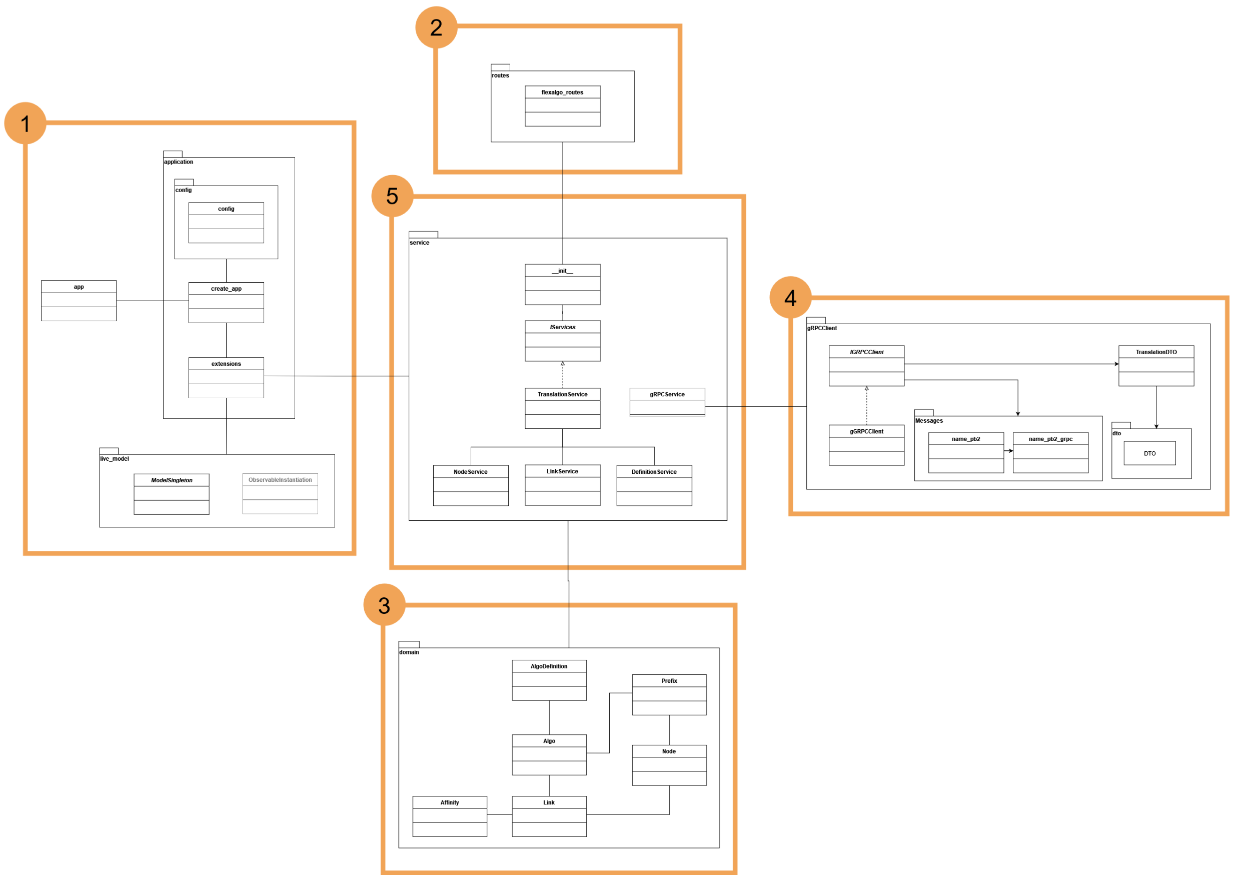


Figure 3.5.: Overview of Class Diagram of Microservice Network Read

The class diagram offers an overview of the project's classes and how they relate to each other. As the class diagram for this project is quite extensive, the diagram is split into five parts and explained in more details individually:

(1) Application and Live Model 3.3.4

(2) Routes 3.3.4

(3) Domain 3.3.4

(4) gRPC Client 3.3.4

(5) Service 3.3.4

Class Diagram - Application and Live Model

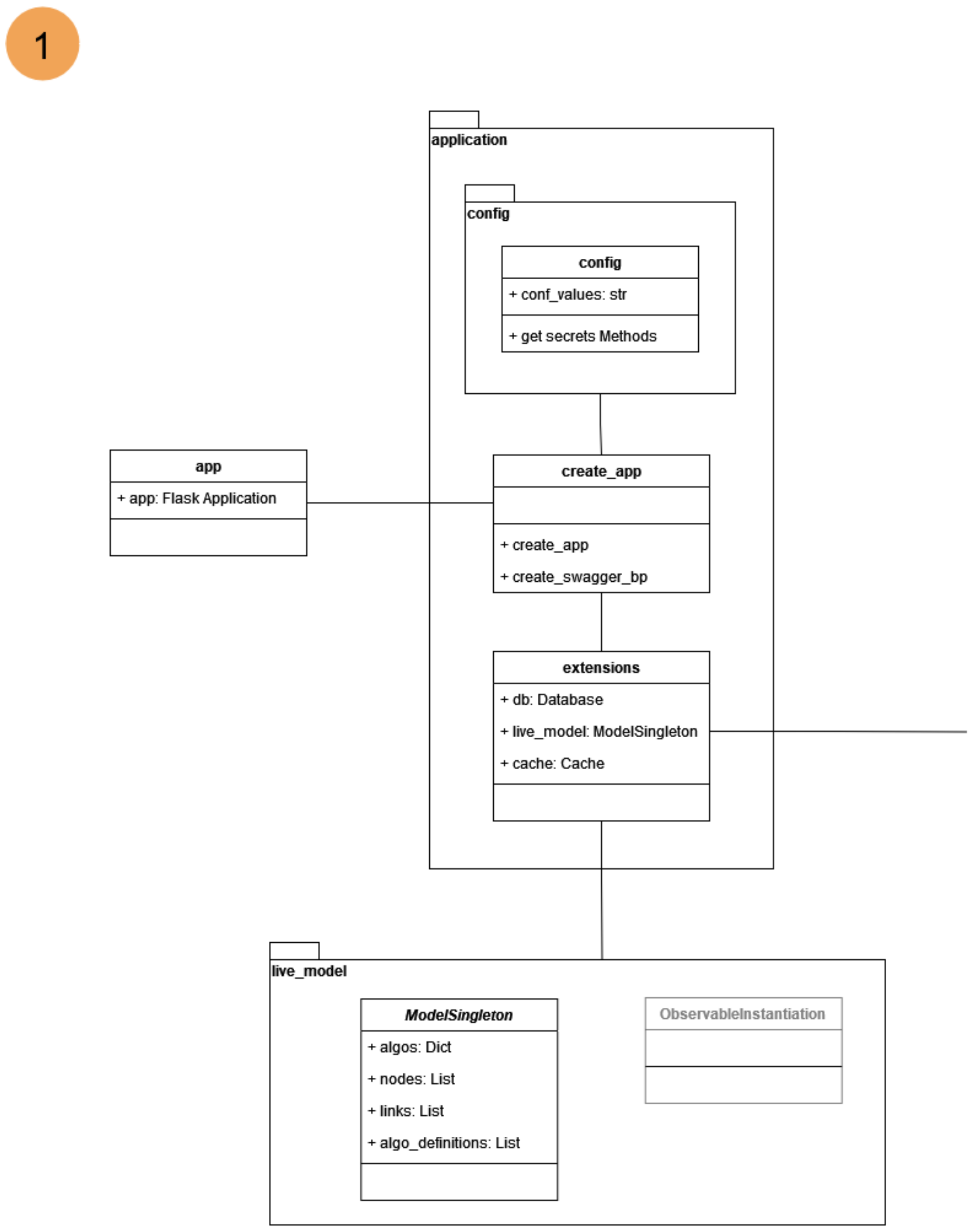


Figure 3.6.: Class Diagram of Microservice Network Read - Application and Live Model

`app`:
The class `app` lies outside the package `application` and provides the entry point of the `network read` service. This allows the app to be started by the CLI directly and provides a root level accessing point for the application. The class uses a `Factory Method`, derived from the `Factory Pattern`, provided by the `application` package, for easy startup of the application and complete

separation of the configuration into it's own package.

Package `application`:

Package `config`:

`config:`
This package isolates and provides all configuration values for the application like database connections, gRPC client initialization and caching information. This isolation is used to fulfill point III and V of the 12-Factor Methodology used for this project 3.1. Sensitive information can be saved in files in the sub folder secrets, like ArangoDB connection values. This allows the usage of the Kubernetes secret functionality and makes deployment into development, test and production stages very easy in accordance with point X of 12-factor Methodology 3.1.

`create_app:`
This class provides a factory method in which the start up of the application is defined. It loads and initializes all necessary basic components like the `Flask` application itself, all `Routes` of the provided API, resources like the `Arango Client`, `Caching Client` and the `gRPC Connection` once it will be utilized.

`extensions:`
This class holds all global variables that need to be accessed system wide, which for the prototype are the `db client`, `cache` and `live_model`.

Package `live_model`:

`ModelSingleton:`
The Live-Model holds an instantiated topology as a global singleton with all FlexAlgo configurations. This allows for better performance, as the network data is only instantiated once and subsequently updated with incoming network changes. This significantly reduces latency with data calculations. Additionally, when push methodology may be implemented in a further project, the observer pattern can be implemented to trigger sending the updates to the frontend. The topology data can be accessed via the `extensions` class.

`ObservableInstantiation:`
This class is not implemented in the prototype. Here an observer pattern can be developed that will receive all changes to the `live_model` and start the push mechanic to calculate any data updates and push them to the frontend.

Class Diagram - Routes

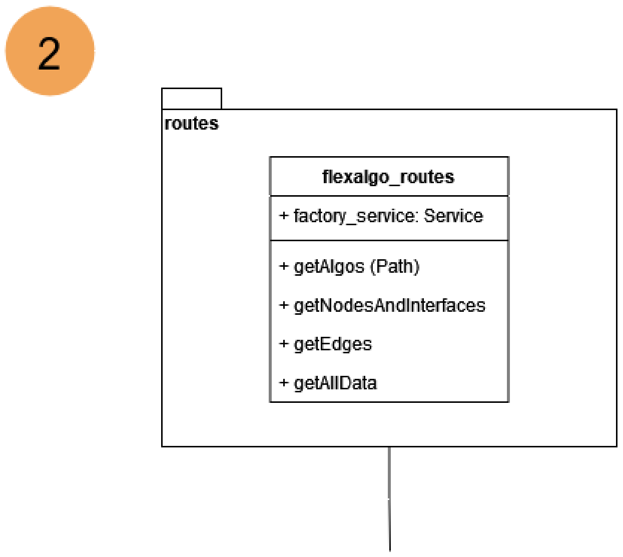


Figure 3.7.: Class Diagram of Microservice Network Read - Routes

`flexalgo_routes`:
To provide all FlexAlgo specific API methods, like html web pages or React.js frontend calls, this class will use the `TranslationServices` functionalities of package 3.10:

- routes for fetching current network configurations from ArangoDB
- routes for getting FlexAlgo related data shown as html lists
- routes to artifically update the network configuration data in the ArangoDB

Class Diagram - Domain

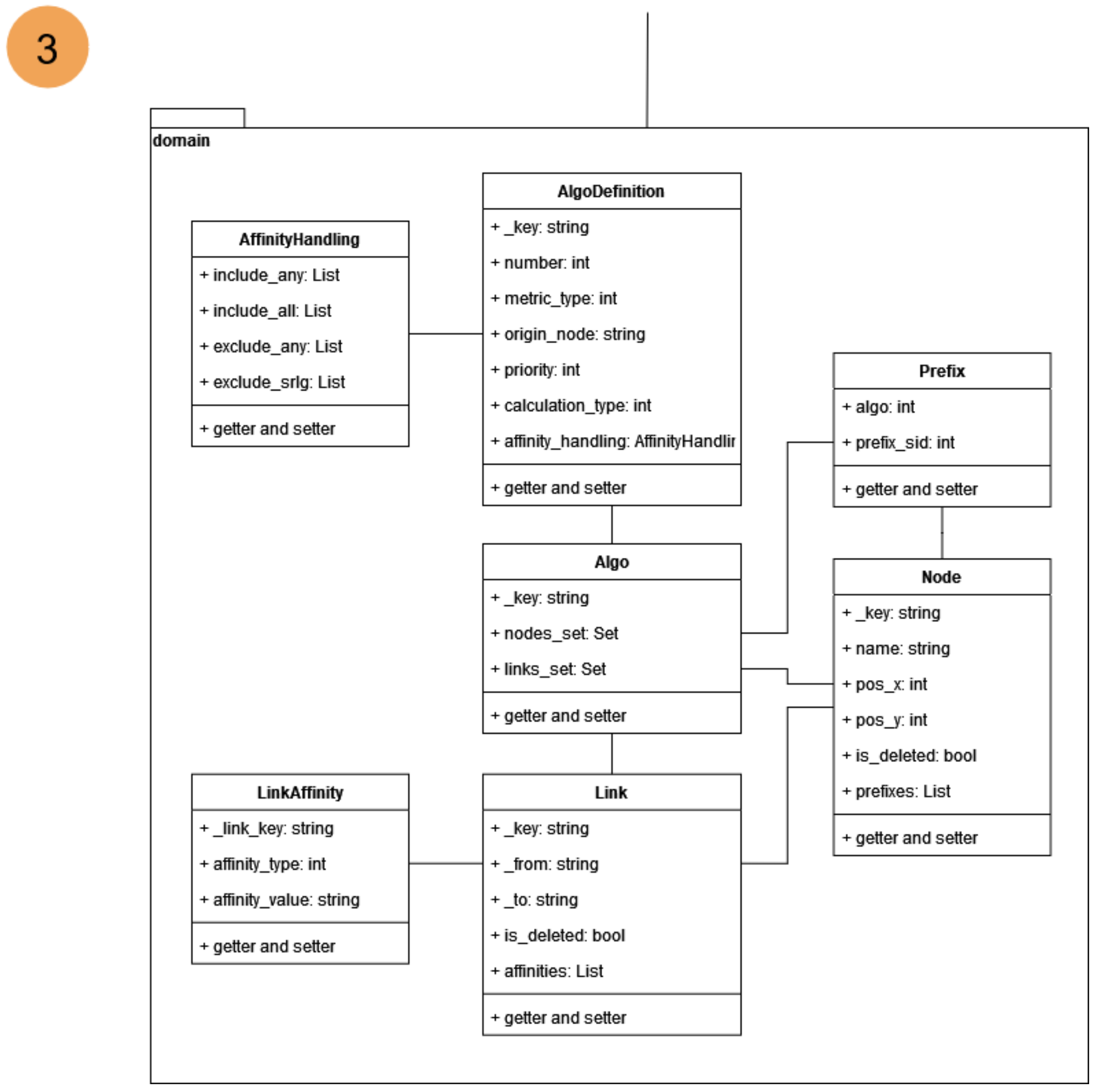


Figure 3.8.: Class Diagram of Microservice Network Read - Domain

The Domain part 3.3.4 of the Class Diagram was constructed to support the frontend tool of Michel Bongard for a clean adaption. As such it differs slightly from the domain analysis model of chapter 2.

`Algo:`
The Algo class represents the graph of a FlexAlgo. It utilizes the base elements of the domain and calculates all nodes and links this algorithm is defined on. It is used by the frontend to show the graphical representation of the algorithm.

`AlgoDefinition:`
The AlgoDefinition represents the algorithms defined on the network. It holds all necessary information for the FlexAlgo to be configured. As there may be inconsistencies in the network between configurations on different nodes, the field `origin_node` was added.This and the priority can be used to calculate the current active definition, as in chater 2 defined.

At the moment all definitions are gathered as a first step. This approach was chosen to help with network configurations once CRUD operations are possible. To have all definitions may be useful to debug possible problems on the network. Should this not be the case filter mechanics can easily be added to the calculations to gather only the active configurations.

`AffinityHandling:`
How each algorithm handles the Affinities (See chapter Segment Routing 2) is represented with the AffinityHandling class, which each AlgoDefinition holds as a variable.

`Prefix:`
Similar to the AlgoDefinitions, that are implemented for fault tolerance in the same situation, the prototype currently reads all `algo` specific prefixes for the nodes.
In a further project this functionality can be refined to handle these prefixes as globally unique `prefix-sid`s independent from the FlexAlgo. For easier debugging of the network configurations an additional check with all active sids to ensure this constraint is advisable.

`Node:`
The Node class represents all network objects capable of handling Segment Routing and FlexAlgo. To be compatible with the existing frontend topology graphic tool that was built by Michel Bongard, the Node class provides the attributes `pos_x` and `pos_y` for the longitude and latitude of a node. As in a normal network with configured FlexAlgo, each participating node depends on a `prefix-sid`, which is represented by the Prefix class attribute.
The BA thesis project SerPro mentions difficulties with deleted nodes and links when it comes to the deployment of new configuration. In accordance with their findings nodes deleted on the network are set to "deleted" with the `is_deleted` property. Should implementation of the CRUD functionality prove this step unnecessary the field can easily be removed.

`Link:`
In a standard configuration of a network with FlexAlgo the admin will define all relevant information on nodes. Link represent only the physical connection between those nodes and are not intelligent. All the same, FlexAlgo configuration theory often handles Links as separate objects on which values can be assigned. To make configuration more userfriendly this way of handling the FlexAlgos is adapted in this project. Links are also necessary for the utilization of the network topology graphic tool mentioned in the Algo 3.3.4 section.

Affinities in a FlexAlgo are defined on an interface of the corresponding node. As mentioned, to reduce complexity this defintion was moved to the link as member class Affinity 3.3.4. Similar to the Node class 3.3.4 a `is_deleted` field is added to the class to prevent problems with the CRUD functionalities of FlexAlgos.

`Affinity:`
Represents the FlexAlgo affinity definitions of the outgoing interface for a given link.

Class Diagram - gRPC Client

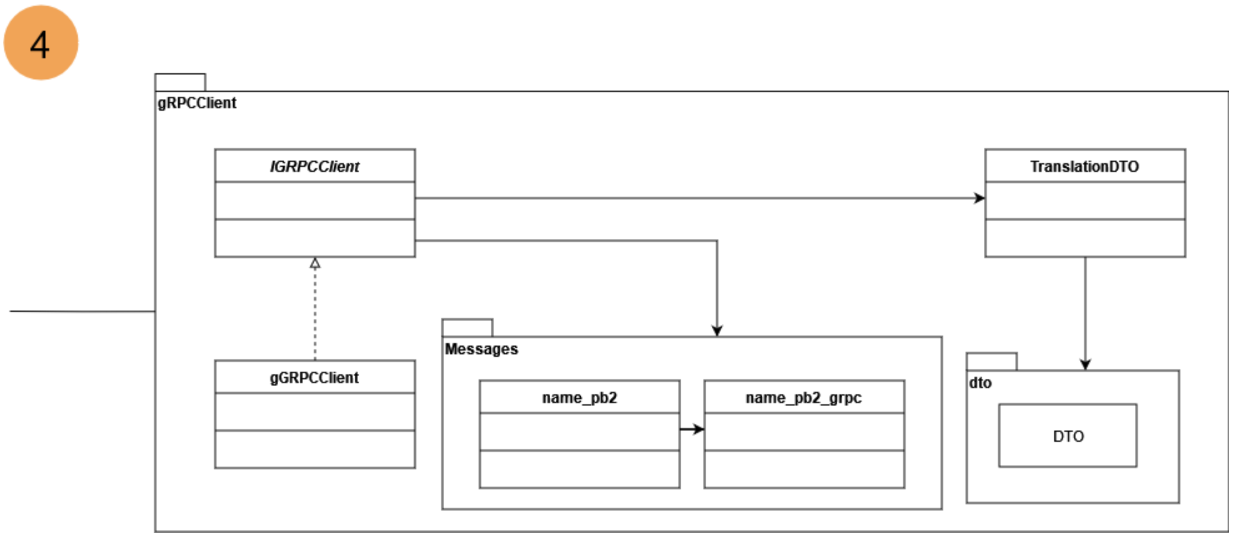


Figure 3.9.: Class Diagram of Microservice Network Read - gRPC Client

As mentioned, it was planned to utilize a gRPC connection to the Jalapeño API Gateway to access network data. During the research and early stages of the development it became apparent that changes had to be made to the Gateway to access the relevant data. As such it was decided to move the gRPC connection to a later project (For further information, please consider chapter 3.5.5). Instead of this connection, a direct connection to the Gateway's database is used, which is further described in package `Service` 3.10.
All the same, a simple connection to the Gateway was implemented and tested out during construction, which can be found on branch feat_add-grpcc. A migration to utilize the Gateway is therefore a simple matter without much technical overhead to be done.

`IGRPCClient:`
Provides an interface for other packages to interact with the gRPC service. This allows for coding against interface and easy interchangeability.

`gRPCClient:`
The implementation for the gRPC interface. There will be two clients implemented, one for requests and one for subscriptions. These clients will hold all logic necessary to connect to the Gateway, handle data retrievals and updates and implement error handling.

`TranslationDTO:`
GRPC connections utilize DTOs and the protobuf language for communication. They are defined by the gRPC service, i.e. the Jalapeño API Gateway. The DTOs are handled and transformed into objects, consumable by the service package. All necessary logic for this will be done in this class.

`dto:`
There are protobuf files predefined by the gRPC service. All data will be handled in DTOs and need to be synchronized between service and client. These DTOs are stored in this package. As such they are easily interchangeable should the gateway be updated.

`Messages:`
These are pre generated files that will be accessed from the client and handle the commu-

nication logic of the gRPC connection. They can be easily replaced should the Gateways' protobufs be updated.

Class Diagram - Service

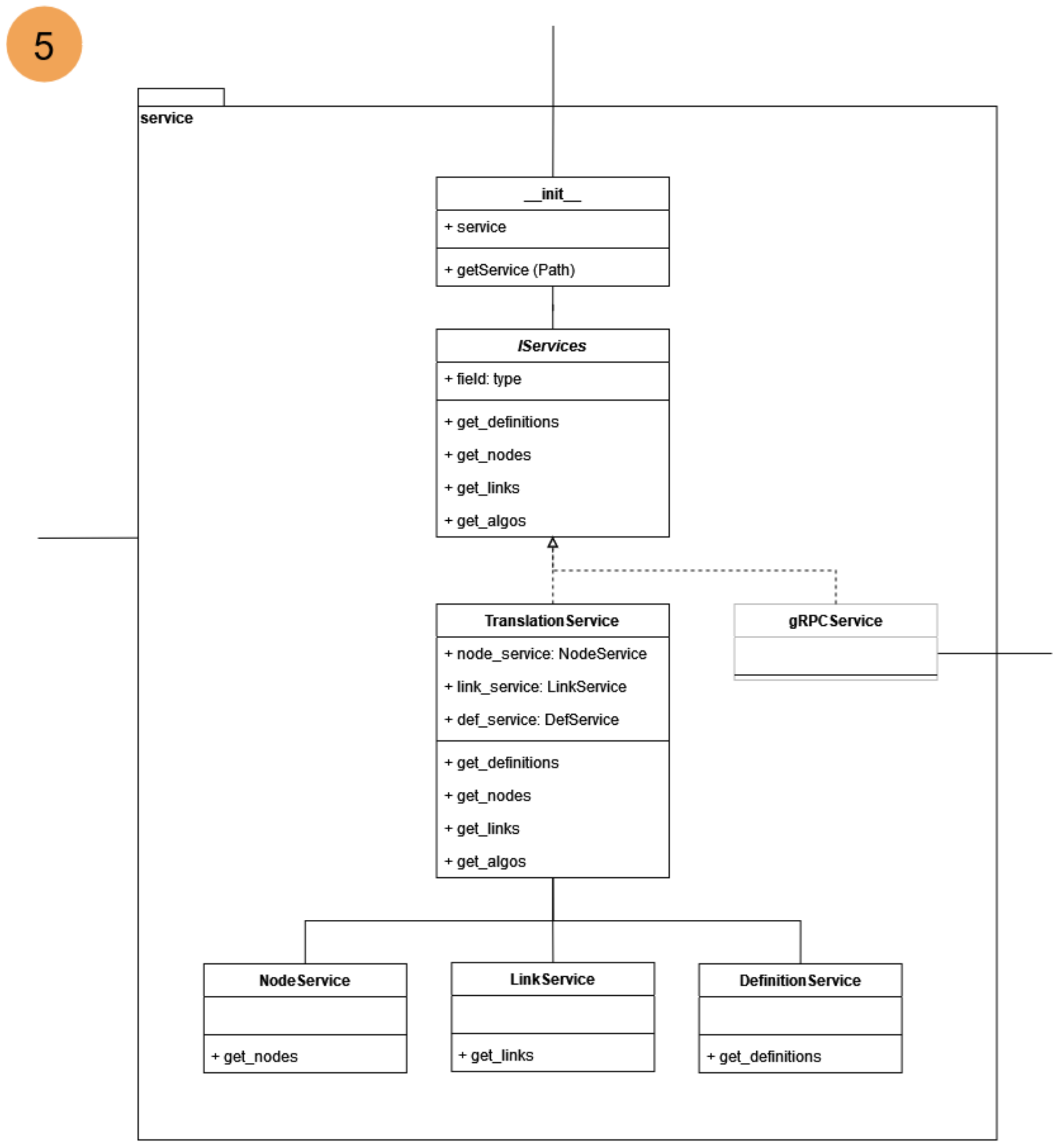


Figure 3.10.: Class Diagram of Microservice Network Read - Service

For the `Service` package a `Factory Pattern` was chosen. Each main service has to implement the basic functionality of the interface `IServices`. This is to ensure callers of this package can easily switch between connection types like gRPC and ArangoDB. Each class in this package is a service provider either for the overlaying routes package 3.3.4 or acts as a sub service for another service class. All business logic is implemented here. Different main services have to be registered in the `Factory Method` to be accessible to external packages.

`__init__`:
The `Factory Pattern Method` is implemented here as a connection point to other packages. Each main service can be accessed via string parameter. This to ensure the consumer packages do not have to know the technical details of this package. Currently all inputs return the `TranslationService` with the ArangoDB connection.
A gRPC request and subscription service are planned for a further project. The request service will be callable for the initialization of the `live_model` and then the subscription for network update handling.
For the prototype only the `arangoDBService` is implemented in the stable source code.

`ServiceInterface`:
This class represents the service interface that all services callable from other packages must implement. It is the blueprint of the service class and allows for programming against an interface. As such replacing the current `arangoDBService` with the correct gRPC services will be easy to implement without having to change the consumer packages.

`TranslationService`:
The main service to handle the ArangoDB connection and all necessary business logic. It implements the `ServiceInterface` interface.
To reduce the complexity of this class and separate the different concerns of the data handling, sub-services were extracted during the development of this class. As such this class merely calls sub-services for `Node`, `Link` and `Definition` handling. Only the graph calculation functionalities still remain in this class.

`NodeService`:
Within this class the functionality to update or read the nodes' related database entries are implemented. Holds all necessary business logic for Nodes.

`LinkService`:
Within this class the functionality to update or read the links' related database entries are implemented. Holds all necessary business logic for Links.

`DefinitionService`:
Within this class the functionality to update or read the FlexAlgo definitions' related database entries are implemented. Holds all necessary business logic for FlexAlgo configurations.

`gRPCService`:
Originally it was planned to build a serialized service with gRPC in a fitting `gRPCService`, which would retrieve the necessary objects from the package `gRPCClient` in 3.9.
At the moment this service is only implemented in branch feat_add-grpcc as the connection to the Gateway is not possible yet.

`UpdateData`:
To simulate the update functionality of the gRPC connection, update methods were implmemented in the Node 3.3.4, Link 3.3.4 and Definition Service classes 3.3.4. The class `UpdateData` was implemented to provide this updated data.

3.4. Sequence Diagrams

Sequence diagrams for the main Use Cases (UC) 1.1 are documented in the following section. They show the workflow of the application while Use Cases are handled. The communication between frontend and backend will be request based, as this research project does not yet implement a real-time frontend.

See a List of Flex Algo Information

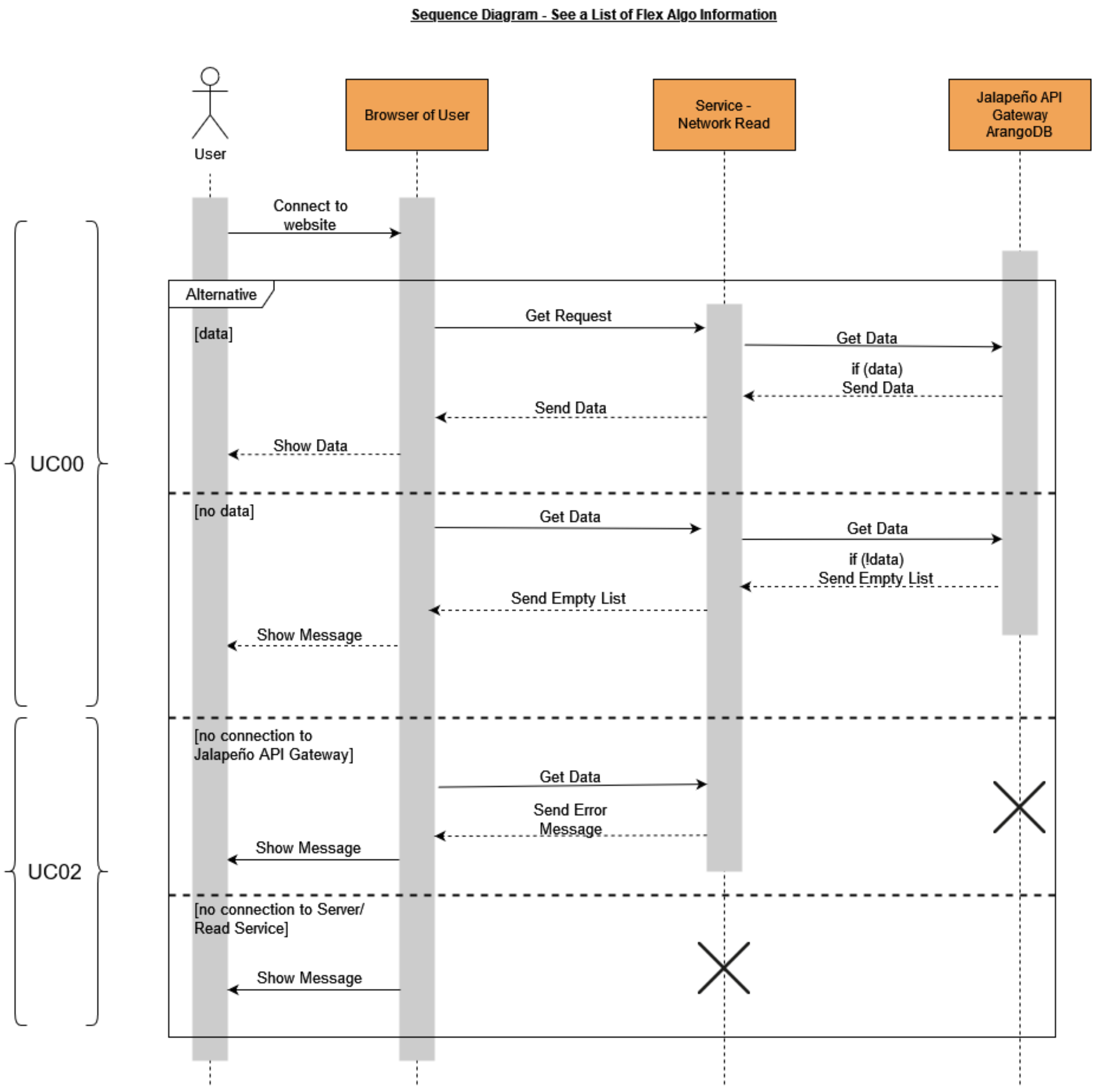


Figure 3.11.: Sequence Diagram - See a List of Flex Algo Information

A User can open a standard browser and call one of the API urls from the backend. This request causes the `network read` service to connect to the ArangoDB of the Jalapeño API Gateway and retrieves the requested network data. After applying the necessary business logic to the data

the `network read` sends a rendered html page or json list back to the browser to display for the User. Is the connection to the Gateway not possible an error message per html or HTTP code is returned. Similarly, the browser will display an error code if the `network read` service is not reachable.
Originally a gRPC connection was planned to the Gateway, but was not possible for this project. In a further project this will be remedied.

Additionally, when the SPA frontend is fully developed a message will be shown should an empty list be returned from the database. This to ensure the User is not confused about an empty display slot. In this prototype a mere demo SPA was implemented for the optional use cases UC05-7 1.1. The User will access the `network read` service via this React.js application instead of directly. The frontend will then call specific API calls of the backend, which will operate as shown in the diagram 3.11.

See Live Updates

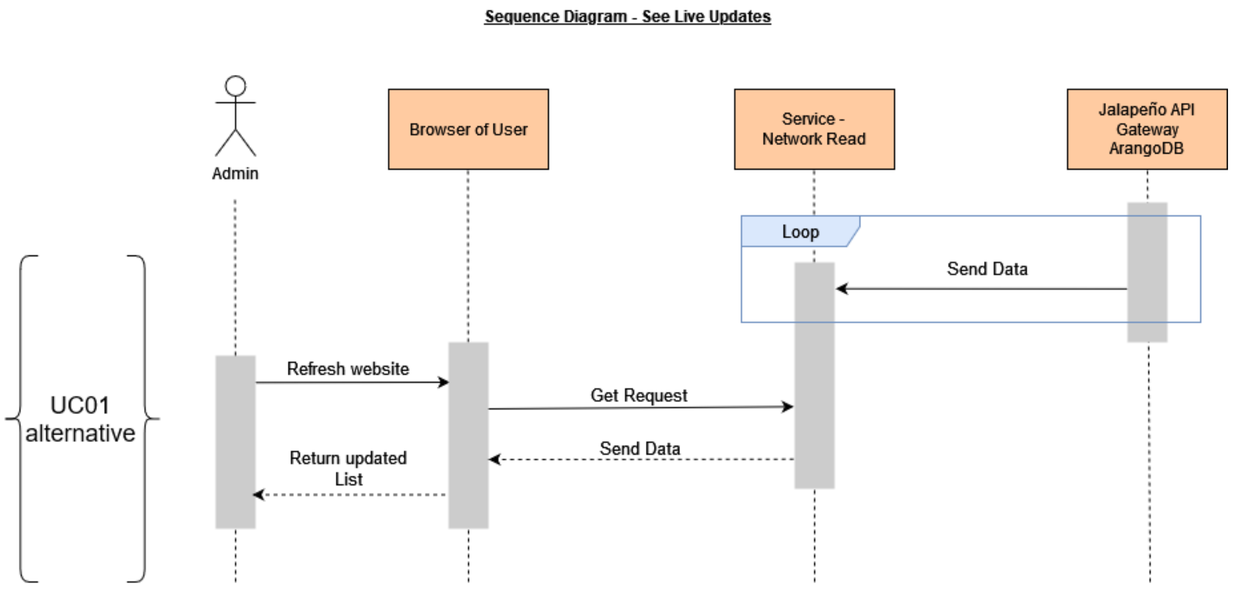


Figure 3.12.: Sequence Diagram - See Live Updates

If there are any updated configurations on the network devices, the User can simply refresh the website manually to retrieve the new information from the `network read` service.
As currently the gRPC connection to the Jalapeño API Gateway is not possible, the loop functionality of the Gateway sending data updates to the server will not be implemented in this project. When the Gateway receives the necessary updates a subscription architecture can be implemented that will update the `network read` service in an automatic loop.

Logging

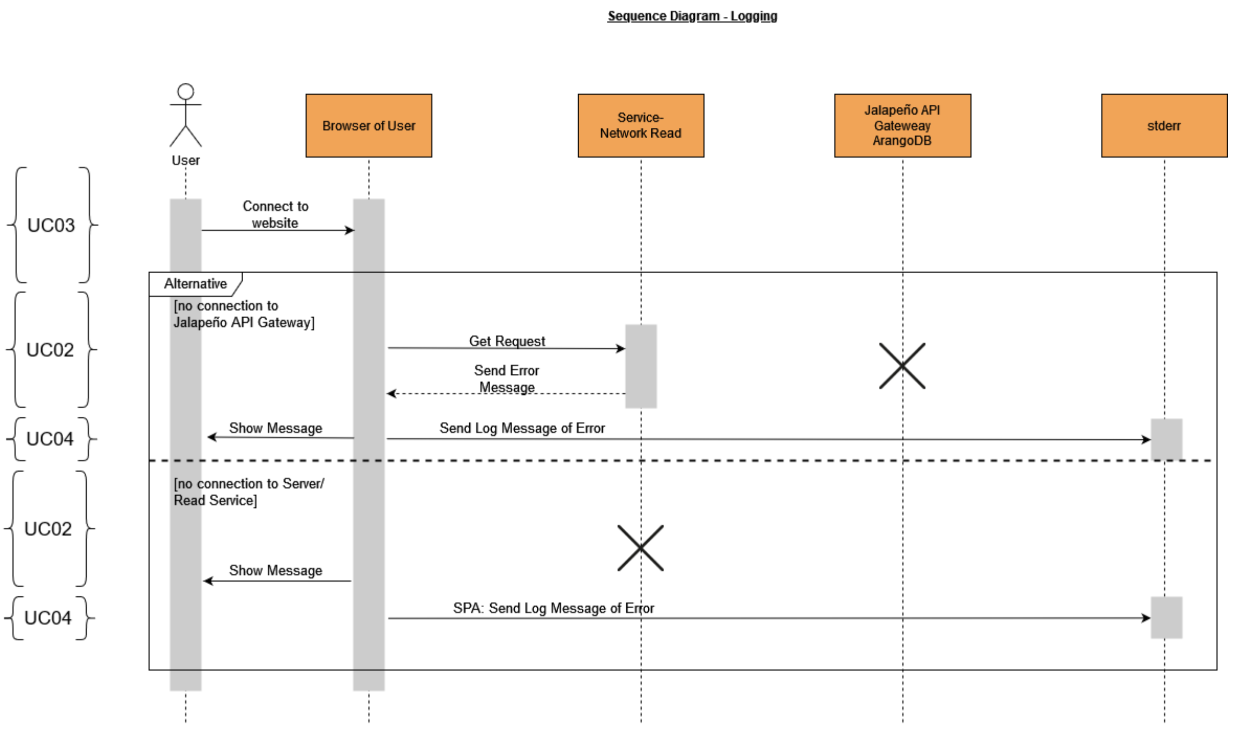


Figure 3.13.: Sequence Diagram - See a List of Flex Algos

Logging of the application will happen in the stederr for simplicities' sake. If an error occurres during the handling of requests in `network read` not only will an error message be sent to the User but a log will be written into the stream. With this design a third party logging service can be implemented with minimal overhead for the persistence of logs. Errors from browser are only logged if the User is utilizing the SPA frontend.

3.5. Technology Decisions

This section shows which languages, framworks and technologies were chosen for the project and why.

3.5.1. Technology Stack

As performance is an important factor in the application, especially in the `Network Read` service, the technology stack was designed to be as lean and fast as possible.

Service/Component	Technology
`network read`	• Flask FW for Python[54] • gRPCio FW for Python client[4] • poetry[25] • Libraries Production – python-arango[32] – arango-orm[33] – Flask-Swagger-Ui[17] – Flask-Caching[14] – Flask-WTF[18] – gunicorn[22] • Libraries Development – pytest[52] – pytest-cov[52] – black – mypy – lxml • Cache[15] – simplecache for prototype – For production environment of the final software filesystemcache or external system like redis.[**caching-gunicorn**]
`mocking tool`	• golang • json
`react demo`	• JavaScript • React.js • Sigma.js[45] • Axios[3]

Table 3.1.: Technology Stack

3.5.2. Language

The SR-App series applications are to be implemented in golang or Python with a React.js frontend for the graphical user interface. As the team members had more interest and experience with python than go, it was decided to write the prototype in this language. At first the Django framework was considered, as other applications had successfully worked with this framework already. Django is a substantial technology that brings a lot of features with it that were unnecessary for the project. On further development of the architecture, especially the decision to use microservices and a largely serverless architecture, Django was therefore deemed to heavy weight. FastAPI and Flask were then considered, as they are much more lightweight frameworks with minimal features out of the box, but with a large library of extensions to supplement the needed functionalities. Upon further researching the team decided on Flask, as it provides an easy way to combine it with a gRPC framework. For unifromities and maintainabilities sake all other microservices are also planned with Flask. But it may be worth the consideration that FastAPI may be utilized for the other services, as it is faster than Flask and may therefore be more suitable. Possible extensions for orm will be possible for both frameworks, with libraries like tortoise-orm.

Python

SR-Apps are to be implemented in either golang or Python, with a React.js frontend for the graphical user interface. Python was chosen for this project as the team members had more interest and experience with this language.

Flask

[31][29] At first the Django framework was considered, as other applications had successfully worked with this framework already. However, Django is a substantial technology that brings a lot of features with it that were unnecessary for the project. On further development of the architecture, especially the decision to use microservices and a largely serverless architecture, Django was therefore deemed too heavy weight. FastAPI and Flask were then considered, as they are much more lightweight frameworks with minimal features out of the box. Both have a large library of extensions to supplement the needed functionalities. The team finally decided on Flask, as it provides an easy way to combine with a gRPC framework. For unifromity and maintainability, all other microservices are also planned with Flask. But it may be worth the consideration that FastAPI could be utilized for the other services, as it is faster than Flask and may therefore be more suitable. Possible extensions for orm will be possible for both frameworks, with libraries like tortoise-orm.

Other Languages

The `mocking tool` was adapted from a previous SR-App project and was therefore already implemented in golang. Similarly the `react demo` was adapted from an application by Michel Bongard and was already implemented in React.js and Sigma.js.

Static Types

[47] To ensure code stability, static typing was added where meaningful to the Python code. The typing is checked in a job in the testing pipeline and will cause a failure if not properly set. Bugs can be found more easily and unexpected behaviour of the application from mistaken data types can be prevented with typing. It also increases readability of code and therefore ensures easier maintenance. This more than makes up of the drawback of making the inherentely very dynamic Python language more static.

3.5.3. Microservices

Scalability and performance needs for the different use cases of the application vary widely in severity. While the process of reading network data has to be able to compute large amounts of data in real time, seeing the logs of configuration deployments will not only be used much more rarely and need significantly less computing power. Additionally deployment of configurations needs to be able to roll out and roll back simultaneous deployment of data over a wide range of objects. To manage these differing needs, a microservice architecture was chosen for the application. Different use cases can be handled separately and isolated from each other. This imporves parallelity, security, isolation of concerns and allows for individual and highly specialized design considerations.

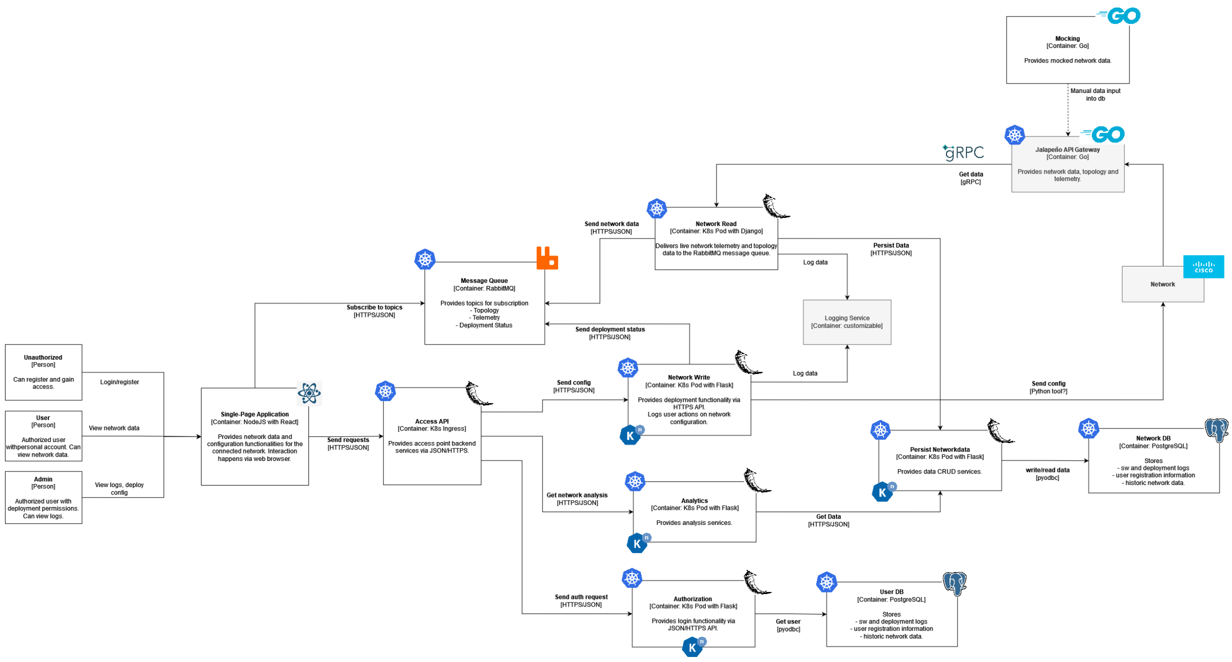

Mocking
[Container: Go]
Provides mocked network data.
Manual data input into db
gRPC
Jalapeño API Gateway
[Container: Go]
Provides network data, topology and telemetry.
Get data
[gRPC]
Network Read
[Container: K8s Pod with Django]
Delivers live network telemetry and topology data to the RabbitMQ message queue.
Send network data
[HTTPS/JSON]
Persist Data
[HTTPS/JSON]
Log data
Network
Message Queue
[Container: RabbitMQ]
Provides topics for subscription
- Topology
- Telemetry
- Deployment Status
Subscribe to topics
[HTTPS/JSON]
Send deployment status
[HTTPS/JSON]
Logging Service
[Container: customizable]
Unauthorized
[Person]
Can register and gain access.
Login/register
User
[Person]
Authorized user withpersonal account. Can view network data.
View network data
Admin
[Person]
Authorized user with deployment permissions. Can view logs.
View logs, deploy config
Single-Page Application
[Container: NodeJS with React]
Provides network data and configuration functionalities for the connected network. Interaction happens via web browser.
Send requests
[HTTPS/JSON]
Access API
[Container: K8s Ingress]
Provides access point backend services via JSON/HTTPS.
Send config
[HTTPS/JSON]
Network Write
[Container: K8s Pod with Flask]
Provides deployment functionality via HTTPS API.
Logs user actions on network configuration.
Log data
Send config
[Python tool?]
Persist Networkdata
[Container: K8s Pod with Flask]
Provides data CRUD services.
write/read data
[pyodbc]
Network DB
[Container: PostgreSQL]
Stores
- sw and deployment logs
- user registration information
- historic network data.
Get network analysis
[HTTPS/JSON]
Analytics
[Container: K8s Pod with Flask]
Provides analysis services.
Get Data
[HTTPS/JSON]
Send auth request
[HTTPS/JSON]
Authorization
[Container: K8s Pod with Flask]
Provides login functionality via JSON/HTTPS API.
Get user
[pyodbc]
User DB
[Container: PostgreSQL]
Stores
- sw and deployment logs
- user registration information
- historic network data.

Figure 3.14.: BA - Container Diagram

A larger Diagram can be seen in the Appendix 6.

Access API

As microservice architecture has a complex adressing system by nature, an uniform entrypoint was chosen to decouple frontend from the individual services in the backend. The `Access API` will provide an extensive API for all user interactions and will then redirect the requests as needed. Additionally this service will provide the SPA application for new users. Consideration to make this service serverless may be made. But as this has to handle all user interactions and a wide variety of requests it was chosen not to.

Technologies: Python framework Flask, Kubernetes

Network Read

The `Network Read` service handles all data retrieval from the network. It is connected to the Jalapeño API Gateway via gRPC and manages all incoming network data. It has the highest demands on scalability and performance, as networks can grow to include more than thousand

nodes. It's functionality includes filtering and aggregation of the network data, handling updates to the existing model data, calculation of the FlexAlgo graphs and pushing data to frontend for live updates.
To be able to analyse the FlexAlgo data and performances the Service will also send the gathered data from the network to be persisted by the service `Persist Network Data`.

Technologies: Python framework Flask, Kubernetes

This microservice was implemented as a prototype of the application during this project.

Network Write

As updating FlexAlgo configurations has vastly different functionality and performance needs, it was decided to separate Read and Write functionalities into two services. The `Network Write` handles all updates to the network configuration and as such communicates directly with the Network objects. How this communication looks, with paralelization, error handling and rollback needs, has not yet been determined.
The frontend will send updates to the network via the `API Application` service. The `Network Write` will then handle the deployment of the new configuration on all network devices and send updates of the deployment status to the frontend via message queue. All user interactions will be logged for traceablility, which will include user information, CRUD operations, changes, status of and any possible errors during deployment.

Technologies: Python framework Flask, Kubernetes, Knative

Message Queue

For communication between frontend SPA and the services that provide push updates, ie `Network Read` and `Network Write`, a message queue technology was choosen. For further details about the communication, see chapter 3.5.5. This message queue will provide network and deployment data in the form of subscriptions to the frontend. It will be hosted in the same Kubernetes cluster and have an endpoint outside the system. This and the `Access API` will provide the only endpoints outside of the cluster. It may be advisable to utilize this queue also for logging and persistency of network data and other communication between services, but this decision will need further research.

Technologies: message queue like RabbitMQ, Kubernetes

Authorization

As the application will need to be secured to prevent tampering with the network, an authorization mechanism is to be implemented. The `Authorization` service will handle all login and authorization needs, to prevent unnecessary duplication of this functionality. To make the authorization easily managed in the whole system a token based authentication will be chosen, which the Flask framework supports. This service will have its own database to handle all persistency needs in user management and cleanly isolate sensitive user data from the rest of the application.

Technologies: Python framework Flask, Kubernetes, Knative, PostgreSQL

Network DB

How FlexAlgo configurations behave long term on a live network proves interesting to a network engineer, especially with historic data to compare it to. As such a persistence of the gathered network data from `Network Read` is planned. Basic functionality of this is already tested in the prototype in form of the caching functionality and proves to be useful not only for persistency but also synchronization between different instances of `Network Read`, see chapter 4.2.0.5.

Technologies: PostgreSQL, Kubernetes

Persist Network Data
The microservice `Persist Networkdata` will handle all accessing of the PostgreSQL database for network data. This reduces duplication and performance overhead of the complex db connection functionality and orm code. Other services can call an API to access and persist their specific data needs. This service may be proven superfluous should the overhead have negligent impact on the performance of `Network Read` and other services accessing the database. This will have to be observed in a more extensive testing system with the impact all services will have on performance of the application. As an alternative the `Analytics` service may handle all database connections for the `NetworkDB` and provide an API for the other services.

Technologies: Python framework Flask, Kubernetes, Knative

Analytics
An analytics view of the network data and the configurations of FlexAlgos may be interesting and educational for network engineers. This is an optional functionality if resources are suffiecient for implementation of this. As such an `Analytics` services is planned that will handle accessing and aggregating network data.

Technologies: Python framework Flask, Kubernetes, Knative

3.5.4. Serverless Architecture

[23] The microservice architecture chosen for the application allows not only for high isolation of functionalities but also an individualized addressing of performance needs. Certainly the highest demands on performance are put on the `Network Read` service, who has a 100% uptime demand to catch all gRPC updates and calculate the computation heavy frontend graphs of the topology and FlexAlgos.
On the other hand a service like `Network Write` only has to run sparingly when a configuration is deployed. Still, while the deployment is running, it has to handle a large workload of managing the simultanous deployment to the different network devices, handle errors, rollbacks and status updates. But after a first initial configuration period with many changes to the network, a stable FlexAlgo configuration will be reached and as such this service will be used much less often. And other use cases like seeing historic data, login into the application, seeing logs, etc. are also only used sparingly for specific functionalities.

As such, while the `Network Read` service will have a more traditional architecture, other services can easily be deployed as serverless. They will therefore only run as long as any given functionality is called. This will largely conserve computing power and impact on other services. Additionally these applications and their deployments can be kept minimalistic and will bring improved observability, easier deployments and easier maintenance.

3.5.5. Communication

gRPC

[4][21] The Jalapeño API Gateway provides a gRPC server with a request and streaming service, where a SR-App can connect to to receive network data. Initialy it was planned for the prototype to utilize the request service and in a further step the streaming service to access the network data.
In contrast to other existing SR-Apps the FlexAlgo technology does not utilize IPv6 but MPLS protocols. As such there is information missing from the router in the network, more precisely the fields in the MultiTopologyIdentifier list are not devilered to the Jalapeño API Gateway. This means the Gateway is not able to consistently send out data at the moment, as this causes problems with its cache. Additionally, the YANG model that contains the FlexAlgo relevant data sends configuration data. While operations data are directly fed into the telemetry Service and can be consumed without changes to the gateway, configuration data is piped to the topology service. This means adjustments to the gateway are necessary to access the releveant information of the network. With these restrictions the gRPC communication to the Gateway are not be implemented in this prototype.

To mitigate the repercussions of this, a direct connection to the ArangoDB of the Jalapeño API Gateway is implemented, where all topology data of the network is stored. The architecture will be done such, that a replacement of the ArangoDB connection with a gRPC client can be done with minimal adjustments to the code. A basic working connection to the gateway is implemented in the `feat_add-grpc` branch and can be build upon, too.

The necessary changes to the proto files for the gRPC connection are proposed in a github fork of the original protorepo of the Jalapeño API Gateway project by the team.

```
message SubTlvNode {
  repeated uint32 include\_any = 1;
  repeated uint32 include\_all = 2;
  repeated uint32 exclude\_any = 3;
  repeated uint32 exclude\_srlg = 4;
}

message FlexAlgoDefinition {
  required uint32 flex\_algo = 1;
  optional uint32 metric\_type = 2;
  optional uint32 calculation\_type = 3;
  optional uint32 priority = 4;
  optional SubTlvNode sub\_tlv = 5;
}

message LsNode {
  required string key = 1;
  optional string id = 2;
  //further fields omitted for brevity
  repeated uint32 sr\_algorithm = 19;
  repeated FlexAlgoDefinition flex\_algo\_definition = 20;
}

message SubTlvsLink {
  optional uint32 sub\_tlv\_type = 1;
  optional string sub\_tlv\_value = 2;
}

message AppSpecLinkAttr {
  repeated SubTlvsLink sub\_tlvs = 1;
}

message LsLink {
  required string key = 1;
  optional string id = 2;
  //further fields omitted for brevity
  repeated AppSpecLinkAttr app\_spec\_link\_attr = 23;
}
```

Listing 3.1: protobuf class topology.proto

python-arangoDB

[32][2] As previously discussed a gRPC connection to the Jalapeño API Gateway is not possible for the prototype. A direct connection to the ArangoDB of the gateway was established instead. As FlexAlgo configurations are handled in the YAND model `Cisco-IOS-XR-clns-isis-cfg`, the Jalapeño handles this as topology data and not telemetry data. The information is therefore stored in the ArangoDB of the gateway. Connection is made via the python-arango and arango-orm packages of python.
A direct translation via the arango-orm package was not utilized, as it meant added complexity that would be rendered useless with the replacement by a gRPC connection. The translation of nested lists and schemas proved to be especially time consuming with the or-mapper.

The translations in the service package are made to be used in gRPC with minimal changes as demonstrated in listings 3.2 and 3.3.

```python
def compute_algo_definition(self, node):
      if "flex_algo_definition" not in node:
          return None
      result = []
      for defin in node["flex_algo_definition"]:
          ad = AlgoDefinition(
              number=defin["flex_algo"],
              origin_node=node["_key"],
              metric_type=defin["metric_type"],
              calculation_type=defin["calculation_type"],
              priority=defin["priority"],
          )
          if "sub_tlv" in defin:
              ad.affinity_handling = self.compute_affinity_handling(defin["sub_tlv"])
          result.append(ad)
      return result
```

Listing 3.2: AlgoDefinitionFunction now

```python
def compute_algo_definition(self, node):
      if node.flex_algo_definition is None:
          return None
      result = []
      for defin in node.flex_algo_definition:
          ad = AlgoDefinition(
              number=defin.flex_algo,
              origin_node=node.key,
              metric_type=defin.metric_type,
              calculation_type=defin.calculation_type,
              priority=defin.priority,
          )
          if defin.sub_tlv is not None:
              ad.affinity_handling = self.compute_affinity_handling(defin.sub_tlv)
          result.append(ad)
      return result
```

Listing 3.3: AlgoDefinitionFunction with grpc models

Only the accessing logic of the fields of the dictionaries has to be changed to work with a class access syntax, otherwise the translation will remain the same. This means migrating to gRPC will result in minimal changes to the services. The Jalapeño API Gateway itself can be made to implement the FlexAlgo definitions as they are on the network without any transformations and simply return the given structure from the underlying YANG model.

API

[24] In the prototype the communication between backend and frontend is handled via an API. Changing the prototype to support real time communication with the frontend, either via web-socket or message queue, will be easily done. The archtiecture was chosen such, that the frontend communication is isolated behind interfaces. The only additional update that will be needed, is the implementation of the observer pattern to the live model topology. This will trigger the communication to the frontend when the network data is updated via the gRPC streaming service and allow for a push architecture.

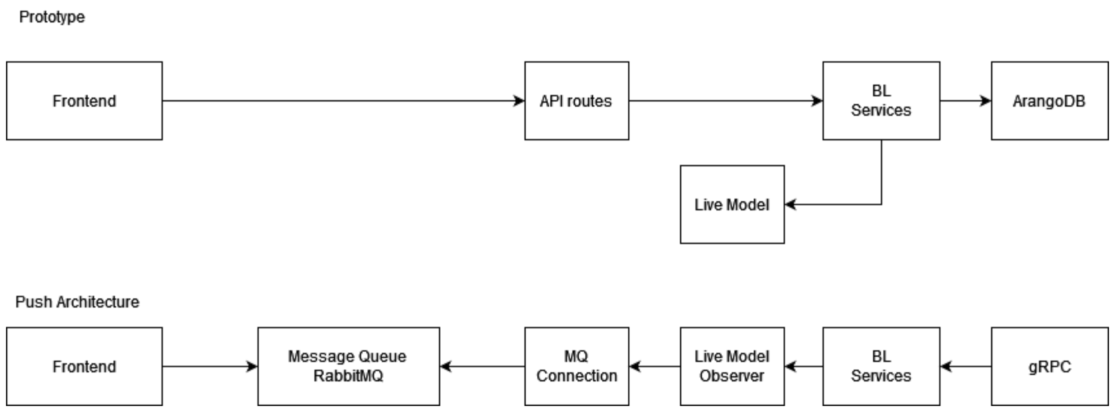


Figure 3.15.: Overview of Communication now and planned

Message Queue

[34] The architecture of the SR-App plans a message queue to handle communication with the frontend. This decouples frontend from the performance intensive `network read` service and allows for easier isolation of the microservices. The frontend can handle pushed updates from the network via subscriptions on the message queue. This can also be used to not only receive network updates from `network read` but also configuration status from `network write` without having to implement direct communication with both services separately. As such the whole userbase can be notified for network updates, receive configuration status updates and can be locked while a deployment is under way. This will make serializing and handling of updates much easier and circumvent problems like lost updates and contradicting configurations.

Alternatively to the queue a websocket[53] technology was considered. A great amount of users could be easily handled with the Kubernetes scaling mechanisms but it will bring a much tighter

coupling between frontend `network read` and `network write`.
To keep communication within the application uniform a gRPC connection between frontend and backend could also be considered. However this technology brings similar limitations to websocket. The biggest advantage, uniform message formats through the whole application, can not be fully utilized either. The frontend needs a vastly different message format than the communication of the backend to the Gateway. Both these solutions were therefore discarded in favor of the queue.

Message Format

As it was unclear if the utilized YANG model will bring telemetry or topology data from the gateway at the beginning, the message format was studied for further optimization. Telemetry data is sent from the gateway every ten seconds and can therefore, in great networks with thousands of router, be a large onslaught of data. This could pose a serious performance issue. As such, a binary message format instead of a textbased one like bson (binary json) could be beneficial in such a situation. As the FlexAlgo information was in the end found in topology data, which only sends updates when changes in the network happen, this is not necessary anymore. The more convenient textbased format can be utilized. All the same, should the situation change in a further project, this should be kept in mind.

3.5.6. Live Model

An architecture with an instantiated live model of the network was chosen. With this architecture the complete separation of the communication with the front end and the connection to the gRPC Server of the Jalapeño API Gateway is possible. This reduces the complexity of the project and makes the communcation modules easily interchangable. Even more importand, the time spent computing the graphical representation of the network and it's flexible algorithms is dramatically reduced after the intial load of the data. Updates fed into the application from the gateway can easily be computed and the live model is merely updated, instead of performing a full refresh. This reduces the work time significantly and ensures updates to the frontend are as real time as possible. In the push architecture of the application this initial load can be run directly at startup of the application, so users have the optimal performance experience. With the model thus instantiated when the Kubernetes pod comes online for users to connect to, all changes to the network can merely be updated into the existing model.

In the prorotype architecture a request based approach is implemented, which runs complete instation on each request with the base API calls. To simulate the update functionality, additional calls were developed that trigger an updated object being handled by the application.

Currently, while in the current prototype the initial load of the nodes in a netowrk with 1000 routers may take up to 3 seconds, any changes to this network will be calculated in less than a seconds time.

When the push functionality will be implemented in the service, these updat methods can be called via observation pattern on the live model. This ensures optimal isolation and separation of concerns.

3.6. API Definition

[8][9][16]

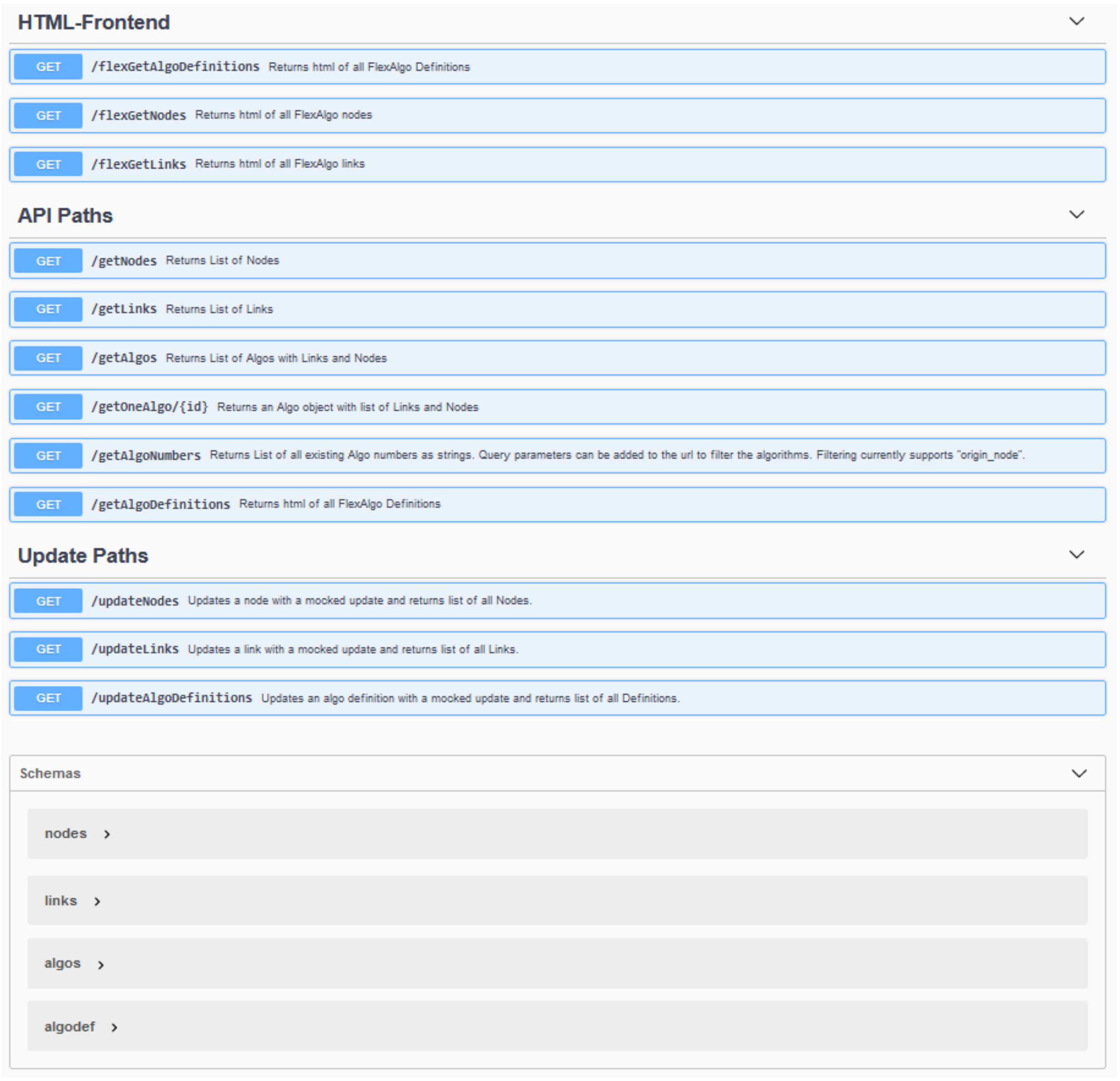
The API is kept minimal, as it will only be used in the prototype for the application. The API will be replaced in a further project with live updates to a frontend or queue, which will be done via push methodology. As such only the methods necessary for the frontend demo application from Michel Bongard are implemented, with the addition of server side rendered html templates for all algorithm specific data. To imitate the event driven design of the gRPC service, three data update calls were also developed.

A Swagger documentation of the API was added to the source code and can be found here. In this chapter onyl a brief overview of the API is given.

3.6.1. Swagger Documentation

GET /api

Returns the Swagger documentation of the API.



HTML-Frontend

GET `/flexGetAlgoDefinitions` Returns html of all FlexAlgo Definitions

GET `/flexGetNodes` Returns html of all FlexAlgo nodes

GET `/flexGetLinks` Returns html of all FlexAlgo links

API Paths

GET `/getNodes` Returns List of Nodes

GET `/getLinks` Returns List of Links

GET `/getAlgos` Returns List of Algos with Links and Nodes

GET `/getOneAlgo/{id}` Returns an Algo object with list of Links and Nodes

GET `/getAlgoNumbers` Returns List of all existing Algo numbers as strings. Query parameters can be added to the url to filter the algorithms. Filtering currently supports "origin_node".

GET `/getAlgoDefinitions` Returns html of all FlexAlgo Definitions

Update Paths

GET `/updateNodes` Updates a node with a mocked update and returns list of all Nodes.

GET `/updateLinks` Updates a link with a mocked update and returns list of all Links.

GET `/updateAlgoDefinitions` Updates an algo definition with a mocked update and returns list of all Definitions.

Schemas

nodes

links

algos

algodef

Figure 3.16.: Swagger Documentation

3.6.2. HTML

GET /flexGetAlgoDefinitions

This API call returns a server side rendered html template that shows a list of FlexAlgo relevant data of the network.

Query parameters: none
Body: none
Returns: server side rendered html

```html
<!DOCTYPE html>

<html>
  <head>
    <title>FlexAlgo Data</title>
    ...
  </head>

  <body>
    ...
    {{content}}
    ...
  </body>
</html>
```

Listing 3.4: flexGetAlgoDefinitions HTML return

GET /flexGetNodes

This API call returns a server side rendered html template that shows a list of all Nodes on the network.

Query parameters: none
Body: none
Returns: server side rendered html

```html
<!DOCTYPE html>

<html>
  <head>
    <title>FlexAlgo Data - Nodes</title>
    ...
  </head>

  <body>
    ...
    {{content}}
    ...
  </body>
</html>
```

Listing 3.5: flexGetNodes HTML return

GET /flexGetLinks

This API call returns a server side rendered html template that shows a list of all Links on the network.

Query parameters: none
Body: none
Returns: server side rendered html

```html
<!DOCTYPE html>

<html>
  <head>
    <title>FlexAlgo Data - Links</title>
    ...
  </head>

  <body>
    ...
    {{content}}
    ...
  </body>
</html>
```

Listing 3.6: flexGetLinks HTML return

3.6.3. React API

In this section the API calls utilized by the React.js frontend are defined. This will be the routes that need to be modified once the push notification functionality is implemented.

GET /getNodes

Used by the React.js frontend to get all nodes in the topology. Returns a list of the nodes of the network, if they have FlexAlgo configured or not. Mocked geographical data is added to make positioning in graphical frontend possible.

Query parameters: none
Body: none
Returns: list of nodes as json

```
[
  {
    key: string,
    name: string,
    prefix-id: number,
    x: number,
    y: number,
    prefixes: [ { algo: number, prefix_sid: number } ]
  }
]
```

Listing 3.7: getNodes json return

GET /getLinks

Used by the React.js frontend to get all links in the topology. Returns a list of links in the network, no matter if they are included in a FlexAlgo or not. The link is represented via an id, a from and to node and the affinities assigned to it. As the affinities are defined on interfaces, the definition of the outgoing interface is chosen for a given link.

Query parameters: none
Body: none
Returns: list of links as json

```
[
  {
    key: string,
    fromNode: string,
    toNode: string,
    affinities: [
      {
        type: number,
        value: string
      }
    ]
  }
]
```

Listing 3.8: getEdges json return

GET /getAlgoDefinitions

Returns all algorithm definitions currently on the network with all configured values. In a further step this method could be parameterized to return only a single algorithms information to display specific configurations in the front end.

Query parameters: none
Body: none
Returns: list of FlexAlgo definitions as json

```
[
  {
    algoKey: number,
    priority: number,
    originNode: number,
    metric: number,
    incl-all: [
      {
        affinities: string
      }
    ],
    incl-any: [
      {
        affinities: string
      }
    ],
    excl-all: [
      {
        affinities: string
      }
    ],
    excl-srlg: [
      {
        affinities: string
      }
    ]
  }
]
```

Listing 3.9: getAlgos json return

GET /getAlgoNumbers

Used by the React.js frontend to get a list of ids of the algorithms contained in the network. This is used in the frontend to select a specific algorithm and show the graphical representation of this FlexAlgo.
To add filter functionality to the algonumbers, query parameters were utilized. At the moment filtering is only applied to the origin node, as filter settings are not yet defined. But possibilities like filtering algorithms on metric type, calculation type or how affinities are handled, are easily implemented with the existing functionality.

Query parameters: origin_node: string

Body: none
Returns: List of FlexAlgo ids as string as json

```
[
  number: string
]
```

Listing 3.10: getOneAlgo json return

GET /getAlgos

Returns all algorithms within the network with their nodes and edges. No additional data save for the FlexAlgo id is returned with this request. Algorithm configuration values have to be called with the `/getAlgoDefinitions` API call.

Query parameters: none
Body: none
Returns: list of FlexAlgo with nodes and edges as json

```
[
  {
    algoKey: number,
    nodes: [
      {
        nodeKey: nummer
      }
    ],
    edges: [
      {
        edgeKey: nummer
      }
    ]
  }
]
```

Listing 3.11: getAlgos json return

GET /getOneAlgo/<id>

Used by the React.js frontend to get all nodes and links contained in the algorithm with the number given. With this the graphical representation of the algorithm is calculated.

Parameters: id: algorithm number
Query parameters: none
Body: none
Returns: Lists of nodes and edges contained in the specified FlexAlgo as json

```
{
  algoKey: number,
  nodes: [
    {
      nodeKey: nummer
    }
  ],
  edges: [
    {
      edgeKey: nummer
    }
  ]
}
```

Listing 3.12: getOneAlgo json return

3.6.4. Update

As previously discussed the Jalapeño API Gateway could not yet be utilized in this project. As such the subscription functionality of the gRPC connection to the gateway and it's update service was mocked by update API calls that simulate an input in the form of a changed object (the code provides the updated source data). The functionality in the code handles all CRUD operations for the objects but only simulates updates. Extensive tests cover all other possible operations.

GET /updateNodes

Simulates an input of an updated node. Changes can be seen in the node with the key "2_0_0_0000.0000.0008". The name value changes with each call.

Query parameters: none
Body: none
Returns: Lists of nodes as json

GET /updateLinks

Simulates an input of an updated link. Changes can be seen in the link with the key "2_0_0_0_0000.0000.0006_
The value of the affinity type changes with each call.

Query parameters: none
Body: none
Returns: Lists of links as json

GET /updateAlgoDefinitions

Simulates an input of an updated FlexAlgo configuration. Changes can be seen in the definitions of the origin node with the key "2_0_0_0000.0000.0008". On the first update call the nodes configuration changes from the algorithms 128 and 129 to the algorithms 128 and 999. On subsequent calls the priority values of both these algorithms change.

where the value of the affinity type changes with each call.

Query parameters: none
Body: none
Returns: Lists of FlexAlgo definitions as json

3.7. Logging

Logging of the application process is handled with the standard error stream stederr. Is an error raised in the application it is caught in a decorator function to turn into a workable error type. Simultanously the raised error is written out into the stederr. In a further step or another service this functionality can be extended to write out more general software or user actions for a more extensive logging experience.

Additionally, as the application is designed to run on Kubernetes and as such can have multiple instances run simultaneously, a central logging service may be adviseable. Kubernetes already provides possible implementations for such a system and the stederr and stedstr can easily be written out onto a logging application for further analysis and persistency.

The Flask application build in this prototype also runs on production servers managed by gunicorn. Gunicorn also implements a basic scaling service that works by running multiple processes of the application. This scaling is implemented with a pre-fork worker model, ie the application can be run as multiple instances that handle the workload of requests. This means a central logging system may be advisable for production implementation on bigger networks at least. For this to work seamlessly the process id is added to the standard error stream before the logged error.[22]

```
SA_FlexAlgo $ python -m newtork_read.app
 * Serving Flask app 'network_read.application.create_app' (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-971-713
 127.0.0.1 - - [30/May/2022 10:06:56] "GET /getNodes HTTP/1.1" 500 -
 2022-05-30 10:14:53.081089 - Process ID 27772 exception.
 2022-05-30 10:14:53.082090 - Object did not have field '_key'.
```

Listing 3.13: Logging Example

3.8. Frontend

3.8.1. Service Side Rendered

The network information that is retrieved from the database can be displayed in a simple, uniform html template, callable over three separate API calls:

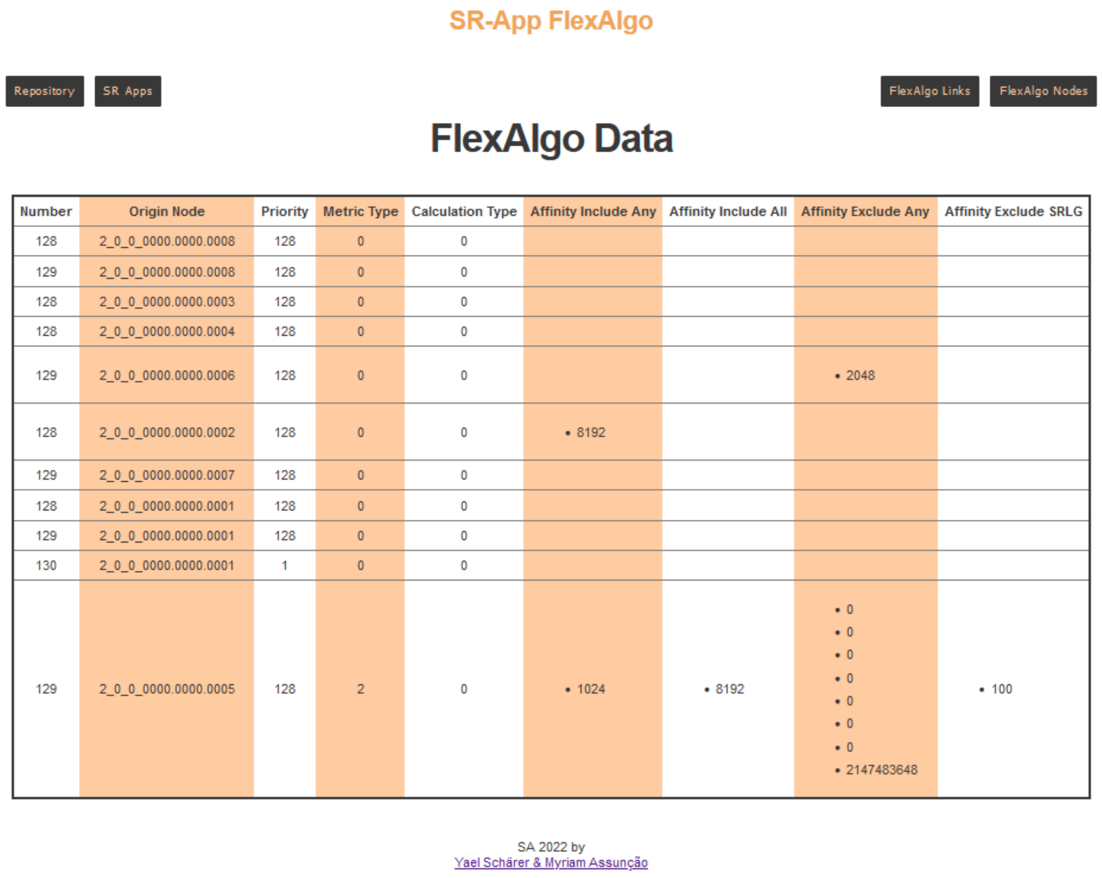
- `/flexGetAlgoDefinitions`
- `/flexGetNodes`
- `/flexGetLinks`

The html provided by these calls are server side rendered templates [49]. These are static files predefined in the Flask application, one for each view. They contain placeholders for the network data that are filled upon rendering of the template into a viable html response. The template library Jinja is used for this process.[48]

The template's header provides four buttons for navigation:

- `Repository:` Leads to the GitLab repository of this project.
- `SR-Apps:` Will lead the user to information about the other SR-AppsSR-Apps on the INS website.
- $2x$ `FlexAlgo:` Navigates to the other two html templates.

/flexGetAlgoDefinitions



FlexAlgo Data

Number	Origin Node	Priority	Metric Type	Calculation Type	Affinity Include Any	Affinity Include All	Affinity Exclude Any	Affinity Exclude SRLG
128	2_0_0_0000.0000.0008	128	0	0				
129	2_0_0_0000.0000.0008	128	0	0				
128	2_0_0_0000.0000.0003	128	0	0				
128	2_0_0_0000.0000.0004	128	0	0				
129	2_0_0_0000.0000.0006	128	0	0			• 2048	
128	2_0_0_0000.0000.0002	128	0	0	• 8192			
129	2_0_0_0000.0000.0007	128	0	0				
128	2_0_0_0000.0000.0001	128	0	0				
129	2_0_0_0000.0000.0001	128	0	0				
130	2_0_0_0000.0000.0001	1	0	0				
129	2_0_0_0000.0000.0005	128	2	0	• 1024	• 8192	• 0 • 0 • 0 • 0 • 0 • 0 • 0 • 2147483648	• 100

SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.17.: Frontend of /flexGetAlgoDefinitions

This site provides an aggregated view of all configured algorithms on the underlaying network. [13]

- `Number:` The algorithm's identification number in this network.
- `Origin Node:` The device on which this definition of the algorithm is configured.
- `Priority:` The property to specify which definition has to be taken if there is a tie in a decision of which path should be taken for a route.
- `Metric Type:` With which metric this algorithm is configured.

  0 - IGP metric

  1 - Link delay (RFC 8570 [40])

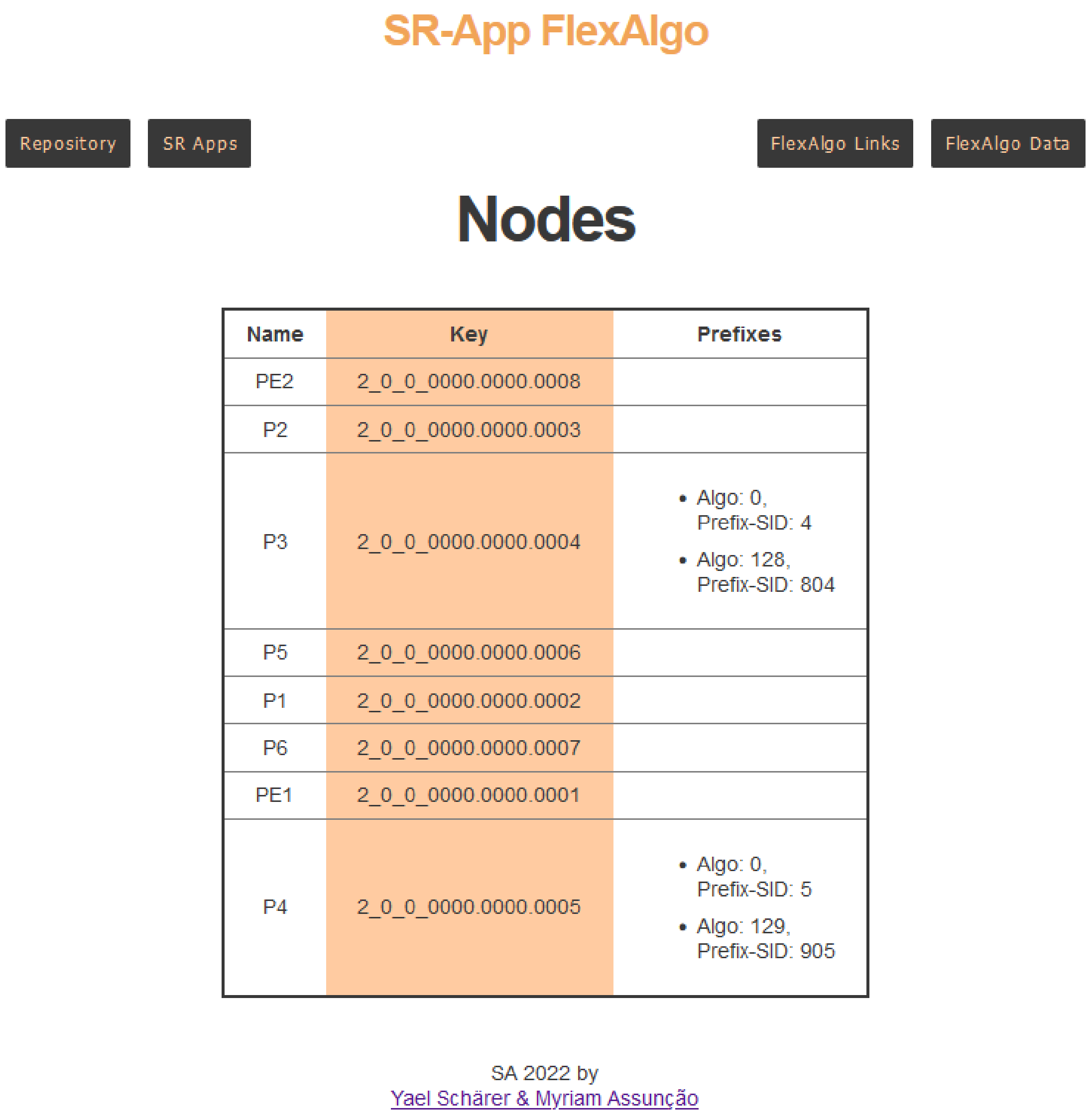
  2 - glste metric (RFC 5305 [36])
- `Calculation Type:`

  0 - SPF algorithm

  1 - strict-SPF algorithm - does not exist on the IOS XR

- `Affinity Include Any:` The link color numbers that can be but don't have to be included in the path of the algorithm.
- `Affinity Include All:` The link colors that have to be included in the configured algorithm.
- `Affinity Exclude Any:` The link colors the algorithm should avoid.
- `Affinity Exclude SRLG:` Which SRLGs the algorithm should avoid in calculating the path.

/flexGetNodes



Nodes

Name	Key	Prefixes
PE2	2_0_0_0000.0000.0008	
P2	2_0_0_0000.0000.0003	
P3	2_0_0_0000.0000.0004	• Algo: 0, Prefix-SID: 4 • Algo: 128, Prefix-SID: 804
P5	2_0_0_0000.0000.0006	
P1	2_0_0_0000.0000.0002	
P6	2_0_0_0000.0000.0007	
PE1	2_0_0_0000.0000.0001	
P4	2_0_0_0000.0000.0005	• Algo: 0, Prefix-SID: 5 • Algo: 129, Prefix-SID: 905

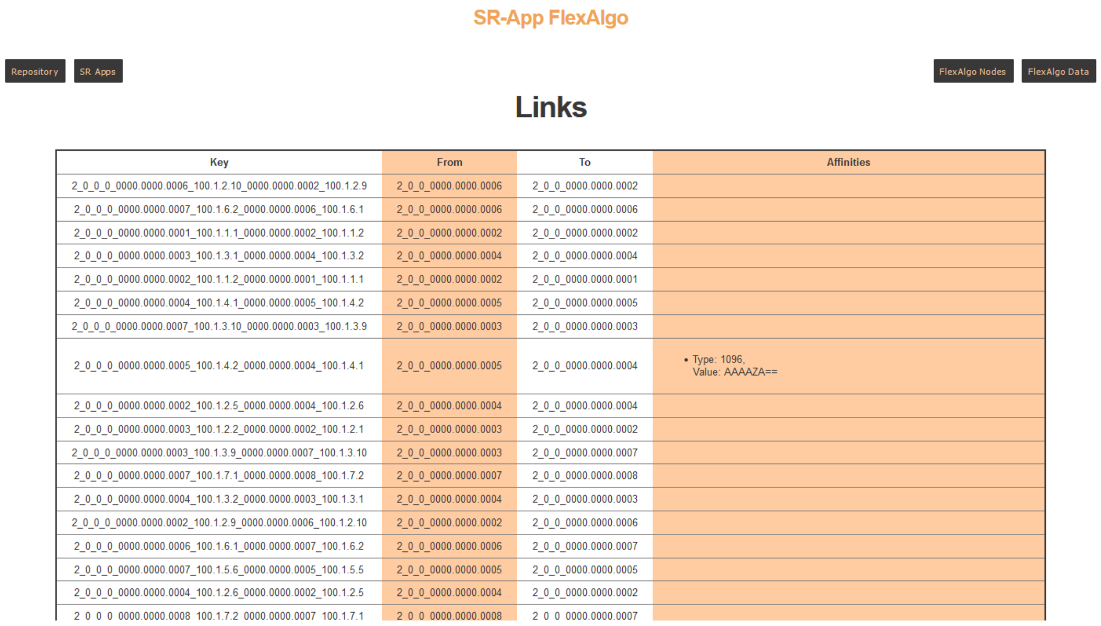
SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.18.: Frontend of /flexGetNodes

The most relevant information for the nodes in the network can be examined on this site.

- `Name:` The device's hostname.
- `Key:` The unique device key that identifies the device in the ArangoDB.
- `Prefixes:` What Prefix-SID this device received according to the specified algorithm.

/flexGetLinks



Links

Key	From	To	Affinities
2_0_0_0_0000.0000.0006_100.1.2.10_0000.0000.0002_100.1.2.9	2_0_0_0000.0000.0006	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0007_100.1.6.2_0000.0000.0006_100.1.6.1	2_0_0_0000.0000.0006	2_0_0_0000.0000.0006	
2_0_0_0_0000.0000.0001_100.1.1.1_0000.0000.0002_100.1.1.2	2_0_0_0000.0000.0002	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0003_100.1.3.1_0000.0000.0004_100.1.3.2	2_0_0_0000.0000.0004	2_0_0_0000.0000.0004	
2_0_0_0_0000.0000.0002_100.1.1.2_0000.0000.0001_100.1.1.1	2_0_0_0000.0000.0002	2_0_0_0000.0000.0001	
2_0_0_0_0000.0000.0004_100.1.4.1_0000.0000.0005_100.1.4.2	2_0_0_0000.0000.0005	2_0_0_0000.0000.0005	
2_0_0_0_0000.0000.0007_100.1.3.10_0000.0000.0003_100.1.3.9	2_0_0_0000.0000.0003	2_0_0_0000.0000.0003	
2_0_0_0_0000.0000.0005_100.1.4.2_0000.0000.0004_100.1.4.1	2_0_0_0000.0000.0005	2_0_0_0000.0000.0004	• Type: 1096, Value: AAAAZA==
2_0_0_0_0000.0000.0002_100.1.2.5_0000.0000.0004_100.1.2.6	2_0_0_0000.0000.0004	2_0_0_0000.0000.0004	
2_0_0_0_0000.0000.0003_100.1.2.2_0000.0000.0002_100.1.2.1	2_0_0_0000.0000.0003	2_0_0_0000.0000.0002	
2_0_0_0_0000.0000.0003_100.1.3.9_0000.0000.0007_100.1.3.10	2_0_0_0000.0000.0003	2_0_0_0000.0000.0007	
2_0_0_0_0000.0000.0007_100.1.7.1_0000.0000.0008_100.1.7.2	2_0_0_0000.0000.0007	2_0_0_0000.0000.0008	
2_0_0_0_0000.0000.0004_100.1.3.2_0000.0000.0003_100.1.3.1	2_0_0_0000.0000.0004	2_0_0_0000.0000.0003	
2_0_0_0_0000.0000.0002_100.1.2.9_0000.0000.0006_100.1.2.10	2_0_0_0000.0000.0002	2_0_0_0000.0000.0006	
2_0_0_0_0000.0000.0006_100.1.6.1_0000.0000.0007_100.1.6.2	2_0_0_0000.0000.0006	2_0_0_0000.0000.0007	
2_0_0_0_0000.0000.0007_100.1.5.6_0000.0000.0005_100.1.5.5	2_0_0_0000.0000.0005	2_0_0_0000.0000.0005	
2_0_0_0_0000.0000.0004_100.1.2.6_0000.0000.0002_100.1.2.5	2_0_0_0000.0000.0004	2_0_0_0000.0000.0002	
2 0 0 0 0000.0000.0008 100.1.7.2 0000.0000.0007 100.1.7.1	2 0 0 0000.0000.0008	2 0 0 0000.0000.0007	

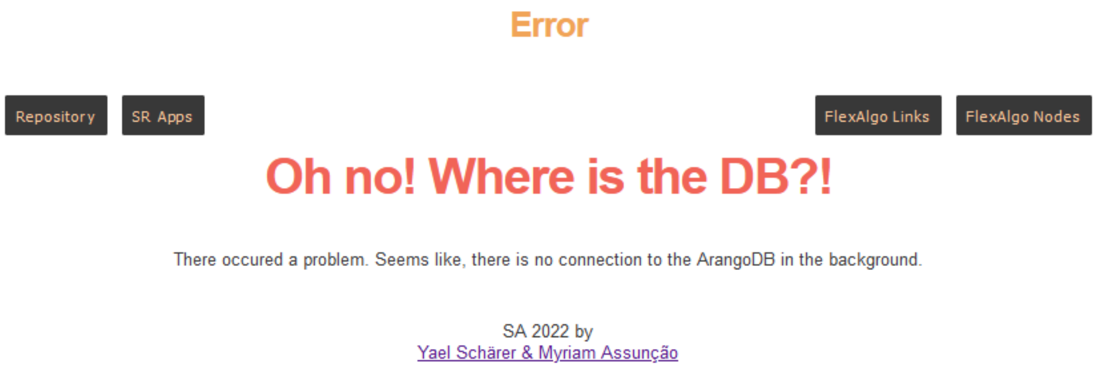
Figure 3.19.: Frontend of /flexGetLinks

This site provides the relevant information related to links in the network.

- `Key:` The unique key from the ArangoDB for the corresponding link from the underlaying network.
- `From:` The ArangoDB key from the device at the start of the link.
- `To:` The ArangoDB key from the device at the end of the link.
- `Affinities:` What customized FlexAlgo properties the link's interfaces has configured:
  - TLV 1088 - Administrative group (color) (RFC 7752 [37])
  - TLV 1092 - TE Default Metric (RFC 7752 [38])
  - TLV 1096 - Shared Risk Link Group (RFC 7752 [39])
  - TLV 1173 - Extended Administrative Group (RFC 9104 [42]) with include-all, include-any, exclude-any or exclude-srlg of the affinities

Error Handling

In case an error occurres, the application will differentiate between communication problems with the `database` ArangoDB, a `key error` (i.e. there is an attribute missing in the ArangoDB entry), a general `server exception` or a `gRPC problem` (which is already implemented to simplify the error handling in production of a further project). Each error will show different information to inform the user of specific actions that may be required.



Oh no! Where is the DB?!

There occured a problem. Seems like, there is no connection to the ArangoDB in the background.

SA 2022 by
Yael Schärer & Myriam Assunção

Figure 3.20.: Frontend of an occurred ArangoDB Error

3.8.2. SPA

This project offers a basic Single Page Application with extended functionalities of the React.js project Sigma.js Demohttps://github.com/mbongard/react-sigmajs-demo[20] of Michel Bongard. The React.js application is fetching the required data from the SR-App FlexAlgo application. This allows the software to visualize the underlying network with all its devices, links and the currently running algorithms of the segment domain.

The SPA application has React.js components implemented that Michel Bongard has developed for the SR-Apps line.Small adaptions were made to the `NetworkGraph` component to call the `network read` service. As these calls can take a few seconds and to prevent freezing of the application while waiting for the API response async methods were developed. To this end the library `axios` was used. Axios provides simple methods for promise based API calls in the form of a HTTP client [3].
The components `Checkbox` and `Controls` were also adapted from the original demo to support a list of FlexAlgo buttons that each implements the highlight functionality for a path on the network topology. Each button, when pressed, calls the `network read` service to get the `Algo` object for the FlexAlgo with the provided button.

The demo by Michel Bongard implements Sigma.js functionality for the graph visualization. Sigma.js is a JavaScript library for graph visualization and can support thousands of nodes and edges. It renders network graphs and allows many different user interactions with the resulting visuals out of the box [45].

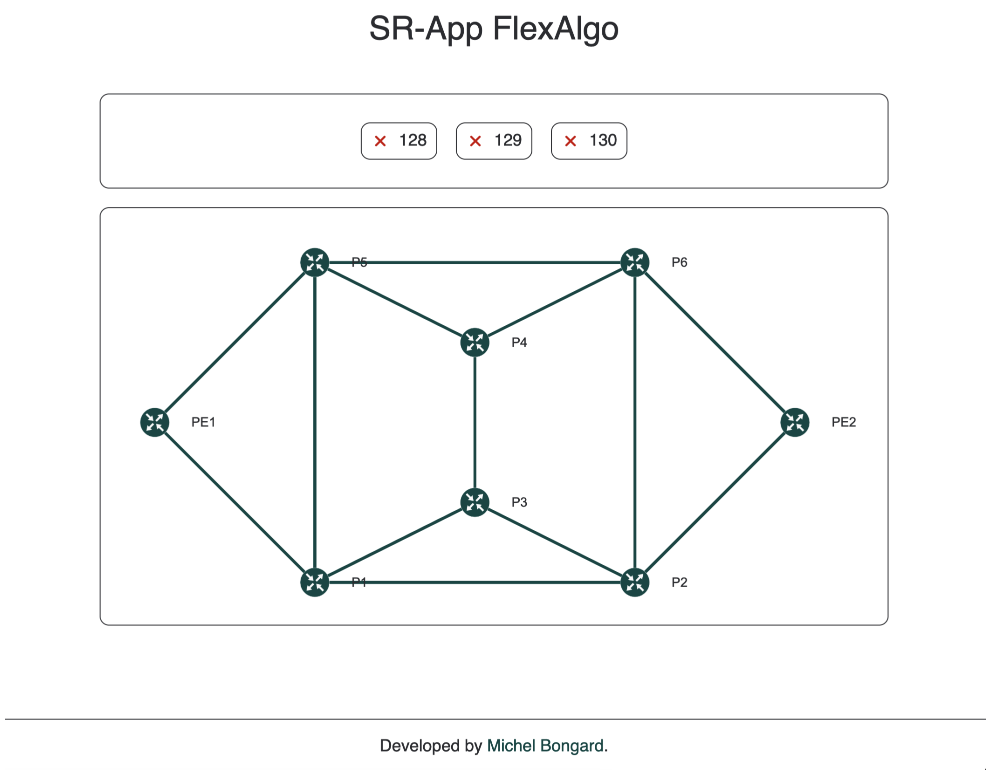


Figure 3.21.: Single Page Application View - No selected FlexAlgo

The placement of the nodes in the graph is handled according to the nodes' coordinates. For further details, the section `Domain` 3.3.4 of the `Network Read` class diagram can be consulted.

The page gives the user a navigation bar that shows all active algorithms of the network. The bar allows the switch between algorithms by clicking on the corresponding buttons.
By activating a button, the nodes and links of the algorithm get highlighted in red. This provides the user with an immediate overview and the current configurations of the network in a very intuitive way.

In section deployment 3.9 the instruction of how to run the SPA can be found.

3.9. Deployment

3.9.1. Network Read

3.9.1.1. Kubernetes

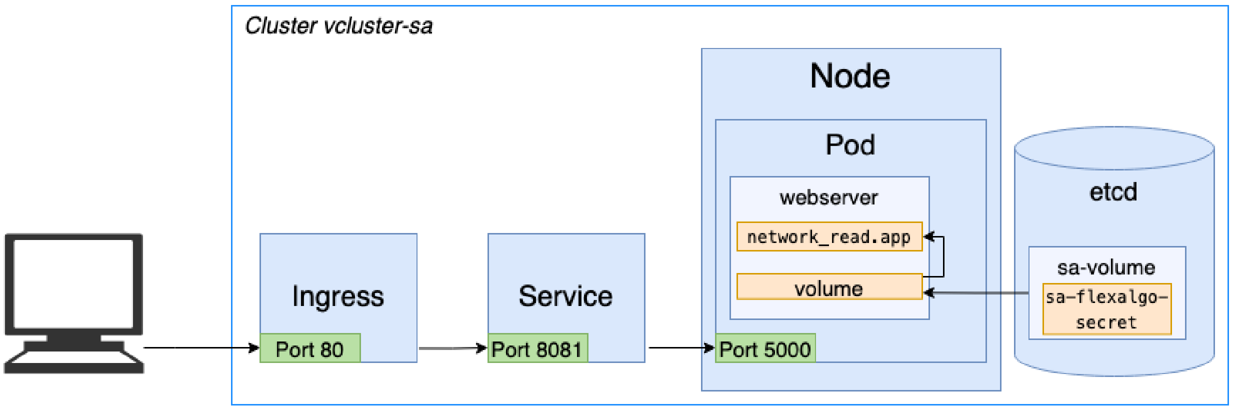


Figure 3.22.: Overview of Kubernetes Deployment

The application is deployed into Kubernetes. For this a cluster named `vcluster-sa` was created, with the help of a cluster YAML file with the same name. A namespace is defined, `sa-flexalgo`, in which the desired pod for the `network read` service gets deployed. Inside the cluster a deployment builds a pod named `webserver` which hosts the `network_read.app` application. The property `replica` will ensure that there will always run at least one pod.

To be able to run the application after deployment, a `volume` inside the pod receives the necessary secrets (from the secret service `sa-flexalgo-secret`). These secrets come from the `sa-volume` in the etcd database. Without these, the application will not start and throws an error. The running `network_read.app` can be reached from outside the cluster over the `Ingress` object on port 80. This ingress will forward the request to the implemented `Service` on port 8081. The `Service` will then reach the application over the port 5000, as the `network_read.app` exposes this explicit port.

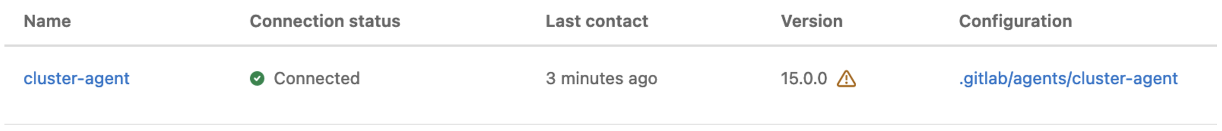
3.9.1.2. Pipeline

The pipeline runs two jobs sequentially that create the deployment for the `network read` service. The first to tun is the job `build-image` 3.9.1.2, which compiles the Docker image. Second, the `kubernetes-deployment` job 3.9.1.2 runs, which updates the running deployment of the Kubernetes pod with the new image. Further insights into the pipeline can be found in the pipeline YAML file in the Appendix 10.1 or the original file `.gitlab-ci.yml` from the GitLab repository.

Pipeline Job: build-image
The Docker image will be created with the new code that is uploaded to the dev-branch by running the `build-image` job. The pipeline will produce an image, with the tag `dev`. These images are accessible in the GitLab repository's container registry.

Pipeline Job: kubernetes-deployment

A cluster agent in the infrastructure of the GitLab repository provides the required connectivity between the repository's container registry with the Docker images and the Kubernetes pod with the deployment on it.



Name	Connection status	Last contact	Version	Configuration
cluster-agent	✓ Connected	3 minutes ago	15.0.0 ⚠	.gitlab/agents/cluster-agent

Figure 3.23.: Cluster Agent on GiLab

To be able to pull the necessary Docker image from the private registry of GitLab, a secret is provided with the access requests. The secret was created with `kubectl` command line tool:

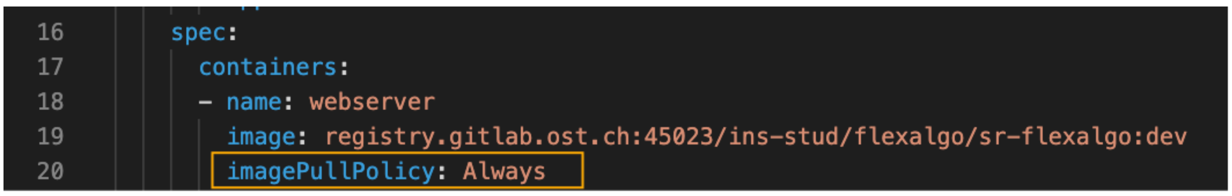
```
$ kubectl create secret docker-registry regcred \
--docker-server=registry.gitlab.ost.ch:45023 \
--docker-username=flexalgo-sa \
--docker-password=<token-from-the-personal-access-token>

# Definition:
# kubectl create secret docker-registry <secret-name> \
# --docker-server=<registry-of-gitlab> \
# --docker-username=<GitLab-username> \
# --docker-password=<token-from-the-personal-access-token>
```

Listing 3.14: Setting the Docker Credentials in Cluster

The Kubernetes deployment YAML file consists of three parts:

- `Deployment:` The Deployment `webserver-deployment` itself, which tells Kubernetes to have exactly 1 running pod with the Docker image from the container registry. The replicas protperty ensures one pod is always deployed. The SR-FlexAlgo application requires three secrets to be able to connect to the ArangoDB. They are saved in a separate `secrets`-file, encoded with Base64. (see Appendix 11.4). The Deployment retrieves the secrets and saves them into the volume in the pod. This makes them accessible to the application that runs in the pod. (see Appendix 11.1)
- `Service:` The service `webserver-svc` creates a connection between the ingress and the Deployment's container with the running application. (see Appendix 11.2)
- `Ingress:` A single ingress `webserver-ingress` is defined, which allows the usage of the url `sa-sr-flexalgo.stu.network.garden` to connect to the pods. (see Appendix 11.3)

The pipeline will execute a `rollout restart` of the Deployment to update the application on the pod with the newest source code of the SR-App FlexAlgo from GitLab. This behaviour is chosen as the deployment file itself doesn't get updated, merely the data that is retrieved from the container registry if the pod gets restarted. To enable this process, the `imagePullPolicy: Always` in the Deployment is set.



```
  spec:
    containers:
    - name: webserver
      image: registry.gitlab.ost.ch:45023/ins-stud/flexalgo/sr-flexalgo:dev
      imagePullPolicy: Always
```

Figure 3.24.: Deployment File Section with imagePullPolicy

The pipeline will then set the environment to the desired url `https://sa-sr-flexalgo.stu.network.garden` to make the SR-FlexAlgo reachable.

```
environment:
    name: dev
    url: https://sa-sr-flexalgo.stu.network.garden
```

Listing 3.15: Setting the Environment for the Pipeline

The url is accessible from within the INS VPN network and can be called from a standard browser over the API calls defined in chapter 3.6.

3.9.2. React Frontend

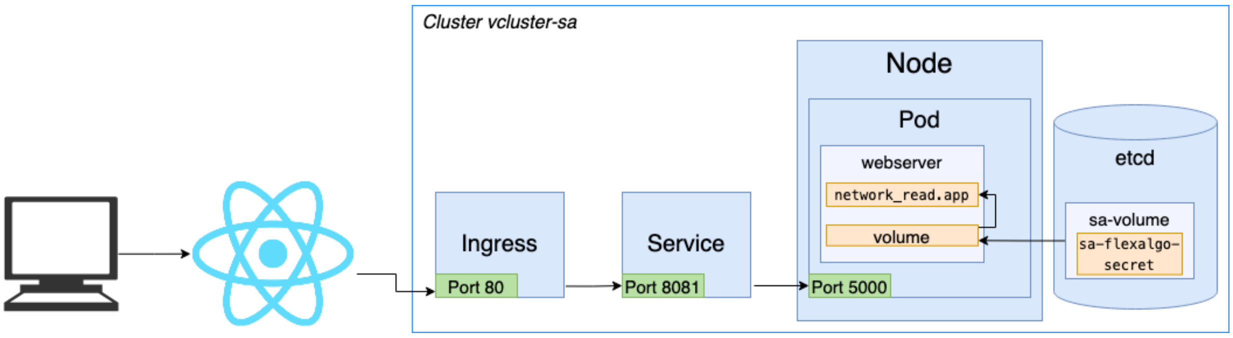


Figure 3.25.: Overview of Kubernetes Deployment with React Frontend

A part of the prototype is the implementation of the React.js SPA, which shows the network topology with the configured algorithms. Currently the React.js application is not automatically deployed via GitLab as it is merely a demo for a possible frontend. The application needs to be run manually on a local machine. For the application to run correctly one has to be logged into the VPN of INS.

The application can be started with the help of the package manager `Yarn`, which will lead the user automatically to the website `localhost:3000`. On startup the application will call the `network read` service in the cluster for topology data and display it.

```
SA_FlexAlgo $ cd development/react-frontend
SA_FlexAlgo $ yarn install    # on first run for dependency installation
SA_FlexAlgo $ yarn start
```

Listing 3.16: Starting React Frontend

The `network read` service can be run with the Poetry shell, should the user wish to run the application system completely local. If this approach is chosen the API calls in the frontend will have to be adabted for `http://127.0.0.1:5000` instead of the pod url. To be able to enable localhost calls, certain browsers need a plugin like CORSE from Mozilla Firefox. Otherwise the API calls throw a "Same-Origin Policy" error.

```
SA_FlexAlgo $ cd development/network-read/
SA_FlexAlgo $ poetry shell
Spawning shell within
/Library/Caches/pypoetry/virtualenvs/network-read-wqd3fn0R-py3.9
./Library/Caches/pypoetry/virtualenvs/network-read-wqd3fn0R-py3.9/bin/activate
bash-5.1$ .
/Library/Caches/pypoetry/virtualenvs/network-read-wqd3fn0R-py3.9/bin/activate
(network-read-wqd3fn0R-py3.9) bash-5.1$ poetry install      # on first run for
dependency installation
(network-read-wqd3fn0R-py3.9) bash-5.1$ python -m newtork_read.app
* Serving Flask app 'network_read.application.create_app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 102-475-289
127.0.0.1 - - [24/May/2022 18:25:49] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [24/May/2022 18:25:55] "GET /flexGetAlgoDefinitions HTTP/1.1" 200 -
127.0.0.1 - - [24/May/2022 18:25:55] "GET /static/css/base.css HTTP/1.1" 200 -
```

Listing 3.17: Starting FlexAlgo application over CLI

3.10. Wireframes

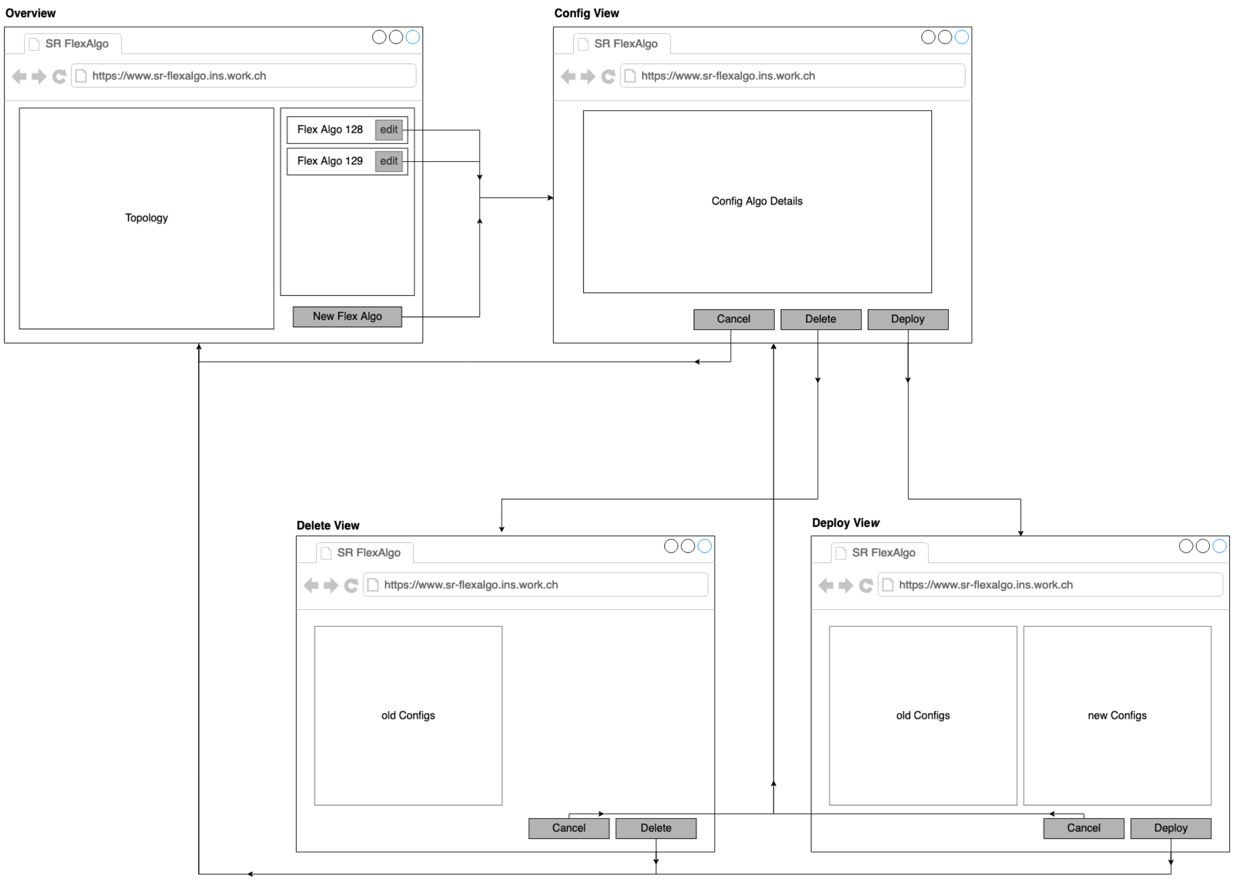


Figure 3.26.: Wireframes

Only a basic wireframe was developed, as the prototype will merely implement a frontend for demonstration purposes and not one with the full functionality of the SR-App FlexAlgo software. It may be build upon in a later project. As such the wireframe is not implemented but merely used for clarifications of use cases. It may be build upon in a later project. The designed was chosen similar to the already existing project SerPro, a SR-App constructed by Severin Dellsperger and Julian Klaiber. This decision was made to better fit the application SR-App FlexAlgo into the SR-Apps series.

The frontend is designed to be simple and intuitive to provide an optimized user experience. As such the landing page of a logged-in user is the `Overview` page. This page shows the topology of the network connected to the application with it's devices. A list with the running algorithms is given to the right side with separate buttons to edit an algorithm individually.

If the user wants to update an existing algorithm or add a new one they can reach the `Config View` via the algorithm's `edit` option or the `New Flex Algo` buttons. There, an overview of the algorithm's parameters will be shown.
After directly updating the desired values a user can either return to the `Overview` page by clicking the `Cancel` button to abort the changes. Or another way is to go on to the `Delete View` by `deleting` the selected algorithm or reach the `Deploy View` by hitting the `Deploy` button.

When the user lands on the `Deploy View` they can see the already conigured information of this algorithm on the left side in the `old Configs` section. On the right site in `new Configs`, the

user can see the new configuration designed in the `Config View`. This comparison will show the changed values highlighted for convenience for a comperhensive review.
If the new configurations are correct, the user can `Deploy` the changes to the network. In case the changes are incorrect the user can `Cancel` the action and return to the `Config View` to rework the values.

The `Delete View` shows the content of the current configurations of the FlexAlgo from the underlying network that will be deleted.
This step acts in the same way against changes by mistakes as the Deploy View by showing the user the subsequent changes of their actions.
The `Cancel` and `Deploy` buttons act in the same way as the ones from the `Deploy View`.

4. Mocking

The INS provided a small virtual network of ten routers, two switches and several virtual PCs for the development of this project. With this, experiments of the FlexAlgo technology could be performed and it gave a solid basis for the development of the prototype.

All the same, todays networks grow larger and more demanding. The SR-App FlexAlgo software will have to handle networks of up of 1'000 routers with a complicated link system and up of 100 algorithms configured. To test out the performance of the prototype on such large networks and find problematic, slow code, a testing network had to be implemented that would simulate such a large network.

Gladly, the SR-App FlexAlgo is not the only application having to simulate such a workload. One other SA application, the research project "Central Frontend for Segment Routing Applications" of the students Davor Gajic and Leonard Obernhuber have created a mocking tool where they generate basic router and link information and prepare it for uploads into the ArangoDB. It is a golang script that creates a customizeable amount of objects for the relevant ArangoDB collections and writes them out into json files. They created a bash script to automatically upload these files directly into the db. As this script needs certain ingress points into the Cluster of the Jalapeño API Gateway we decided against adapting it for our porject. Manually uploading the five json files into the db is easy as the db provides easy to use functionality for bulk upload of a json list.

To simulate the FlexAlgo data this project needed to get an apporpriate perfromance estimation for the prototype, adaptions to the mocking tool were made. Mostly these consisted of additional fields to the generated objects with pseudo random input values. This mocked data was then uploaded into new collections in the db that where named with the prefix `test_` and the original name of the mocked collection. For example the `test_ls_links` collection was created to replace the `ls_links` one in the simulation.

As such the application could be easily switched to the test collections for performance tests and back afterwards. Results of these tests on the mocked objects can be found 5.

5. Testing

General Information

System and acceptance tests were made at every sprint review after the begin of the construction phase. This to ensure the prototype performs at a high level and provide a good starting base for a further project. The use cases of section 1.1 were tested in the system tests, the non-functional requirements 1.2 in the acceptance tests. The test procedure and templates for the protocol where written in the elaboration phase to ensure uniform testing.
For the development of the software a small virtual network of 10 routeres, of which 8 act in the segment domain, was provided by the INS. To test the application's performance a more extensive network is mocked, see chapter mocking 4. This provides a basis to make a well-defined prediction for the progress of the project and the status of the current stable code.

In this chapter the last documented tests are shown, to provide a view of the functionality and performance of the finished prototype. To see the progress of all system and acceptance tests, please consider the Appendix 5 of this paper.

5.1. System Test - Review of Iteration 7

This document will provide an overview of the performed System Tests of this project.

Preconditions
The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation
Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed System Tests

- System Test of iteration 6 was done on tag Dev-I6.testing.
- This System Test is done on tag Dev-I7-final-submission.

5.1.1. Tests - SA MVPs

UC00 - List of Flex Algo

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC01 - Live Updates

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
Live updates work for everything. Additional routes were implemented to simulate gRPC updates of links, nodes and definitions.		

UC02 - Failure to connect

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		

UC03 - List all Flex Algos

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC04 - Logging Software

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC05 - Graph Topology

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC06 - Graph 1 Algo

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
Works, sometimes needs a second try.		

UC07 - Graph all Algos

Implemented	**Process**	**Status**
Yes	Refresh website.	FROZEN
Shortcomings		
Not implemented. See results of thesis section for further information.		

Pending Improvements

-

5.2. Acceptance Test - Review of Iteration 7

Preconditions

The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.
For the Performance tests the data from the Mocking Tool will be used to simluate a large network.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation

Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed Acceptance Tests

- System Test of iteration 6 was done on tag Dev-I6.testing.
- This System Test is done on tag Dev-I7-final-submission.

5.2.1. Tests

Functionality

`Non-Functional Requirement`	`Implementation`	`Shortcomings`	`Status`
`Accuracy`	Data retrieved from Jalapeño API Gateway is up-to-date.	—	Implemented
`Interoperability`	The connection to the Jalapeño API Gateway is established over gRPC.	Will not be implemented for this project.	Frozen

Reliability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Fault Tolerance`	Server remains running if connection to API is down.	Instead of API, connection to ArangoDB was handled successfully.	Implemented

`Recoverability`	The integrated Kubernetes mechanism for failovers handles failed pods.	—	Implemented

Performance

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Data Integrity`	Changes in network are visible with a browser refresh.	—	Implemented
`Scalability`	The application runs with 1'000 routers in the network.	—	Implemented
`Time Behavior`	Changes in the network are visible wihtin 3 seconds after refreshing the browser.	Works on everything. At the moment we have 102 flex algorithms.	Implemented

Maintainability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Analysability`	The user receives a message to inform about occuring errors.	The error handling will be implemented as a HTML template or react frontend message.	Implemented
`Analysability`	Error messages will be written into stdout.	Is written into stderr.	Implemented
`Testability`	All Unit and Integration Tests run successfully.	—	Implemented

Pending Improvements

-

Part V.

Project Management

1. Project Plan

1.1. Introduction

For this SA project a prototype will be built for a Segment Routing application that configures the used FlexAlgos in any given network. The work will be done in 14 weeks by Myriam Assunção and Yael Schärer.

1.1.1. Purpose

This document defines the frame and the process of the SA project SR-Apps FlexAlgo. It will be used as basis for the project management.

1.1.2. Scope

This document will be valid for the whole time of the SA research project during spring semester 2022. Deviations and additions of the initial planning will be noted and justified directly in the corresponding part of the project documentation.

1.2. References

Books

- Applying UML and Patterns, Craig Larman, ISBN: 0-13-148906-2

Websites

- Git Feature Branch Workflow
- GitLab Dokumentation
- Segment Routing Projects

1.2.1. Glossary

All abbreviations and technical terms will be defined in the Glossary.

1.2.2. Project Overview

This project has the goal of an application for the complete configuration of the FlexAlgo on any given network. In the scope of the SA a prototype will be built that can access a test network via Jalapeño API Gateway and visually displays all relevant data of the FlexAlgo for said network. Objective of this prototype during the SA is to check the feasability of such an application. If this is successful, a fully operational application will be built in the following BA project of the team members.

This application is part of a wider project of the INS for a series of SR-Apps. Our prototype will conform to the already existing projects.

1.2.2.1. Scope of Delivery

The product of this project will be a functional prototype of an SR-App that can access an existing network and display all corresponding information in a website. The prototype will include a Pythonbackend service, an interface to the Jalapeño API Gateway, a server rendered frontend and all communication between the components of the app. Should time allow, additional experimentation with a React.js framework (from the INS institute) and a database for data historization may be added.
Additionally, we will deliver extensive documentations for the software, the research and the project management.

Technical Details:

- Frontend via server side rendering
- Backend as Source-Code
- Relational database (PostgreSQL) if historization is implemented
- Communication between components (gRPC and/or message queues)
- Interface documentation (Jalapeño API Gateway)
- Software documentation
- Installation instructions if necessary
- Project documentation

The documentation will be written in LaTeX and delivered as pdf files.
The submission of code will be done via GitLab. The final SA documentation will be submitted digitally via the official AVT website at the end of the project.

1.2.3. Assumptions and Constraints

Important aspects of Segment Routing are the large networks and as such scalability and performance are important factors. To address this the application will be cloud native and the prototype built accordingly.
The project effort is scheduled to be 240 hours per person, which will result in 480 hours for the projet. Is the budgeted time consumed without achieving a workable prototype, the team members will increase the workload per week accordingly. If the prototype is ready early, additional features will be implemented.

1.3. Project Organisation

The organsiation will be kept flat with both team members overseeing different parts of the project. We will use the agil project management of SCRUM+ with a two week sprint interval.

1.4. Internal Structure

Responsibility	**Team Member**
Frontend	Yael Schärer
Backend	Yael Schärer
Database	Yael Schärer
CI/CD	Myriam Assunção
Testing	Yael Schärer
Research	Myriam Assunção
Documentation	Myriam Assunção
SCRUM+ Master	Myriam Assunção

1.5. External Contacts

Prof. Laurent Metzger	Project Supervisor
Urs Baumann	Project Advisor, Jalapeño, CI/CD
Michel Bongard	Jalapeño API Gateway
Severin Dellsperger, Julian Klaiber, Dominique Illi, Michel Bongard	SR-Apps
Yannick Zwicker	Kubernetes

1.6. Management

1.6.1. Meetings

The review meetings with the advisors are held weekly at `Tuesday 10.00 - 11.00 o'clock` at school or remotely per Teams.
sprint plannings and reviews are held every `two weeks` (per SCRUM+ sprint interval) on `Monday at 9.00 o'clock`. Afterwards problems and open questions can be discussed where necessary.
Daily SCRUM+ standups are held on `Monday and Tuesday 17.00 o'clock`.
The team members will alternate between either leading meetings or writing the protocol. For each meeting an agenda will be prepared beforhand. The agenda items will be documented in GitLab together with the protocols of the corresponding meetings. Items can be added by all team members until 22.00 o'clock the day before. The responsibility of organisation and presentation lays with the meeting leader.
Results and decisions of meetings will be translated directly into Issues of the GitLab Board or recorded in the documentation.

1.6.2. Organisation Git

Project management in GitLab

- Organisational matters like meeting notes, etc.
- Documentations like projectplan, time tracking, risk analysis, etc.

Documentation in GitLab

- Use Cases, domain model, etc.

- Research

Development in GitLab

- Software Documentationen
- Installation notes
- Source Code

1.7. Schedule

Project starts at 21.02.2022 and ends 03.06.2022 at 17.00 o'clock.

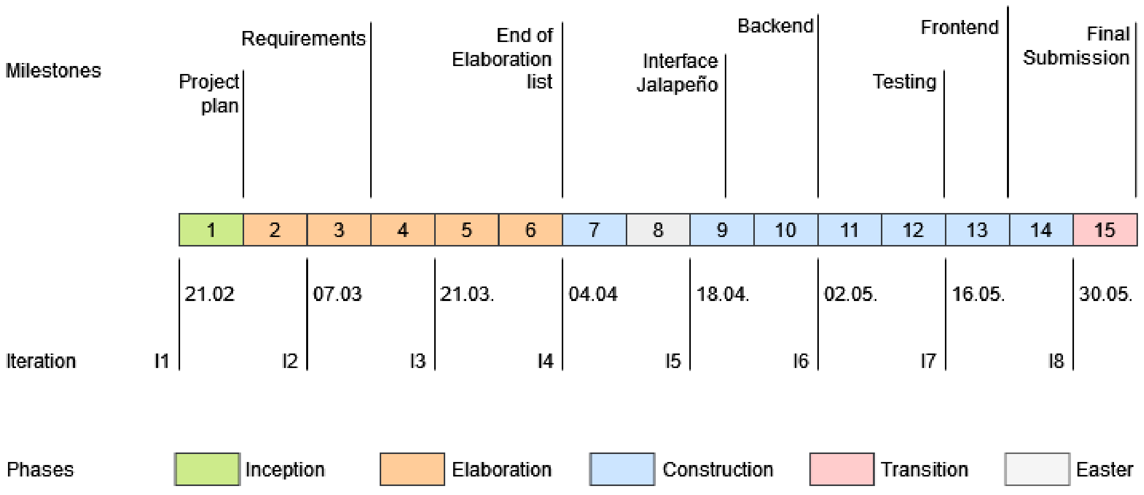


Figure 1.1.: Timeline for SA

1.7.1. Workflow

Phase	**Description**	**From**	**To**	**Duration** [Week]
`Inception`	Project plan finalisation	21.02.22	27.02.22	1
`Elaboration`	Domain analysis and research	28.02.22	29.03.22	5
`Construction`	Construction and testing of prototype	30.03.22	20.05.22	7
`Transition`	Submission	21.05.22	29.05.22	1

1.7.2. Milestones

Milestone	**Duration** [Week]	**Description**	**Assessment**	**Proceed further**
`M1:` Project Planning	1	Create project plan and organisation.	Review Projectplan with advisors.	Work out requirements, use cases, etc.
`M2:` Requirements	2	Requirements, use case minimal in brief format, analysis Gateway.	Requirements must be approved by customer (advisors). Possible changes to gateway must be requested.	Work on items on the End of Elaboration List.
`M3:` End of Elaboration	3	Define all points on the EoE list.	End of Elaboration List is finished. YANG models are defined.	Construct interface to Jalapeño API Gateway. Begin writing *unit and integration tests.*
`M4:` Interface to Jalapeño API Gateway	2.5	Build interface with all for prototype necessary functionalities.	Interface tests are *green.*	Begin construction backend services.
`M5:` Backend	1.5	Build complete backend and begin *acceptance tests.*	Functional backend.	Continue building *system tests.*
`M6:` Testing	2	Build all necessary tests (*unit, integration, system, acceptance*).	Tests run *green.*	Build frontend.
`M7:` Frontend	1	Build simple frontend and begin documentation for final submission.	Fully functional app.	Revice documentation.
`M8:` Final Submission	2	Prepare and submit final submission. Additionally we have one week of reserve time.	Drink beer.	Sober up.

1.8. Risk Management

The risk management document can be found in the section `risk_analysis` with a quick overview of the weighted risks as a graphic.

1.9. Issues

The issues that are remaining, in progress and closed are defined and organised on the GitLab Boards.

1.10. Infrastructure

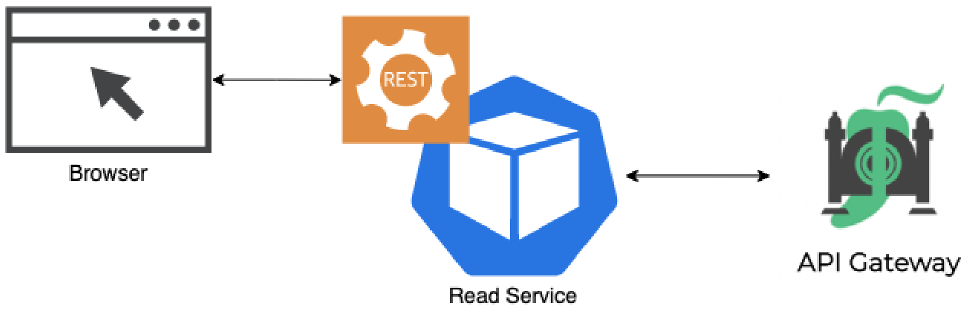


Figure 1.2.: Project Plan Architecture

The architecture of the application will contain Python backend microservices in Kubernetes pods. In this project the service that reads network data will be implemented and connected to the Jalapeño API Gateway to query all necessary information of a network. The server will be built with the Flask framework. The results of this will be shown in a server side rendered website. For analysis purposes the data managed in the network read backend may be saved in a later project.

The team will receive a server from Urs Baumann from the INS institute for networked solutions, to better work with the existing infrastructure.

The team members will be working on their own laptops with following software installed:

- Python 3.9 with Flask framework
- Visual Studio Code with Duckly extension
- Kubernetes
- Git
- Swagger
- LaTeX

Additionally, the team members have accounts on following platforms:

- Clockify
- Teams
- Duckly
- GitLab

1.11. Quality Assurance

Method	Time Frame	Goal	Description
`Unit Tests`	Construction phase	min. 80% Code Coverage	All components of the application are sufficiently tested.
`Integration Tests`	Construction phase	Components work efficiently together.	All possible connections are testet at least once.
`System Tests`	Construction phase	All use cases are implemented	At least once per iteration a complete system test is manually done.
`Acceptance Tests`	Construction phase	Non-functional requirements, especially scalability are met.	At least once per iteration a acceptance test is manually done.
`Acceptance Tests`	Construction phase	Non-functional requirements, especially scalability are met.	At least once per iteration a acceptance test is manually done.
`Continuous integration`	Whole project	Integration is always assured.	Per issue there will be at least one merge request.
`Continuous deployment`	Construction and transition phase	Deployment is always updated with newest changes	All merges into main branch will be automatically deployed via pipeline.
`Peer Review`	Whole project	Improvement of quality of work.	Code is always at least once reviewed by all team members.

1.12. Development

The source code will be versioned on GitLab and with regular intergration and reviews checked for quality. We will use the Git Feature Branch Workflow workflow for this.

1.12.1. CI/CD

Continous integration and deployment will be done via GitLab with the inbuilt pipeline.

For deployment the inbuilt package management of Python will be used with Poetry. Kubernetes containers will be furnished with new images and started from the pipeline.

1.12.2. Code Reviews

Code reviews will be done via the merge request workflow of GitLab with the Definition of Done 1.13. Additionally, if problems in code arise they may be discussed after sprint planning meetings.

1.12.3. Code Style Guidelines

The guidelines of the Visual Studio extension of Microsoft (IntelliSense Pylance) with Black linter will be used for all Python code.

1.13. Definition of Done

Merge requests to the **main** `branch` will only be granted if all points of Definition of Done are met:

1. Code works without errors and warnings.
2. Tests are written for all code changes.
3. Tests are green.
4. Defined code guidelines and standards are met.
5. Code was reviewed by team partner.
6. Non functional requirements are met.
7. Integration into master without any merge conflicts.

1.13.1. Documentation

1. Pipeline works without errors.
2. Spelling check done.
3. Content was reviewed by project partner.
4. Integrations into master without any merge conflicts.

1.14. Tests

1.14.1. Automated Tests

Unit and integration tests will be written for all components of the software. They will be updated and adapted as needed to ensure a quality codebase.

The testing will be integrated into the CI/CD pipeline feature of GitLab for unit and integration tests. Tests will run with each commit to one of the two main branches **mngt** or **dev** and if a pull request to the **main** `branch` is created. Merges into the **master** will only be possible if all tests are successful.

1.14.2. System Tests

System tests will check that all (currently implemented) use cases are correctly implemented. The tests will be done manually at the end of each iteration and the results and measures for corrections will be documented.

1.14.3. Acceptance Tests

Acceptance tests will check that all [non-functional requirements] are fulfilled. The tests will be done manually at the end of each iteration and the results and measures for corrections will be documented.

1.14.4. Usability Tests

The frontend will be kept minimal in the prototype with only a call to synchronize with the network. As such, usability tests will be done in the BA workload.

Should there be enough time to add more UI functionalities to this prototype will it be integrated and tested with usability tests as well.

2. Risk Analysis

2.1. Purpose

This document shows all risks to this project during the SA and they will be managed. It will be updated at each sprint review to adapt risk management.

2.2. Risk Matrix

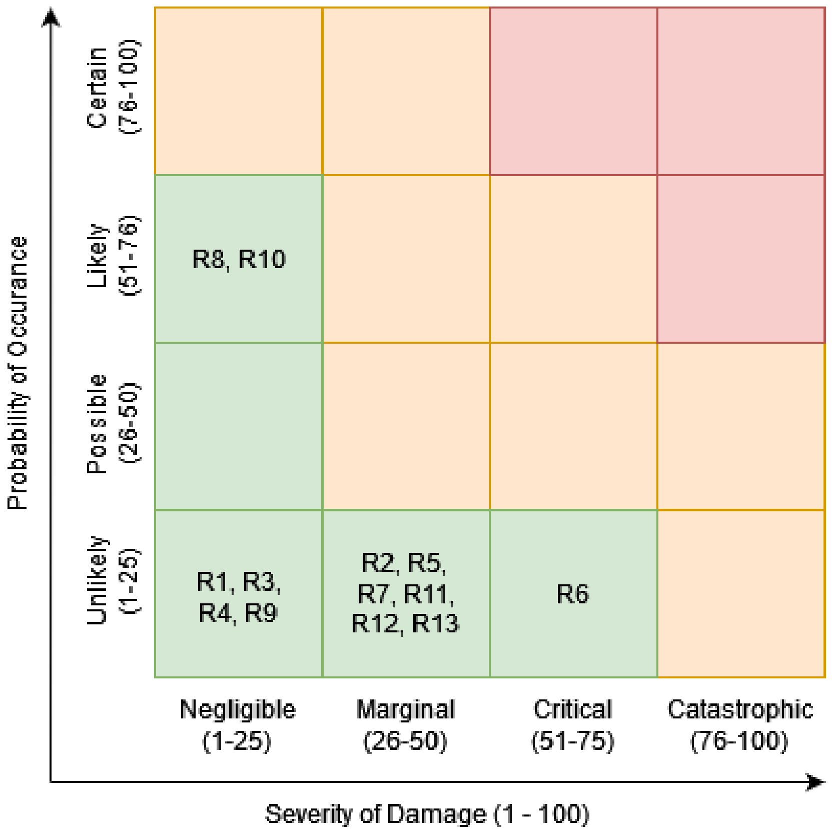


Figure 2.1.: Risk Matrix

2.3. Risk Management

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**
R1	**Requirements are insufficient**	Incorrect, uncompleted or missing functional or non-functional requirements lead to insufficient functionalities in application.	20	20	4
	Prevention		**Behaviour on occurrence**		
	Intensive review of requirements with advisors and industrial partner.		Revise of all requirements and discussion of any necessary edits with advisors and industrial partner. Replanning of sw architecture if necessary.		

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**
R2	**Unfitting architecture design**	Chosen design is not realisable or does not fulfill non-functional or functional requirements.	30	20	6
	Prevention		**Behaviour on occurrence**		
	Thoroughly plan architecture and review with advisors. Develop domain analysis and at least C4 diagrams.		Review of the architecture design with external experts.		

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**
R3	**Time management failure**	The team fails to build a working prototype in the allotted time frame.	25	20	5
	Prevention		**Behaviour on occurrence**		
	Make a time table for the project and include buffer time. In the sprint reveiw spent hours are evaluated and compared to the projected time table.		If there is an insurmountable gap to the time table team members will invest extra hours.		

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**

R4	**Unknown Technologies**	The team does not have experience in the chosen technologies.	20	20	4
	Prevention		**Behaviour on occurrence**		
	Team members will research and practice with unknown technologies.		Replace technology with adequate alternative where possible or reduce sw complexity if feature is not implementable as is.		

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**
R5	**Inadequate Research of flex algo and YANG model**	The team does not understand the network technologies enough to build sw properly.	30	20	6
	Prevention		**Behaviour on occurrence**		
	Research flex algo and carefully choose YANG model. Regularly discuss findings with advisors.		Consult external expert (Michel Bongard, others).		

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**
R6	**Cisco IOS-XR version insufficient**	Required functionalities can't be implemented with the given image of the virtual routers.	60	20	12
	Prevention		**Behaviour on occurrence**		
	Check with advisor which version is used and check for necessary features.		Consult with advisors about changing image. If not possible adjust sw features.		

NR	**Title**	**Description**	**max. Damage [%]**	**Prob. of occurrence [%]**	**Weighted Damage**
R7	**Jalapeño fails**	Jalapeño or the underlying virtual network fails to be provided or stops working and there is no network data to use in sw.	40	20	8
	Prevention		**Behaviour on occurrence**		

	Network is requested and will be provided before start of construction phase. Mocking will be implement directly after successfull connection to Jalapeño API Gateway is established.	Priorisation of mocking implementation.

NR	Title	Description	max. Damage [%]	Prob. of occurrence [%]	Weighted Damage
R8	**Jalapeño API Gateway failure**	Jalapeño API Gateway cannot be migrated to dev server in useful timeframe.	10	60	6
	Prevention		**Behaviour on occurrence**		
	Server is requested and Gateway will be installed as soon as server is provided. Responsible people are informed.		First consult the corresponding internal employee (Urs Baumann). Further, contact external expert (Michel Bongard, others).		

NR	Title	Description	max. Damage [%]	Prob. of occurrence [%]	Weighted Damage
R9	**Failure to implement CI/CD pipeline**	Features like automated testing, metric control (SonarCube), docker deployment, etc. are incomplete or missing.	10	10	1
	Prevention		**Behaviour on occurrence**		
	Both team members attend the module CloudOps. Pipelines of other projects will be researched.		Consult corresponding internal experts.		

NR	Title	Description	max. Damage [%]	Prob. of occurrence [%]	Weighted Damage
R10	**Failure to utilise Jalapeño API Gateway**	Failure to properly use the Jalapeño API Gateway to get the network and YANG model data.	20	60	12
	Prevention		**Behaviour on occurrence**		
	Attended API Gateway demo from Michel Bongard. YANG model reserached and chosen. First step in construction phase is the interface to the gateway.		First consult the corresponding internal employee (Urs). Further, consult external expert (Michel Bongard, others).		

NR	Title	Description	max. Damage [%]	Prob. of occurrence [%]	Weighted Damage
R11	**Software bug**	Insufficient tests may lead to undiscovered bugs in code.	40	25	10
	Prevention		**Behaviour on occurrence**		
	Debug and review code reqularly. Testing is prioritized in the project.		Write tests that catch bug and fix software accordingly.		

NR	Title	Description	max. Damage [%]	Prob. of occurrence [%]	Weighted Damage
R12	**Absence of team members**	Team members miss working days because of sickness, accidents, etc.	30	20	6
	Prevention		**Behaviour on occurrence**		
	Time buffer is planned and in extreme cases an extension of submission deadline can be requested. Remote working is possible with Teams and Duckly.		If possible work remotely over MS Teams and/or Duckly. If not, adjust time planning.		

NR	Title	Description	max. Damage [%]	Prob. of occurrence [%]	Weighted Damage
R13	**Pandemic / War**	Because of this cruel world with it's even more cruel events (free UK!), we have to consider this possibility.	30	20	6
	Prevention		**Behaviour on occurrence**		
	Time buffer is planned and in extreme cases an extension of submission deadline can be requested. Remote working is possible with Teams and Duckly.		If possible work remotely over MS Teams and/or Duckly. If not, adjust time planning.		

Table 2.1.: List of Project Risks

3. Quality Measures

3.1. Coding Guidelines

This project is written in Python following the Black style guide. Both team members have the Black extension for Visual Studio Code and have auto formatting by saving enabled. Additionally Black compliance is checked in the pipeline on each merge request.

3.2. Definition of Done

There are "Definition of Done" checklists 2 defined for both documentation and source code. Before branches can be merged into the `main`, `dev` or `doc_thesis` branches these checklists need to be ticked off by the other team member. Should there be an issue, the author of the merge request is notified and the merge is put on hold until sufficiently corrected.

3.3. End of Elaboration Checklist

The Checklist was consulted at the end of the elaboration phase. The following results were noted.

We have understood the customer. [6]
Both the non-functional and functional requirements are defined and reviewed with the project advisors. They can be found in chapters 1.2.

We have mastered all tools [6]
Team members have completed research and tutorials on all tools of the project.

The architecture is known to all project members. There is a working product with the core architecture, large interfaces defined. [6]
The architecture is defined in chapter 3.3. A first prototype of the architecture was tagged at the start of the construction phase Prototype-EoE.

There are user interface designs (graphics, wireframes) that the customer likes. [6]
There will not be an elaborate frontend in the product of the SA project. As such there is only a rudimentary wireframe in chapter 3.10 that will not be implemented.

The next two iterations are roughly planned, so many work packages created, and an accurate time estimate provided to the customer. [6]
Issues for the first two sprints are defined in Gitlab.

All big risks and big question marks are gone. [6]
The risk analysis can be seen in chapter risk analysis of the management document. The history of the risiko matrix can be viewed in Gitlab.

3.4. Time Tracking

Time tracking is done in clockify. In each sprint review a separate report for both team members and a combined one is exported and saved in gitLab. The reports can be seen in the appendix in section 4. The time slots are mapped to the issues via their descriptions, to milestone via projects and to iterations and work categories via tags. Reports are made at each sprint review and can be viewed here.

3.5. GitLab Workflow

The project uses the GitFeatureBranchWorkflow. The branch graphs can be found here. All issues are managed in gitlab. All work to be done, including new functionalities, corrections, etc. are directly translated into issues.

3.6. Code Reviews

Code Review is done by the other team member, if the changes are not worked on in pair programming. What is to be reviewed is defined in the "definition of done" 2. Any deviations to the checklist need to be discussed in the team and noted in the comments section of the merge request.

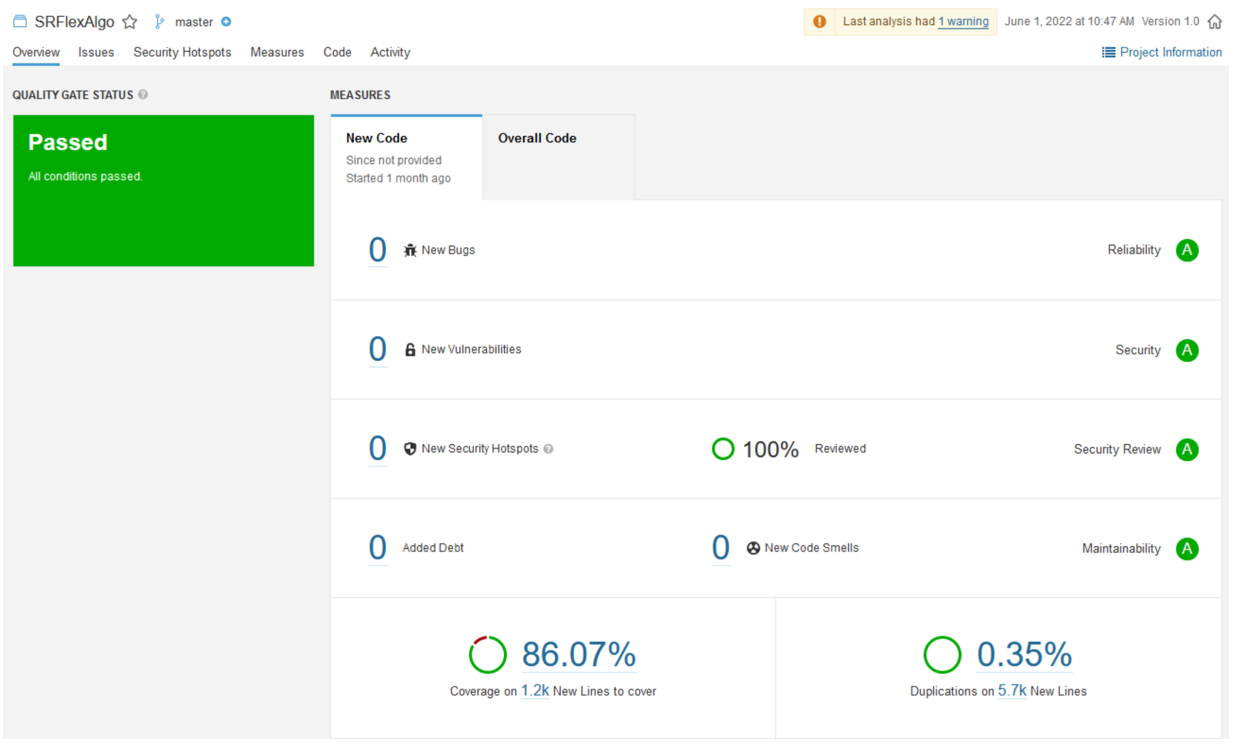
3.7. Sprint Reviews

At the end of each sprint the team holds a review. In this the team discusses the progress of the project and time table, adapt the risk analysis, overview the issue board, review the code and solve any problems. Additionally in the construction phase system and acceptance tests are performed on the current stable source code 5. There will be releases tagged to track the progression and noted in the test protocols.

3.8. Metrics and Code Analysis

3.8.1. SonarQube

The SonarQubeDashboard shows the status of the latest code pushed into the dev branch and its sub-branches. It is updated on each merge request to ensure that the code can be checked properly before merging into the stable code base. This ensures only checked and tested code is deployed to Kubernetes. In SonarQube metrics like tests, code coverage, common security hotspots, vulnerabilities and code smells are covered.


SRFlexAlgo master

Last analysis had 1 warning June 1, 2022 at 10:47 AM Version 1.0

Overview Issues Security Hotspots Measures Code Activity

Project Information

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since not provided
Started 1 month ago

Overall Code

0 New Bugs — Reliability A

0 New Vulnerabilities — Security A

0 New Security Hotspots — 100% Reviewed — Security Review A

0 Added Debt — 0 New Code Smells — Maintainability A

86.07% Coverage on 1.2k New Lines to cover

0.35% Duplications on 5.7k New Lines

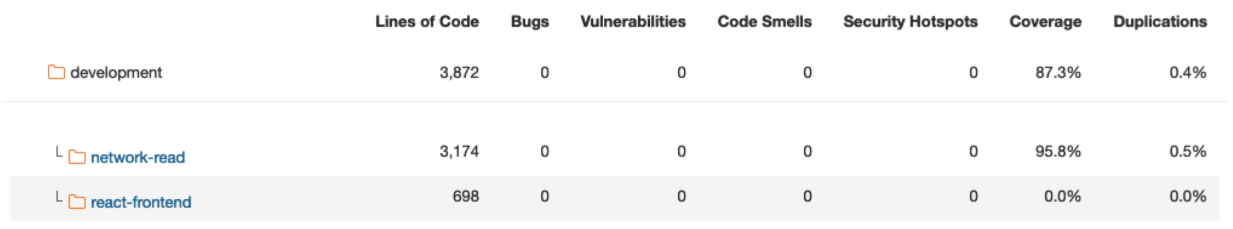
Figure 3.1.: SonarQube Dashboard - All Code

SonarQube will ignore the folder of the mocking tool for accurate code analysis. The tool will only be used locally to generate the required mocked data which will be entered manually into the ArangoDB and does not need to conform to strict coding guidelines.

While the rudimentary React.js frontend is not excempt from SonarQube, it was not made with clean coding in mind, as it will need to be reworked for a further project. Additionally a BA project from another group is currently developing an uniform frontend for all SR-Apps that may be implemented in a following project into the SR-FlexAlgo.

The dashboard of SonarQube shows an overview of the currently deployed code and provides a ranking from A to F, in which A is the highest and most desired score. Actions need to be taken before a merge can commence if the rankings on the dashboard page of the SonarQube dip under "A".
The code smells "duplicated code" for test data are reviewd and set to ignore, as there would not be much gain for solving them. Additionally security hotspots for the `randint` function are similarly ignored, as they are not used for cryptography and therefore don't impact the applications security. At the end of the construction phase time is scheduled to remove all unnecessary comments and code (such as the development routes save for the clearDB one, etc).

3.8.2. Network Read



	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
development	3,872	0	0	0	0	87.3%	0.4%
└ network-read	3,174	0	0	0	0	95.8%	0.5%
└ react-frontend	698	0	0	0	0	0.0%	0.0%

Figure 3.2.: SonarQube Dashboard - Code Network Read

- *Lines of code* 3'156
- *Classes* 18
- *Code coverage* 95.8%
- *Cognitive Complexity* 104

3.8.3. Frontend

The Frontend is not covered separately as we have server generated html templates included in the `Network Read` statistics above. The React.js SPA frontend is a prototype developed by Michel Bongard and we only added minimal functionality to it. Here only code smells of our code were reviewed and solved where meaningful. Other metrics were largely ignored.

3.9. Testing

3.9.1. Unit Tests

- Each method in the backend is testet at least once with a valid and invalid input, where possible and meaningful.
- Frontend will not be tested in this project, as we use either server side rendered html (tested in integration tests) or later the premade project react-sigmajs-demo for our frontend.
- Access to different layers is mocked.

3.9.2. Integration Tests

- Methods that have layer crossing access are tested where unit tests are not sufficient. The server side rendered html is tested in the integration tests of the `routes` package.
- For each method there is at least one valid and invalid test.

3.9.3. System and Acceptance Tests

Will be done according to the test templates 5. They are conducted at each sprint review on a tag of the latest stable source code, protocols are documented in the appendix 5. Results are directly translated into new issues.

3.9.4. Usability Tests

There will be no usability tests as there is not yet a proper frontend implemented in the prototype.

3.10. Pipeline

There are two main parts to the pipeline. The first one is for the construction of the documentation files. This one will only complete successfully if the LaTeX standards are met and the runner was able to build the pdfs.

The other part is for the source code of the application. It checks coding guidelines with the formatter Black, controls the static types of the code with MyPy, runs the Unit- and Integration tests and generates a coverage report with the help of Pytest and Cobertura (which can be viewed directly at the GitLab's pipeline under its Tests tab). To finish it runs the SonarQube scanner with the provided results from the previous Pytests.

If the jobs of format or static type checking fail the pipeline will be stopped. If any of the other jobs fail, the pipeline will go on, in order to be able to run the SonarQube scanner which will provide a better overview of any problems in the code. This is also the reason why the pipeline will run up to the SonarQube scanner job if there is a merge request of a developer branch.

The SonarQube Job will fail if the code does not correspond to the defined quality gate status that is set in the background of SonarQube. This is visible in the browser SonarQube in the overview page.

To ensure good code quality, the creation of the Docker file will only be done if the pipeline ran through the build container job successfully and there was a successful merge into the `dev` branch.

In case the pipeline's SonarQube job wasn't 100% successful a team member can still merge manually at GitLab. This case has to be discussed by both team members to ensure no instable or unfitting code is deployed.

3.11. Non-Functional Requirements

The state of the non-functional Requirements was extensively tested in each sprint review. Testing protocols can be found 5.

3.11.1. Security

Not implemented in this project, will be handled in a further project.

3.11.2. Reliability

The code is build with a try catch functionality that handles all exceptions and give appropriate error messages back. For the API calls used in the React.js frontend HTTP status 5** is returned if an error occurs. Additionally the twelve-factor methodology was implemented to ensure a high standard of recoverability was reached.

3.11.3. Usability

Not implemented in this project, will be handled in a further project.

3.11.4. Performance

Performance of the source code can be found in the testing protocols.

3.11.5. Maintainability

Extensive unit and integration tests were written for the software. Results and metrics can be found in SonarQube. Additionally logs to the standard error stream are implemented in the error handling of the software and can be viewed and persisted as needed.

Part VI.

Indexes

List of Figures

List of Tables

Listings

Part VII.

Appendix

The project repo can be found on GitLab.

1. Task Definition

1 Task Formulation

1.1 Institute

The SR-App FlexAlgo project is being conducted in partnership with the Institute for Networked Solutions (INS) and is a new SR-App that will be part of the SR-App Ecosystem.

1.2 Supervisor

Prof.Laurent Metzger, Institute for Networked Solutions (INS), laurent.metzger@ost.ch

1.3 Co-Supervisor

Urs Baumann, Institute for Networked Solutions (INS), urs.baumann@ost.ch

1.4 Initial Situation

Flexible algorithms (FlexAlgo) is in a draft state at the IETF (https://datatracker.ietf.org/doc/draft-ietf-lsr-flex-algo/) and in the process of being standardized. It is a dynamic segment routing technology and manage the traffic on a network in a high granularity. This freedom comes with additional complexity. Building and maintaining all the flexible algorithms in a network is a manual task, time consuming and needs intimate knowledge of network configurations. The SR-App FlexAlgo should make configurations fast and easy and allow not only a graphical view of the configurations but also show possible problems in the network. In this research paper a prototype for the SR-App is made, to check the feasability of a segment routing application that handles the configuration of flexible algorithms.

1.5 Expectations and Goals

The goal of this research paper is a proof of concept for a SR-App FlexAlgo application. For this, research on the segment routing's flexible algorithm technology is to be done and a prototype is to be built that displays all relevant information to the technology.

Expected minimum work results:

- Research on where all FlexAlgo information is stored and how to access it via the Jalapeño API Gateway.
- Application prototype that shows FlexAlgo relevant information in a minimal frontend, i.e. a json or hmtl list.
- Prototype needs to handle a network of up to 1000 routers in a reasonable amount of time.

Optional work results:

- Single Page Application frontend with graphical representation of the network topology.
- Graphical representation of the running algorithm configurations.
- Prototype needs to run on Kubernetes and must be developed for cloud native environments.
- Working pipeline for the prototype.

Additional goals for success:

- Code must be written in Python or Go.
- Optional Single Page Application frontend needs to be written in React.
- Research of basic functionality of Flexible Algorithm technology, as far as application prototype needs.
- The relevant YANG models for flex algo need to be researched.
- Jalapeño API Gateway needs to be utilized via gRPC connection. If the chosen YANG model depends on changes to the Gateway that can not be implemeneted in time, direct connections to the databases are also possible.

As this is a software project the resulting application code should follow best practices known in software engineering. As this is a prototype, the application needs to give a representative proof of concept for the adaption into the final software.

1.6 Definition of Implementation

This task formulation is written by the students and has been reviewed and validated by the supervisor. As this paper is written within the framework of the "Semesterarbeit" module, the students have the right of weekly meetings with the supervisors. Additional meetings can be held at the students' or supervisors' requests. The work process should be transparent, continuous and documented with repository versioning and time tracking. A project plan is to be created in the first week of the project with a timetable, milestones and basic procedures defined. The requirements engineering is done by the students and reviewed by the supervisors. It should define functionality of a minimum viable product and additional features.

1.7 Documentation

Part of this student research project is a documentation based on the regulations of the department of computer science at the OST university of applied sciences. All documents need to be finished at the time of submission. Since this project may be of international interest, it should be written in english. All additional submissions can be written in english as well, but are not mandatory so.

1.8 Important Dates

21.02.2022 Start of the project 03.06.2022 Submission

1.9 Additional Notes

This research paper was developed based on the oral assignment by the supervisor. This task formulation was written by the students and approved by Prof. Laurent Metzger.

Date: 26.05.2022

Signature Prof. Laurent Metzger: [illegible]

2. Definition of Done Checklists

Code

- Code works without errors and warnings.
- Tests are written for all code changes.
- Tests are green.
- Defined code guidelines and standards are met.
- Code was reviewed by team partner.
- Non functional requirements are met.
- Integration into master without any merge conflicts.

Documentation and Management

- Pipeline works without errors.
- Spelling check done.
- Content was reviewed by project partner.
- Integrations into master without any merge conflicts.

3. Meeting Protocols

The folder **/project_management/meeting/** of the GitLab repository contains all meeting protocols of this project.

The files ***_AdvisorMeeting.pdf** provide protocols of the weekly meetings held with the project advisors.

The file **02-28_jalapeno_meeting.pdf** provides a protocol of the meeting held with Michel Bongard for the topic of the Jalapeño API Gateway.

The file **03-21_SegmentRouting.pdf** provides a protocol of the meeting held with the project advisors for the topic of Segment Routing.

The file **03-21_Kubernetes.pdf** provides a protocol of the meeting held with Yannick Zwicker for the topic of Kubernetes.

4. Time Reports

The files in the folder **/poject_management/time_report/** of the GitLab repository are the time reports for the team members of this project. For each sprint a report was made for each team member and one for both.

5. Testing Protocols

5.1. Acceptance Test - Review of Iteration 7

Preconditions
The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.
For the Performance tests the data from the Mocking Tool will be used to simluate a large network.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation
Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed Acceptance Tests

- System Test of iteration 6 was done on tag Dev-I6.testing.
- This System Test is done on tag Dev-I7-final-submission.

5.1.1. Tests

Functionality

`Non-Functional Requirement`	`Implementation`	`Shortcomings`	`Status`
`Accuracy`	Data retrieved from Jalapeño API Gateway is up-to-date.	—	Implemented
`Interoperability`	The connection to the Jalapeño API Gateway is established over gRPC.	Will not be implemented for this project.	Frozen

Reliability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Fault Tolerance`	Server remains running if connection to API is down.	Instead of API, connection to ArangoDB was handled successfully.	Implemented

`Recoverability`	The integrated Kubernetes mechanism for failovers handles failed pods.	—	Implemented

Performance

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Data Integrity`	Changes in network are visible with a browser refresh.	—	Implemented
`Scalability`	The application runs with 1'000 routers in the network.	—	Implemented
`Time Behavior`	Changes in the network are visible wihtin 3 seconds after refreshing the browser.	Works on everything. At the moment we have 102 flex algorithms.	Implemented

Maintainability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Analysability`	The user receives a message to inform about occuring errors.	The error handling will be implemented as a HTML template or react frontend message.	Implemented
`Analysability`	Error messages will be written into stdout.	Is written into stderr.	Implemented
`Testability`	All Unit and Integration Tests run successfully.	—	Implemented

Pending Improvements

-

5.2. Acceptance Test - Review of Iteration 6

Preconditions

The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.
For the Performance tests the data from the Mocking Tool will be used to simluate a large network.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation

Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed Acceptance Tests

- System Test of iteration 5 was done on tag Dev-I5.
- This System Test is done on tag Dev-I6.testing.

5.2.1. Tests

Functionality

`Non-Functional Requirement`	`Implementation`	`Shortcomings`	`Status`
`Accuracy`	Data retrieved from Jalapeño API Gateway is up-to-date.	—	Implemented
`Interoperability`	The connection to the Jalapeño API Gateway is established over gRPC.	Will not be implemented for this project.	Frozen

Reliability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Fault Tolerance`	Server remains running if connection to API is down.	Instead of API, connection to ArangoDB was handled successfully.	Implemented

`Recoverability`	The integrated Kubernetes mechanism for failovers handles failed pods.	—	Implemented

Performance

Non-Functional Requirement	Implementation	Shortcomings	Status
`Data Integrity`	Changes in network are visible with a browser refresh.	—	Implemented
`Scalability`	The application runs with 1'000 routers in the network.	—	Implemented
`Time Behavior`	Changes in the network are visible wihtin 3 seconds after refreshing the browser.	Works on everything. At the moment we only have 102 flex algorithms.	Implemented

Maintainability

Non-Functional Requirement	Implementation	Shortcomings	Status
`Analysability`	The user receives a message to inform about occuring errors.	The error handling will be implemented as a HTML template. (in process)	Implemented
`Analysability`	Error messages will be written into stdout.	—	Implemented
`Testability`	All Unit and Integration Tests run successfully.	—	Implemented

Pending Improvements

- Error handling with HTML.

5.3. Acceptance Test - Review of Iteration 5

Preconditions
The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.
For the Performance tests the data from the Mocking Tool will be used to simluate a large network.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation
Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed Acceptance Tests

- Acceptance Test of Iteration 4 was done on tag Dev-I4.
- This Acceptance Test is done on tag Dev-I5-backend.

5.3.1. Tests

Functionality

`Non-Functional Requirement`	`Implementation`	`Shortcomings`	`Status`
`Accuracy`	Data retrieved from Jalapeño API Gateway is up-to-date.	—	Implemented
`Interoperability`	The connection to the Jalapeño API Gateway is established over gRPC.	Will not be implemented for this project.	Frozen

Reliability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Fault Tolerance`	Server remains running if connection to API is down.	Instead of API, connection to ArangoDB was handled successfully.	Implemented

`Recoverability`	The integrated Kubernetes mechanism for failovers handles failed pods.	—	In process

Performance

Non-Functional Requirement	Implementation	Shortcomings	Status
`Data Integrity`	Changes in network are visible with a browser refresh.	—	Implemented
`Scalability`	The application runs with 1'000 routers in the network.	—	Implemented
`Time Behavior`	Changes in the network are visible wihtin 3 seconds after refreshing the browser.	Works on everything. At the moment we only have 12 flex algorithms.	In process

Maintainability

Non-Functional Requirement	Implementation	Shortcomings	Status
`Analysability`	The user receives a message to inform about occuring errors.	—	Implemented
`Analysability`	Error messages will be written into stdout.	—	Implemented
`Testability`	All Unit and Integration Tests run successfully.	—	Implemented

Pending Improvements

- Adapt mocking tool with more algorithms.
- Html return template.
- Error handling with HTML.
- Implementation of pipeline with Kubernetes integration.

5.4. Acceptance Test - Review of Iteration 4

Preconditions
The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.
For the Performance tests the data from the Mocking Tool will be used to simluate a large network.

Status: Access to Jalapeño API Gateway not possible at the moment, get data via ArangoDB.

Preparation
Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed Acceptance Tests
This acceptance test is done on tag Dev-I4.

5.4.1. Tests

Functionality

`Non-Functional Requirement`	`Implementation`	`Shortcomings`	`Status`
`Accuracy`	Data retrieved from Jalapeño API Gateway is up-to-date.	Cannot access Gateway but go over ArangoDb. This data is live. A connection to the gateway is implemented on branch `feat_add-grpc`, but there are cache problems in the gateway.	Partially implemented
`Interoperability`	The connection to the Jalapeño API Gateway is established over gRPC.	See shortcomings `Accuracy`.	Partially implemented

Reliability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Fault Tolerance`	Server remains running if connection to API is down.	Is not implemented. Began implementation in branch `feat_error-handling`.	Not implemented

`Recoverability`	The integrated Kubernetes mechanism for failovers handles failed pods.	—	Not implemented

Performance

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Data Integrity`	Changes in network are visible with a browser refresh.	Momentarily via ArangoDB connection and not gRPC.	Implemented
`Scalability`	The application runs with 1'000 routers in the network.	Not tested yet because of missing error handling and current construction in the gateway.	Pending
`Time Behavior`	Changes in the network are visible wihtin 3 seconds after refreshing the browser.	Route `getAlgos` is most heavy in computation needs and runs under a second. Limitation is that there are only 8 routers in the network.	Implemented

Maintainability

Non-Functional Requirement	**Implementation**	**Shortcomings**	**Status**
`Analysability`	The user receives a message to inform about occuring errors.	Will be addded with error handling. Began implementation in branch `feat_error-handling`.	Pending
`Analysability`	Error messages will be written into stdout.	Will be addded with error handling. Began implementation in branch `feat_error-handling`.	Pending
`Testability`	All Unit and Integration Tests run successfully.	Ok.	Implemented

Pending Improvements

- Complete implementation of error handling.
- Complete implementation of gRPC connection.
- Implementation of Pipeline with Kubernetes integration.

5.5. System Test - Review of Iteration 7

This document will provide an overview of the performed System Tests of this project.

Preconditions

The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation

Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed System Tests

- System Test of iteration 6 was done on tag Dev-I6.testing.
- This System Test is done on tag Dev-I7-final-submission.

5.5.1. Tests - SA MVPs

UC00 - List of Flex Algo

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC01 - Live Updates

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED

Shortcomings
Live updates work for everything. Additional routes were implemented to simulate gRPC updates of links, nodes and definitions.

UC02 - Failure to connect

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		

UC03 - List all Flex Algos

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC04 - Logging Software

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC05 - Graph Topology

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC06 - Graph 1 Algo

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
Works, sometimes needs a second try.		

UC07 - Graph all Algos

Implemented	**Process**	**Status**
Yes	Refresh website.	FROZEN
Shortcomings		
Not implemented. See results of thesis section for further information.		

Pending Improvements

-

5.6. System Test - Review of Iteration 6

This document will provide an overview of the performed System Tests of this project.

Preconditions

The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation

Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed System Tests

- System Test of iteration 5 was done on tag Dev-I5.
- This System Test is done on tag Dev-I6.testing.

5.6.1. Tests - SA MVPs

UC00 - List of Flex Algo

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC01 - Live Updates

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
Live updates work for everything. Nodes has an additional update mechanic implemented, the others not.		

UC02 - Failure to connect

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
Not yet as HTML, only string message or HTTPS error.		

UC03 - List all Flex Algos

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC04 - Logging Software

Implemented	Process	Status

Yes	Refresh website.	PASSED
Shortcomings		
-		

5.6.2. Tests - SA Optionals

`Use Case`	`Implemented`	`Shortcomings`	`Status`
UC05 `Graph Topology`	no	—	In process
UC06 `Graph 1 Algo`	no	—	In process
UC07 `Graph all Algos`	no	—	In process

Pending Improvements

- HTML return template.
- Error handling with HTML.
- Implementation of pipeline with Kubernetes integration.

5.7. System Test - Review of Iteration 5

This document will provide an overview of the performed System Tests of this project.

Preconditions

The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.

Status: Access to Jalapeño API Gateway not possible for duration of this project, get data via ArangoDB.

Preparation

Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed System Tests

- System Test of iteration 4 was done on tag Dev-I4.
- This System Test is done on tag Dev-I5-backend.

5.7.1. Tests - SA MVPs

UC00 - List of Flex Algo

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC01 - Live Updates

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
Live updates work for everything. Nodes has an additional update mechanic implemented, the others not.		

UC02 - Failure to connect

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
Not yet as HTML, only string message or HTTPS error.		

UC03 - List all Flex Algos

Implemented	Process	Status
Yes	Refresh website.	PASSED
Shortcomings		
-		

UC04 - Logging Software

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
-		

5.7.2. Tests - SA Optionals

Use Case	Implemented	Shortcomings	Status
UC05 `Graph Topology`	no	—	In process
UC06 `Graph 1 Algo`	no	—	In process
UC07 `Graph all Algos`	no	—	In process

Pending Improvements

- HTML return template.
- Error handling with HTML.
- Implementation of pipeline with Kubernetes integration.

5.8. System Test - Review of Iteration 4

Preconditions

The Unit and following Integration Tests run successfully.
The Jalapeño API Gateway sends real time data from the virtual network. Should this not be possible then mocked data from the Mocking Tool will be fed into the ArangoDB of the Gateway to simluate the network data instead.

Status: Access to Jalapeño API Gateway not possible at the moment, get data via ArangoDB.

Preparation

Enter VPN of INS to access ArangoDB sources. Check ArangoDB collections have data.
`ls_node_coordinates` does not currently have data. One mocked data set will be inserted.

Information about the performed System Tests

This acceptance test is done on tag Dev-I4.

5.8.1. Tests - SA MVPs

UC00 - List of Flex Algo

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
Logic for affinity mappings not yet adapted to new db schema. Bit-map fields not yet in db and therefore not implemented in code.		
Expected Output	**Actual Output**	
Information list of all existing Flex Algos	Information list of all existing Flex Algos	
Next Steps		
Add affinity mappings. Check with Urs Baumann for missing fields in db.		

UC01 - Live Updates

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED
Shortcomings		
None		
Expected Output	**Actual Output**	
Changed information.	Changed information.	

UC02 - Failure to connect

Implemented	**Process**	**Status**
Not yet	Refresh website.	FAILED
Shortcomings		
Error handling not implemented yet. Began implementation in branch `feat_error-handling`.		
Expected Output	**Actual Output**	
Message about failed connection.	HTTP time out message.	
Next Steps		
Complete implementation of error handling.		

UC03 - List all Flex Algos

Implemented	**Process**	**Status**
Yes	Refresh website.	PASSED

Shortcomings	
Currently only with direct ArangoDB connection not via gRPC.	
Expected Output	**Actual Output**
Calculated list of all algos with their nodes and links.	Calculated list of all algos with their nodes and links.
Next Steps	
Implement gRPC.	

UC04 - Logging Software

Implemented	**Process**	**Status**
Not yet	Refresh website.	FAILED
Shortcomings		
Not yet implemented. Began implementation in branch `feat_error-handling`.		
Expected Output	**Actual Output**	
Log information about errors in stderr.	Nothing.	
Next Steps		
Complete implementation of error handling.		

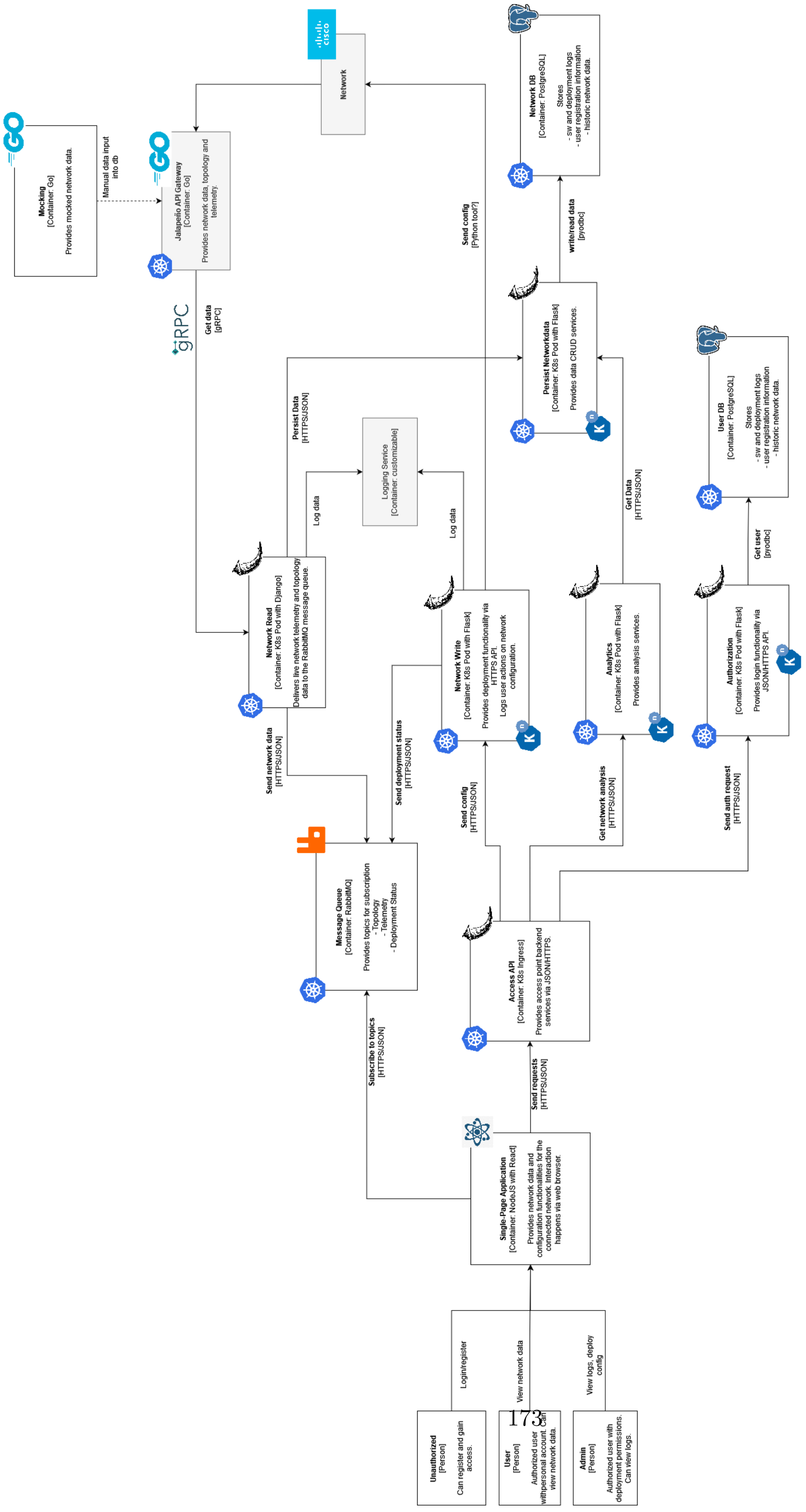
5.8.2. Tests - SA Optionals

`Use Case`	`Implemented`	`Shortcomings`	`Status`
UC05 `Graph Topology`	no	—	Planned
UC06 `Graph 1 Algo`	no	—	Planned
UC07 `Graph all Algos`	no	—	Planned

Pending Improvements

- Complete implementation of error handling.
- Complete implementation of gRPC connection.

6. Container Diagram


Mocking
[Container: Go]
Provides mocked network data.
Manual data input into db
Jalapeño API Gateway
[Container: Go]
Provides network data, topology and telemetry.
Network
Send config
[Python tool?]
Network DB
[Container: PostgreSQL]
Stores
- sw and deployment logs
- user registration information
- historic network data.
write/read data
[pyodbc]
gRPC
Get data
[gRPC]
Persist Networkdata
[Container: K8s Pod with Flask]
Provides data CRUD services.
Persist Data
[HTTPS/JSON]
Logging Service
[Container: customizable]
Log data
Log data
User DB
[Container: PostgreSQL]
Stores
- sw and deployment logs
- user registration information
- historic network data.
Get Data
[HTTPS/JSON]
Get user
[pyodbc]
Network Read
[Container: K8s Pod with Django]
Delivers live network telemetry and topology data to the RabbitMQ message queue.
Network Write
[Container: K8s Pod with Flask]
Provides deployment functionality via HTTPS API.
Logs user actions on network configuration.
Analytics
[Container: K8s Pod with Flask]
Provides analysis services.
Authorization
[Container: K8s Pod with Flask]
Provides login functionality via JSON/HTTPS API.
Send network data
[HTTPS/JSON]
Send deployment status
[HTTPS/JSON]
Send config
[HTTPS/JSON]
Get network analysis
[HTTPS/JSON]
Send auth request
[HTTPS/JSON]
Message Queue
[Container: RabbitMQ]
Provides topics for subscription
- Topology
- Telemetry
- Deployment Status
Access API
[Container: K8s Ingress]
Provides access point backend services via JSON/HTTPS.
Subscribe to topics
[HTTPS/JSON]
Send requests
[HTTPS/JSON]
Single-Page Application
[Container: NodeJS with React]
Provides network data and configuration functionalities for the connected network. Interaction happens via web browser.
Login/register
View network data
View logs, deploy config
Unauthorized
[Person]
Can register and gain access.
User
[Person]
Authorized user withpersonal account. Can view network data.
Admin
[Person]
Authorized user with deployment permissions. Can view logs.

7. YANG model

The YANG model chosen is `Cisco-IOS-XR-clns-isis-cfg@2021-04-02` which can be seen here.

The relevant FlexAlgo data are as follows:

Cisco-IOS-XR-clns-isis-cfg/isis/instances/instance/interfaces/interface/interface-afs/interface-af/topology-name/algorithm-prefix-sids/algorithm-prefix-sid/

- algo
- type
- value (prefix-sid)

Cisco-IOS-XR-clns-isis-cfg/isis/instances/instance/

- flex-algos/flex-algo/
  - includes
  - excludes
  - srlg-exclude-anies
  - running (active)
  - metric-type (metric)
  - priority
  - advertise-definition
  - flex-algo (algo number)
- instance-id (originNode)

Cisco-IOS-XR-clns-isis-cfg/affinity-mappings/affinity-mapping/

- affinity-name
- value

Cisco-IOS-XR-clns-isis-cfg/interfaces/interface/

- interface-name (interface id)
- int-affinity-tyble/flex-algos/
  - flex-algo (list Affinities)

8. Decoding Affinities

The file **/documentation/src/decodings/jalapeno-mapping.txt** of the GitLab repository provides a translation of the Affinity values saved in the objects of the ArangoDB and the binary values made by Severin Dellsperger.

The file **packet-121-trace1.txt** provides the analysis notes of Severin Dellsperger of a packet with the Affinity values on the network.

9. Network Configurations

The current configurations of the single devices of the project can be read in the folder **/project_management/topology_configs** of the GitLab repository.

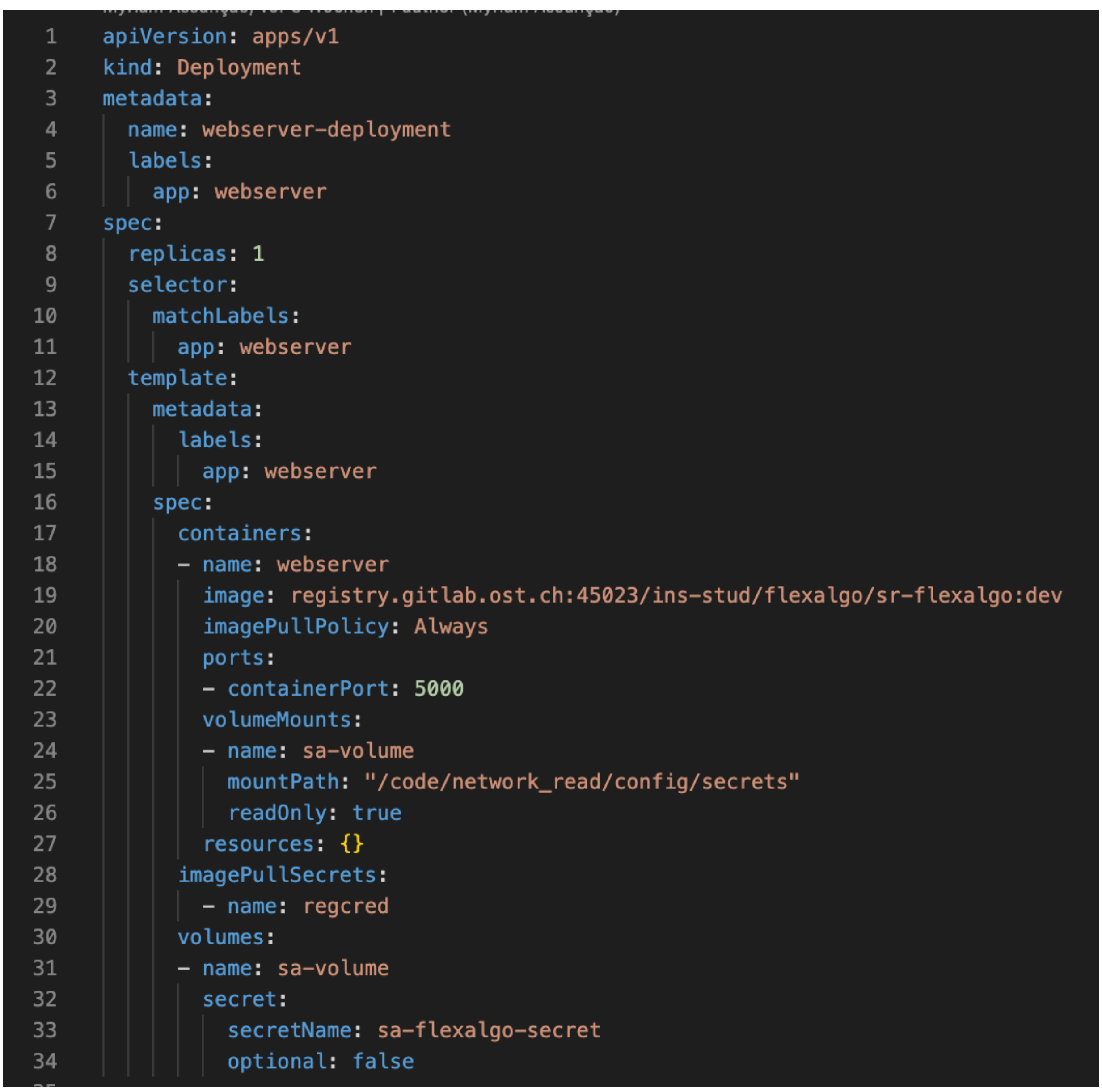
10. Pipeline

The complete pipeline can be inspected in the file `.gitlab-ci.yml` of the repository.

```yaml
workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH && $CI_OPEN_MERGE_REQUESTS'
      when: never
    - if: '$CI_COMMIT_BRANCH'


.base-python:
<-- shortened -->


stages:
  - documentation
  - formatting
  - check_static_types
  - test
  - sonar-scanner
  - build-image
  - deployment

latex:
<-- shortened -->

check_format:
<-- shortened -->


check_static_types:
<-- shortened -->


pytest:
<-- shortened -->

sonarqube-check:
<-- shortened -->


build-image:
  stage: build-image
  tags:
    - ins-student
  variables:
    IMAGE_TAG: ${CI_COMMIT_REF_SLUG}
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""]
  before_script:
    - mkdir -p /kaniko/.docker
    - echo "{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\", \
      \"password\":\"$CI_REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
  script:
    - /kaniko/executor --context $CI_PROJECT_DIR --dockerfile
    $CI_PROJECT_DIR/Dockerfile --destination $CI_REGISTRY_IMAGE:$IMAGE_TAG
    --destination $CI_REGISTRY_IMAGE:latest --use-new-run --single-snapshot
    --snapshotMode=redo
  rules:
    - if: $CI_COMMIT_BRANCH == "dev"


kubernetes-deployment:dev:
  stage: deployment
  tags:
    - ins-student
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  before_script:
    - kubectl config use-context ins-stud/flexalgo/sr-flexalgo:cluster-agent
  script:
    - cd deployment/k8s/base/
    - kubectl -n sa-flexalgo apply -f secrets.yaml
    - kubectl -n sa-flexalgo apply -f deployment.yaml
    - kubectl -n sa-flexalgo rollout restart deployment webserver-deployment
  environment:
    name: dev
    url: https://sa-sr-flexalgo.stu.network.garden
  rules:
    - if: $CI_COMMIT_BRANCH == "dev"
```

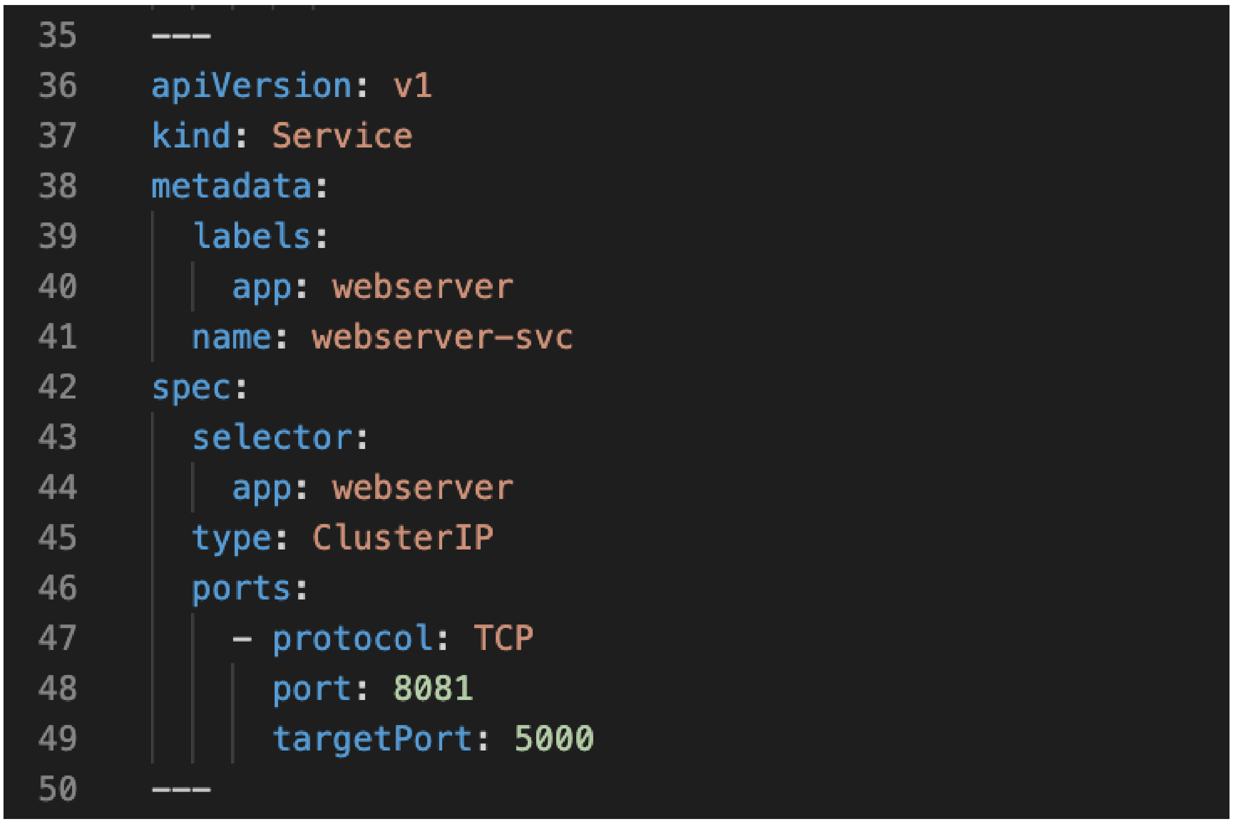
Listing 10.1: Pipeline

11. Kubernetes Files



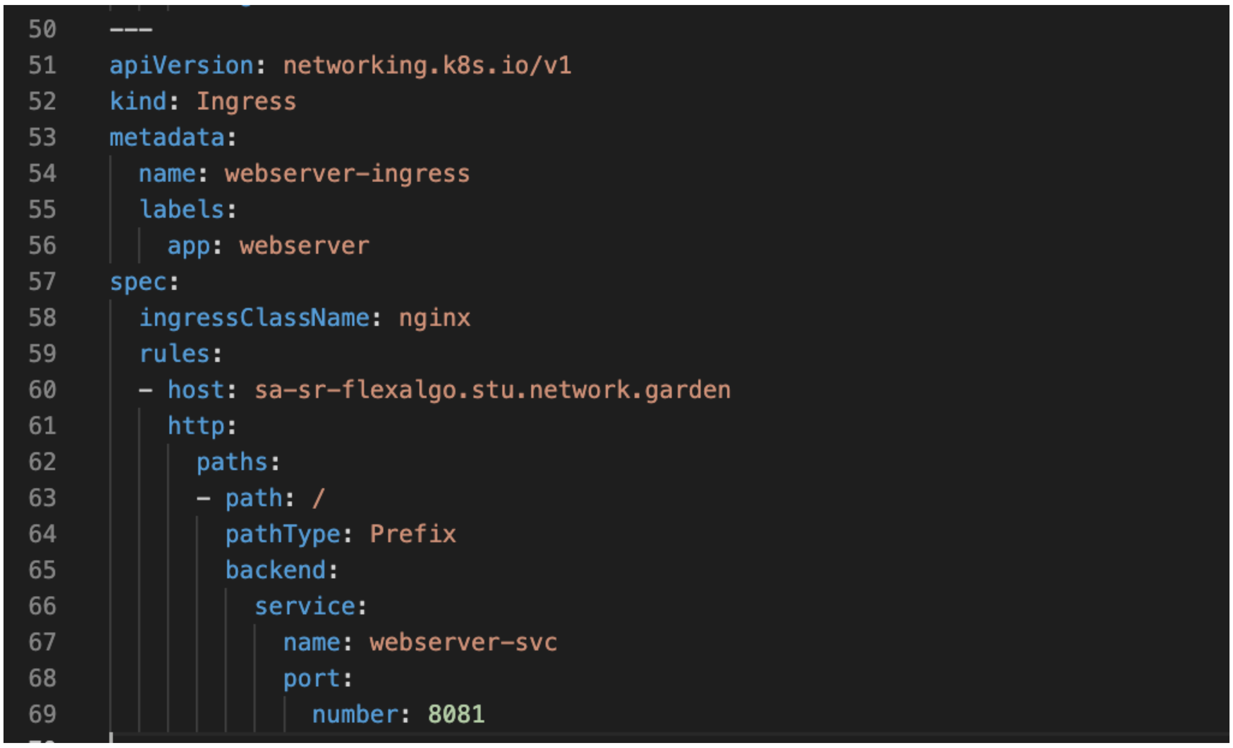
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver-deployment
  labels:
    app: webserver
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: webserver
        image: registry.gitlab.ost.ch:45023/ins-stud/flexalgo/sr-flexalgo:dev
        imagePullPolicy: Always
        ports:
        - containerPort: 5000
        volumeMounts:
        - name: sa-volume
          mountPath: "/code/network_read/config/secrets"
          readOnly: true
        resources: {}
      imagePullSecrets:
        - name: regcred
      volumes:
      - name: sa-volume
        secret:
          secretName: sa-flexalgo-secret
          optional: false
```

Figure 11.1.: Deployment File Section Deployment



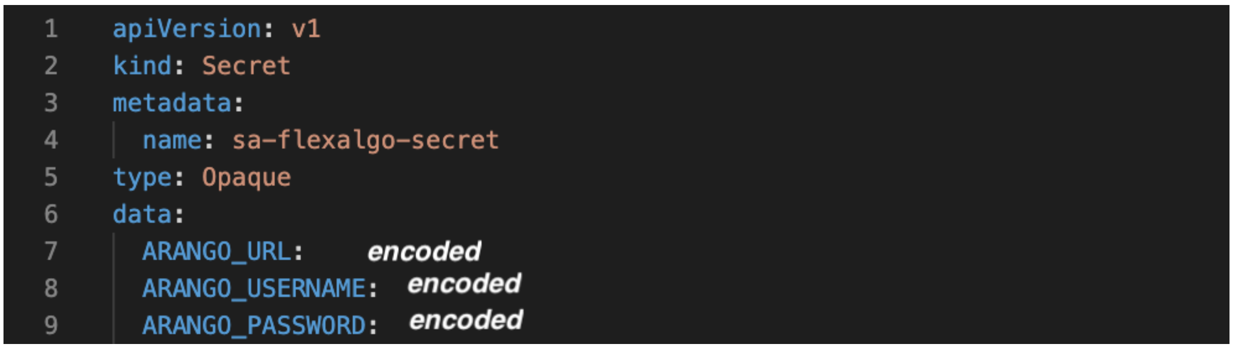
```yaml
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: webserver
  name: webserver-svc
spec:
  selector:
    app: webserver
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 5000
---
```

Figure 11.2.: Deployment File Section Service



```yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webserver-ingress
  labels:
    app: webserver
spec:
  ingressClassName: nginx
  rules:
  - host: sa-sr-flexalgo.stu.network.garden
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: webserver-svc
            port:
              number: 8081
```

Figure 11.3.: Deployment File Section Ingress



```yaml
apiVersion: v1
kind: Secret
metadata:
  name: sa-flexalgo-secret
type: Opaque
data:
  ARANGO_URL:      encoded
  ARANGO_USERNAME:  encoded
  ARANGO_PASSWORD:  encoded
```

Figure 11.4.: Secrets File