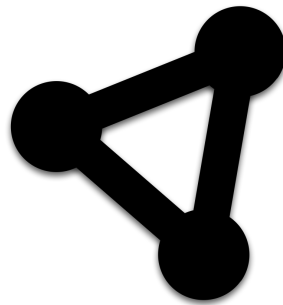


Term Project

# Graph properties of a telecommunication network

Autumn Term 2022



Version: December 22, 2022

**Authors:** Lukas Ribi  
Pascal Christen

**Advisors:** Prof. Laurent Metzger  
Severin Dellsperger

**Project Partner:** Cisco Systems Belgium - François Clad



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

# Abstract

## Objective

Every telecommunication network has a different topology. Additionally, the topology is often complex and unstructured. In the current telecommunications industry, graph properties are not broadly used for network capacity planning or network comparisons. The goal of this thesis is to create a system that conveys the structure of a graph in an understandable way. This is achieved through the visualization of a multitude of graph properties. Included explanations provide context and link to additional sources. Furthermore, it should be possible to obtain network topology data from the Jalapeño API Gateway, which is an ongoing project by the INS.

## Approach

The system is composed of three separate applications that interact with each other. The Data Collector is responsible for reading network topology data, be it from the Jalapeño API Gateway via gRPC or from an alternative source. The read data is persisted in a graph database. Calculations of the graph properties are triggered by requests to the API. These calculations are performed based on the graph that is kept in the graph database, the results are exposed via a REST API. The Frontend consumes data from the API and displays the various graph properties. In order to provide context, explanations are provided for each property.

## Conclusion

A system composed of multiple applications was created that allows network administrators to analyze a provided network based on graph properties. Additional features like querying for cut edges and vertices have also been implemented. The system is developed in a cloud-native way in order to achieve a high scalability and availability. It currently supports the import of network topology data from the Jalapeño API Gateway and files in the GEXF format. The system is designed in an extensible way so that other data sources can be added in the future. It also acts as a platform for future works in the area of network topology analysis.

# Management Summary

## Initial Situation

Telecommunication networks have diverse topologies which are frequently complex and unstructured. In the telecommunications industry, graph properties are currently not widely utilized for network capacity planning or comparisons.

This lack of utilization of graph properties should be addressed by creating a software system that analyzes a provided graph. Network resilience regarding failures can be increased through gained information. It can also serve as a debugging tool and support the planning of network topologies. It assists in planning and maintaining a robust, efficient and highly available network.

## Procedure and Technology

Initially, it was necessary to understand the concepts found in graph theory. Specific graph properties can be complex, and the time for their calculation increases exponentially based on the size of the graph. Thus, technologies that could handle these demands needed to be utilized. Functional and non-functional requirements were defined as part of the conception phase in collaboration with the advisors. The use cases were extended to incorporate new ideas like querying functionality and an import function.

Before the implementation started, technologies and chosen approaches were validated with various prototypes and test scenarios. Risks were mitigated through this approach, and it clarified encountered issues in advance.

During the implementation, three applications were built as part of the system, the API, Data Collector and Frontend. The API is responsible for the graph property calculation and provides the results to the Frontend. Importing the graph from a data source is handled by the Data Collector. A graph database holds the to-be-analyzed graph and provides additional functionality, such as querying capabilities. The Frontend retrieves data from the API and displays it.

## Results

The result of this term thesis is the Graph Analyzer system. It is capable of analyzing networks composed of more than 10'000 nodes. Each graph property is explained and contextualized using a computer network as an example. Additional resources are provided for further information. An overview of the dashboard and query view can be found in figures 1 and 2.

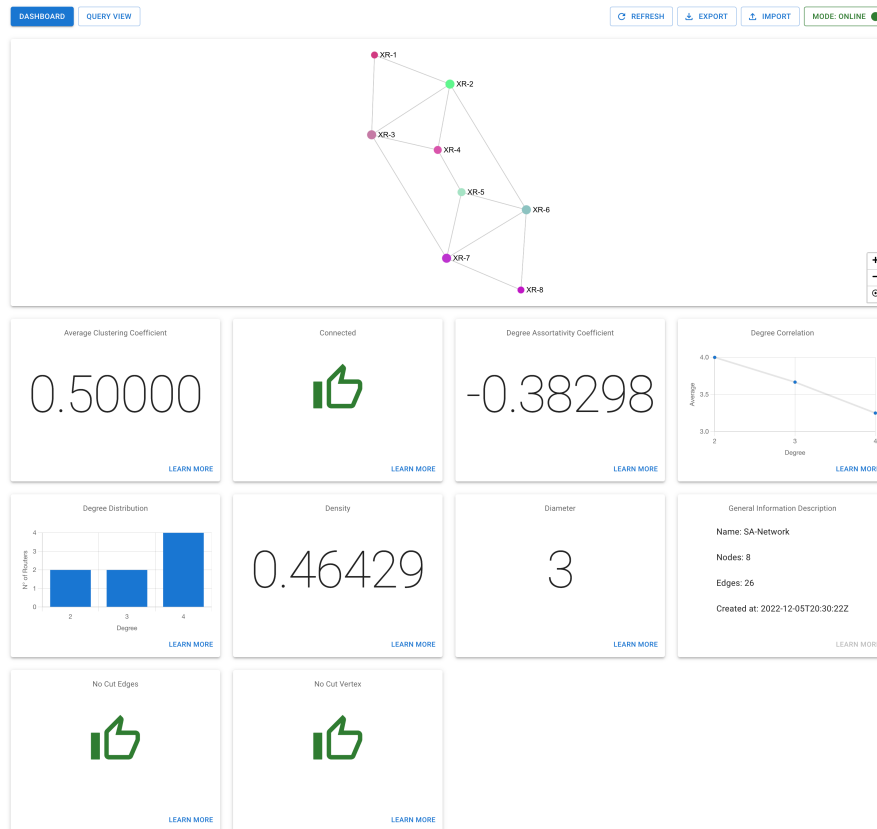


Figure 1: Dashboard



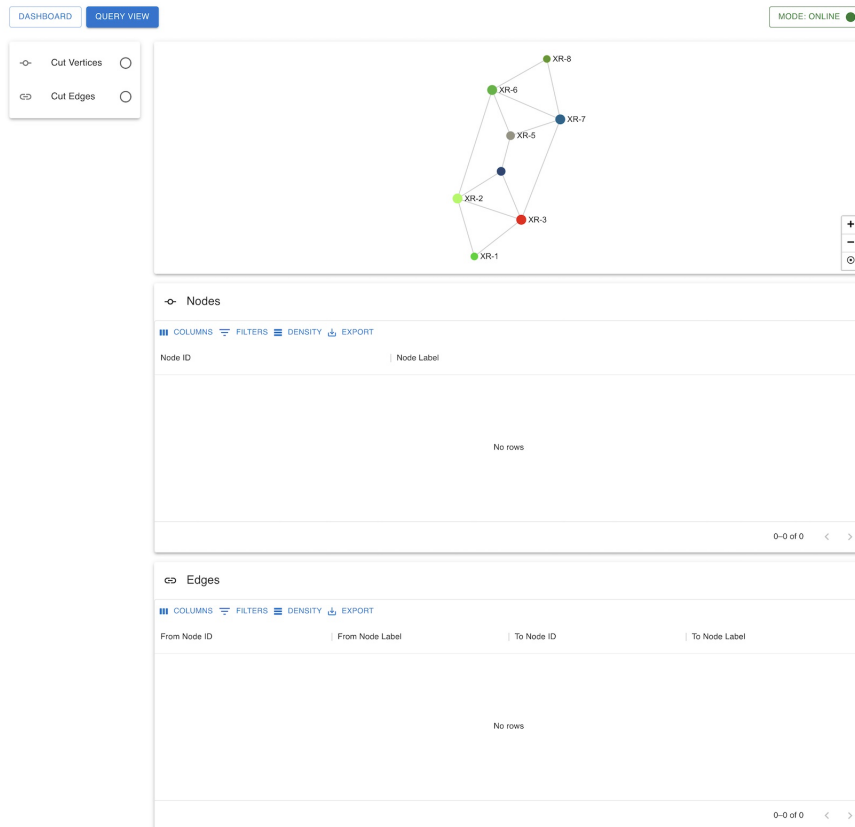


Figure 2: Query View

## Outlook

The current version of the Graph Analyzer offers a solid foundation for future works in the area of network topology analysis. Its structure allows the easy addition of new graph properties and is designed to support additional data sources. A follow-up bachelor thesis will utilize the gained knowledge to research network model generation.

# Contents

|  |             |
|--|-------------|
| <b>Abstract</b>                                | <b>i</b>    |
| <b>Management Summary</b>                      | <b>ii</b>   |
| <b>Glossary</b>                                | <b>x</b>    |
| <b>List of Figures</b>                         | <b>xiii</b> |
| <b>List of Tables</b>                          | <b>xv</b>   |
| <b>I Technical Report</b>                      | <b>1</b>    |
| <b>1 Introduction</b>                          | <b>2</b>    |
| 1.1 Thesis Structure . . . . .                 | 2           |
| 1.1.1 Technical Report . . . . .               | 2           |
| 1.1.2 Project Documentation . . . . .          | 2           |
| 1.2 Graph Property Introduction . . . . .      | 3           |
| 1.2.1 Connectedness . . . . .                  | 3           |
| 1.2.2 Diameter . . . . .                       | 3           |
| 1.2.3 Degree Distribution . . . . .            | 3           |
| 1.2.4 Density . . . . .                        | 3           |
| 1.2.5 Assortativity Coefficient . . . . .      | 3           |
| 1.2.6 Degree Correlation . . . . .             | 4           |
| 1.2.7 Average Clustering Coefficient . . . . . | 4           |
| 1.2.8 Cut Edge . . . . .                       | 4           |
| 1.2.9 Articulation Point . . . . .             | 4           |
| 1.3 Aims and Objectives . . . . .              | 5           |
| 1.3.1 Problem . . . . .                        | 5           |
| 1.3.2 Solution . . . . .                       | 5           |
| <b>2 Requirements</b>                          | <b>6</b>    |
| 2.1 Functional Requirements . . . . .          | 6           |
| 2.1.1 Actor - User . . . . .                   | 6           |
| 2.1.2 Use Cases . . . . .                      | 6           |
| 2.2 Non-Functional Requirements . . . . .      | 12          |
| 2.2.1 Validating NFRs . . . . .                | 13          |
| <b>3 Design Decisions</b>                      | <b>14</b>   |
| 3.1 Basic System Structure . . . . .           | 14          |

|          |  |           |
|----------|--|-----------|
| 3.1.1    | Frontend                                       | 14        |
| 3.1.2    | API  | 14        |
| 3.1.3    | Data Collector                                 | 15        |
| 3.1.4    | Graph Database                                 | 15        |
| 3.2      | Backend Programming Language                   | 15        |
| 3.2.1    | Analyzed Libraries                             | 15        |
| 3.2.2    | Performance Test Setup                         | 16        |
| 3.2.3    | Performance Test                               | 16        |
| 3.2.4    | Conclusion                                     | 17        |
| 3.3      | API Communication                              | 18        |
| 3.3.1    | Technology Comparison                          | 18        |
| 3.3.2    | Conclusion                                     | 18        |
| 3.4      | API Framework                                  | 19        |
| 3.4.1    | Framework Comparison                           | 19        |
| 3.4.2    | Conclusion                                     | 19        |
| 3.5      | API Specification                              | 20        |
| 3.5.1    | Specification Definition                       | 20        |
| 3.6      | Graph Database                                 | 20        |
| 3.6.1    | Analyzed Graph Databases                       | 21        |
| 3.6.2    | Software Licensing                             | 21        |
| 3.6.3    | Test Setup                                     | 21        |
| 3.6.4    | Graph Databases Test                           | 22        |
| 3.6.5    | Conclusion                                     | 22        |
| 3.7      | Graph Visualization                            | 23        |
| 3.7.1    | Previous Work                                  | 23        |
| 3.7.2    | Chosen Library                                 | 23        |
| 3.8      | State Container                                | 23        |
| 3.9      | UI Library                                     | 24        |
| 3.9.1    | Colors   | 24        |
| <b>4</b> | <b>Architecture</b>                            | <b>25</b> |
| 4.1      | Architecture Model                             | 25        |
| 4.1.1    | System Context Diagram                         | 26        |
| 4.1.2    | Container Diagram                              | 27        |
| 4.1.3    | Component Diagram - Single-Page Application    | 29        |
| 4.1.4    | Component Diagram - API Application            | 31        |
| 4.1.5    | Component Diagram - Data Collector Application | 33        |
| 4.2      | UI and UX                                      | 35        |
| 4.3      | Technologies                                   | 39        |
| 4.3.1    | API  | 39        |
| 4.3.2    | Data Collector                                 | 39        |
| 4.3.3    | Frontend                                       | 39        |
| <b>5</b> | <b>Implementation</b>                          | <b>40</b> |
| 5.1      | API  | 40        |

|          |  |           |
|----------|--|-----------|
| 5.1.1    | GDS Access . . . . .                                   | 40        |
| 5.1.2    | General Information Retrieval . . . . .                | 41        |
| 5.1.3    | Density Calculation . . . . .                          | 41        |
| 5.1.4    | Average Clustering Coefficient Calculation . . . . .   | 41        |
| 5.1.5    | Connectedness Calculation . . . . .                    | 41        |
| 5.1.6    | Degree Distribution Calculation . . . . .              | 42        |
| 5.1.7    | Full Graph Retrieval . . . . .                         | 42        |
| 5.1.8    | Degree Assortativity Coefficient Calculation . . . . . | 43        |
| 5.1.9    | Degree Correlation Calculation . . . . .               | 44        |
| 5.1.10   | Diameter Calculation . . . . .                         | 45        |
| 5.1.11   | Cut Edge Calculation . . . . .                         | 46        |
| 5.1.12   | Cut Vertex Calculation . . . . .                       | 48        |
| 5.1.13   | Query Endpoint . . . . .                               | 49        |
| 5.1.14   | Infrastructure Endpoints . . . . .                     | 49        |
| 5.1.15   | Testing . . . . .                                      | 49        |
| 5.1.16   | Usage . . . . .  | 50        |
| 5.2      | Data Collector . . . . .                               | 51        |
| 5.2.1    | Implementation . . . . .                               | 51        |
| 5.2.2    | Jalapeño / JAGW . . . . .                              | 51        |
| 5.2.3    | GEXF . . . . .   | 52        |
| 5.2.4    | Data Structure . . . . .                               | 52        |
| 5.2.5    | GDS Graph . . . . .                                    | 54        |
| 5.2.6    | Testing . . . . .                                      | 54        |
| 5.2.7    | Usage . . . . .  | 54        |
| 5.3      | Frontend . . . . .                                     | 55        |
| 5.3.1    | API Generation . . . . .                               | 55        |
| 5.3.2    | Dashboard - Page . . . . .                             | 55        |
| 5.3.3    | Query View - Page . . . . .                            | 55        |
| 5.3.4    | Components . . . . .                                   | 55        |
| 5.3.5    | Navigation - Component . . . . .                       | 56        |
| 5.3.6    | Refresh - Component . . . . .                          | 56        |
| 5.3.7    | Export - Component . . . . .                           | 56        |
| 5.3.8    | Import - Component . . . . .                           | 57        |
| 5.3.9    | Graph - Component . . . . .                            | 57        |
| 5.3.10   | Graph property - Component . . . . .                   | 58        |
| 5.4      | Neo4j Image . . . . .                                  | 59        |
| 5.4.1    | Implementation . . . . .                               | 59        |
| 5.4.2    | Usage . . . . .  | 59        |
| 5.5      | Helm Chart . . . . .                                   | 60        |
| 5.5.1    | Implementation . . . . .                               | 60        |
| 5.5.2    | Usage . . . . .  | 60        |
| <b>6</b> | <b>Results</b>   | <b>61</b> |
| 6.1      | Deployment . . . . .                                   | 61        |

|           |  |           |
|-----------|--|-----------|
| 6.2       | Use Cases . . . . .                        | 63        |
| 6.3       | NFR Validation . . . . .                   | 64        |
| 6.4       | API . . . . .                              | 67        |
| 6.5       | Frontend . . . . .                         | 68        |
| 6.5.1     | Dashboard . . . . .                        | 68        |
| 6.5.2     | Query View . . . . .                       | 69        |
| 6.5.3     | Graph Visualization . . . . .              | 70        |
| 6.5.4     | Average Clustering Coefficient . . . . .   | 71        |
| 6.5.5     | Connected . . . . .                        | 72        |
| 6.5.6     | Degree Assortativity Coefficient . . . . . | 73        |
| 6.5.7     | Degree Correlation . . . . .               | 74        |
| 6.5.8     | Degree Distribution . . . . .              | 75        |
| 6.5.9     | Density . . . . .                          | 76        |
| 6.5.10    | Diameter . . . . .                         | 77        |
| 6.5.11    | General Information Description . . . . .  | 78        |
| 6.5.12    | No Cut Edges . . . . .                     | 79        |
| 6.5.13    | No Cut Vertex . . . . .                    | 80        |
| <b>7</b>  | <b>Quality Measures</b>                    | <b>81</b> |
| 7.1       | Git Process . . . . .                      | 81        |
| 7.1.1     | Workflow . . . . .                         | 81        |
| 7.1.2     | Code Review . . . . .                      | 81        |
| 7.2       | CI/CD . . . . .                            | 82        |
| 7.2.1     | API and Data Collector . . . . .           | 82        |
| 7.2.2     | Frontend . . . . .                         | 84        |
| 7.3       | Metric Tools . . . . .                     | 85        |
| 7.3.1     | Test Coverage . . . . .                    | 85        |
| 7.3.2     | SonarQube . . . . .                        | 85        |
| <b>8</b>  | <b>Conclusion</b>                          | <b>86</b> |
| 8.1       | Outlook . . . . .                          | 86        |
| 8.2       | Limitations . . . . .                      | 86        |
| <b>II</b> | <b>Project Documentation</b>               | <b>87</b> |
| <b>9</b>  | <b>Project Plan</b>                        | <b>88</b> |
| 9.1       | Project Plan . . . . .                     | 88        |
| 9.1.1     | Development Process . . . . .              | 88        |
| 9.1.2     | Phases . . . . .                           | 88        |
| 9.1.3     | Project Milestones . . . . .               | 89        |
| 9.1.4     | Application Milestones . . . . .           | 89        |
| 9.1.5     | Roadmap . . . . .                          | 90        |
| 9.1.6     | Key Dates and Numbers . . . . .            | 91        |
| 9.2       | Meetings . . . . .                         | 91        |
| 9.2.1     | Status Meetings . . . . .                  | 91        |
| 9.2.2     | Scrum Meetings . . . . .                   | 91        |

|            |  |            |
|------------|--|------------|
| 9.3        | Roles . . . . .                            | 92         |
| 9.3.1      | Details About The Assigned Roles . . . . . | 92         |
| 9.4        | Risk Management . . . . .                  | 93         |
| 9.4.1      | Risks . . . . .                            | 93         |
| 9.4.2      | Risk Matrix . . . . .                      | 95         |
| 9.4.3      | Risk Management and Mitigation . . . . .   | 95         |
| 9.5        | Planning Tools . . . . .                   | 96         |
| 9.5.1      | Issue Tracker . . . . .                    | 96         |
| 9.5.2      | Time Tracker . . . . .                     | 96         |
| <b>III</b> | <b>Appendix</b>                            | <b>97</b>  |
|            | <b>Personal Reports</b>                    | <b>98</b>  |
| 9.6        | Lukas Ribi . . . . .                       | 98         |
| 9.7        | Pascal Christen . . . . .                  | 98         |
|            | <b>Time Tracking Report</b>                | <b>99</b>  |
|            | <b>Meeting Minutes</b>                     | <b>100</b> |
|            | <b>Bibliography</b>                        | <b>113</b> |

# Glossary

**API** Application Programming Interface - A set of rules and protocols that allow different software applications to communicate with each other.

**APOC** Awesome Procedures on Cypher - An add-on library for Neo4j that offers numerous procedures and functions, significantly expanding the functionality of Neo4j.

**AQL** ArangoDB Query Language - A querying language for ArangoDB.

**BSL** Business Source License - A non-open-source software license.

**Central Frontend** A collection of Micro Frontends hosted in a central location.

**CLI** Command-line Interface - A type of user interface that allows users to interact with a computer by typing commands into a text-based console.

**Cypher** A declarative graph querying language used to perform complex data manipulation operations on a graph database such as Neo4j.

**GDS** Graph Data Science Library - A Neo4j graph database plugin that implements common graph algorithms.

**Gin** A web framework written in Golang that features a minimalist design and convenient middleware support.

**GPL** GNU General Public License - An open-source software license.

**Graph database** A graph database is a type of database that stores and organizes data in the form of interconnected nodes. This allows efficient querying and analysis of complex relationships between data.

**Graphology** A robust and versatile graph object implemented in JavaScript and TypeScript.

**gRPC** Google Remote Procedure Calls - A remote procedure call framework.

**IDE** Integrated Development Environment - A program that provides a range of tools and features to facilitate software development.

**INS** Institute for Network and Security - An institute at the Eastern Switzerland University of Applied Sciences.

**k8s** Short form of Kubernetes.

**Kubernetes** An open-source container orchestration system that automates the deployment, scaling, and management of containerized applications.

**MUI** Material UI - A comprehensive library that offers an implementation of Google's Material Design system through a variety of components for React.

**Neo4j** A popular graph database.

**npm** A package manager for the JavaScript and TypeScript programming language that allows developers to install and manage packages.

**OpenAPI** OpenAPI (formerly known as Swagger) is a specification for building APIs that allows developers to describe the structure and behavior of APIs in a standardized way, enabling automated documentation and code generation.

**React** An open-source JavaScript library designed for the development of user interfaces.

**Redux** An open-source JavaScript library that helps the management of application states.

**REST** Representational State Transfer - A software architecture style popular with APIs.

**RSAL** Redis Source Available License - A non-open-source software license.

**RTK** Redux Toolkit - A set of tools that integrates with Redux.

**RTK Query** A tool that facilitates the fetching and caching of data in a web application. It is designed to simplify the process of loading data, reducing the need for manual coding of data fetching and caching logic.

**Swagger** Swagger has since been rebranded as OpenAPI and is now often referred to as OpenAPI 2.0.

**Swagger UI** A web-based interface for visualizing and interacting with the API endpoints of a web application, allowing developers to easily test and debug their APIs. It is part of the Swagger/OpenAPI toolset for building and documenting APIs.



**Testcontainers** A library that allows developers to easily run tests against ephemeral instances of software (in Docker Containers), such as databases, within a test environment, providing a consistent and reliable way to test code that depends on these external instances.

**WebGL** Web Graphics Library - A JavaScript API that enables the rendering of high-performance interactive 3D and 2D graphics within any compatible web browser.

# List of Figures

|      |   |     |
|------|---|-----|
| 1    | Dashboard . . . . .   | iii |
| 2    | Query View . . . . .  | iv  |
| 2.1  | Use Case Diagram . . . . .  | 7   |
| 3.1  | Default Color Palette . . . . .   | 24  |
| 4.1  | C4 Model Level 1 - System Context Diagram . . . . .                         | 26  |
| 4.2  | C4 Model Level 2 - Container Diagram . . . . .                              | 28  |
| 4.3  | C4 Model Level 3 - Component Diagram - Single-Page Application . . . . .    | 30  |
| 4.4  | C4 Model Level 3 - Component Diagram - API Application . . . . .            | 32  |
| 4.5  | C4 Model Level 3 - Component Diagram - Data Collector Application . . . . . | 34  |
| 4.6  | Dashboard (Live) Wireframe . . . . .  | 35  |
| 4.7  | Dashboard (Offline) Wireframe . . . . .                                     | 36  |
| 4.8  | Property Tile Number Wireframe (Variant 1) . . . . .                        | 36  |
| 4.9  | Property Tile Number Wireframe (Variant 2) . . . . .                        | 37  |
| 4.10 | Property Tile Statistics Wireframe . . . . .                                | 37  |
| 4.11 | Property Tile Description Wireframe . . . . .                               | 37  |
| 4.12 | Query View Wireframe . . . . .  | 38  |
| 5.1  | Neo4j Graph Of The Provided INS network . . . . .                           | 52  |
| 5.2  | Neo4j Node Of The Provided INS network . . . . .                            | 53  |
| 5.3  | Neo4j Edge Of The Provided INS network . . . . .                            | 54  |
| 5.4  | Graph Analyzer Frontend - Navigation - Dashboard . . . . .                  | 56  |
| 5.5  | Graph Analyzer Frontend - Navigation - Query View . . . . .                 | 56  |
| 5.6  | Available Neo4j Image Tags . . . . .  | 59  |
| 6.1  | Current Deployment . . . . .  | 62  |
| 6.2  | Responsive Frontend On A Smaller Viewport . . . . .                         | 66  |
| 6.3  | Swagger UI . . . . .  | 67  |
| 6.4  | Graph Analyzer Frontend - Dashboard . . . . .                               | 68  |
| 6.5  | Graph Analyzer Frontend - Query View . . . . .                              | 69  |
| 6.6  | Graph Analyzer Frontend - Graph Visualization . . . . .                     | 70  |
| 6.7  | Graph Analyzer Frontend - Graph Visualization With Selected Node . . . . .  | 70  |
| 6.8  | Graph Analyzer Frontend - Average Clustering Coefficient Tile . . . . .     | 71  |

|      |   |    |
|------|---|----|
| 6.9  | Graph Analyzer Frontend - Average Clustering Coefficient Description . .  | 71 |
| 6.10 | Graph Analyzer Frontend - Connected Tile . . . . .                        | 72 |
| 6.11 | Graph Analyzer Frontend - Connected Description . . . . .                 | 72 |
| 6.12 | Graph Analyzer Frontend - Degree Assortativity Coefficient Tile . . . . . | 73 |
| 6.13 | Graph Analyzer Frontend - Degree Assortativity Coefficient Description .  | 73 |
| 6.14 | Graph Analyzer Frontend - Degree Correlation Tile . . . . .               | 74 |
| 6.15 | Graph Analyzer Frontend - Degree Correlation Description . . . . .        | 74 |
| 6.16 | Graph Analyzer Frontend - Degree Distribution Tile . . . . .              | 75 |
| 6.17 | Graph Analyzer Frontend - Degree Distribution Description . . . . .       | 75 |
| 6.18 | Graph Analyzer Frontend - Density Tile . . . . .                          | 76 |
| 6.19 | Graph Analyzer Frontend - Density Description . . . . .                   | 76 |
| 6.20 | Graph Analyzer Frontend - Diameter Tile . . . . .                         | 77 |
| 6.21 | Graph Analyzer Frontend - Diameter Description . . . . .                  | 77 |
| 6.22 | Graph Analyzer Frontend - General Information Tile . . . . .              | 78 |
| 6.23 | Graph Analyzer Frontend - No Cut Edges Tile . . . . .                     | 79 |
| 6.24 | Graph Analyzer Frontend - No Cut Edges Description . . . . .              | 79 |
| 6.25 | Graph Analyzer Frontend - No Cut Vertex Tile . . . . .                    | 80 |
| 6.26 | Graph Analyzer Frontend - No Cut Vertex Description . . . . .             | 80 |
|      |   |    |
| 7.1  | API Pipeline For Main Branch Tags . . . . .                               | 82 |
| 7.2  | API Merge Request With Artifacts . . . . .                                | 83 |
| 7.3  | Frontend Pipeline For Main Branch Tags . . . . .                          | 84 |
| 7.4  | SonarQube Report For The Data Collector And API . . . . .                 | 85 |
|      |   |    |
| 9.1  | Roadmap . . . . .   | 90 |
| 9.2  | RUP Phases With Project Milestones And Sprints . . . . .                  | 90 |
| 9.3  | Risk Matrix . . . . .   | 95 |
|      |   |    |
| 9.4  | Time Tracking Report . . . . .  | 99 |

# List of Tables

|     |  |    |
|-----|--|----|
| 2.1 | How NFRs Are Validated . . . . .             | 13 |
| 3.1 | Performance Test With 100 Nodes . . . . .    | 16 |
| 3.2 | Performance Test With 1'000 Nodes . . . . .  | 17 |
| 3.3 | Performance Test With 10'000 Nodes . . . . . | 17 |
| 3.4 | Technology Comparison . . . . .              | 18 |
| 3.5 | Framework Comparison . . . . .               | 19 |
| 3.6 | Graph Databases Test . . . . .               | 22 |
| 6.1 | Fulfillment Of Use Cases . . . . .           | 63 |
| 6.2 | NFR Validation . . . . .                     | 64 |
| 9.1 | Project Milestoness . . . . .                | 89 |
| 9.2 | Application Milestones . . . . .             | 89 |
| 9.3 | Scrum Meetings . . . . .                     | 91 |
| 9.4 | Scrum Roles . . . . .                        | 92 |
| 9.5 | Risk Management . . . . .                    | 94 |
| 9.6 | Risk Mitigation . . . . .                    | 96 |

# Listings

|      |   |    |
|------|---|----|
| 5.1  | GDS Latest Graph Name Retrieval . . . . .                 | 40 |
| 5.2  | GDS General Information Retrieval . . . . .               | 41 |
| 5.3  | GDS Density Retrieval . . . . .                           | 41 |
| 5.4  | GDS Average Clustering Coefficient Retrieval . . . . .    | 41 |
| 5.5  | GDS Component Count Retrieval . . . . .                   | 41 |
| 5.6  | GDS Degree Distribution Retrieval . . . . .               | 42 |
| 5.7  | Cypher Full Graph Retrieval . . . . .                     | 42 |
| 5.8  | Go Degree Assortativity Coefficient Calculation . . . . . | 43 |
| 5.9  | Go Degree Correlation Calculation . . . . .               | 44 |
| 5.10 | Go Diameter Calculation . . . . .                         | 45 |
| 5.11 | Go Bridge Detection . . . . .                             | 46 |
| 5.12 | Go Articulation Points Detection . . . . .                | 48 |
| 5.13 | GEXF Structure . . . . .                                  | 52 |
| 5.14 | Invalidating RTK Query Tags . . . . .                     | 56 |
| 5.15 | Helm Deployment Output . . . . .                          | 60 |
| 6.1  | Apache Bench Output . . . . .                             | 65 |
| 6.2  | Data Collector 10'000 Nodes Import Output . . . . .       | 65 |
| 6.3  | Self Healing Test Output . . . . .                        | 65 |

**Part I**

**Technical Report**

# Chapter 1

## Introduction

In this chapter, an introduction to the thesis is given.

The thesis is written under the assumption that the reader has existing knowledge of basic computer science topics such as computer networks, algorithms, application architecture and distributed systems.

### 1.1 Thesis Structure

The following two sections will describe how the thesis is structured. Essential concepts and terminology are also introduced. Sources are given if necessary to provide additional information.

#### 1.1.1 Technical Report

The Technical Report is divided into eight chapters. In the Introduction, the given problem description and goal of the thesis are described. The Requirements chapter lists the capabilities and criteria that the final result of the thesis must fulfill. Decisions that were made before and during the implementation can be found in the Design Decisions chapter. The chosen architecture is located in the following Architecture chapter. The Implementation chapter contains details about how the final result was realized. The final three Results, Quality Measures and Conclusions chapters analyze the final result in retrospect and how code quality is ensured. Additionally, they contain an outlook and thoughts about continuative works that could result from this thesis.

#### 1.1.2 Project Documentation

The Project Documentation provides details on how the results of the thesis were achieved. Project management-related information can be found in this part of the thesis.

## 1.2 Graph Property Introduction

Each implemented graph property will get a short introduction in this section. Graphs in this thesis are undirected and unweighted if not specified otherwise. A graph consists of the following properties:

- Graph
- Node
- Edge

To avoid confusion about the various terminologies used in graph theory, these terms will be used hence.

Nodes that connect to a node via an edge are called the neighbors of the node. The number of edges that connect to a node represents the degree that a node has [1].

### 1.2.1 Connectedness

A connected graph describes if there is a path, with which a node can reach every other node of the graph. If this is not the case, the graph is disconnected [1].

### 1.2.2 Diameter

The diameter is the longest shortest path in a network. All shortest paths between all pairs of nodes need to be calculated and the longest of them will be the so-called diameter. The number of edges that compose the diameter act as its value [1].

### 1.2.3 Degree Distribution

The degree distribution shows the number of nodes that have a certain degree.

### 1.2.4 Density

Density is the ratio between the present edges and the maximum possible edges in a graph. Therefore, a complete graph (every node connects to every other node) has a value of 1, whereas the value for a large graph with only a few edges tends to be closer to zero. Most large real-world networks often have a very low density [1].

### 1.2.5 Assortativity Coefficient

The degree assortativity coefficient is defined by the Pearson correlation between the degrees of pairs of connected nodes. How nodes of one degree connect to nodes of another degree is known as assortativity. It represents the Degree-degree correlation. A positive value indicates that high degree nodes tend to connect to nodes with high degree and low degree nodes to nodes with low degree. If the value is negative, it indicates that



high degree nodes tend to connect to nodes with low degree and low degree nodes to nodes with high degree. A value near zero indicates that there is no tendency on how nodes connect, they connect randomly. Computer networks tend to have a negative degree assortativity coefficient [1].

### **1.2.6 Degree Correlation**

The average degree connectivity correlation is the average nearest neighbor degree of nodes with degree  $k$ . It represents the same information as the degree assortativity coefficient but in a function form. An increasing function indicates the same as a negative degree assortativity coefficient, and a decreasing function a positive one. If the function is constant, it can be interpreted similarly to a degree assortativity coefficient near zero [1].

### **1.2.7 Average Clustering Coefficient**

The local clustering coefficient of a node represents how close its neighboring nodes are to being a complete graph. It represents the average probability that two neighbors of a node are themselves, neighbors. The average clustering coefficient is the average of all local clustering coefficients in the graph. In practice, a high average clustering coefficient indicates that nodes tend to form clusters [1].

### **1.2.8 Cut Edge**

A cut edge is a single edge in the graph whose removal would split the graph [2].

### **1.2.9 Articulation Point**

An articulation point is a single node in the graph whose removal would split the graph [2], [3].

## 1.3 Aims and Objectives

This section introduces the given problem description of the thesis and the solution to it.

### 1.3.1 Problem

Every telecommunication network has a different topology. In addition, the topology is often complex and unstructured. In the telecommunication industry, graph properties so far are not broadly used for network capacity planning or network comparisons.

As a first step, these graph properties need to be calculated and visualized.

### 1.3.2 Solution

Cisco has developed a software called Jalapeño. It is an infrastructure platform for network services. The collected network data is stored in a time series database (for telemetry data) or graph database. To access the data collected by Jalapeño, the INS developed the Jalapeño API Gateway. It provides a unified interface that supports API requests or subscriptions via gRPC to access the underlying data.

This thesis aims to develop a system that can analyze a provided network and calculate various graph properties, which will be visually represented with accompanying explanations. The system consists of two parts. A backend processes the data received from the Jalapeño API Gateway and calculates the graph properties, while a frontend displays and explains the calculated properties. The network data should be automatically updated via gRPC subscriptions. In addition, the system should stay performant even when analyzing large networks. As an optional feature, the system should also have the ability to search for specific nodes or edges based on particular graph properties, allowing for more detailed analysis and debugging of the network.

## Chapter 2

# Requirements

This chapter describes the various requirements that the final system of the thesis needs to fulfill.

### 2.1 Functional Requirements

#### 2.1.1 Actor - User

The final system has only one actor, a regular user. His goal is to get insights about a network via its graph properties. They should be displayed on a website, understandably and with useful explanations.

#### 2.1.2 Use Cases

The use cases shown in figure 2.1 are specified in a brief form.

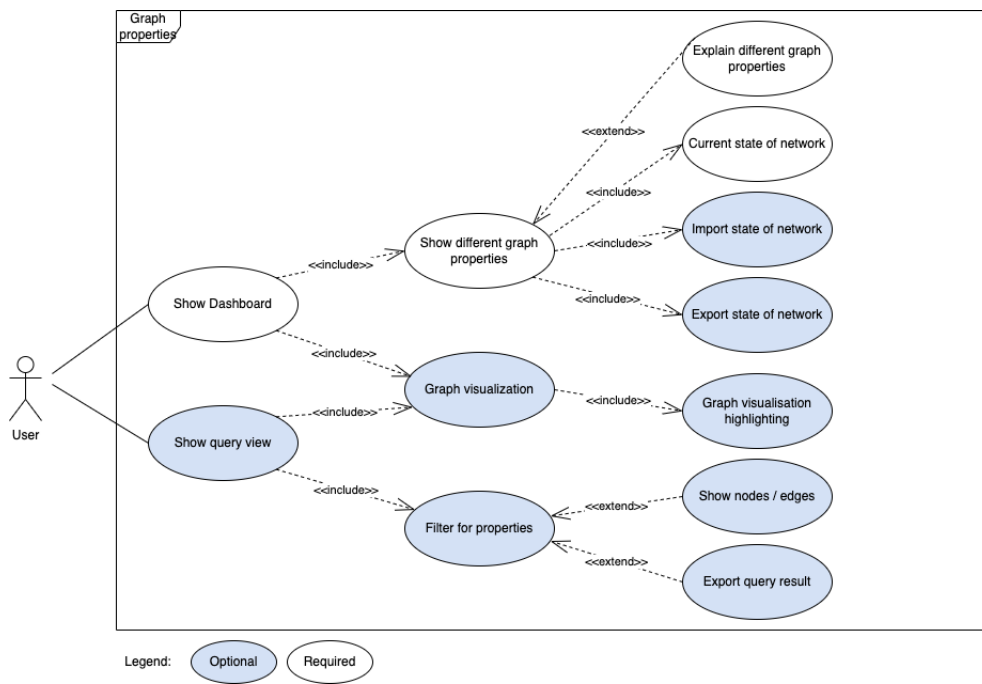


Figure 2.1: Use Case Diagram

***FR-01***

**Use case:** Show dashboard

**Brief:** User accesses the application directly or via the Central Frontend and is shown a dashboard that shows various graph properties of the network. This can be seen as the entry point of the application.

**Electivity:** Required

---

***FR-01.a***

**Use case:** Dashboard - Show number of nodes

**Brief:** The number of nodes should be shown on the dashboard.

**Electivity:** Required

---

***FR-01.b***

**Use case:** Dashboard - Show number of edges

**Brief:** The number of edges should be shown on the dashboard.

**Electivity:** Required

---

***FR-01.c***

**Use case:** Dashboard - Show degree correlation

**Brief:** The degree correlation should be shown on the dashboard.

**Electivity:** Required

---

***FR-01.d***

**Use case:** Dashboard - Show assortativity coefficient

**Brief:** The assortativity coefficient should be shown on the dashboard.

**Electivity:** Required

---

***FR-01.e***

**Use case:** Dashboard - Show degree distribution

**Brief:** The degree distribution should be shown on the dashboard.

**Electivity:** Required

---

***FR-01.f***

**Use case:** Dashboard - Show cut edge status  
**Brief:** The existence of cut edges should be shown on the dashboard.  
**Electivity:** Required

---

***FR-01.g***

**Use case:** Dashboard - Show network diameter  
**Brief:** The network diameter should be shown on the dashboard.  
**Electivity:** Required

---

***FR-01.h***

**Use case:** Dashboard - Show average clustering coefficient  
**Brief:** The average clustering coefficient should be shown on the dashboard.  
**Electivity:** Required

---

***FR-01.i***

**Use case:** Dashboard - Show articulation point status  
**Brief:** The existence of articulation points should be shown on the dashboard.  
**Electivity:** Optional

---

***FR-01.j***

**Use case:** Dashboard - Show density  
**Brief:** The density of the graph should be shown on the dashboard.  
**Electivity:** Optional

---

***FR-01.k***

**Use case:** Dashboard - Show graph connectivity  
**Brief:** The connectedness of the graph (connected, disconnected) should be shown on the dashboard.  
**Electivity:** Optional

---

***FR-01.l***

**Use case:** Dashboard - Show graph communities  
**Brief:** The communities of the graph should be shown on the dashboard.  
**Electivity:** Optional

---

#### ***FR-01.m***

|                    |  |
|--------------------|--|
| <b>Use case:</b>   | Dashboard - Current state of network   |
| <b>Brief:</b>      | The various shown graph properties on the dashboard reflect the current state of the network. The graph properties get updated based on network changes. |
| <b>Electivity:</b> | Required   |

---

#### ***FR-01.n***

|                    |   |
|--------------------|---|
| <b>Use case:</b>   | Dashboard - Export graph properties                             |
| <b>Brief:</b>      | The current state of the dashboard can be exported into a file. |
| <b>Electivity:</b> | Optional  |

---

#### ***FR-01.o***

|                    |   |
|--------------------|---|
| <b>Use case:</b>   | Dashboard - File-based state of the network   |
| <b>Brief:</b>      | The various shown graph properties on the dashboard reflect the data contained in a previously exported file when imported. |
| <b>Electivity:</b> | Optional  |

---

#### ***FR-01.p***

|                    |   |
|--------------------|---|
| <b>Use case:</b>   | Dashboard - Get background information  |
| <b>Brief:</b>      | For each graph property on the dashboard an option should be present to get more information about it. It should explain what the graph property represents and give context to its relevance using a computer network as an example. |
| <b>Electivity:</b> | Required  |

---

#### ***FR-01.q***

|                    |  |
|--------------------|--|
| <b>Use case:</b>   | Dashboard - Graph visualization  |
| <b>Brief:</b>      | The underlying network on which all graph properties on the dashboard are based should be visualized. Its intended use is not as a detailed tool for further analysis, but to give the user some visual context for graph properties that are only represented as numbers or text. |
| <b>Electivity:</b> | Optional   |

---

#### ***FR-02***

|                    |  |
|--------------------|--|
| <b>Use case:</b>   | Show query view  |
| <b>Brief:</b>      | The user should be able to switch to a second page from the dashboard, this separate page provides the user with options to query the network for specific graph properties. |
| <b>Electivity:</b> | Optional   |

---

***FR-02.a***

**Use case:** Query view - Filter for properties  
**Brief:** Preselected query options are provided to the user with which he can execute a query. Only graph properties that are not based on the network as a whole can be queried. The goal is to have some simple options that can be used as the base for future extensions.  
**Electivity:** Optional

---

***FR-02.b***

**Use case:** Query view - Show nodes/edges  
**Brief:** The nodes and edges that match the query are listed as the result of the query. The user can read more details from this listing. For example, to identify the affected nodes in another tool.  
**Electivity:** Optional

---

***FR-02.c***

**Use case:** Query view - Export query result  
**Brief:** The result list of the query can be exported by the user into a common file format like CSV.  
**Electivity:** Optional

---



## 2.2 Non-Functional Requirements

These are the eight defined non-functional requirements.

| <i>NFR-01</i>      |  |
|--------------------|--|
| <b>Category:</b>   | Performance  |
| <b>Brief:</b>      | The solution must be able to support 100 active users. |
| <b>Electivity:</b> | Required   |

---

| <i>NFR-02</i>      |  |
|--------------------|--|
| <b>Category:</b>   | Performance  |
| <b>Brief:</b>      | The solution must be able to handle at least 10'000 nodes in a sparse network. |
| <b>Electivity:</b> | Required   |

---

| <i>NFR-03</i>      |  |
|--------------------|--|
| <b>Category:</b>   | Availability   |
| <b>Brief:</b>      | In case of a software fault the system is self-healing without manual interaction within 30 seconds. |
| <b>Electivity:</b> | Required   |

---

| <i>NFR-04</i>      |  |
|--------------------|--|
| <b>Category:</b>   | Usability  |
| <b>Brief:</b>      | The solution must validate the input data of the user. |
| <b>Electivity:</b> | Required   |

---

| <i>NFR-05</i>      |  |
|--------------------|--|
| <b>Category:</b>   | Maintainability                                |
| <b>Brief:</b>      | The solution must be developed cloud-natively. |
| <b>Electivity:</b> | Required                                       |

---

| <i>NFR-06</i>      |  |
|--------------------|--|
| <b>Category:</b>   | Maintainability                                      |
| <b>Brief:</b>      | The solution must be tested and built through CI/CD. |
| <b>Electivity:</b> | Required   |

---

### *NFR-07*

**Category:** Usability  
**Brief:** The solution should be responsive (Mobile/Web).  
**Electivity:** Optional

---

### *NFR-08*

**Category:** Maintainability  
**Brief:** The chosen graph visualization library should be exchangeable.  
**Electivity:** Optional

---

## 2.2.1 Validating NFRs

Table 2.1 shows how and when the NFRs are validated.

| <b>ID</b> | <b>How</b>  | <b>When</b>                         |
|-----------|---|-------------------------------------|
| NFR-01    | Perform a load test on the API with at least 100 concurrent connections.  | Before releasing the Alpha version. |
| NFR-02    | Mocking a large graph with 10'000 nodes, which has the properties of a sparse graph.                                      | Before releasing the Alpha version. |
| NFR-03    | Kill a container and measure the time until a new container has been spun up.   | Before releasing the Alpha version. |
| NFR-04    | Test the system's resistance to invalid user input and ensure that it displays helpful error messages.                    | Before releasing the Beta version.  |
| NFR-05    | With the help of available cloud-native concepts, it is checked whether the architecture meets the requirements.          | Before releasing the Alpha version. |
| NFR-06    | It is checked whether a sufficient CI/CD pipeline exists for each part of the system.                                     | Before releasing the Alpha version. |
| NFR-07    | Browser tools are used to check whether the website is responsive and can be used on devices with different screen sizes. | Before releasing the Beta version.  |
| NFR-08    | Check if the library has been encapsulated and if it would be possible to replace it.                                     | Before releasing the Beta version.  |

Table 2.1: How NFRs Are Validated

# Chapter 3

## Design Decisions

This chapter contains design decisions that were made before and during the implementation.

### 3.1 Basic System Structure

The rough structure of the backend and frontend that compose the system was already included in the thesis description. Based on optional requirements like the query view, the backend was split into further components to achieve higher flexibility. Importing graphs from data sources other than Jalapeño into the system should be possible. To achieve this, two additional components were introduced, the graph database and Data Collector.

#### 3.1.1 Frontend

The role of the Frontend did not change compared to how it was initially envisioned. It interacts with the backend, specifically the API, and displays the retrieved data for the user. In the beginning, the requirement was to embed the Frontend in the Central Frontend of the INS as a Micro Frontend. The Central Frontend is based on React<sup>1</sup>, so it made sense to use React as a Frontend framework as there was already some experience present regarding React. This requirement was later abandoned because it was determined that the current version of the Central Frontend is not suitable for production usage.

#### 3.1.2 API

The API encapsulates a lot of the ideas that were initially referred to as just the backend. Graph property calculations and delivering the results to the Frontend are the core responsibilities of the API.

---

<sup>1</sup><https://reactjs.org>

### 3.1.3 Data Collector

The Data Collector is an independent application of the backend, that is solely responsible for retrieving graphs from different data sources. All logic concerning the import from various data sources is contained within it. The graph data is transformed into a defined structure and saved in the graph database.

### 3.1.4 Graph Database

The graph database acts as the connecting component between the API and Data Collector. It contains the graph data that the Data Collector has collected. The API can retrieve the graph from it to calculate graph properties. Additionally, it provides certain graph property calculations out of the box and supports querying the graph.

## 3.2 Backend Programming Language

The initial thesis description required that the backend would be programmed with either Python or Go.

To determine the to-be-used programming language, performance comparisons between popular graph libraries were conducted for each language. While many of the algorithms in the final system can be executed directly on a graph database, the results of this analysis are still important as they indicate the performance that can be expected from the language itself. In addition, the results serve as a backup option if the implementation in the graph database is insufficient for a particular use case and needs to be implemented in a customized manner.

### 3.2.1 Analyzed Libraries

The analysis included the following libraries:

- Python
  - NetworkX (Python Module) <sup>2</sup>
  - graph-tool (C++ Module) <sup>3</sup>
- Go
  - Gonum <sup>4</sup>
  - thcyron/graphs <sup>5</sup>

The thcyron/graphs library had too few algorithms implemented, so it was not analyzed further.

---

<sup>2</sup><https://networkx.org>

<sup>3</sup><https://graph-tool.skewed.de>

<sup>4</sup><https://www.gonum.org>

<sup>5</sup><https://pkg.go.dev/github.com/thcyron/graphs>

### 3.2.2 Performance Test Setup

Three graphs using the NetworkX Internet Autonomous System Network generator <sup>6</sup> were generated and used for the performance measurements.

- 100 nodes and 141 Edges
- 1'000 nodes and 1'501 Edges
- 10'000 nodes and 26'361 Edges

The time for the following graph properties was measured for the following calculations.

- All shortest paths from Node 0
- BFS from Node 0
- All shortest paths in the graph
- Betweenness
- Minimum spanning tree

### 3.2.3 Performance Test

The test setup was a MacBook with an Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz processor.

100 Nodes:

| Property           | NetworkX | graph-tool | gonum      |
|--------------------|----------|------------|------------|
| node shortest path | 0.0001 s | 0.0019 s   | 0.0001 s   |
| bfs                | 0.0002 s | 0.0013 s   | 0.0001 s   |
| all shortest paths | 0.0149 s | no support | 0.0124 s   |
| betweenness        | 0.0294 s | 0.0023 s   | 0.0102 s   |
| min span tree      | 0.0013 s | 0.0007 s   | no support |

Table 3.1: Performance Test With 100 Nodes

---

<sup>6</sup>[https://networkx.org/documentation/stable/reference/generated/networkx.generators.internet\\_as\\_graphs.random\\_internet\\_as\\_graph.html](https://networkx.org/documentation/stable/reference/generated/networkx.generators.internet_as_graphs.random_internet_as_graph.html)

1'000 Nodes:

| <b>Property</b>    | <b>NetworkX</b> | <b>graph-tool</b> | <b>gonum</b> |
|--------------------|-----------------|-------------------|--------------|
| node shortest path | 0.0021 s        | 0.0006 s          | 0.0009 s     |
| bfs                | 0.0048 s        | 0.0014 s          | 0.0013 s     |
| all shortest paths | 1.5610 s        | no support        | 0.9815 s     |
| betweenness        | 2.2551 s        | 0.0560 s          | 0.9100 s     |
| min span tree      | 0.0099 s        | 0.0005 s          | no support   |

Table 3.2: Performance Test With 1'000 Nodes

10'000 Nodes:

| <b>Property</b>    | <b>NetworkX</b> | <b>graph-tool</b> | <b>gonum</b> |
|--------------------|-----------------|-------------------|--------------|
| node shortest path | 0.0207 s        | 0.0028 s          | 0.0171 s     |
| bfs                | 0.0341 s        | 0.0113 s          | 0.0088 s     |
| all shortest paths | 425.9755 s      | no support        | 238.7183 s   |
| betweenness        | 520.0862 s      | 10.4052 s         | 182.4155 s   |
| min span tree      | 0.1823 s        | 0.0017 s          | no support   |

Table 3.3: Performance Test With 10'000 Nodes

### 3.2.4 Conclusion

Based on the results in tables 3.1 to 3.3, it was decided to use Golang because of the following reasons:

- NetworkX is too slow to handle the performance requirements.
- graph-tool is fast enough (sometimes even the fastest) but the developer experience of the library was not great. The installation and integration are not straightforward (manual compilation on some platforms). The IDE integration is also not as good as expected.
- Golang combined with Gonum combines the best parts of both Python libraries. Speed, IDE integration, code integration and library functionality were overall the best.

### 3.3 API Communication

To handle incoming requests from the Frontend to the API, two technologies were evaluated:

- REST
- GraphQL

gRPC was not considered because grpc-web had several limitations, such as experimental TypeScript support, which prevented its use in the Frontend [4]. In addition, an additional proxy service would be required to facilitate the communication.

#### 3.3.1 Technology Comparison

| Property     | REST                           | GraphQL                  |
|--------------|--------------------------------|--------------------------|
| Architecture | Server-driven                  | Client-driven            |
| State        | Stateless                      | Stateless                |
| HTTP Methods | GET, POST, UPDATE, PUT, DELETE | POST                     |
| Organization | Multiple Endpoints             | Single Endpoint (Schema) |
| Fetching     | Often under/over-fetching      | Exact response           |
| Subscription | Not possible                   | WebSocket                |

Table 3.4: Technology Comparison

#### 3.3.2 Conclusion

Both technologies have their advantages and disadvantages. Based on the results in table 3.4, REST was chosen for the backend API for the following reasons:

- Simple and easily understandable
- Future maintenance (Currently, the INS does not use GraphQL)
- Prior experience with REST

## 3.4 API Framework

To implement the REST API the following Golang REST API frameworks were analyzed:

- Echo <sup>7</sup>
- Gin <sup>8</sup>
- Beego <sup>9</sup>
- Fiber <sup>10</sup>

### 3.4.1 Framework Comparison

|                        | <b>Echo</b> | <b>Gin</b> | <b>Beego</b>                                | <b>Fiber</b>  |
|------------------------|-------------|------------|---|---|
| <b>Github Stars</b>    | 24'000      | 63'700     | 29'000                                      | 22'800  |
| <b>License</b>         | MIT         | MIT        | Apache License 2.0                          | MIT   |
| <b>Maintained</b>      | Active      | Active     | Active                                      | Active  |
| <b>Hot Code Reload</b> | With Air    | Yes        | Yes   | Yes   |
| <b>Based on</b>        | net/http    | net/http   | net/http                                    | fasthttp  |
| <b>Limitations</b>     |             |            | Integrated MVC - may not be suitable for us | Not possible to use net/http interfaces (go-swagger, ...) and the use of unsafe can conflict with new Golang versions [5] |

Table 3.5: Framework Comparison

### 3.4.2 Conclusion

Gin was chosen as the backend REST API framework because of the following reasons based on the results in table 3.5:

- Well regarded in the Golang community
- Many middlewares available
- Existing experience with Gin from previous projects

<sup>7</sup><https://github.com/labstack/echo>

<sup>8</sup><https://github.com/gin-gonic/gin>

<sup>9</sup><https://github.com/beego/beego>

<sup>10</sup><https://github.com/gofiber/fiber>



## 3.5 API Specification

Since Gin was chosen for the REST API in section 3.4, a descriptive API specification was needed. An API specification can be defined through OpenAPI <sup>11</sup>. The OpenAPI specification allows for the description of the structure of an API in a manner that is easily comprehensible to both humans and machines. OpenAPI makes it easier to create, maintain, and use the API. Additionally, OpenAPI can be used to generate client libraries and other artifacts that can be used to easily integrate an API.

### 3.5.1 Specification Definition

The swag <sup>12</sup> CLI combined with the gin-swagger <sup>13</sup> middleware is suitable for Gin. It automatically generates the OpenAPI 2.0 specification by working with the Declarative Comments Format during development [6]. The latest version would be version 3.1 of the OpenAPI specification, but since there is no easily implementable package for Gin, it was chosen regardless.

## 3.6 Graph Database

As one of the optional requirements of the system is a query view, a simple graph analysis to calculate graph properties is not enough. A query engine is needed to support this use case. Jalapeño already keeps the network in a graph database (ArangoDB), and the Jalapeño API gateway keeps a cached version of it, which is accessed.

It was decided to implement a separate graph database as part of the system based on the following reasons:

- Using the already existing ArangoDB, which is part of Jalapeño, is not an option as system boundaries would be crossed and a dependency on Jalapeño be created.
- The storage of the graph by the system enables it to connect to other data sources.
- The goal was not to create a graph query engine, existing solutions in form of graph databases can be used.

---

<sup>11</sup><https://www.openapis.org>

<sup>12</sup><https://github.com/swaggo/swag>

<sup>13</sup><https://github.com/swaggo/gin-swagger>

### 3.6.1 Analyzed Graph Databases

The analysis included the following graph databases:

- ArangoDB <sup>14</sup>
- Memgraph <sup>15</sup>
- Neo4j <sup>16</sup>
- RedisGraph <sup>17</sup>

### 3.6.2 Software Licensing

Memgraph and RedisGraph were eliminated due to their problematic licenses. The BSL 1.1 and RSAL licenses are too restrictive with their field of use restrictions [7], [8]. ArangoDB uses the Apache 2.0 license, which is the least problematic in the selection [9]. Neo4j uses the often controversial GPLv3 license <sup>18</sup>. After looking into it, Neo4j was chosen in the end. The license should not interfere as long as Neo4j is not bundled into one of the container images [10].

### 3.6.3 Test Setup

Two tests were run for the selected graph databases and the time measured.

- Betweenness centrality
  - Calculating the betweenness centrality gives us an indication on how well the graph database supports graph analysis internally.
- Get all nodes and edges
  - Querying the entirety of the graph shows us the expected impact of handling graph analysis externally. The whole graph needs to be extracted if the internal graph analysis capabilities do not cover certain graph properties or are not performant enough.

The tests were done based on the previously used 1'000 node NetworkX AS graph in section 3.2.2.

---

<sup>14</sup><https://github.com/arangodb/arangodb>

<sup>15</sup><https://github.com/memgraph/memgraph>

<sup>16</sup><https://github.com/neo4j/neo4j>

<sup>17</sup><https://github.com/RedisGraph/RedisGraph>

<sup>18</sup><https://github.com/neo4j/neo4j/blob/5.1/LICENSE.txt>

### 3.6.4 Graph Databases Test

The test setup was a MacBook with an Apple M1 Max 10 core processor.

| Property                | ArangoDB               | Neo4j                 |
|-------------------------|------------------------|-----------------------|
| Betweenness centrality  | 23.780 s <sup>19</sup> | 0.154 s <sup>20</sup> |
| Get all nodes and edges | 3.901 s <sup>21</sup>  | 0.007 s               |

Table 3.6: Graph Databases Test

### 3.6.5 Conclusion

Based on the results in table 3.6, Neo4j was chosen due to the following reasons:

- Better overall performance in general.
- Easier query language because of familiarity with Cypher compared to AQL.
- Better internal graph analysis capabilities through the GDS <sup>22</sup> library.

---

<sup>19</sup>The ArangoDB general graph <https://www.arangodb.com/docs/stable/graphs-general-graphs.html> module was used.

<sup>20</sup>The Neo4j GDS library was used.

<sup>21</sup>The collections for nodes and edges were individually retrieved.

<sup>22</sup><https://github.com/neo4j/graph-data-science>

## 3.7 Graph Visualization

To visualize the graph in the Frontend, a graph visualization library that had the following properties was needed:

- Performant
  - The graphs, that need to be rendered out, can be composed out of thousands of nodes and edges.
- Allow different layouts e.g. force layout.
- Allow highlighting of nodes and edges.

### 3.7.1 Previous Work

No extensive comparisons between different libraries were made, as such research has already been done in the "Central Frontend for Segment Routing Applications" bachelor thesis [11]. The results of their work already form the basis of the Frontend, and the technical decisions they made are still relevant as it was done in the Spring Term 2022.

### 3.7.2 Chosen Library

Sigma.js<sup>23</sup> was chosen in combination with React Sigma<sup>24</sup>. Reagraph<sup>25</sup> was another library that was tested but did not get chosen as the performance was not acceptable when tested with the test graphs. Sigma.js was able to render and allow interaction with the test graph in a fast and responsive way, through the usage of WebGL. The customization options cover the intended use case of node and edge highlighting. Additionally, through the usage of Graphology<sup>26</sup> to manage the graph, a wide range of layouts and helper functions are available.

## 3.8 State Container

As an optional requirement is the ability to export and import the current state of the dashboard, it was decided to implement a state container in the Frontend. Having the state of the application managed in a central location enables easier implementation of the required data export. Redux<sup>27</sup> was chosen to implement this based on the extensive documentation and available community resources. Redux has also several helper libraries<sup>28</sup> like RTK Query that provide useful functionality.

---

<sup>23</sup><https://github.com/jacomyal/sigma.js>

<sup>24</sup><https://github.com/sim51/react-sigma>

<sup>25</sup><https://github.com/reaviz/reagraph>

<sup>26</sup><https://github.com/graphology/graphology>

<sup>27</sup><https://redux.js.org>

<sup>28</sup><https://redux-toolkit.js.org>

## 3.9 UI Library

It was decided to use Material UI <sup>29</sup> in the Frontend. The reasons were the huge amount of available components, customization options, detailed documentation and most importantly native integration into React. MUI allows the creation of visually pleasing and responsive web applications, without the need to write a lot of individual CSS.

### 3.9.1 Colors

The default color palette is provided by MUI. The predefined colors offer good contrast and are very well-matched for various purposes, such as a dark mode or light mode as seen in figure 3.1.

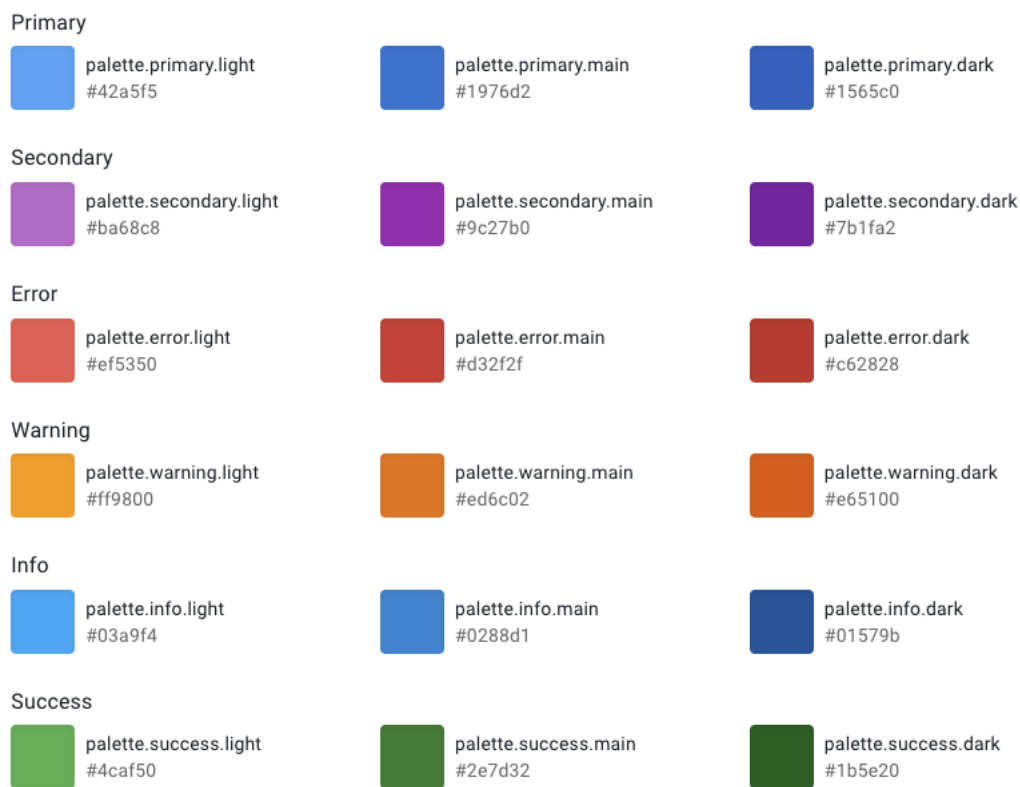


Figure 3.1: Default Color Palette

---

<sup>29</sup><https://mui.com>

## Chapter 4

# Architecture

In this chapter, the detailed architecture of the system is detailed and analyzed. Decisions stated in chapter 3 are incorporated as part of it.

### 4.1 Architecture Model

The C4 model was used to visualize the system and application architecture [12]. It was decided to only implement the first three levels as the fourth level is not necessary because it is too detailed. If it is required at some point, it can be generated automatically by modern IDEs after the implementation.

### 4.1.1 System Context Diagram

Figure 4.1 shows the System Context diagram. The user of the system can be seen interacting with the Graph Analyzer system. Dependencies on other systems are also visualized, the Graph Analyzer system depends on their provided network topology information.

**C4 Model Level 1 (System Context) - Graph Analyzer architecture**

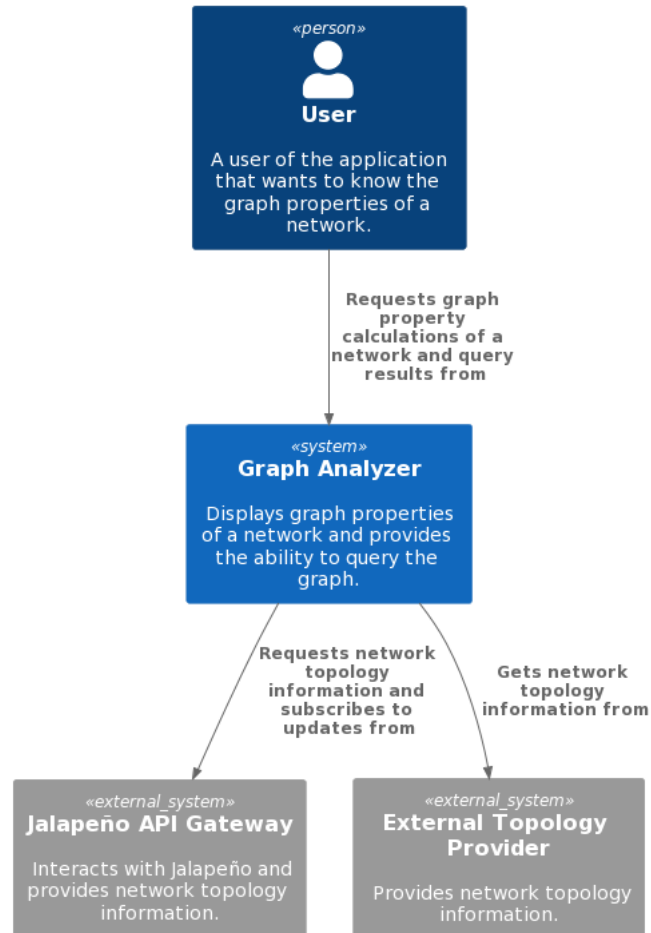


Figure 4.1: C4 Model Level 1 - System Context Diagram

### 4.1.2 Container Diagram

Based on the previous design decisions in section 3.1, the figure 4.2 shows the Container diagram of the Graph Analyzer system. The user mainly interacts with the Single-Page Application, which is the Frontend and responsible for data visualization. The Single-Page Application gets its data from the API Application. As explained in section 3.1 the backend is split into three components. The API Application retrieves and calculates graph properties, which are then exposed to the Single-Page Application. The introduced Graph Database holds the graph for the API Application and already provides some included graph properties. Retrieval of network topology data is handled by the Data Collector Application, data is written into the Graph Database so that the API Application can further process it. The retrieval can be initiated automatically through a Container Orchestrator or manually through the user, the second option is mainly possible for development reasons.



C4 Model Level 2 (Containers) - Graph Analyzer architecture

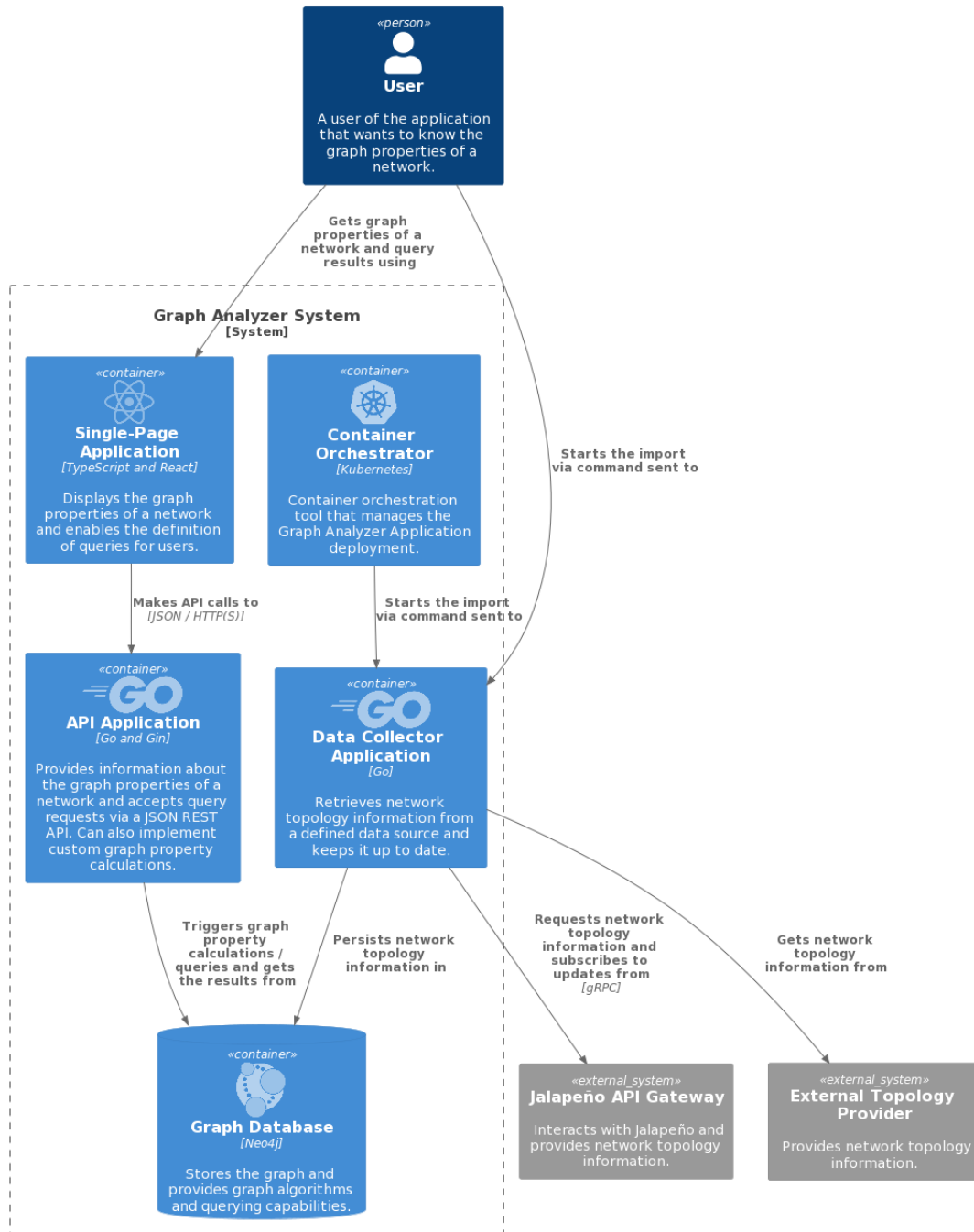


Figure 4.2: C4 Model Level 2 - Container Diagram

### 4.1.3 Component Diagram - Single-Page Application

The Component diagram of the Single-Page Application can be found in figure 4.3. As the name implies it is a React TypeScript Single-Page Application, additionally Redux is used as the state container to manage a global state. RTK Query is used to communicate with the API Application.

C4 Model Level 3 (Components) | Single-Page Application - Graph Analyzer architecture

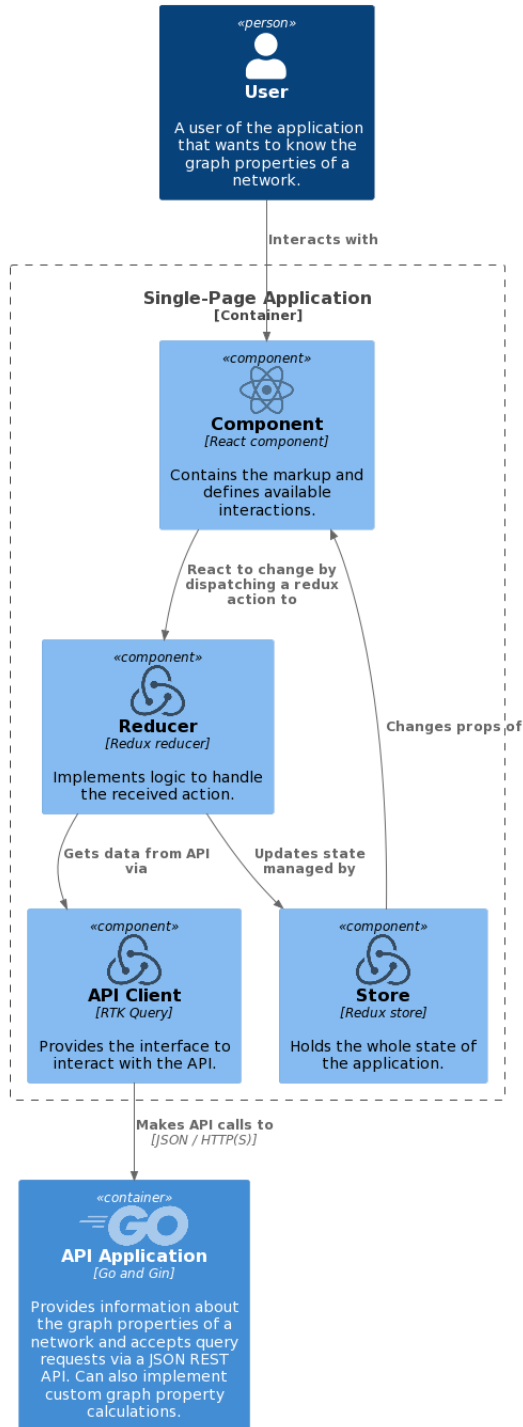


Figure 4.3: C4 Model Level 3 - Component Diagram - Single-Page Application

#### 4.1.4 Component Diagram - API Application

Figure 4.4 shows the Component diagram of the API Application. It is designed to use the ADR pattern [13]. The goal is to have a high separation of concerns and to adhere to the single responsibility principle. Interaction with the Graph Database is handled by the Repository component.

C4 Model Level 3 (Components) | API Application - Graph Analyzer architecture

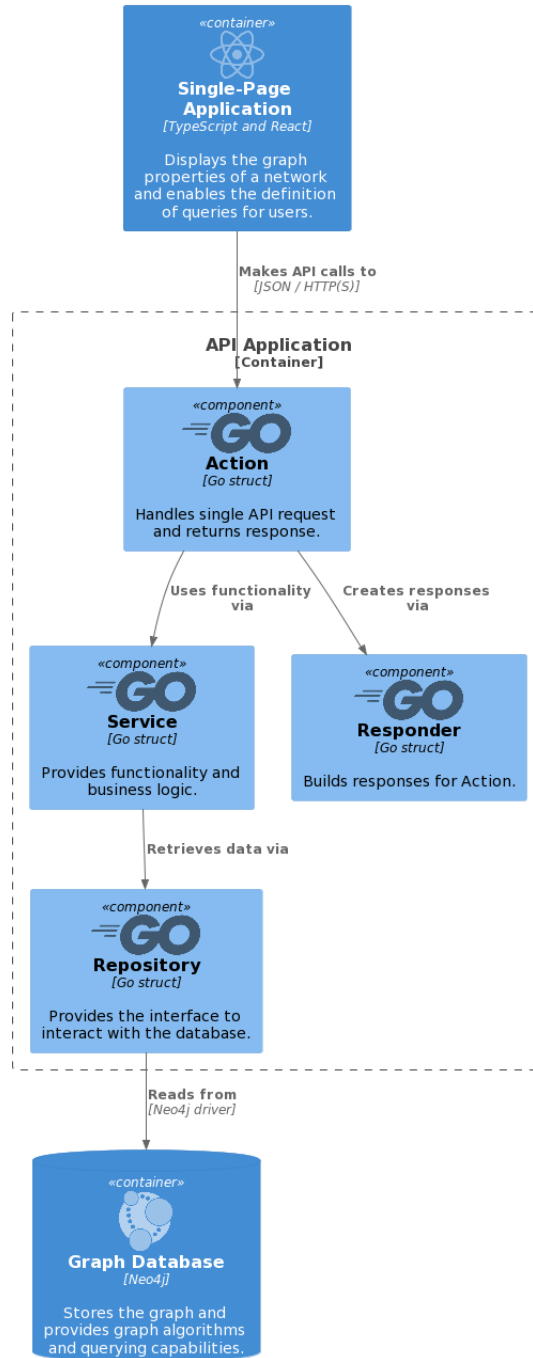


Figure 4.4: C4 Model Level 3 - Component Diagram - API Application

#### **4.1.5 Component Diagram - Data Collector Application**

To provide the flexibility to read the graph from different sources, a CLI application is used. As a result, the Data Collector can be easily expanded and supports the user when importing a graph with the available input formats.

The Component diagram of the Data Collector Application can be found in figure 4.5. It is implemented as a CLI with different commands that originate from a root command. Each command implements the data import from a separate topology provider. The structure originates from the used Cobra library that is used to create the CLI [14]. The Graph Database interaction is handled by the Repository component.

C4 Model Level 3 (Components) | Data Collector Application - Graph Analyzer architecture

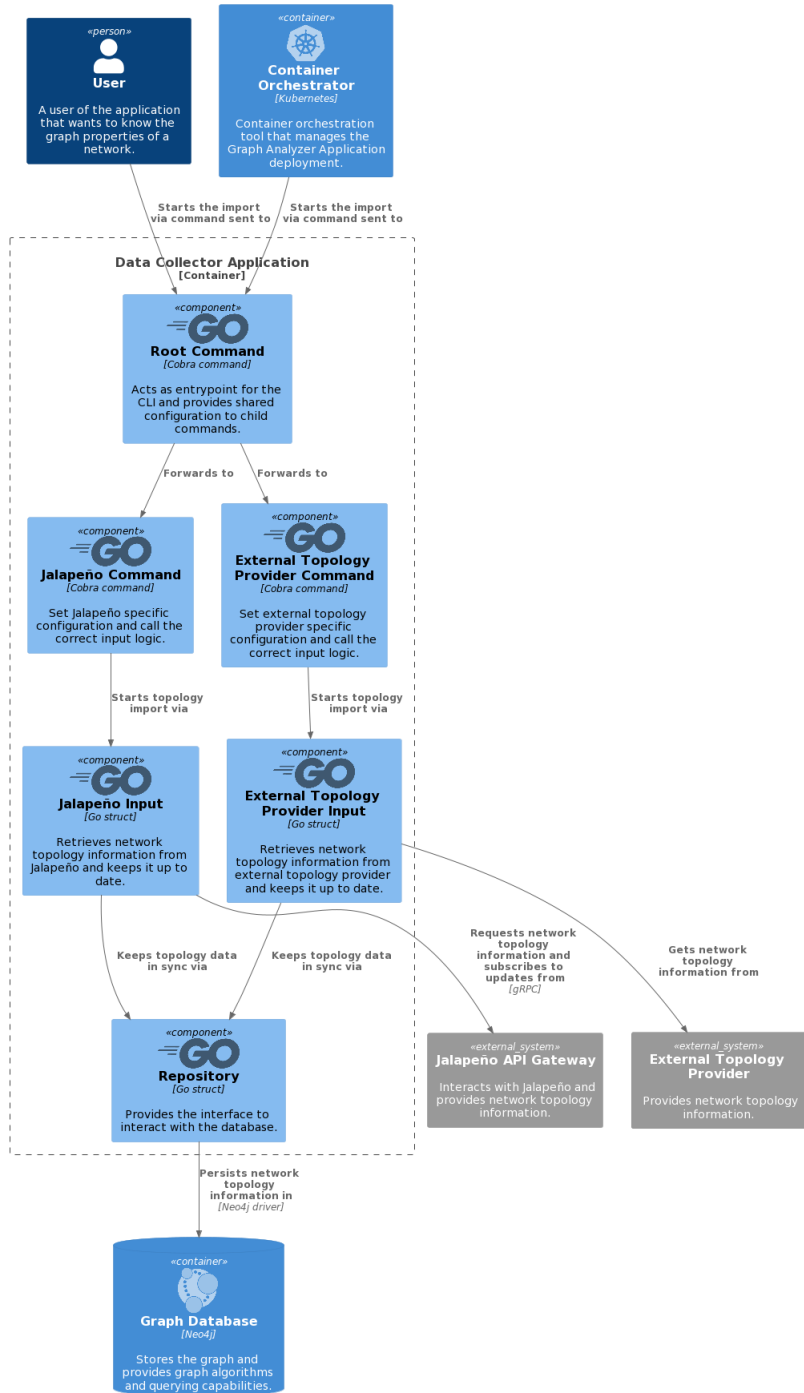


Figure 4.5: C4 Model Level 3 - Component Diagram - Data Collector Application

## 4.2 UI and UX

The UI and UX were not the primary goals of the thesis, despite that some basic wireframes were created. The goal was to have an initial idea of how the Frontend could look and provide some orientation during the latter implementation. Figure 4.6 shows the main dashboard embedded into the Central Frontend. A graph visualization and tiles for various graph properties dominate the space. The navigation bar with file import and export buttons can also be found. Figure 4.7 shows the dashboard in the offline state, data from an imported file would be used. The query view and download function are not available anymore. Various variants for the graph property tiles and their description can be found in figures 4.8 to 4.11. The query view can be seen in figure 4.12. A column of different filters, a result list with an export function and another graph visualization can be found.

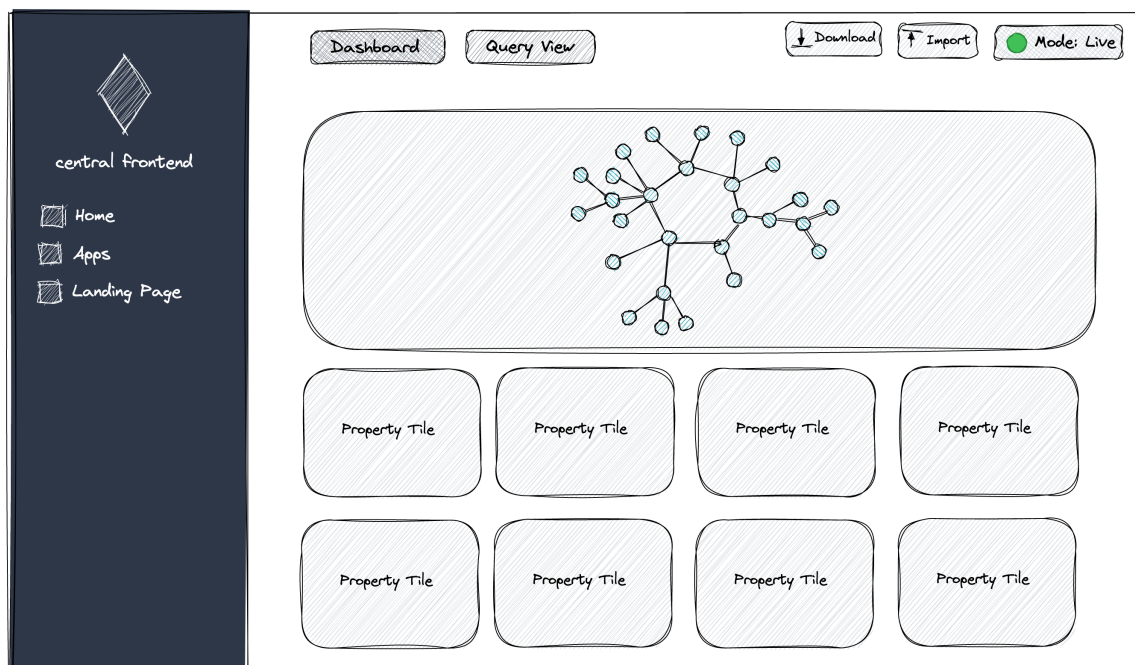


Figure 4.6: Dashboard (Live) Wireframe



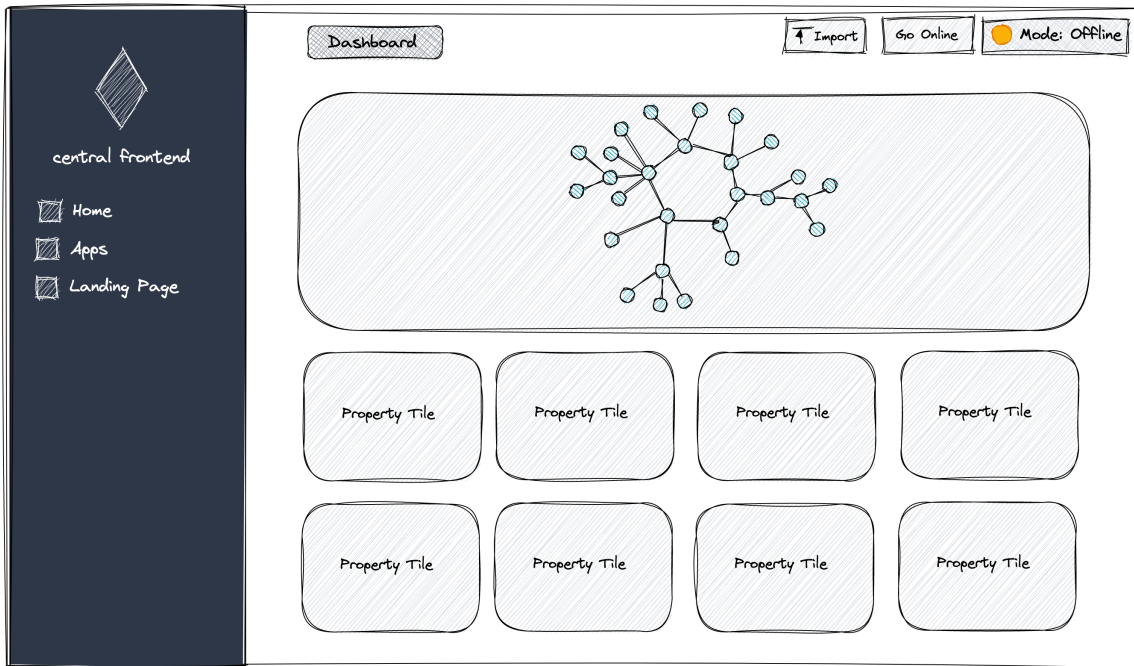


Figure 4.7: Dashboard (Offline) Wireframe

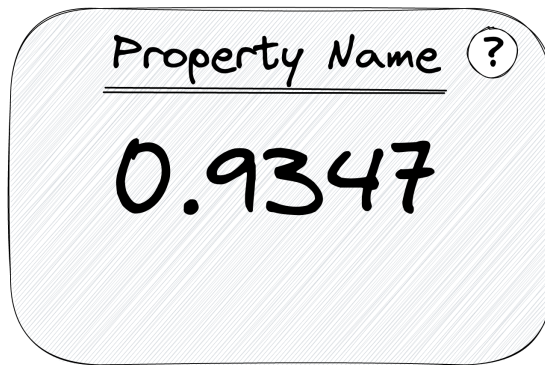


Figure 4.8: Property Tile Number Wireframe (Variant 1)

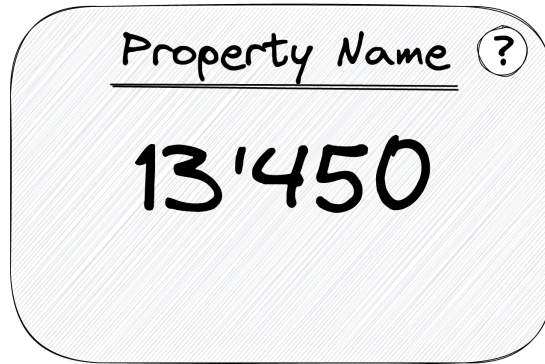


Figure 4.9: Property Tile Number Wireframe (Variant 2)

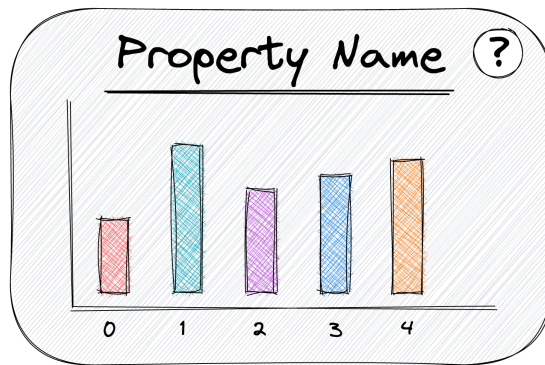


Figure 4.10: Property Tile Statistics Wireframe

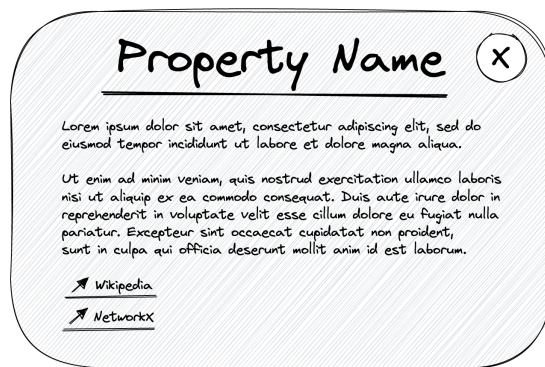


Figure 4.11: Property Tile Description Wireframe

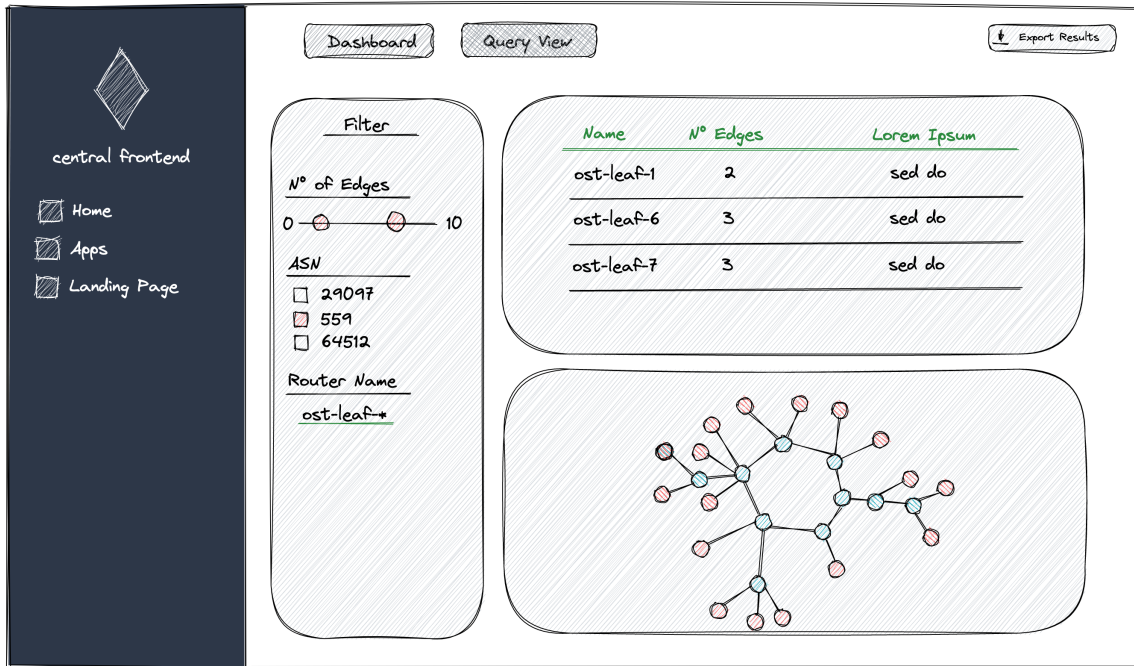


Figure 4.12: Query View Wireframe

## 4.3 Technologies

All libraries and frameworks used in this thesis are published under an open-source license. The list below shows the most important technologies used in individual applications. They will be discussed in more detail in later chapters.

### 4.3.1 API

- Programming language: Golang
- Framework: Gin
- Libraries
  - Gonum
  - Viper
  - Swag

### 4.3.2 Data Collector

- Programming language: Golang
- Libraries
  - Cobra
  - Viper
  - Logrus
  - Testcontainers

### 4.3.3 Frontend

- Programming language: TypeScript
- Framework: React
- Libraries
  - Sigma.js
  - Chart.js
  - MUI (Material UI)
  - RTK Query
  - Redux Toolkit

# Chapter 5

## Implementation

The next few sections deal with the implementation of the various applications of the Graph Analyzer system.

### 5.1 API

The API is responsible for the calculation of the graph properties and providing them to the Frontend. These calculations can be requested directly from the graph database or are alternatively implemented as part of the API.

#### 5.1.1 GDS Access

Before the API can interact with the GDS graph in Neo4j, it needs to know the name of the graph so that it can be referenced in the following procedure call. Before a GDS procedure call is done via the repository, the latest graph name is retrieved by a procedure call to `gds.graph.list` [15]. An example can be found in listing 5.1.

```
1 CALL gds.graph.list()
2 YIELD graphName, creationTime
3 RETURN graphName
4 ORDER BY creationTime DESC
5 LIMIT 1
```

Listing 5.1: GDS Latest Graph Name Retrieval

## 5.1.2 General Information Retrieval

`gds.graph.list` is also used to retrieve information about the underlying graph [15]. By calling the procedure with the graph name, the node and edge count can be retrieved. Additionally, the creation time is returned. See listing 5.2.

```
1 CALL gds.graph.list($graphName)
2 YIELD graphName, nodeCount, relationshipCount, creationTime
3 RETURN graphName AS name, nodeCount AS nodes, relationshipCount AS edges,
   creationTime AS createdAt
```

Listing 5.2: GDS General Information Retrieval

## 5.1.3 Density Calculation

The density of the graph is also contained as a result of the `gds.graph.list` procedure when called with a graph name [15]. Listing 5.3 shows how it is returned.

```
1 CALL gds.graph.list($graphName)
2 YIELD density
3 RETURN density
```

Listing 5.3: GDS Density Retrieval

## 5.1.4 Average Clustering Coefficient Calculation

The average clustering coefficient is retrieved by calling the `gds.localClusteringCoefficient.stats` procedure and extracting the value from `averageClusteringCoefficient` in the result [16]. Listing 5.4 contains an example.

```
1 CALL gds.localClusteringCoefficient.stats($graphName)
2 YIELD averageClusteringCoefficient
3 RETURN averageClusteringCoefficient
```

Listing 5.4: GDS Average Clustering Coefficient Retrieval

## 5.1.5 Connectedness Calculation

Another graph property that can directly be retrieved via GDS is the connectedness of the graph. A call to `gds.wcc.stats` is made and the returned `componentCount` is checked [17]. See listing 5.5 for an example of getting the components of a graph.

```
1 CALL gds.wcc.stats($graphName)
2 YIELD componentCount
3 RETURN componentCount
```

Listing 5.5: GDS Component Count Retrieval

### 5.1.6 Degree Distribution Calculation

The degree distribution of the graph is retrieved by calling the `gds.degree.stream` procedure [18]. Listing 5.6 contains further details.

```
1 CALL gds.degree.stream($graphName)
2 YIELD score
3 RETURN score AS degree, COUNT(*) AS amount
4 ORDER BY degree ASC
```

Listing 5.6: GDS Degree Distribution Retrieval

### 5.1.7 Full Graph Retrieval

Retrieving the full graph from the graph database is done in two cases. Firstly to provide the Frontend with data to render a visualization of it. Secondly internally when graph property calculations in the API are required, the full graph needs to be extracted for further processing. Two simple Cypher queries to retrieve the nodes and edges are used. Listing 5.7 shows the respective Cypher queries.

```
1 MATCH (n:Router)-[r:CONNECTS_TO_HIGHER|CONNECTS_TO_LOWER]->(r:Router)
2 RETURN id(n), n.RouterID, n.Label, COUNT(r)
3
4 MATCH (from:Router)-[r:CONNECTS_TO_HIGHER|CONNECTS_TO_LOWER]->(to:Router)
5 RETURN r.EdgeID, id(from), from.RouterID, id(to), to.RouterID
```

Listing 5.7: Cypher Full Graph Retrieval



### 5.1.8 Degree Assortativity Coefficient Calculation

GDS currently does not support the calculation of the degree assortativity coefficient, thus it was implemented in the API. First, the full graph is extracted from the graph database and then a Gonom undirected graph gets created based on it [19]. Every node degree and the degrees of its neighbors are retrieved by iterating through the graph. Finally, the collected degrees are used to calculate the correlation [20]. Listing 5.8 contains a simplified version of this calculation.

```
1 x := make([]float64, 0)
2 y := make([]float64, 0)
3
4 nodes := graph.Nodes()
5
6 for nodes.Next() {
7     neighbors := graph.From(nodes.Node().ID())
8     nodeDegree := neighbors.Len()
9
10    for neighbors.Next() {
11        neighbor := neighbors.Node()
12        neighborDegree := graph.From(neighbor.ID()).Len()
13
14        x = append(x, float64(nodeDegree))
15        y = append(y, float64(neighborDegree))
16    }
17 }
18
19 correlation := stat.Correlation(x, y, nil)
```

Listing 5.8: Go Degree Assortativity Coefficient Calculation



### 5.1.9 Degree Correlation Calculation

To calculate the degree correlation, again a Gonom undirected graph is created. By iterating through the graph a map is built, containing the degree of a node and the respective neighbor degrees. In the end, an array containing items with the node degree and the respective mean of its neighbor degrees is returned [21]. See listing 5.9.

```
1 var nodeNeighborDegrees = make(map[int64][]float64)
2 nodes := graph.Nodes()
3
4 for nodes.Next() {
5     neighbors := graph.From(nodes.Node().ID())
6     nodeDegree := int64(neighbors.Len())
7
8     for neighbors.Next() {
9         neighbor := neighbors.Node()
10        neighborDegree := float64(graph.From(neighbor.ID()).Len())
11
12        nodeNeighborDegrees[nodeDegree] = append(nodeNeighborDegrees[
13            nodeDegree], neighborDegree)
14    }
15 }
16 nodeDegrees := make([]int64, 0, len(nodeNeighborDegrees))
17
18 for nodeDegree := range nodeNeighborDegrees {
19     nodeDegrees = append(nodeDegrees, nodeDegree)
20 }
21
22 values := make([]degreeCorrelationValue, 0, len(nodeDegrees))
23
24 for _, nodeDegree := range nodeDegrees {
25     values = append(values, degreeCorrelationValue{
26         Degree: nodeDegree,
27         Average: stat.Mean(nodeNeighborDegrees[nodeDegree], nil),
28     })
29 }
```

Listing 5.9: Go Degree Correlation Calculation

### 5.1.10 Diameter Calculation

To calculate the diameter of a graph, the shortest path between all pairs of nodes in a graph needs to be calculated. This is an expensive calculation and can run for up to a few minutes for a graph with a few thousand nodes and edges. GDS provides the `gds.alpha.allShortestPaths` procedure for this [22]. This procedure is classified as alpha and is not considered ready for production usage. Important functionality like memory estimation is not available yet, a huge graph could thus lead to the graph database running out of memory [23]. The diameter calculation is done in the API. Using Dijkstra's algorithm, all shortest paths get calculated and the longest of them is the diameter. The time complexity of the Gonum DijkstraAllPaths function is  $O(|V| * |E| + |V|^2 * \log|V|)$  [24]. A simplified implementation can be found in listing 5.10.

```
1 shortestPaths := path.DijkstraAllPaths(graph)
2
3 nodesDone := make(map[int64]bool, 0)
4 nodesFrom := graph.Nodes()
5 nodesTo := graph.Nodes()
6 distance := int64(0)
7
8 for nodesFrom.Next() {
9     nodeFrom := nodesFrom.Node()
10
11     for nodesTo.Next() {
12         nodeTo := nodesTo.Node()
13
14         if _, ok := nodesDone[nodeTo.ID()]; !ok {
15             weight := shortestPaths.Weight(nodeFrom.ID(), nodeTo.ID())
16
17             distance = int64(
18                 math.Max(float64(distance), weight),
19             )
20         }
21     }
22
23     nodesDone[nodeFrom.ID()] = true
24     nodesTo.Reset()
25 }
```

Listing 5.10: Go Diameter Calculation

### 5.1.11 Cut Edge Calculation

Finding cut edges or bridges is implemented in the API based on Tarjan's bridge-finding algorithm [25]. It has a time complexity of  $O(|V| + |E|)$ . Listing 5.11 shows the implementation.

```
1 type bridgeDetectionCalculation struct {
2     id float64
3     ids map[int64]float64
4     low map[int64]float64
5 }
6
7 type bridge struct {
8     From int64
9     To   int64
10 }
11
12 func DetectBridgesInGraph(graph *simple.UndirectedGraph) []bridge {
13     bdc := &bridgeDetectionCalculation{
14         id: 0,
15         ids: make(map[int64]float64),
16         low: make(map[int64]float64),
17     }
18
19     visited := make(map[int64]bool)
20     bridges := make([]bridge, 0)
21
22     nodes := graph.Nodes()
23
24     for nodes.Next() {
25         node := nodes.Node()
26
27         if !visited[node.ID()] {
28             bridges = detectBridgesInGraphRecursive(bdc, graph, node.ID(),
29             visited, -1, bridges)
30         }
31     }
32     return bridges
33 }
34
35 func detectBridgesInGraphRecursive(
36     bdc *bridgeDetectionCalculation,
37     graph *simple.UndirectedGraph,
38     at int64,
39     visited map[int64]bool,
40     parent int64,
41     bridges []bridge,
42 ) []bridge {
43     visited[at] = true
44     bdc.id++
45     bdc.low[at] = bdc.id
46     bdc.ids[at] = bdc.id
47 }
```

```

48 neighbours := graph.From(at)
49
50 for neighbours.Next() {
51     to := neighbours.Node().ID()
52
53     if to == parent {
54         continue
55     }
56
57     if visited[to] {
58         bdc.low[at] = math.Min(bdc.low[at], bdc.ids[to])
59     } else {
60         bridges = detectBridgesInGraphRecursive(bdc, graph, to, visited, at
, bridges)
61         bdc.low[at] = math.Min(bdc.low[at], bdc.low[to])
62
63         if bdc.ids[at] < bdc.low[to] {
64             if at < to {
65                 bridges = append(bridges, bridge{
66                     From: at,
67                     To:    to,
68                 })
69             } else {
70                 bridges = append(bridges, bridge{
71                     From: to,
72                     To:    at,
73                 })
74             }
75         }
76     }
77 }
78
79 return bridges
80 }

```

Listing 5.11: Go Bridge Detection

### 5.1.12 Cut Vertex Calculation

Finding cut vertices or articulation points is implemented in the API based on Tarjan's strongly connected components algorithm [26]. It also has a time complexity of  $O(|V| + |E|)$ . The implementation can be found in listing 5.12.

```
1 type articulationPointDetectionCalculation struct {
2     id          float64
3     ids         map[int64]float64
4     low        map[int64]float64
5     articulationPoints map[int64]int64
6 }
7
8 func DetectArticulationPointsInGraph(graph *simple.UndirectedGraph) map[
9     int64]int64 {
10     apd := &articulationPointDetectionCalculation{
11         id:          0,
12         ids:         make(map[int64]float64),
13         low:        make(map[int64]float64),
14         articulationPoints: make(map[int64]int64),
15     }
16     visited := make(map[int64]bool)
17
18     nodes := graph.Nodes()
19
20     for nodes.Next() {
21         node := nodes.Node()
22
23         if !visited[node.ID()] {
24             detectArticulationPointsInGraphRecursive(apd, graph, node.ID(),
25                 visited, node.ID(), -1)
26         }
27     }
28     return apd.articulationPoints
29 }
30
31 func detectArticulationPointsInGraphRecursive(
32     apd *articulationPointDetectionCalculation,
33     graph *simple.UndirectedGraph,
34     root int64,
35     visited map[int64]bool,
36     at int64,
37     parent int64,
38 ) {
39     visited[at] = true
40     apd.id++
41     apd.low[at] = apd.id
42     apd.ids[at] = apd.id
43
44     children := 0
45
46     neighbours := graph.From(at)
```

```

47
48 for neighbours.Next() {
49     to := neighbours.Node().ID()
50
51     if to == parent {
52         continue
53     }
54
55     if visited[to] {
56         apd.low[at] = math.Min(apd.low[at], apd.ids[to])
57     } else {
58         detectArticulationPointsInGraphRecursive(apd, graph, root, visited,
59             to, at)
60         apd.low[at] = math.Min(apd.low[at], apd.low[to])
61
62         if apd.ids[at] <= apd.low[to] && parent != -1 {
63             apd.articulationPoints[at] = at
64         }
65
66         children++
67     }
68 }
69
70 if parent == -1 && children > 1 {
71     apd.articulationPoints[at] = at
72 }

```

Listing 5.12: Go Articulation Points Detection

### 5.1.13 Query Endpoint

As the querying function was defined as an optional requirement, it was decided to implement the query endpoint as an early-stage proof of concept. Only the two basic query options for cut edges and cut vertices are implemented. The previously described algorithms for bridge and articulation point detection are reused. Instead of checking for existence, the detected nodes and edges are returned and can thus be identified.

### 5.1.14 Infrastructure Endpoints

The API implements two additional endpoints for liveness and readiness. These are used to monitor the status of the deployed application in Kubernetes.

### 5.1.15 Testing

Each endpoint is covered by an integration test. Because of the limited time, it was decided to create a mocked repository to simulate access to the graph database. The other option would have been to start a real graph database in the background on each test run via Testcontainers. In addition, specific logic and calculations that are implemented by the API itself are covered with unit tests.

### 5.1.16 Usage

All the endpoints contain annotations to generate an OpenAPI 2.0 specification. Using Swagger UI the specification is rendered in an understandable and interactive form. Additionally, the raw specification can be accessed directly by API consumers, to enable them to generate API clients automatically based on it.

## 5.2 Data Collector

The main task of the Data Collector is to consume a graph data source and persist it in the graph database. Depending on the source, the graph is imported once or continually.

### 5.2.1 Implementation

The application is implemented as a CLI and is controlled by provided parameters and flags. The CLI is implemented with the Cobra library. Handling configurations is done by Viper. Configurations can be provided via flags, referenced configuration files or through set environment variables. The Data Collector supports two types of data sources when importing a graph. JAGW is an example of an interaction with an external system and GEXF files for a one-time file-based import. Since other data sources and formats should be supported in the future, the import logic was implemented using the strategy pattern [27]. The core logic of the import can stay generic, while other data source formats can be added on top.

### 5.2.2 Jalapeño / JAGW

The JAGW offers two options for obtaining the network topology:

1. The JAGW Request Service has several endpoints to get the required data from Jalapeño [28]. It is currently enough to call the `GetLsNodes()` and `GetLsNodeEdges()` gRPC endpoints to receive the needed information.
2. To satisfy the live update requirement, the JAGW Subscription Service is used to receive direct updates via gRPC [29]. Same as with the Request Service, the base methods `SubscribeToLsNodes()` and `SubscribeToLsNodeEdges()` are used to continuously receive the needed information.

Since the subscription endpoints are used, it needs to be ensured that the data is written consistently to the database when the application is initializing. This is achieved by subscribing to events first and writing them into a queue for later consumption. The request endpoint is then called to retrieve the full topology at the time of the call. The consumption of events kept in the queue can then begin.

Go offers an elegant solution in the form of channels [30]. They can be used to implement a queue as previously described. Subscription events are continuously written into channels, their processing only starts when the data returned by the request service has been completely processed. The processing is handled by two separate go routines that are concurrently running and are being kept alive.



### 5.2.3 GEXF

GEXF (Graph Exchange XML Format) is an XML-based language format that is used to describe graphs [31]. Files in this format are able to be parsed and imported by the Data Collector. Listing 5.13 contains a minimal example of how the structure of such a file looks.

```
1 <?xml version='1.0' encoding='utf-8'?>
2 <gexf xmlns="http://gexf.net/1.2" version="1.2">
3   <graph defaultedgetype="undirected" mode="static">
4     <nodes>
5       <node id="1" label="XR-1">
6     </node>
7       <node id="2" label="XR-2">
8     </node>
9     </nodes>
10    <edges>
11      <edge source="1" target="2" id="0">
12    </edge>
13  </edges>
14 </graph>
</gexf>
```

Listing 5.13: GEXF Structure

### 5.2.4 Data Structure

The Neo4j graph database holds two types of data. Nodes and edges, which can both be enriched with metadata. Edges are also referred to as relationships in Neo4j. Figure 5.1 shows how a graph is stored in the graph database.

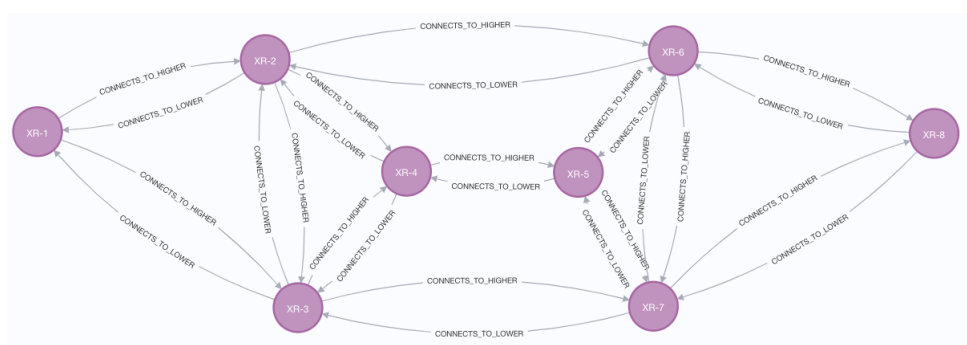


Figure 5.1: Neo4j Graph Of The Provided INS network

#### Nodes

A node has the following properties:

- `<id>`: Internal Neo4j identifier
- `Label`: Display name of the node

- **RouterID**: ID of the associated router
- **RouterKey**: Key of the associated router

It was required to store both the **RouterID** and **RouterKey** due to how the JAGW is designed. The **RouterID** is used to correlate nodes to edges when the edges are retrieved. Nodes need to be added to the graph before edges can. While **RouterID** is used mainly by the JAGW Request Service, **RouterKey** is used by the JAGW Subscription Service. Due to this behavior, it is required to store both identifiers. An example of how a node is stored can be found in figure 5.2.

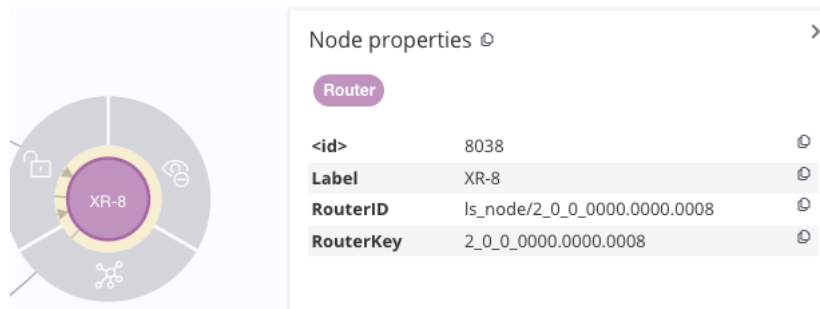


Figure 5.2: Neo4j Node Of The Provided INS network

## Edges

An edge has the following properties:

- **relationshipType**: CONNECTS\_TO\_HIGHER | CONNECTS\_TO\_LOWER
- **<id>**: Internal Neo4j identifier
- **EdgeID**: ID of the associated link
- **EdgeKey**: Key of the associated link
- **FromRouterID**: Start node of the edge
- **ToRouterID**: End node of the edge

It was required to use two types of relationships between nodes because Neo4j does not natively support undirected relationships. Due to the reason that GDS just duplicates existing relationships when creating a projection of an undirected graph, a solution was required that enables the consistent extraction of just one relationship direction. Otherwise, the GDS graph would contain duplicated edges. The relationship type **CONNECTS\_TO\_LOWER** indicates whether the directed edge connects from a node with a higher **RouterID** to a lower one. **CONNECTS\_TO\_HIGHER** indicates the opposite. The structure of how an edge is stored can be found in figure 5.3.

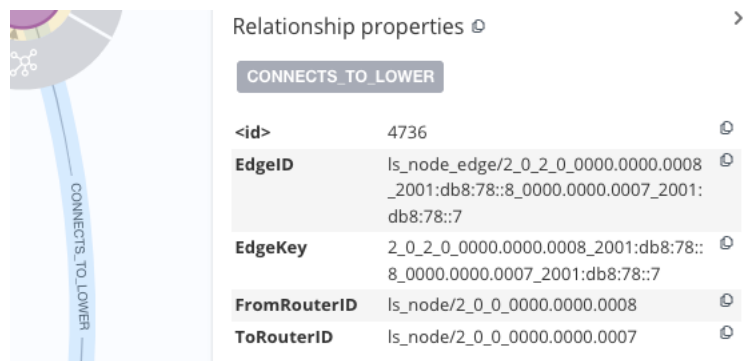


Figure 5.3: Neo4j Edge Of The Provided INS network

### 5.2.5 GDS Graph

To use the Neo4j GDS algorithms that are called via procedures in the API, a projection of the existing graph in Neo4j needs to be created. The state of the graph at the moment of creation is then kept in memory. The Data Collector is responsible for the creation of these projections. This is done during the initial import of a graph. If a change event from JAGW is received while an active subscription exists, the Data Collector also regularly recreates the projection.

### 5.2.6 Testing

It was decided to only write some basic integration tests for the Data Collector. Using Testcontainers a Neo4j database is started, and it is validated that both types of imports still work. The GEXF import is tested using an example file and the JAGW is tested via a Camouflage<sup>1</sup> gRPC server that provides mock responses.

### 5.2.7 Usage

How the Data Collector can be used is documented as part of the implementation via a help command [14]. It is started as part of the deployment by a container orchestrator or locally for development purposes.

<sup>1</sup><https://github.com/testinggospels/camouflage>

## **5.3 Frontend**

The Frontend has the goal of displaying the calculated properties from the API and displaying them visually and with explanations. It is based on React and written in Typescript. For the state store, Redux toolkit as well as RTK Query for fetching and caching API requests are used.

### **5.3.1 API Generation**

With the help of RTK Query Code Generator for OpenAPI, a typed API client from an OpenAPI schema defined in the API application is generated [32].

### **5.3.2 Dashboard - Page**

The Dashboard is the main page in the current version. All required graph properties are displayed and explained on this page.

### **5.3.3 Query View - Page**

The Query View was introduced as an optional requirement to understand the network even better and to use it as a debugging tool. Two pieces of information are currently available. Which nodes are cut vertices and which links are cut edges.

### **5.3.4 Components**

All properties are split into components to keep the React application reusable, modular, and easier to understand.

### 5.3.5 Navigation - Component

Depending on the functionality of the page, the navigation bar contains additional features as shown in figures 5.4 and 5.5.

- Refresh
- Export
- Import



Figure 5.4: Graph Analyzer Frontend - Navigation - Dashboard



Figure 5.5: Graph Analyzer Frontend - Navigation - Query View

### 5.3.6 Refresh - Component

The refresh button invalidates the RTK Query API cache and re-requests the data from the API on click to get the latest graph information and properties.

```
1 onClick={() => {  
2   dispatch(graphAnalyzerAPI.util.invalidateTags(['graph-properties']));  
3 }}
```

Listing 5.14: Invalidating RTK Query Tags

### 5.3.7 Export - Component

The export function makes it possible to export the current status of the network to a JSON file and later read it back in using the import function described in section 5.3.8. The name of the JSON file is based on the network name that may have been specified in the Data Collector. The values are extracted from the Redux Store, converted to JSON and downloaded to the user's download folder with the help of file-saver <sup>2</sup>.

<sup>2</sup><https://www.npmjs.com/package/file-saver>

### **5.3.8 Import - Component**

The import function lets the user select the previously exported JSON file and display the information it contains. The function was developed to compare different networks as simply as possible without changing the underlying network via Data Collector. It is important to know that the import function does not have any interaction with the API and everything is processed locally.

### **5.3.9 Graph - Component**

The graph component displays the network graph using Sigma.JS, which is using WebGL under the hood. Graphology in combination with forceatlas2 is used to handle the graph data model and algorithm [33]. For performance reasons, the layout computation has been encapsulated inside a web worker to not block the render process. The nodes are given different sizes, depending on the degree of the respective node. They also get random colors assigned to improve visibility on high-degree nodes. It is also possible to click on an individual node and thus see what other connections it has to other nodes.

### 5.3.10 Graph property - Component

For the graph properties, strict attention was paid to keeping the components as modular as possible and making them reusable. The properties have been split according to the function it is supposed to represent. A total of 6 components were created to cover all use cases.

- Base component
  - Correlation component
  - Distribution component
  - MultiValue component
  - Value component
- InformationDialog component

#### **Base component**

All graph properties are using this component. It defines the basic look and allows further specifying the children using React props.

#### **Correlation component**

The correlation component is about displaying a line chart using Chart.js [34].

#### **Distribution component**

The distribution component is about displaying a bar chart using Chart.js [35].

#### **MultiValue component**

This component is used to render multiple lines of text in a card.

#### **Value component**

The most used component is the value component. It represents a single value.

## 5.4 Neo4j Image

A custom Neo4j image is needed because of the reliance on the APOC<sup>3</sup> and GDS plugins. For deployment on Kubernetes using the official Helm chart, Neo4j recommends building a custom image [36].

### 5.4.1 Implementation

The image is built by the Gitlab CI/CD pipeline. It is a so-called multi-stage build [37]. The build process is divided into two steps. In the first step, the required sources (in this case plugins) are downloaded using an Alpine image and then integrated into a productive Neo4j image. This has the advantage that the required image size remains relatively small.

### 5.4.2 Usage

The pipeline runs on normal commits as well as on tags. The images can then be obtained from the Container Registry of the GitLab repository as seen in figure 5.6.

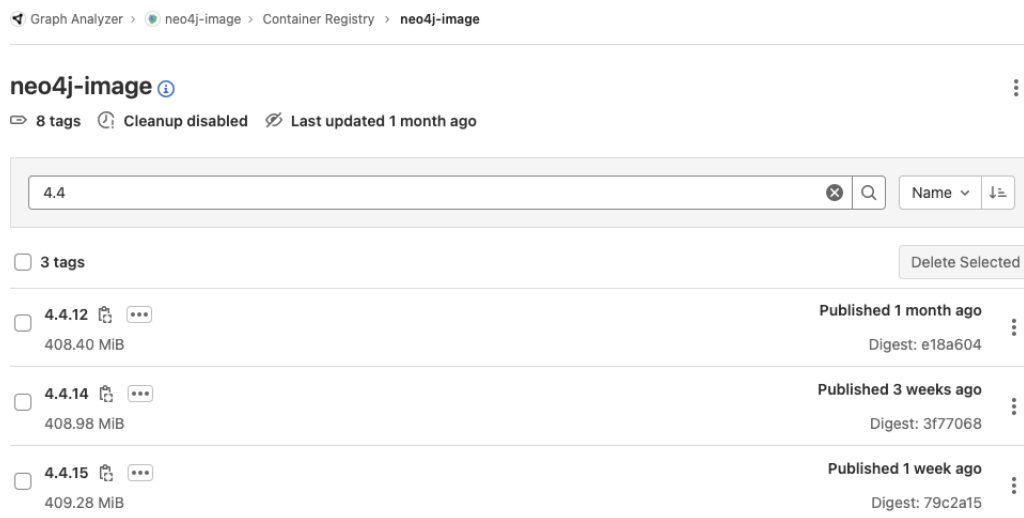


Figure 5.6: Available Neo4j Image Tags

<sup>3</sup><https://neo4j.com/labs/apoc>



## 5.5 Helm Chart

In order to carry out the deployment quickly and without much effort, a helm chart was created for the different components. The Helm chart consists out of 5 parts:

1. Neo4j helm dependency
2. API
3. Data Collector
4. Standalone Frontend
5. Micro-Frontend (for the Central Frontend)

### 5.5.1 Implementation

The last four components mentioned each have their sub-folder in the "templates" folder, while the Neo4j helm dependency can be found in the "charts" folder. To adapt the deployment, the `values.yaml` file can be easily configured so that it meets the needs of another infrastructure.

### 5.5.2 Usage

The usage of the Helm Chart is well documented in the README of the repository. After running the Helm Chart, the output displays the deployed URLs as seen in listing 5.15.

```
1 Release "graph-analyzer" has been upgraded. Happy Helming!  
2 NAME: graph-analyzer  
3 LAST DEPLOYED: Wed Dec 7 17:51:48 2022  
4 NAMESPACE: graph-analyzer  
5 STATUS: deployed  
6 REVISION: 61  
7 TEST SUITE: None  
8 NOTES:  
9 Access your graph-analyzer:  
10  
11 API:  
12 https://api-sa-graph-properties.stu.network.garden  
13  
14 Standalone Frontend:  
15 https://frontend-sa-graph-properties.stu.network.garden  
16  
17 Micro Frontend (Central Frontend):  
18 https://frontend-micro-sa-graph-properties.stu.network.garden
```

Listing 5.15: Helm Deployment Output

# Chapter 6

## Results

### 6.1 Deployment

The Graph Analyzer system is currently deployed to a Kubernetes cluster using the Helm Chart in section 5.5. The cluster is provided by the INS and can only be accessed from the INS network or via VPN.

- Frontend: <https://frontend-sa-graph-properties.stu.network.garden>
- API: <https://api-sa-graph-properties.stu.network.garden>

An overview of the deployment in Kubernetes can be found in figure 6.1.

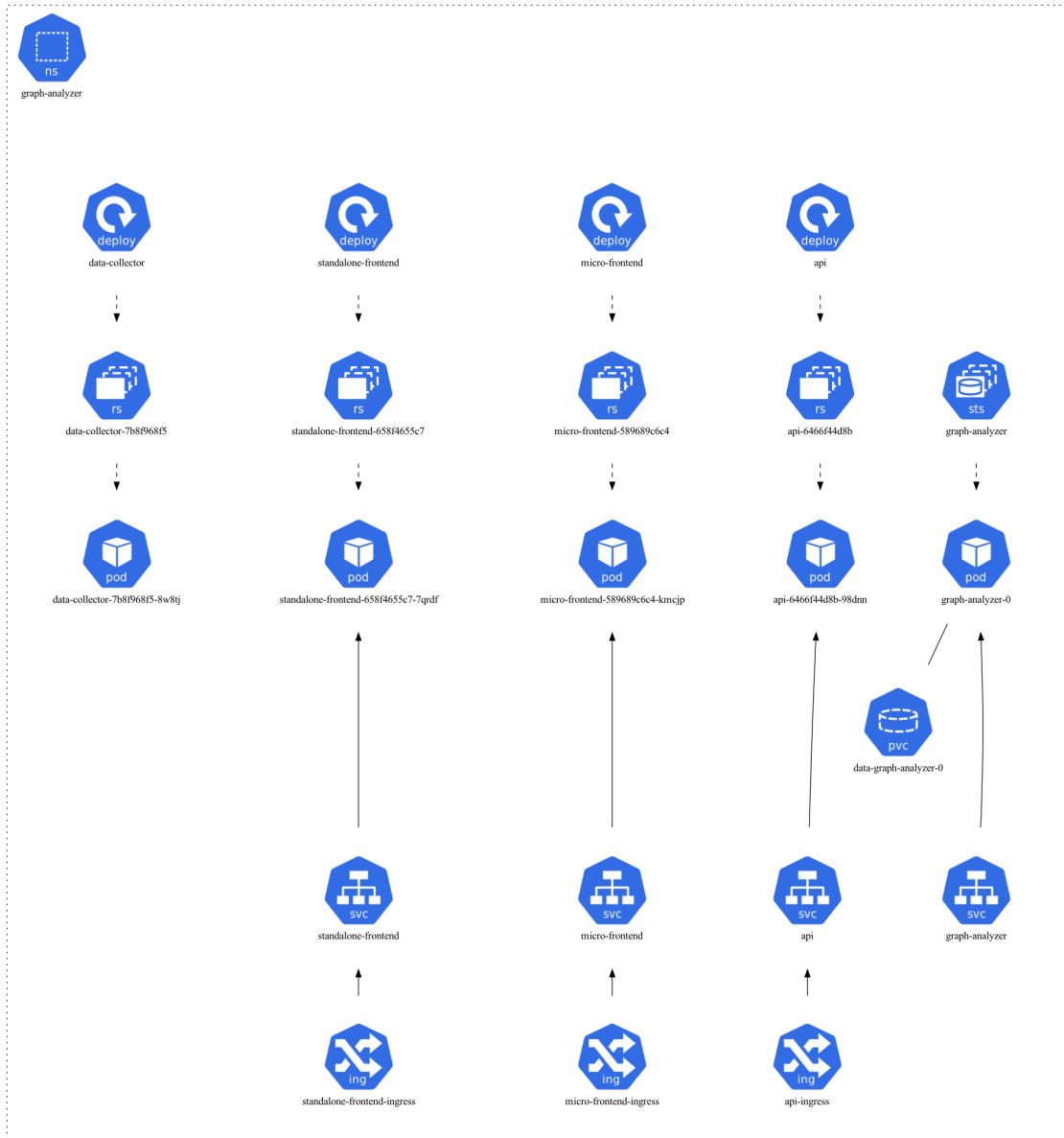


Figure 6.1: Current Deployment

## 6.2 Use Cases

Table 6.1 lists all defined use cases and indicates via traffic light colors if they have been fulfilled.

| ID      | Use case  |
|---------|---|
| FR-01   | Required: Show dashboard                                  |
| FR-01.a | Required: Dashboard - Show number of nodes                |
| FR-01.b | Required: Dashboard - Show number of edges                |
| FR-01.c | Required: Dashboard - Show degree correlation             |
| FR-01.d | Required: Dashboard - Show assortativity coefficient      |
| FR-01.e | Required: Dashboard - Show degree distribution            |
| FR-01.f | Required: Dashboard - Show cut edge status                |
| FR-01.g | Required: Dashboard - Show network diameter               |
| FR-01.h | Required: Dashboard - Show average clustering coefficient |
| FR-01.i | Optional: Dashboard - Show articulation point status      |
| FR-01.j | Optional: Dashboard - Show density                        |
| FR-01.k | Optional: Dashboard - Show graph connectivity             |
| FR-01.l | Optional: Dashboard - Show graph communities              |
| FR-01.m | Required: Dashboard - Current state of network            |
| FR-01.n | Optional: Dashboard - Export graph properties             |
| FR-01.o | Optional: Dashboard - File-based state of the network     |
| FR-01.p | Required: Dashboard - Get background information          |
| FR-01.q | Optional: Dashboard - Graph visualization                 |
| FR-02   | Optional: Show query view                                 |
| FR-02.a | Optional: Query view - Filter for properties              |
| FR-02.b | Optional: Query view - Show nodes / edges                 |

Table 6.1: Fulfillment Of Use Cases

## 6.3 NFR Validation

Table 6.2 contains the results of the NFR validation process defined in table 2.1. Traffic light colors indicate if an NFR has been met.

| ID     | Result  |
|--------|---|
| NFR-01 | The API response time is logged by using Apache Bench <sup>1</sup> . 100 concurrent requests and 1'000 requests were sent, and the output can be found in listing 6.1. The output shows that the API can handle 100 concurrent users and that the NFR is therefore met.   |
| NFR-02 | Importing a large graph composed of 10'000 nodes only takes a few seconds as can be seen in listing 6.2. The API and Frontend are also able to handle it in all but one case. The calculation of the diameter graph property can take up to a few minutes to complete. This is due to the underlying nature of this property, as all shortest paths in the graph need to be calculated.   |
| NFR-03 | To test the self-healing capability of the applications, it was tested if the API restarts after it has been killed. As can be seen in listing 6.3, the API was up and running again after about 7 seconds without any manual intervention.   |
| NFR-04 | It was tried to cover most cases where a user can enter invalid data. The Data Collector checks if all necessary or optional flags have been set correctly. The GEXF input is checked to see whether it is a valid XML and if the required GEXF types are present in the file. In the Frontend the only user input is the importable Graph Analyzer file that has been previously exported. Only a very rudimentary check of the imported data takes place. The API validates input data when handling a request and throws an error if not successful. |
| NFR-05 | All applications were created cloud-native right from the start. Thus, all applications run in containers and are created and validated by pipelines. Logging takes place directly after stdout/stderr and the logic of the import and the backend is divided into its services.  |
| NFR-06 | Each part of the system has its pipeline that performs linting, testing, building and tagging of the container images.  |
| NFR-07 | Despite the likely usage of the Graph Analyzer system on desktop-sized devices, the Frontend has been programmed in a responsive way that adapts to smaller devices and viewports. Figure 6.2 shows the dashboard on a mobile device.   |
| NFR-08 | The graph visualization component in the Frontend is isolated in its component. In addition, the full graph provided by the API is not in a graphology-specific format and is converted to the correct format when used in the component itself.  |

Table 6.2: NFR Validation

<sup>1</sup><https://httpd.apache.org/docs/2.4/programs/ab.html>

```

# ab -n 1000 -c 100
https://api-sa-graph-properties.stu.network.garden/api/
graph-property/degree-correlation
...
Concurrency Level:      100
Time taken for tests:   1.807 seconds
Complete requests:     1000
Failed requests:       0
Total transferred:     314000 bytes
HTML transferred:     107000 bytes
Requests per second:   553.39 [#/sec] (mean)
Time per request:      180.706 [ms] (mean)
Time per request:      1.807 [ms]
                        (mean, across all concurrent requests)
Transfer rate:         169.69 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median  max
Connect:     36   68  21.1    62   156
Processing:  37   97  93.8    64   411
Waiting:     37   97  93.8    64   411
Total:       84  166 104.8   128  493

Percentage of the requests served within a certain time (ms)
 50%    128
 66%    142
 75%    158
 80%    169
 90%    260
 95%    476
 98%    484
 99%    488
100%    493 (longest request)

```

Listing 6.1: Apache Bench Output

```

# ./data-collector gexf -f testgraph/graph_10000.gexf
INFO [2022-12-14T08:31:08+01:00] Database connection established
INFO [2022-12-14T08:31:10+01:00] Using gexf file testgraph/graph_10000.
gexf
INFO [2022-12-14T08:31:11+01:00] Parsed 10000 nodes from input file
INFO [2022-12-14T08:31:13+01:00] Parsed 52722 edges from input file
INFO [2022-12-14T08:31:22+01:00] Finished importing gexf file

```

Listing 6.2: Data Collector 10'000 Nodes Import Output

```

# while true; do date; curl -s -o /dev/null -w "%{http_code}" "https://
  api-sa-graph-properties.stu.network.garden/api/graph-property/degree-
  correlation"; sleep 1; echo; done
Wed Dec 14 08:40:43 CET 2022
200
Wed Dec 14 08:40:44 CET 2022
200
Wed Dec 14 08:40:45 CET 2022

```

```
404
Wed Dec 14 08:40:46 CET 2022
404
Wed Dec 14 08:40:47 CET 2022
404
Wed Dec 14 08:40:49 CET 2022
404
Wed Dec 14 08:40:50 CET 2022
404
Wed Dec 14 08:40:51 CET 2022
404
Wed Dec 14 08:40:52 CET 2022
404
Wed Dec 14 08:40:53 CET 2022
200
Wed Dec 14 08:40:54 CET 2022
200
```

Listing 6.3: Self Healing Test Output

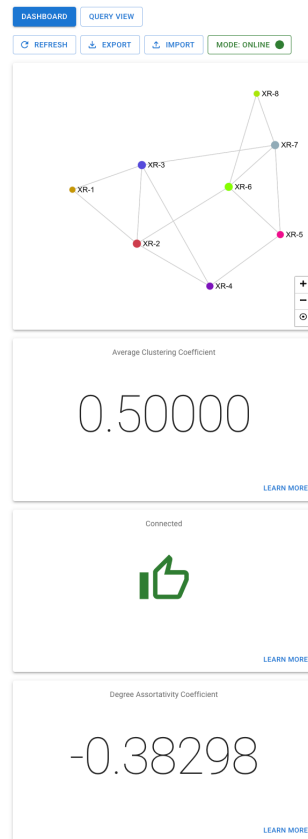


Figure 6.2: Responsive Frontend On A Smaller Viewport

## 6.4 API

Figure 6.3 shows an excerpt of the final OpenAPI specification displayed in Swagger-UI. All implemented graph properties with their respective endpoints can be seen. The query endpoint is also present in addition to the liveness and readiness endpoints.

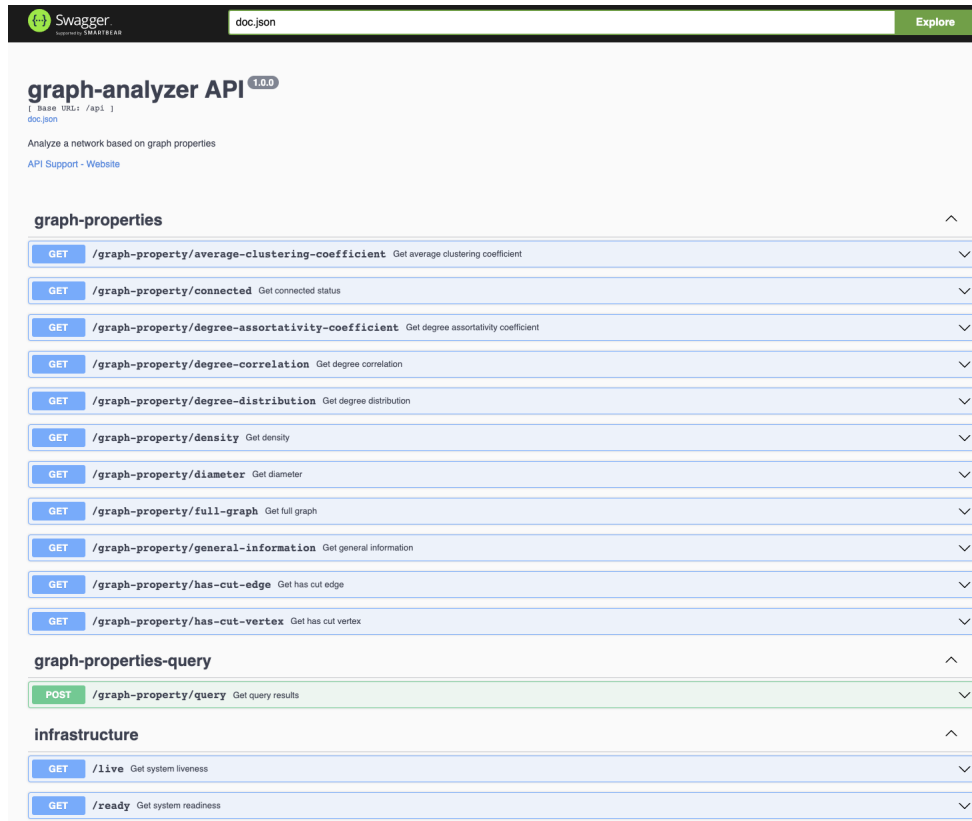


Figure 6.3: Swagger UI



## 6.5 Frontend

In this section, the final implementation of the Frontend is illustrated with the help of screenshots.

### 6.5.1 Dashboard

The dashboard can be seen in figure 6.4. It contains the graph visualization and individual graph property tiles. In addition, the navigation containing the refresh, import and export buttons can be seen.

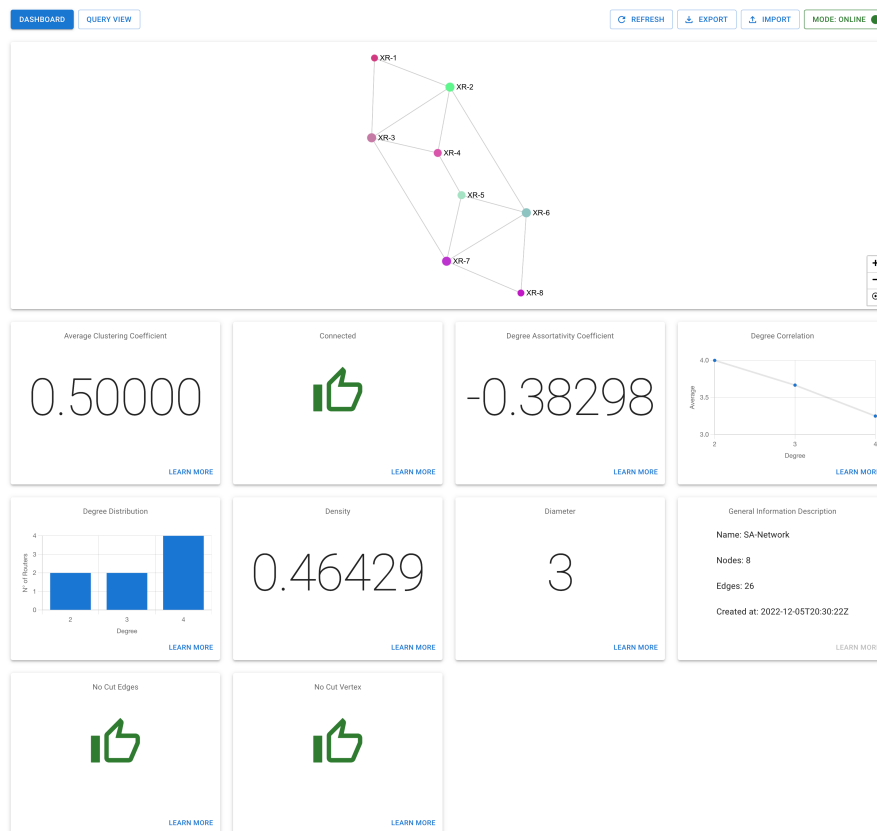


Figure 6.4: Graph Analyzer Frontend - Dashboard

## 6.5.2 Query View

The query view can be found in figure 6.5. On the left, the two current query options can be seen. The current graph is visualized above the two result tables for nodes and edges.

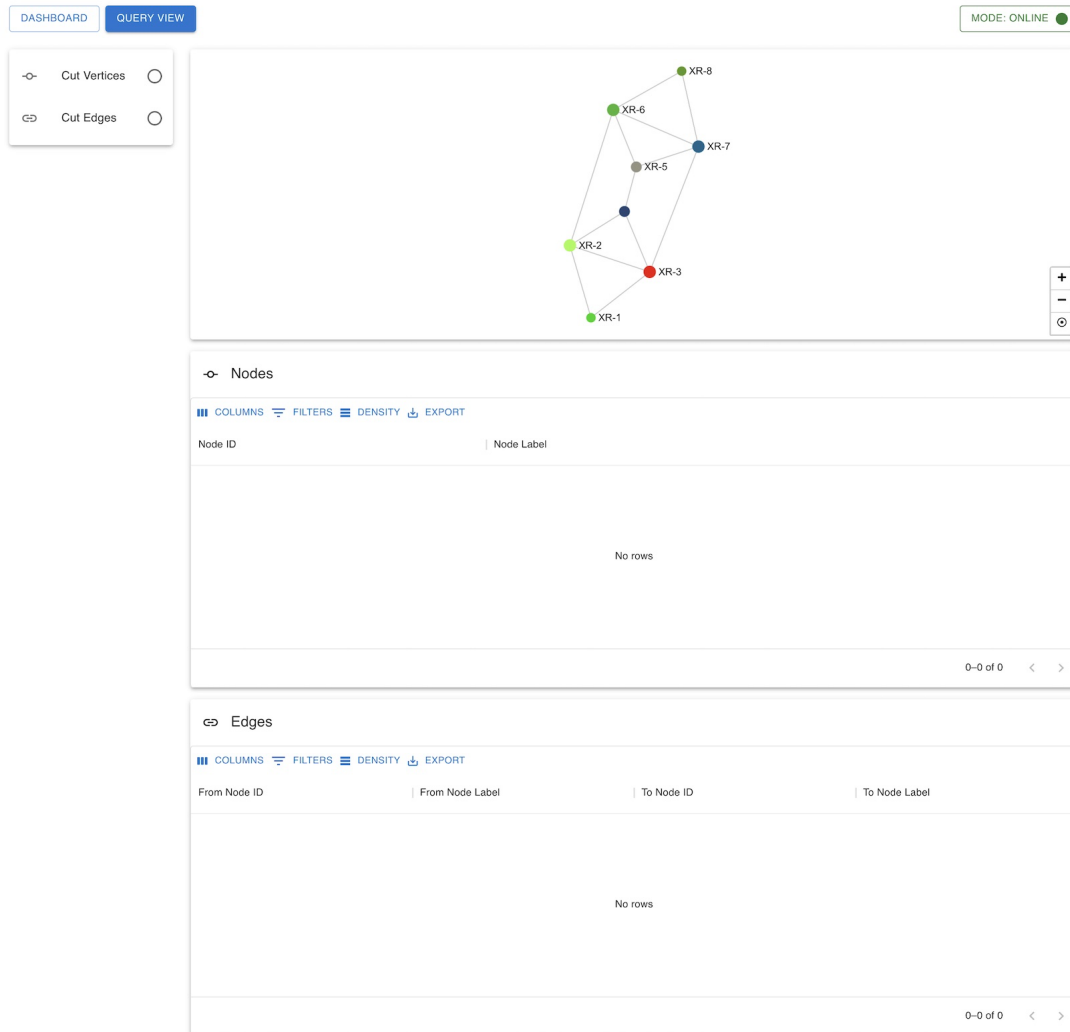


Figure 6.5: Graph Analyzer Frontend - Query View

### 6.5.3 Graph Visualization

Figures 6.6 and 6.7 show the graph visualization component in the Frontend.

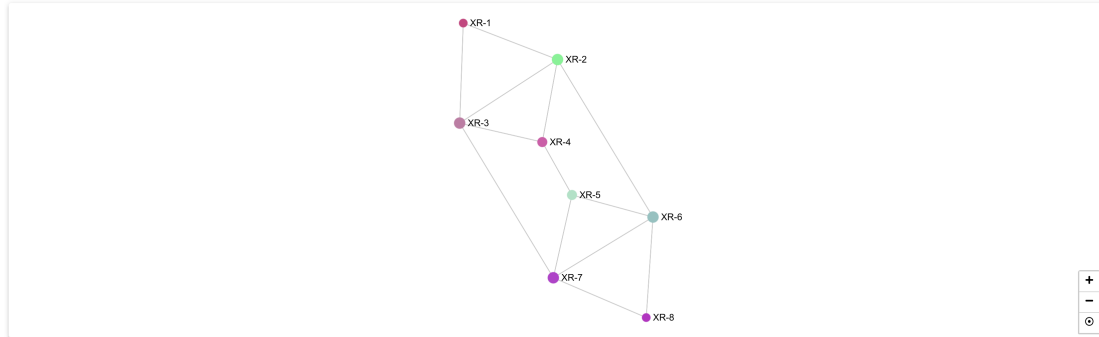


Figure 6.6: Graph Analyzer Frontend - Graph Visualization



Figure 6.7: Graph Analyzer Frontend - Graph Visualization With Selected Node

### 6.5.4 Average Clustering Coefficient

Figures 6.8 and 6.9 display the mandatory average clustering coefficient property and its description.

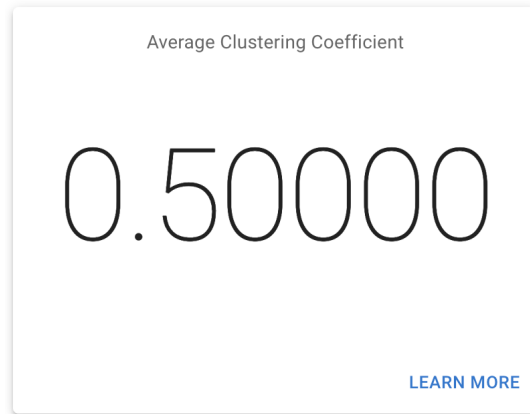


Figure 6.8: Graph Analyzer Frontend - Average Clustering Coefficient Tile

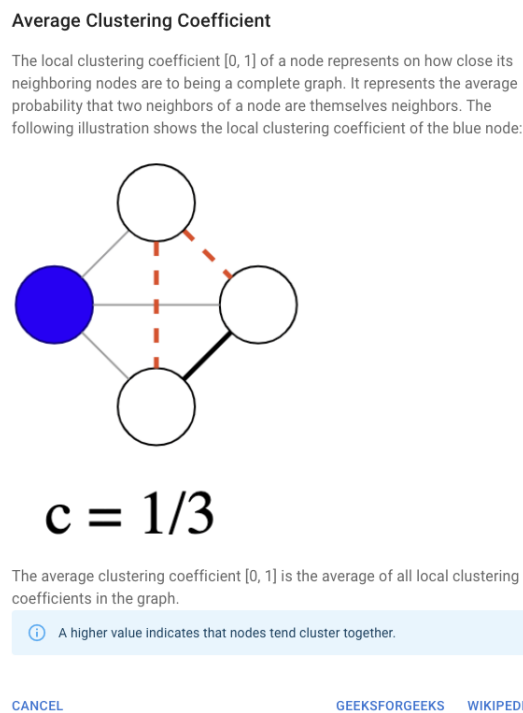


Figure 6.9: Graph Analyzer Frontend - Average Clustering Coefficient Description

### 6.5.5 Connected

Figures 6.10 and 6.11 display the optional connected property and its description.

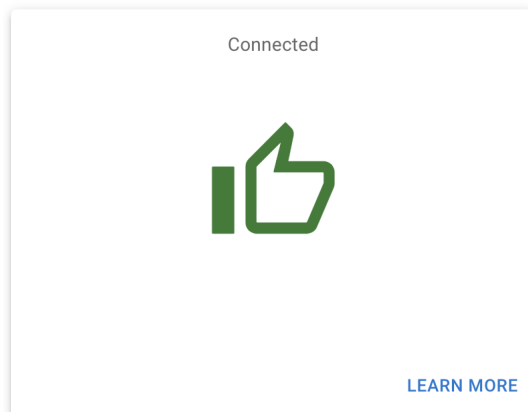




Figure 6.10: Graph Analyzer Frontend - Connected Tile

#### Connected

A connected graph describes that there is a **path**, with which a node can reach **every** other node of the graph. If this is not the case, the graph is disconnected.

-  Graph is connected
-  Graph is disconnected

[CANCEL](#)

[WIKIPEDIA](#)

[WOLFRAM MATHWORLD](#)

Figure 6.11: Graph Analyzer Frontend - Connected Description

## 6.5.6 Degree Assortativity Coefficient

Figures 6.12 and 6.13 display the mandatory degree assortativity coefficient property and its description.

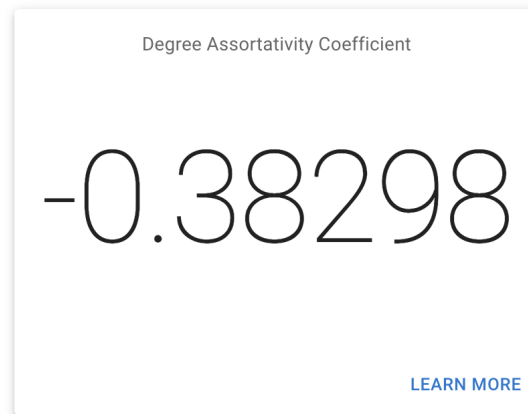


Figure 6.12: Graph Analyzer Frontend - Degree Assortativity Coefficient Tile

### Degree Assortativity Coefficient

The degree assortativity coefficient  $[-1, 1]$  is defined by the Pearson correlation between the degrees of pairs of linked nodes. The way in which nodes of one degree connect to nodes of another degree is known as assortativity. It represents the Degree-degree correlation.

- Positive  
A positive value indicates that high degree nodes tend to connect to nodes with high degree. Low degree nodes tend to connect to nodes with low degree.
- Negative  
A negative value indicates that high degree nodes tend to connect to nodes with low degree. Low degree nodes tend to connect to nodes with high degree.
- Zero  
A value near zero indicates that there is no tendency on how nodes connect to each other, they connect randomly.

 Computer networks tend to have a negative degree assortativity coefficient.

[CANCEL](#)

[NETWORKX](#) [WIKIPEDIA](#)

Figure 6.13: Graph Analyzer Frontend - Degree Assortativity Coefficient Description

## 6.5.7 Degree Correlation

Figures 6.14 and 6.15 display the mandatory degree correlation property and its description.

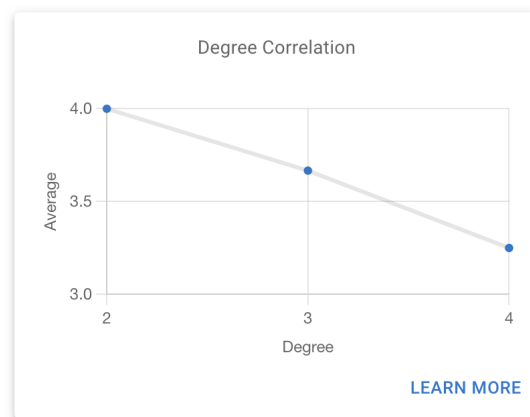





Figure 6.14: Graph Analyzer Frontend - Degree Correlation Tile

### Degree Correlation

The average degree connectivity correlation is the average nearest neighbor degree of nodes with degree  $k$ .

-  An increasing function indicates that the network is assortative. High degree nodes tend to connect to nodes with high degree. Low degree nodes tend to connect to nodes with low degree.
-  A decreasing function indicates that the network is disassortative. High degree nodes tend to connect to nodes with low degree. Low degree nodes tend to connect to nodes with high degree.
-  A constant function indicates that the network is uncorrelated. There is no clear tendency on how nodes connect to each other, they connect randomly.

 Computer networks tend to be disassortative.

CANCEL

NETWORKX UNICAMP

Figure 6.15: Graph Analyzer Frontend - Degree Correlation Description

### 6.5.8 Degree Distribution

Figures 6.16 and 6.17 display the mandatory degree distribution property and its description.

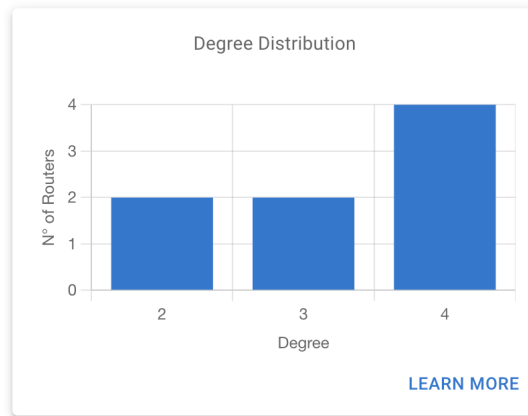


Figure 6.16: Graph Analyzer Frontend - Degree Distribution Tile

#### Degree Distribution

A degree is the number of directly connected other nodes (neighbors) a node has. The degree distribution shows a chart of the number of nodes that have a certain degree.

[CANCEL](#)

[UNICH](#) [WIKIPEDIA](#)

Figure 6.17: Graph Analyzer Frontend - Degree Distribution Description



## 6.5.9 Density

Figures 6.18 and 6.19 display the optional density property and its description.

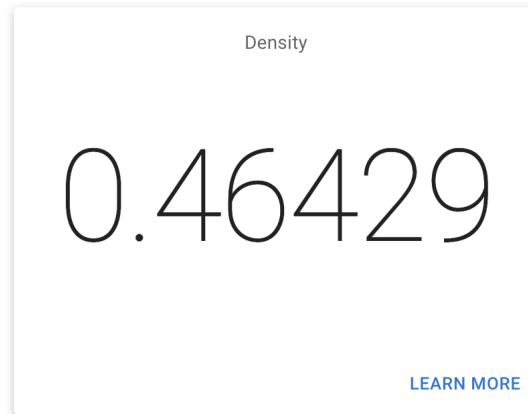


Figure 6.18: Graph Analyzer Frontend - Density Tile

### Density

Density  $[0, 1]$  is the ratio between the present edges and the maximum possible edges in a graph. Therefore, a complete graph (every node connects to every other node) has a value of 1, whereas the value for a large graph with only a few edges tends towards zero.

 Most large real-world networks often have a very low density.

[CANCEL](#)

[BAELDUNG](#) [WIKIPEDIA](#)

Figure 6.19: Graph Analyzer Frontend - Density Description

### 6.5.10 Diameter

Figures 6.20 and 6.21 display the mandatory diameter property and its description.

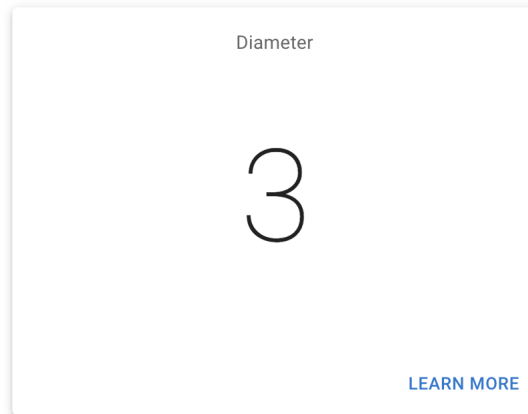


Figure 6.20: Graph Analyzer Frontend - Diameter Tile

#### Diameter

The diameter is the longest shortest path in a network. All shortest paths between all pairs of nodes need to be calculated, the longest of them will be the so-called diameter. The number of edges that compose the diameter act as its value.

 Currently the underlying graph is **unweighted** and **undirected**.

[CANCEL](#)

[WIKIPEDIA](#)

[WOLFRAM MATHWORLD](#)

Figure 6.21: Graph Analyzer Frontend - Diameter Description

### 6.5.11 General Information Description

Figure 6.22 displays the optional general information of the analyzed graph.

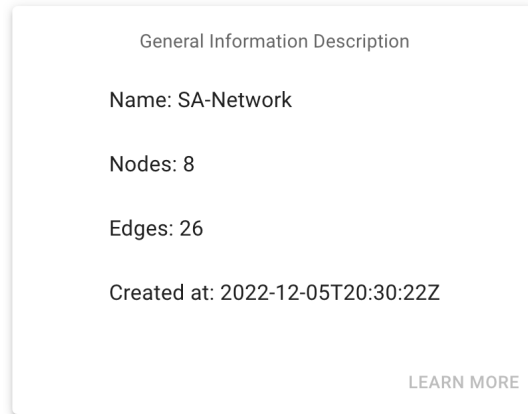


Figure 6.22: Graph Analyzer Frontend - General Information Tile

### 6.5.12 No Cut Edges

Figures 6.23 and 6.24 display the mandatory cut edges property and its description.

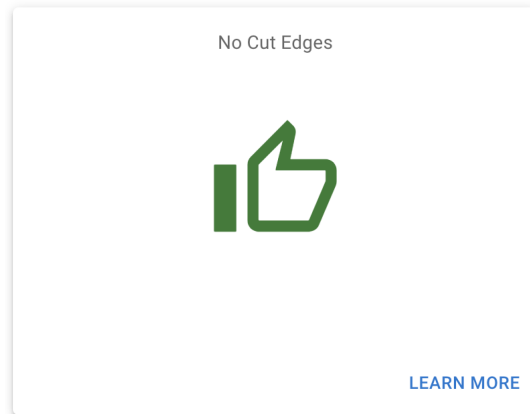





Figure 6.23: Graph Analyzer Frontend - No Cut Edges Tile

#### No Cut Edges

This graph property describes if there are any cut edges in the graph. A cut edge is a single edge whose removal would split the graph.

-  Graph has no cut edge
-  Graph has at least one cut edge

 It is **strongly** recommended not to have a single point of failure, such as a cut edge, in a network.

[CANCEL](#)

[WIKIPEDIA](#)

[WOLFRAM MATHWORLD](#)

Figure 6.24: Graph Analyzer Frontend - No Cut Edges Description

### 6.5.13 No Cut Vertex

Figures 6.25 and 6.26 display the optional cut vertex property and its description.

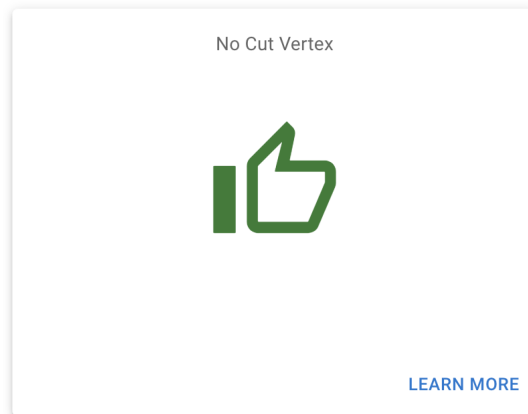





Figure 6.25: Graph Analyzer Frontend - No Cut Vertex Tile

#### No Cut Vertex

This graph property describes if there are any cut vertices (articulation points) in the graph. A cut vertex is a single node whose removal would split the graph.

-  Graph has no cut vertex
-  Graph has at least one cut vertex

 It is **strongly** recommended not to have a single point of failure, such as a cut vertex, in a network.

[CANCEL](#)

[GEEKSFORGEEEKS](#) [WIKIPEDIA](#)

Figure 6.26: Graph Analyzer Frontend - No Cut Vertex Description

# Chapter 7

## Quality Measures

### 7.1 Git Process

It was decided to use a multi-repo approach, which involves splitting the system's source code across multiple repositories.

#### 7.1.1 Workflow

Every task is present as an issue in Jira. The issue is attached to a corresponding application milestone. Every issue is equal to a branch, meaning that a separate branch is created for every task. When a task is completed, the feature branch is merged into the main branch. Before a review is requested, the assigned person should make sure that the feature branch incorporates the latest changes from the main branch and is therefore considered mergeable. If that has been done, a merge request is created and a reviewer is assigned. If the review passes, the reviewer approves the merge request and the feature branch is merged into the main branch. The feature branch is deleted as soon as the merge request has been merged.

#### 7.1.2 Code Review

As already described in section 7.1.1, merge requests are used in GitLab. To do this, a merge request is created and the other person (since the team is currently only composed of two people) is assigned as a reviewer. The reviewing person is then responsible for conducting the review and making suggestions for improvements. In the end, the merge request is approved and merged. GitLab features that support code reviews like threads and suggestions are used.

## 7.2 CI/CD

GitLab's CI/CD pipeline is used to keep code quality standards high and simplify processes.

### 7.2.1 API and Data Collector

The same pipeline is used for the Data Collector and API, as both are Golang-based applications. The following 4 stages are used for the API and Data Collector pipeline:

- lint
- test
- build
- release

Depending on whether the commit is in the main branch or not, other checks run in the pipeline. Execution of the API pipeline triggered by a tag is shown in figure 7.1.

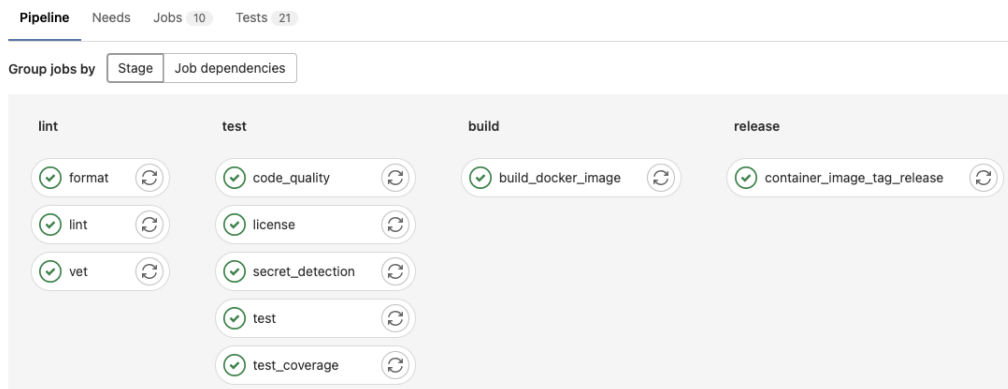


Figure 7.1: API Pipeline For Main Branch Tags

#### lint

During the lint stage, it is checked whether the code is correctly formatted and if there are any obvious flaws or bugs that can be discovered using static code analysis.

#### test

During the test stage, various checks are performed. GitLab-specific tests like secret detection and code quality are used. Other checks including the test suite, test coverage and verifying license compliance are also executed.

## build

During the build stage, the container image is built and pushed to the container repository.

## release

Depending on whether it is a commit in the main branch or a tag, the correct image tag is set during the release stage so that a release tag like `api:1.0.0` can be set during deployment.

## Artifacts

To support the reviewer, some artifacts created in the pipeline are displayed directly in the merge request. Thus, an initial assessment of the quality of the merge request is directly possible. An example of the displayed artifacts can be seen in figure 7.2.

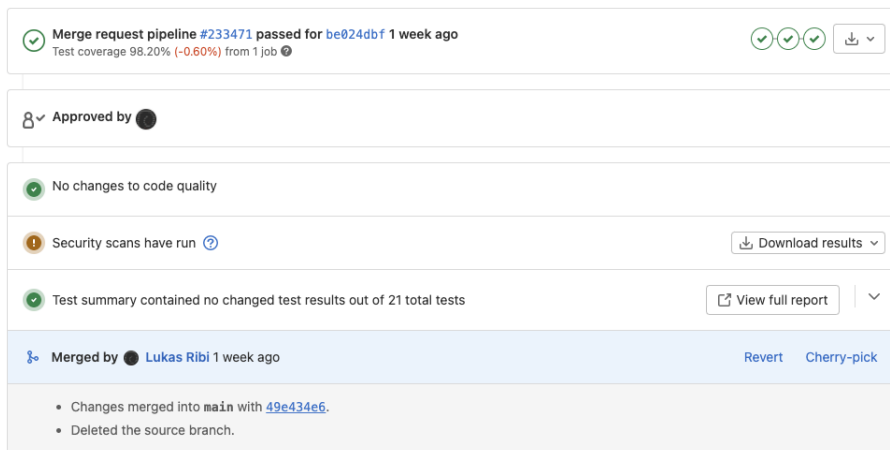


Figure 7.2: API Merge Request With Artifacts



## 7.2.2 Frontend

The following 5 stages are used for the Frontend pipeline:

- install
- lint
- test
- build
- release

Depending on whether the commit is in the main branch or not, other checks run in the pipeline. Execution of the Frontend pipeline triggered by a tag is shown in figure 7.3.

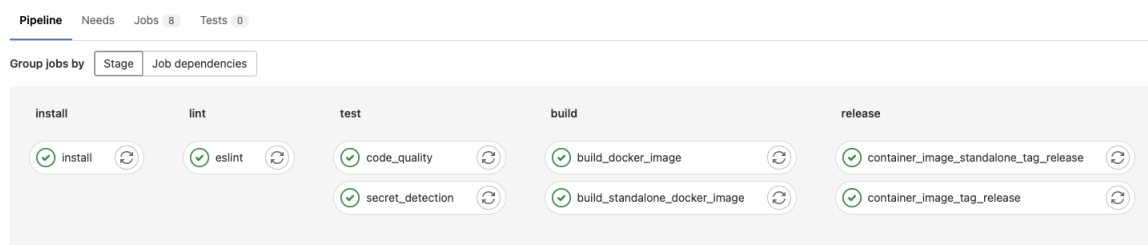


Figure 7.3: Frontend Pipeline For Main Branch Tags

### install

In this stage, the required npm modules are installed and cached for 30 minutes so that they do not need to be reinstalled for each stage.

### lint

During the lint stage, eslint <sup>1</sup> statically analyzes the code to find obvious problems.

### test

During the test stage, GitLab-specific checks like secret detection and code quality are executed.

### build

During the build stage, the container image is built and pushed to the container repository. There are two different builds for the Frontend. A standalone React app and a Micro Frontend app, which can be used for the Central Frontend.

---

<sup>1</sup><https://eslint.org>

## release

Depending on whether it is a commit in the main branch or a tag, the correct image tag is set during the release stage so that a release tag like `frontend:1.0.0` can be set during deployment.

## 7.3 Metric Tools

### 7.3.1 Test Coverage

As already mentioned, the test coverage is displayed in a merge request and is an important indicator during the review. The test-coverage visualization feature from GitLab also shows whether new lines of code are covered by a test case or not [38]. For GitLab to understand the code coverage report, the output from the `gotestsum`<sup>2</sup> library is converted to the required Cobertura format using `gocover-cobertura`<sup>3</sup>.

### 7.3.2 SonarQube

Since the SonarQube community edition has limited functionality, no continuous checks are done. The checks are done on a manual basis. An A grade was received for both the API and the Data Collector as seen in figure 7.4.

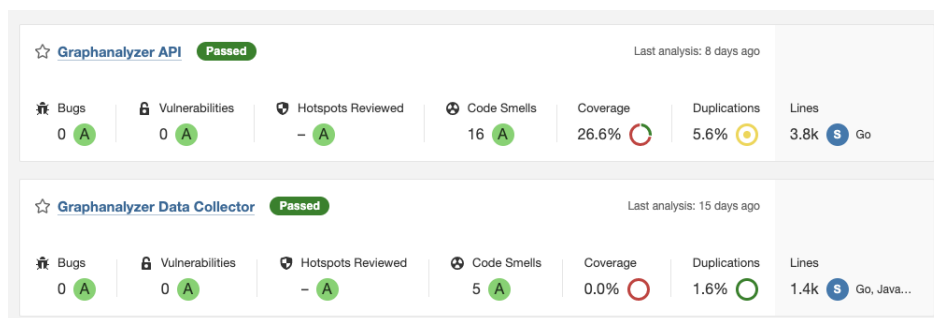


Figure 7.4: SonarQube Report For The Data Collector And API

<sup>2</sup><https://github.com/gotestyourself/gotestsum>

<sup>3</sup><https://github.com/t-yuki/gocover-cobertura>

# Chapter 8

## Conclusion

### 8.1 Outlook

The current implementation of the Graph Analyzer provides a solid foundation for future works in the area of network topology research. The system is designed in a way that allows for the integration of new properties with minimal effort. The Data Collector is also easily extendable to accommodate additional data sources. Additionally, the query view is constructed in a manner that facilitates easy extension. A follow-up bachelor thesis will use the developed system to research network model generation.

### 8.2 Limitations

Currently, the system is designed to handle undirected and unweighted graphs. This has consequences on the calculation of certain properties like the diameter. Weights are currently not considered when it is calculated. Adjusting the system for this use case is however possible with a reasonable amount of effort.

An optional use case involving the detection of graph communities was not pursued due to time constraints. A graph community is defined as a subset of nodes that have a high level of connectivity within the subset and a lower level of connectivity with nodes in other communities within the same graph. To effectively convey this information, it would be necessary to expand the graph visualization.

The current validation of user input is basic. There are opportunities to enhance this process, such as validating the GEXF schema in the Data Collector or comparing the provided JSON file with the desired structure during the import in the Frontend.

The current Helm chart only supports Jalapeño as a data source. This limitation is due to the primary focus on the internal operation of the system using Jalapeño.

**Part II**

**Project Documentation**

# Chapter 9

## Project Plan

### 9.1 Project Plan

#### 9.1.1 Development Process

For the project, the Rational Unified Process (RUP) is used for planning in the long term. For short-term planning within iterations, Scrum will be used. RUP splits the project into the following 4 phases:

#### 9.1.2 Phases

1. Inception: Vision, Initial risk assessment, Project description
2. Elaboration: Use-case model, Description of software architecture
3. Construction: Building the software
4. Transition: Rollout, Quality checks

### 9.1.3 Project Milestones

Progress is tracked via two milestone types, project and application. Project milestones are based on the project phases and show the progress of the project as a whole. Application milestones are more geared toward the actual development of the system. They track specific parts or features of the to-be-developed system. Seven project milestones are defined for the project, these are present as epics in the issue tracker. Epics are not exclusive to these milestones and can also contain issues that span multiple project phases. Project milestones are present as epics in the issue tracker, but no issues will be assigned to them. They are mainly there to give a better context to the roadmap. Table 9.1 lists all project milestones.

| <b>Milestone</b>              | <b>Planned</b>                       |
|-------------------------------|--------------------------------------|
| M1 - Project Plan             | Semester week 2 (27.09.2022)         |
| M2 - Requirements & Research  | Semester week 4 (11.10.2022)         |
| M3 - End of Elaboration Phase | Semester week 6 (25.10.2022)         |
| M4 - Architecture             | Semester week 7 (01.11.2022)         |
| M5 - Alpha                    | Semester week 10 (22.11.2022)        |
| M6 - Beta                     | Semester week 12 (06.12.2022)        |
| M7 - Final Submission         | End of semester week 14 (23.12.2022) |

Table 9.1: Project Milestones

### 9.1.4 Application Milestones

There are six application milestones defined. These application milestones also act as epics in the issue tracker and can be mapped to parts of the system. Table 9.2 lists all application milestones.

| <b>Milestone</b>                    | <b>Planned</b> |
|-------------------------------------|----------------|
| AM1 - Proof of Concept              | 18.10.2022     |
| AM2 - Prototype                     | 08.11.2022     |
| AM3 - Dashboard MVP                 | 29.11.2022     |
| AM4 - Dashboard Graph Visualization | 29.11.2022     |
| AM5 - Dashboard File-Based View     | 06.12.2022     |
| AM6 - Query View                    | 06.12.2022     |

Table 9.2: Application Milestones

### 9.1.5 Roadmap

The roadmap and schedule, which includes planned sprints and milestones, are presented in figures 9.1 and 9.2.

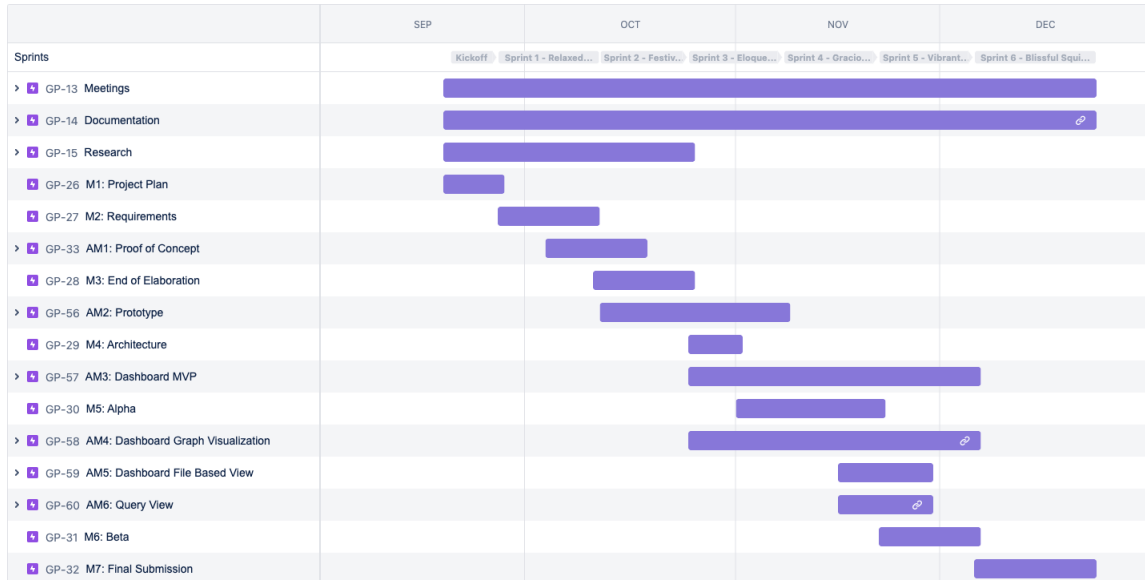


Figure 9.1: Roadmap

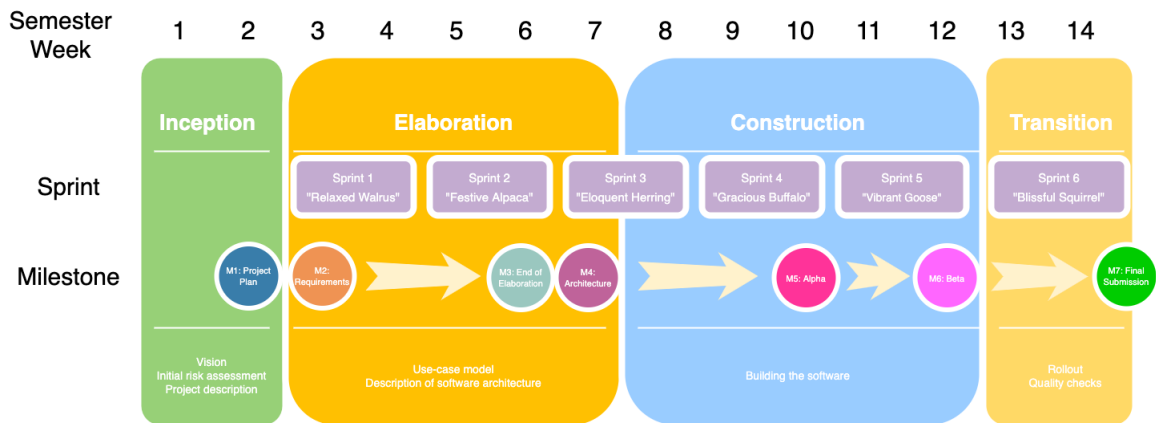


Figure 9.2: RUP Phases With Project Milestones And Sprints

### 9.1.6 Key Dates and Numbers

- Project start: 19.09.2022
- Project end: 23.12.2022 17:00
- Time budget: 480 hours (1 ECTS = 30 hours)
- Working days: Tuesday, Saturday, Sunday

## 9.2 Meetings

### 9.2.1 Status Meetings

It is planned that there will be weekly meetings with the advisors. They are scheduled for each Tuesday afternoon but can be skipped if there is no need for one (for example in the Construction phase). Physical presence at the meetings is preferred, but remote participation is possible if needed.

### 9.2.2 Scrum Meetings

A weekly scrum is done instead of a daily one. The reason is the part-time work schedule of the team members and that a big part of the work is planned to happen on the weekends.

In addition, there are previously documented status meetings with the advisors. Table 9.3 lists the planned Scrum meetings.

| Meeting              | Frequency                             | When   | Where      |
|----------------------|---------------------------------------|--|------------|
| Sprint Planning      | Every two weeks (before every sprint) | Tuesday, 15:00 - 18:00                                     | Rapperswil |
| Sprint Review        | Every two weeks (after every sprint)  | Tuesday, 15:00 - 18:00                                     | Rapperswil |
| Sprint Retrospective | Every two weeks (after every sprint)  | Tuesday, 15:00 - 18:00                                     | Rapperswil |
| Weekly Scrum         | Every week                            | Saturday, 15:00 - 16:00<br>(Backup: Sunday: 15:00 - 16:00) | Online     |
| Backlog Refinement   | Every other week (with Weekly Scrum)  | Saturday, 15:00 - 16:00<br>(Backup: Sunday: 15:00 - 16:00) | Online     |

Table 9.3: Scrum Meetings



## 9.3 Roles

The role assignment can be seen in table 9.4.

| <b>Role</b>   | <b>Member(s)</b>                     |
|---------------|--------------------------------------|
| Minute Keeper | Pascal Christen                      |
| Scrum Master  | Lukas Ribl                           |
| Product Owner | Pascal Christen                      |
| Developer     | Lukas Ribl, Pascal Christen          |
| Advisors      | Laurent Metzger, Severin Dellsperger |

Table 9.4: Scrum Roles

### 9.3.1 Details About The Assigned Roles

- Scrum roles (Scrum Master, Product Owner, Developer) as defined in Scrum
  - Managing the product backlog is the main responsibility of the product owner, however, the actual work concerning it is done collaboratively anyway.
- Minute keeper: Takes notes during the status and team meetings.

## 9.4 Risk Management

### 9.4.1 Risks

Legend:

- Likelihood: Rare, Unlikely, Possible, Likely, Certain
- Severity: Negligible, Marginal, Critical, Catastrophic
- Impact: Low, Medium, High, Very High

| <b>ID</b> | <b>Topic</b>      | <b>Description</b>   | <b>Likelihood</b> | <b>Severity</b> | <b>Impact</b> |
|-----------|-------------------|--|-------------------|-----------------|---------------|
| R01       | Knowledge         | The team members lack knowledge concerning the technical or theoretical aspects of the project. This can delay phases of the project and lead to missed milestones. Missing theoretical knowledge hugely influences the project as understanding is a key requirement for displaying the graph properties correctly. | Rare              | Critical        | Medium        |
| R02       | Performance       | It turns out that calculating certain graph properties of the network requires a lot of processing power or time. The calculation of other graph properties could be affected by this and the displayed graph properties are a mix of current and outdated data.   | Possible          | Marginal        | Medium        |
| R03       | Health and Safety | A team member gets sick or is involved in an accident. The member is unable to work for a certain period and the pressure on the other team member increases significantly. This can lead to project delays and not implemented features.  | Possible          | Critical        | High          |
| R04       | Communication     | There is no or insufficient communication inside the team or with the advisors. A misconception could be the result of the lack of communication. Therefore, the final result may not have the desired outcome.  | Rare              | Marginal        | Low           |

|     |                  |   |          |          |        |
|-----|------------------|---|----------|----------|--------|
| R05 | Scope            | There are too many mandatory requirements that can not be fulfilled, or the existing ones are too big for the scope of this thesis. This can lead to not implemented features   | Unlikely | Critical | Medium |
| R06 | Technology       | In the middle of the construction phase, the used technology stack needs to be changed because of technical or personal limitations. The final result is at high risk because of this huge conceptional change.   | Rare     | Critical | Medium |
| R07 | External Hazards | There is a non-influenceable outside event happening that prevents work on the project as is. This could be a winter storm, energy blackout or even war for example. Work on the project is not possible as done previously and the final form of the project will vary greatly from the planned goals. | Possible | Critical | High   |

Table 9.5: Risk Management

## 9.4.2 Risk Matrix

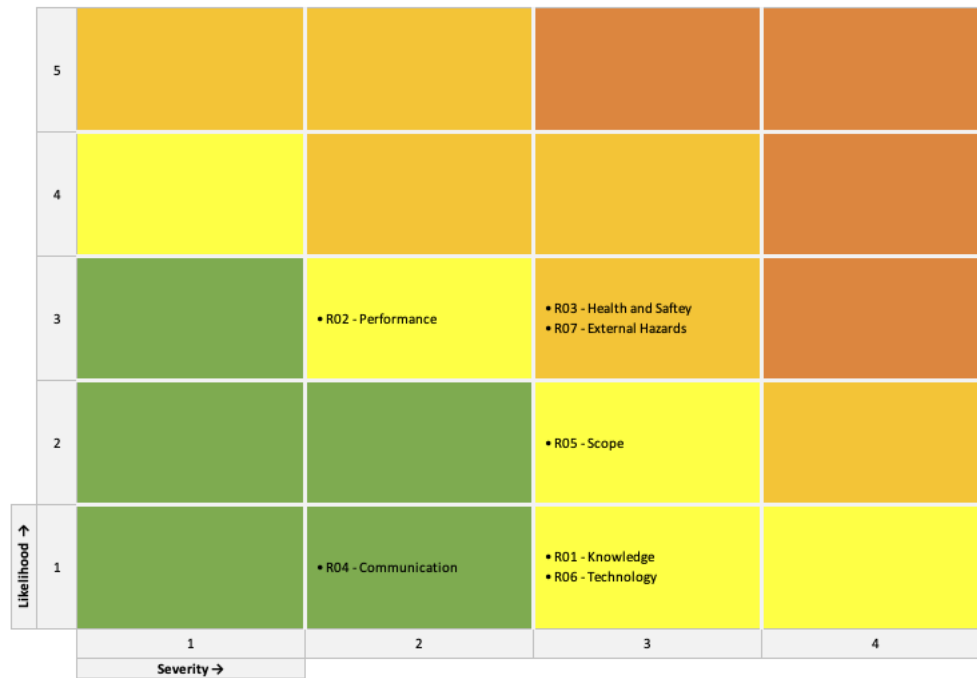


Figure 9.3: Risk Matrix

## 9.4.3 Risk Management and Mitigation

Besides the external hazard risk, every mentioned risk in table 9.5, especially those with "medium" and "high" impact according to the risk matrix in figure 9.3, can be mitigated to a certain degree by engaging via direct communication, frequent meetings and status updates. The mitigations are listed in table 9.6.

| ID  | Mitigation / Action  |
|-----|--|
| R01 | During the elaboration phase, the research (book) was able to be done and the prototype with all mandatory graph properties was implemented.   |
| R02 | During the elaboration phase benchmark were done. Neither the backend nor the graph database showed very slow performance under real data conditions. With the project advisors, it was agreed upon that the system is for "educational research" and not a live monitoring system, where the properties needed to be calculated instantly. It can thus be concluded that the likelihood and severity decrease to Unlikely and Negligible. |
| R03 | There is nothing one can do about external hazard risks (besides good planning).   |
| R04 | After 5 weeks of working together, no obvious lack of communication or misunderstandings occurred. Therefore, the severity is decreased to Negligible.   |

|     |  |
|-----|--|
| R05 | During the elaboration phase, research was performed and a prototype with all mandatory graph properties was implemented. In this case, the severity decreased to marginal.                |
| R06 | Technologies were evaluated with appropriate detail and it is not expected anything breaks or does not fulfill defined requirements. Considering that, the severity decreases to marginal. |
| R07 | As mentioned before, there is nothing one can do about external hazard risks (besides good planning).  |

Table 9.6: Risk Mitigation

## 9.5 Planning Tools

### 9.5.1 Issue Tracker

Jira is used to track issues and handle project management as a whole. From experience, it was known that GitLab’s issue tracker needs a lot of workarounds for a Scrum workflow. Thus, it was decided to use a tool that supports it natively, namely Jira.

### 9.5.2 Time Tracker

Time tracking is done directly in Jira on tickets. Every team member is responsible for tracking his spent time on the correct ticket.

**Part III**  
**Appendix**

# Personal Reports

## 9.6 Lukas Ribí

I am pretty happy with how this thesis turned out. Collaboration with Pascal went without a hitch, this was already the case in previous projects. The topic that we chose, turned out to be the right choice. We wanted to do our thesis about a topic that we were not familiar with yet but still interested us. As we both already work professionally, we wanted to do something that included a research part on an unfamiliar topic. The initially given requirements were reasonable and left us a lot of room for our ideas. It was great that the advisors gave us a lot of freedom over how we implemented the system. They were responsive and helpful when we had questions or needed some advice. I was also able to use some tools in the implementation that I had on my bucket list. Graph databases, Redux, gRPC and Go channels were new tools that I was able to use as part of this thesis. Not continuing work on the documentation during the implementation phase is something that I would classify as a mistake that we made. In the bachelor thesis, I will aim to improve on this.

## 9.7 Pascal Christen

Overall, I believe our thesis was a complete success due to the strong communication and teamwork between Lukas and me. We were able to utilize each other's strengths effectively and had productive discussions with our advisors. The topic of our thesis was engaging and I enjoyed the challenge of balancing scientific research with implementing our solution. The tools we used helped streamline our work and make it more efficient. If I had to change one thing, I would have utilized the "Toogle Track" tool for more extensive time tracking earlier on in the process. During this time, I had the opportunity to learn about a diverse range of new technologies or further expand my understanding of those I was already familiar with.

I want to express my gratitude to Lukas for the excellent collaboration and to Laurent Metzger and Severin Dellsperger for their valuable support.

# Time Tracking Report

Figure 9.4 shows the time evaluation for the project. Practically all of the available 480 hours were used. The time was divided equally between the two team members.

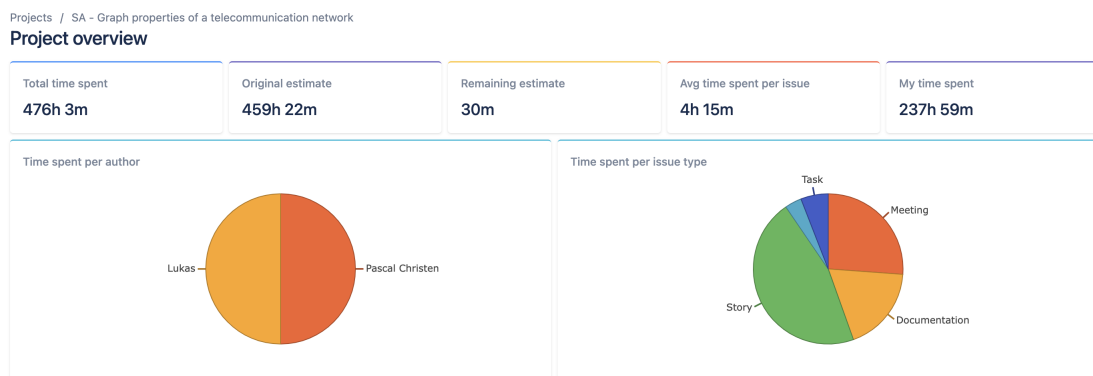


Figure 9.4: Time Tracking Report



# Meeting Minutes

## Weekly Meeting SW1 - 20.09.2022 / 16:00

### Participants

- Laurent Metzger
- Severin Dellsperger (Remote)
- Lukas Ribi
- Pascal Christen

### Agenda

- Introduction to SA

### Notes

- Integration into Central Frontend
- Goal is a working system
- Mocking a large network is out of scope for this SA
- Multiple graph properties needed:
  - Degree distributions
  - Degree correlations
  - Cut edges
  - Assortativity coefficient
  - Clustering coefficient
  - Network diameter
  - ...
- Book: A First Course in Network Science
- No RBAC needed
- Filtering (e.g. Router with only 2 edges, ...) (maybe optional)

### To-Do

- Create a project plan
- Start research on graph properties
- Start work on use cases
- Grant GitLab permissions to supervisors

## Weekly Meeting SW2 - 27.09.2022 / 16:00

### Participants

- Severin Dellsperger (Remote)
- Lukas Ribl
- Pascal Christen

### Agenda

- Demonstration of ArangoDB, Central Frontend, Grafana (optional)
- Show current status

### Notes

- No need to use an existing CI (Corporate Identity) & CD (Corporate Design)
- Use cases:
  - Split graph properties into separate use cases
  - Visualization of the graph is mandatory (no need to map the location)
  - No need to add the Central Frontend to the use case diagram
- NFR:
  - Possibility to have 10'000 nodes in a sparse network
  - Frontend is accessible on mobile and desktop browsers

### To-Do

- VPN access for INS (Severin)
- Deploy Jalapeño API Gateway (Severin)
- Network screenshot (Severin)
- Create a use case diagram and add a brief description
- Add alpha milestone (SW10)
- Ask Laurent Metzger about documents we need to sign

## **Weekly Meeting SW3 - 04.10.2022 / 16:00**

### **Participants**

- Lukas Ribi (Remote)
- Pascal Christen (Remote)
- Laurent Metzger (Remote)

### **Agenda**

- Show current status (Jira, Documentation)
- Question about documents we have to sign

### **Notes**

- Idea: Compare different networks (Sunrise, Swisscom, UPC) - Export/Import. No need to be implemented but keep it in mind
- Milestone 1 reached (FR/NFR) and approved
- There is no need to sign a document right now (the license will be open source)

### **To-Do**

- -

## Weekly Meeting SW4 - 12.10.2022 / 09:00

### Participants

- Laurent Metzger
- Severin Dellsperger (Remote)
- Lukas Ribi
- Pascal Christen

### Agenda

- Current status
- Architecture Q&A
- Bug in Jalapeño

### Notes

- Architecture:
  - Query view will be a fundamental component of this system (BA/INS/another thesis)
  - Therefore a Jalapeño independent GraphDB is ok
  - No need to have "live updates" in the Frontend. It is an analysis tool, not a monitoring system
  - Calculating the properties on demand is ok in the beginning. Outlook: Tasks like caching or auto-updates can be implemented in another thesis
  - No need to store link properties (ASN, Area id, ...) in the GraphDB for now
  - Links can be seen as undirected in this SA
  - Both advisors like our architectural approach

### To-Do

- Severin will debug Jalapeño

## Weekly Meeting SW5 - 18.10.2022 / 16:00

### Participants

- Laurent Metzger
- Severin Dellsperger (Remote)
- Lukas Ribl
- Pascal Christen

### Agenda

- Current status
  - Wireframes
  - GraphDB
  - API
- Open bugs

### Notes

- Query-view - accordion may be needed
- GraphDB (Licenses): Neo4j - ok
- API: Rest - ok

### To-Do

- Severin will provide network logins to test gRPC subscriptions
- Severin will update us regarding the Jalapeño link bug

## Weekly Meeting SW6 - 25.10.2022 / 16:00

### Participants

- Laurent Metzger
- Severin Dellsperger (Remote)
- Lukas Ribi
- Pascal Christen

### Agenda

- Current status
  - Neo4j - OK
  - JS graph library - OK
  - Architecture - OK
  - API - OK
- Deployment
  - Single Helm chart
  - No need to deploy it with CI/CD
- Open bugs
  - gRPC subscription for LsNodeEdges will be implemented in 3 weeks by the INS
- Next week?
  - No meeting due to Allerheiligen

### Notes

- Degree correlations vs Assortativity coefficient - Slides from Laurent
- General Project: On track and looks good to the supervisors

### To-Do

- Deploy k8s Cluster - Severin
- Send the graph properties document to us - Laurent

## Weekly Meeting SW8 - 09.11.2022 / 15:00

### Participants

- Laurent Metzger (Remote)
- Severin Dellsperger (Remote)
- Lukas Ribi (Remote)
- Pascal Christen (Remote)

### Agenda

- Current status
  - C4 Model - ok
  - Data Collector - ok
  - deployment (helm) - ok
  - API - ok
- Open bugs
  - gRPC subscription for LsNodeEdges - PR open. Should be done in 1 week
- BA - Follow-up thesis possible. We will discuss it in the next meeting.

### Notes

- Pagination - Use it if needed and when it makes sense

### To-Do

- -



## Weekly Meeting SW9 - 15.11.2022 / 16:00

### Participants

- Laurent Metzger
- Severin Dellsperger
- Lukas Ribl
- Pascal Christen

### Agenda

- Current status
  - Frontend - ok
- gRPC subscriptions
  - Severin demonstrated the gRPC subscription by taking up/down routers/links
- BA - Follow-up thesis possible. We will discuss it in the next meeting.

### Notes

- Presentation in the last week (in English)

### To-Do

- -

## **Weekly Meeting SW11 - 29.11.2022 / 16:00**

### **Participants**

- Severin Dellsperger
- Lukas Ribl
- Pascal Christen

### **Agenda**

- Show Current status to Severin

### **Notes**

- -

### **To-Do**

- -

## Weekly Meeting SW12 - 06.12.2022 / 16:00

### Participants

- Laurent Metzger
- Severin Dellsperger (Remote)
- Lukas Ribl
- Pascal Christen

### Agenda

- Show Current status
  - All good
- Follow-up thesis (BA)
  - Not ready now. Laurent Metzger will have a meeting with Cisco later this week
- Central Frontend (CSS)
  - Because of some problems (CSS integration, Variables, ...) the Central Frontend is not mandatory for the final project.

### Notes

- Presentation will be held in the last week (INS & Cisco)

### To-Do

- Laurent Metzger will send us a brief description of the BA thesis until Monday morning.

## **Weekly Meeting SW13 - 13.12.2022 / 16:00**

### **Participants**

- Laurent Metzger (Remote)
- Severin Dellsperger (Remote)
- Lukas Ribi
- Pascal Christen

### **Agenda**

- Feedback Graph properties
- Question about last week (presentation)

### **Notes**

- Maybe public deployment in January (for Cisco, Swisscom, ...)

### **To-Do**

- Sign document (Laurent)
- Provide feedback for Graph properties (Laurent)

## **Weekly Meeting SW14 - 20.12.2022 / 16:00**

### **Participants**

- Laurent Metzger (Remote)
- Severin Dellsperger (Remote)
- Lukas Ribl
- Pascal Christen
- Viewers from the INS (Presentation)

### **Agenda**

- Presentation
- Feedback

### **Notes**

- Cooperation was good from both sides. And it was fun working with us. The supervisors are very satisfied with what has been achieved.
- A date for the discussion of the bachelor thesis was agreed on

### **To-Do**

- -

# Bibliography

- [1] F. Menczer, S. Fortunato, and C. A. Davis, *A First Course in Network Science*. Cambridge: Cambridge university press, 2020, ISBN: 978-1-108-47113-8.
- [2] J. W. ESSAM and M. E. FISHER, “Some Basic Definitions in Graph Theory,” *Reviews of Modern Physics*, vol. 42, no. 2, pp. 271–288, 1970-04-01. DOI: 10.1103/RevModPhys.42.271. [Online]. Available: <https://link.aps.org/doi/10.1103/RevModPhys.42.271> (visited on 2022-11-21).
- [3] L. Tian, A. Bashan, D.-N. Shi, and Y.-Y. Liu, “Articulation points in complex networks,” *Nature Communications*, vol. 8, no. 1, p. 14223, 1 2017-01-31, ISSN: 2041-1723. DOI: 10.1038/ncomms14223. [Online]. Available: <https://www.nature.com/articles/ncomms14223> (visited on 2022-10-15).
- [4] “gRPC Web Roadmap.” (2022-10-21), [Online]. Available: <https://github.com/grpc/grpc-web> (visited on 2022-10-21).
- [5] “Gofiber/fiber.” (2022-10-21), [Online]. Available: <https://github.com/gofiber/fiber#readme> (visited on 2022-10-21).
- [6] “Swag Declarative Comments Format.” (2022-10-21), [Online]. Available: <https://github.com/swaggo/swag> (visited on 2022-10-21).
- [7] “Adopting and Developing BSL Software,” MariaDB. (2022-10-21), [Online]. Available: <https://mariadb.com/bsl-faq-adopting/> (visited on 2022-10-21).
- [8] “Redis Licensing Overview,” Redis. (2022-10-21), [Online]. Available: <https://redis.com/legal/licenses/> (visited on 2022-10-21).
- [9] “ArangoDB License.” (2022-10-21), [Online]. Available: <https://github.com/arangodb/arangodb> (visited on 2022-10-21).
- [10] “Containers, the GPL, and copyleft: No reason for concern — Opensource.com.” (2022-10-21), [Online]. Available: <https://opensource.com/article/18/1/containers-gpl-and-copyleft> (visited on 2022-10-21).
- [11] L. Oberhuber and D. Gajic, “Central Frontend for Segment Routing Applications (Folgearbeit),” other, OST Ostschweizer Fachhochschule, 2022. [Online]. Available: <https://eprints.ost.ch/id/eprint/1053/> (visited on 2022-10-21).
- [12] “The C4 model for visualising software architecture.” (2022-10-22), [Online]. Available: <https://c4model.com/> (visited on 2022-10-21).

- [13] “Paul M. Jones — Action Domain Responder.” (2022-10-22), [Online]. Available: <http://pmjones.io/adr/> (visited on 2022-10-21).
- [14] “Cobra.Dev,” Cobra.Dev. (2022-10-22), [Online]. Available: <https://cobra.dev/> (visited on 2022-10-21).
- [15] “Listing graphs - Neo4j Graph Data Science,” Neo4j Graph Data Platform. (2022-11-15), [Online]. Available: <https://neo4j.com/docs/graph-data-science/2.2/graph-list/> (visited on 2022-11-14).
- [16] “Local Clustering Coefficient - Neo4j Graph Data Science,” Neo4j Graph Data Platform. (2022-11-22), [Online]. Available: <https://neo4j.com/docs/graph-data-science/2.2/algorithms/local-clustering-coefficient/> (visited on 2022-11-21).
- [17] “Weakly Connected Components - Neo4j Graph Data Science,” Neo4j Graph Data Platform. (2022-11-25), [Online]. Available: <https://neo4j.com/docs/graph-data-science/2.2/algorithms/wcc/> (visited on 2022-11-24).
- [18] “Degree Centrality - Neo4j Graph Data Science,” Neo4j Graph Data Platform. (2022-11-25), [Online]. Available: <https://neo4j.com/docs/graph-data-science/2.2/algorithms/degree-centrality/> (visited on 2022-11-24).
- [19] “Undirected Graph - gonum.org/v1/gonum/graph/simple - Go Packages.” (2022-11-25), [Online]. Available: <https://pkg.go.dev/gonum.org/v1/gonum/graph/simple#UndirectedGraph> (visited on 2022-11-24).
- [20] “Correlation - gonum.org/v1/gonum/stat - Go Packages.” (2022-11-25), [Online]. Available: <https://pkg.go.dev/gonum.org/v1/gonum/stat#Correlation> (visited on 2022-11-24).
- [21] “Mean - gonum.org/v1/gonum/stat - Go Packages.” (2022-11-25), [Online]. Available: <https://pkg.go.dev/gonum.org/v1/gonum/stat#Mean> (visited on 2022-11-24).
- [22] “All Pairs Shortest Path - Neo4j Graph Data Science,” Neo4j Graph Data Platform. (2022-11-25), [Online]. Available: <https://neo4j.com/docs/graph-data-science/2.2/alpha-algorithms/all-pairs-shortest-path/> (visited on 2022-11-24).
- [23] “Memory Estimation - Neo4j Graph Data Science,” Neo4j Graph Data Platform. (2022-11-25), [Online]. Available: <https://neo4j.com/docs/graph-data-science/2.2/common-usage/memory-estimation/> (visited on 2022-11-24).
- [24] “DijkstraAllPaths - gonum.org/v1/gonum/graph/path - Go Packages.” (2022-11-25), [Online]. Available: <https://pkg.go.dev/gonum.org/v1/gonum/graph/path#DijkstraAllPaths> (visited on 2022-11-24).
- [25] R. E. Tarjan, “A note on finding the bridges of a graph,” *Information Processing Letters*, vol. 2, no. 6, pp. 160–161, 1974-04-01, ISSN: 0020-0190. DOI: 10.1016/0020-0190(74)90003-9. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0020019074900039> (visited on 2022-11-24).

- [26] R. Tarjan, “Depth-First Search and Linear Graph Algorithms,” *SIAM Journal on Computing*, 2006-07-13. DOI: 10.1137/0201010. [Online]. Available: <https://epubs.siam.org/doi/10.1137/0201010> (visited on 2022-11-24).
- [27] M. R. Bala and M. K. K. Kaswan, “Strategy Design Pattern,” *Global Journal of Computer Science and Technology: C Software & Data Engineering*, vol. 14, no. 6, 2014, ISSN: 0975-4172.
- [28] “Request Service — Jalapeño API Gateway.” (2022-11-25), [Online]. Available: <https://jalapeno-api-gateway.github.io/jagw/docs/api/request-service/> (visited on 2022-11-24).
- [29] “Subscription Service — Jalapeño API Gateway.” (2022-11-25), [Online]. Available: <https://jalapeno-api-gateway.github.io/jagw/docs/api/subscription-service/> (visited on 2022-11-24).
- [30] “Effective Go - The Go Programming Language.” (2022-11-25), [Online]. Available: [https://go.dev/doc/effective\\_go](https://go.dev/doc/effective_go) (visited on 2022-11-24).
- [31] “GEXF File Format.” (2022-11-25), [Online]. Available: <https://gexf.net/> (visited on 2022-11-24).
- [32] “Code Generation — Redux Toolkit.” (2022-11-25), [Online]. Available: <https://redux-toolkit.js.org/rtk-query/usage/code-generation> (visited on 2022-11-24).
- [33] “Layout-forceatlas2,” Graphology. (2022-11-22), [Online]. Available: <https://graphology.github.io/standard-library/layout-forceatlas2.html> (visited on 2022-11-21).
- [34] “Line Chart — Chart.js.” (2022-11-22), [Online]. Available: <https://www.chartjs.org/docs/latest/charts/line.html> (visited on 2022-11-21).
- [35] “Bar Chart — Chart.js.” (2022-11-22), [Online]. Available: <https://www.chartjs.org/docs/latest/charts/bar.html> (visited on 2022-11-21).
- [36] “Configure a Neo4j Helm deployment - Operations Manual,” Neo4j Graph Data Platform. (2022-11-15), [Online]. Available: <https://neo4j.com/docs/operations-manual/5/kubernetes/configuration/> (visited on 2022-11-15).
- [37] “Docker Multi-stage builds,” Docker Documentation. (2022-12-20), [Online]. Available: <https://docs.docker.com/build/building/multi-stage/> (visited on 2022-10-10).
- [38] “Test coverage visualization — GitLab.” (2022-12-22), [Online]. Available: [https://docs.gitlab.com/ee/ci/testing/test\\_coverage\\_visualization.html](https://docs.gitlab.com/ee/ci/testing/test_coverage_visualization.html) (visited on 2022-12-22).