Term Project Green Routing

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

Autumn Term 2021

Authors:	Jonas Hauser, Pascal Schlumpf
Advisor:	Prof. Laurent Metzger
Co-Advisors:	Severin Dellsperger, Julian Klaiber
Project Partner:	Cisco Systems represented by Francois Clad

Abstract

Traditional routing protocols and techniques are often used in today's networks, and their basics were generally established before the millennium. In recent years, the network area has not experienced the same level of fast transformation as other IT industries. With the development of the digital world and the introduction of new industries and technologies like 5G and cloud computing, the volume of data transferred through networks today is massive and will continue to expand in the future. Modern networks must not only deal with an unprecedented amount of data transmissions, but many new requirements have emerged in order to meet client demands. In our time with climate change a new requirement on the energy efficiency of routing was developed.

Ecological aspects are becoming increasingly important in our world. The latest estimates for the ICT sector indicate emissions of around 1.4Gt of CO_2 per year. Internet backbone networks are responsible for six percent of this ecological footprint. The growing bandwidth creates new opportunities to consider other metrics and aspects in addition to the traditional ones which mostly only tend to use more and more bandwidths. A new possibility of this is to measure the energy consumption of ASR 9000 routers and to draw conclusions for the definition of the path.
This thesis is to look for a solution to implement a green routing approach, where in a network the most ecological paths are to be computed. The solution should be able to compute paths efficiently underlying a defined green index based on sensor data from routers. The calculated paths should be meaningful on the basis of power consumptions on a specified time range to prevent possible route flapping.

Through the thesis a first approach to the definition of the green index could be defined by setting paths with the cumulative least electricity consumption over the whole route from the source to the destination router. The overlying calculation of the best paths based on the green index could be implemented by a Green SR-App software in form of a REST-API. This backend API, which is written in modern Golang, can synchronize all network data via the Jalapeño API Gateway, process it, store it optimally in the sense of statistical purposes and then calculate the best path over a predefined period of sensor data based on Dijkstra's algorithm. The software is designed to be very performant despite very large networks of up to 1000 routers with many times more links in between. High Quality, mature scalability and a proven architecture through a Domain Driven Design have been deliberately written, because this thesis is only the beginning of larger extensions.

Management Summary

Initial Situation

Traditional routing protocols and techniques are often used in today's networks, and their specifications were generally established before the millennium. In recent years, the network area has not experienced the same level of fast transformation as other IT industries. With the advancement of the digital world and the introduction of new industries and technologies such as 5G and cloud computing, the volume of data transferred through networks today is massive and will continue to expand in the future. Modern networks must not only deal with an unprecedented amount of data transmissions, but many new requirements have emerged in order to meet demands of network users. In times of climate change a new requirement regarding the energy efficiency of routing was formed.

Ecological aspects are becoming increasingly important in our world. The latest estimates for the ICT sector indicate emissions of around 1.4Gt of CO_2 per year. Internet backbone networks are responsible for six percent of this ecological footprint. The increasing bandwidth creates new opportunities to consider other metrics and aspects in addition to the traditional ones which mostly only tend to use more and more bandwidths. Segment routing gives the possibility to route packages in new ways, which were not possible before. If we combine the desire to route more environment friendly with the possibility to define our own routes, we end up with the theme of our green routing thesis.

This thesis is to prototype a green routing approach, in which the most ecological paths in a network are computed. The solution should be able to compute paths efficiently by using a defined green index based on sensor data from routers. The calculated paths should be meaningful on the basis of power consumption over a specified time range to prevent possible route flapping.

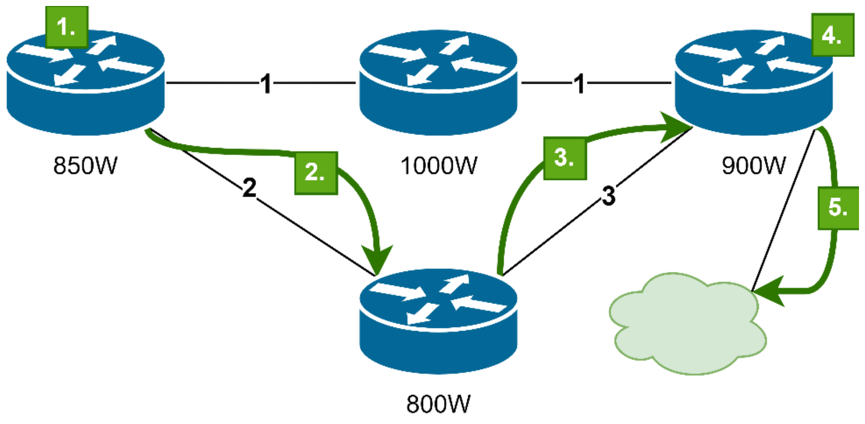


Figure Management Summary 1.1: Green Route example

Procedures and Technologies

In an initial phase, the requirements and use cases were developed together with our supervisors. This created the context where this project took place in. In regard to non-functional requirements, many constraints regarding performance, scalability and quality were defined. These had a significant impact on the overall architectural and design decisions.

In the elaboration phase we defined the whole architecture and ended up with the domain driven design approach. Together with the requirement for a cloud native compliant software, the direction was already clear. In this phase the frameworks were decided on, with a focus on performance. This was also the reason for the decision for a MariaDB over PostgreSQL. We also defined in what way the green index would be defined and in what manner the green route would be calculated.

The construction phase was reserved for the implementation of the app. The project was developed in Golang, with an API built on top of the Gin Web Framework. To get the data from the network, we also needed an interface to the Jalapeño API Gateway to synchronize the topology and request the telemetry data for the calculation. The data was stored in the MariaDB with the object relation mapper gorm. The main function of the backend is to receive an incoming node as well as an outgoing node and to calculate and return the greenest path between those two nodes.

It was decided that this project would be continued in the bachelor thesis with the consent of the supervisor. This had an impact on the further development of the application. The focus was set on more stability and an easier expandability as well as performance to lay a solid foundation for the continuation of the project.

Results

We were able to fulfill all the mandatory use cases, which include the ability to calculate green routes based on the synchronized data from the Jalapeño API Gateway. We developed an API which can verify the stored nodes and links, and allow for the calculation of a path upon request. For a network with 1000 nodes and 25000 links the synchronization is completed in less than 2.5 seconds on average.

We also mocked the lowest possible application layer per area to be able to write integration tests without having an in-memory or local database running. Furthermore, we wrote a router message faker to send telemetry data to Jalapeño, since we could not get access to real router data to test our app.

To decrease the development time, we build a local docker image with all the pipeline functionalities imitated to get faster results during local development for the linter, test and code analysis stage.

Outlook

The base of our app is provided with a stable, performant and easy to expand code. This allows us to focus on enhancements and new features, such as the installation of the segment list and the configuration of the routers to work with segments. If there is more than one greenest route, we want to be able to return multiple paths. We would like to include throughput into the calculation of the green index and other indicators. Finally, we want to deploy the app to a Kubernetes cluster and connect it to a frontend.

Acknowledgments

We would like to thank the following people for their valuable support and or supervision during this work.

Laurent Metzger is the initial idea carrier for this work and made us aware that this work is an important part for the future in internet backbone networks. This motivated us strongly during the whole project. Additionally for the always pleasant and enriching support during our work as main supervisor.

Severin Dellsperger who was co-supervisor of Laurent Metzger and always supported us with very valuable feedback and ideas and was always available for questions.

Julian Klaiber who was also co-supervisor of Laurent Metzger and supported us with very valuable feedback and ideas.

Michel Bongard who has been actively working on the new Jalapeño API gateway simultaneously with the start of this thesis, providing us with an important base dependency for the development of our application and for always being available to answer questions from us.

Francois Clad who joined a weekly meeting in the middle of our project construction phase and was able to provide important feedback from Cisco Systems on our research.

Matteias Collet who helped us with the correction of our English and gave useful feedback on the content.

Contents

A. Technical Report

Change history

Version	Date	Changes	Responsible
1.0	24.12.2021	Finished the final technical report.	Jonas H. and Pascal S.

1 Introduction

This document is aimed at engineers in the field of computer science. A basic understanding, especially in networking and software engineering, is necessary.

1.1 Thesis composition

This thesis is written according to the guidelines of the Eastern University of Applied Sciences and is divided into the following three main areas.

1.1.1 Technical Report

The technical report is divided into three chapters.

The first chapter contains a rough introduction and overview of the work. Techniques used, if any, are outlined. The goals and tasks are specified based on the problem definition and solution with the methods used. In the chapter results and discussion, the outcome of this work is discussed and the implementation reflected in rough form. The third and last chapter contains important parts for the conclusion. Experiences and lessons learned are analysed for each part of this thesis. In addition, possible next steps are shown, potential improvements are pointed out and a look into the future is given.

1.1.2 Project Documentation

The second part documents the whole project, especially how the results were achieved.

In the first chapter, the requirements are presented in the form of use cases and split into functional and non-functional requirements. The second chapter contains everything about project management. The methods used and their key data are explained in detail, including milestones, schedule, meetings, responsibilities, as well as risk management beside additional parts. After the project management chapter, it is discussed how we want to develop in this thesis. Principles, version management, quality measures, error handling, development environment and CI/CD are outlined in detail. Those measures should support the team during development. In the next chapter the architecture and design specifications as well as a domain analysis are provided.
The last three chapters contain the declaration of independence, the terms of use of this work and the reference to the meeting minutes.

1.1.3 Appendix

Supporting documents and figures with lower relevance are attached in the appendix.

1.2 Techniques

1.2.1 Traditional Routing in Networks

Many of today's networks, which our infrastructure builds on, were created 20 years ago. [1] That is because the basic principles of the used routing protocols have not changed much over time. In contrast to the rest of the digital world, the networking area has not made significant advancements in the last two decades. If you have fast and reliable protocols, there is no big need to change something. In this chapter we illustrate how networking is done today and we investigate the near future where segment routing is introduced, which could eventually replace part of the networking protocols used today.

1.2.1.1 Destination-based Routing

Most networks and network protocols follow the destination-based routing approach. From a high level perspective it works as follows: When someone wants to transmit a message over a network, the data is split up into many smaller sized packets. Each of these packets consist of a header, which contains information such as where the packet should be routed to, as well as the payload containing the actual data. The field with the information where the packet should be routed to is called the destination field. Each router who receives a packet inspects its destination field and routes the packet according to its routing table, which is aware of the shortest path to each destination. The content of this routing table is calculated by a network protocol.

In Figure Part A 1.1 you can see a small example network. Router R1 receives a packet with destination A. According to its routing table it knows that it needs to forward the packet to router R4. R4 forwards the packet over the fastest path to R3. This is repeated until the traffic reaches its destination.

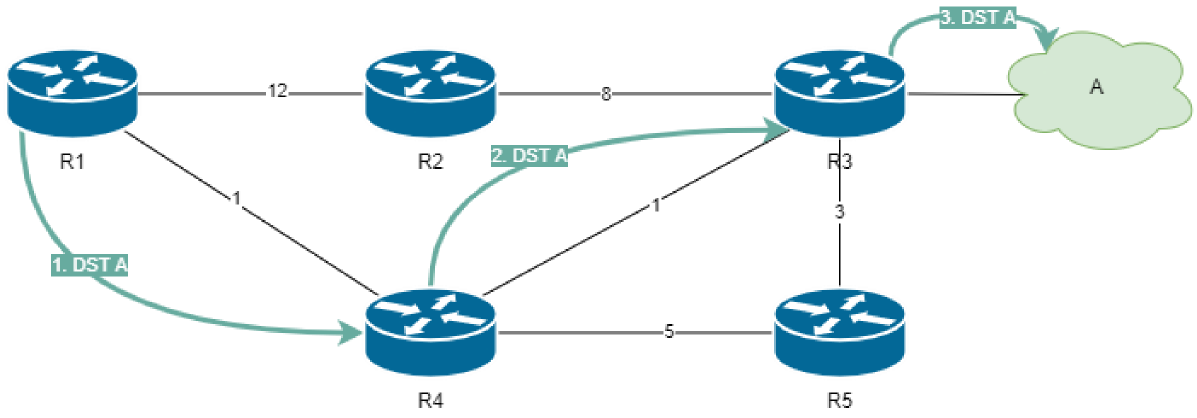


Figure Part A 1.1: Example destination-based Routing [2]

1.2.1.2 Traditional Approaches

The destination-based routing method is not enough in many networks, especially if they are bigger. For this reason, some additional concepts were added on top to gain some flexibility in traffic management. One of these concepts utilizes different labels which tell a router to use a certain rule when forwarding traffic.

If data had to be conveyed by such networks, the packets would be constructed with a single unique label. In this scenario, the label reflects the entire path. The packet is transferred from router to router over the network after the label has been affixed. Each packet is treated independently, since each router keeps track of where it should be forwarded to based on its unique label.

Thanks to the labels, it is possible to use different pathfinding algorithms with different optimizations based on the label of the package. Now you can not only choose the shortest path but you may want the path with the fewest hops or the least probability of lost packets. This way, a router can decide which path to take for each individual package. However, there are also some downsides which will be discussed later.

In Figure Part A 1.2 you can see the same network topology as before with the only difference being that in addition to the destination address a label is attached to the package. This label instructs the router to take a specific path, such as the lowest Interior Gateway Protocol (IGP) path.

Two packets with different labels are sent over the network. R1 sends them both to R4 where the paths split up and label A2 takes the shortest path whereas label 1F takes a different path over router R5. Both end up at R3 and finally at their destination.

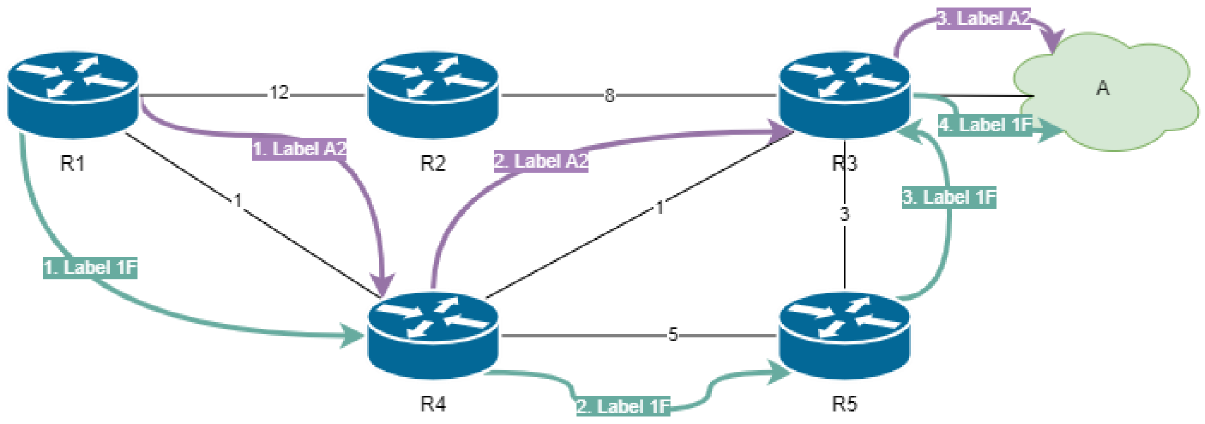


Figure Part A 1.2: Example Routing with Labels [2]

1.2.1.3 Limitations

The techniques outlined above have improved older networks and demonstrated new capabilities that were a perfect match for networks a few years ago. However, as the digital world evolves and introduces new technologies such as 5G and Cloud Computing, current networks face new issues. Traditional approaches no longer satisfy the needs of modern networking and have several disadvantages.

One of the drawbacks is the fact that the network must maintain a state with all the labels and the different rules resulting from the labels. This means that there is a lot of intelligence distributed in the network requiring specialised protocols to ensure that every router always has the newest information. This leads to many different protocols in use which increases complexity and costs due to more difficult maintenance and troubleshooting.

Convergence is another problem that older techniques have. The network state must be recalculated whenever the topology changes, such as when a connection disconnects. This is an important operation since packets may not be transmitted continuously while the recalculation is being performed, therefore the recalculation must be completed as quickly as possible to resume with the package routing. This is referred to as network convergence in network terminology.

Traditional networks operate on the notion that each router must keep track of the network's status. In terms of network convergence, this concept has two major drawbacks:

- Changes in the network have to be broadcasted to each router, which then performs the recalculations. These messages can put a strain on the network and each router's resources, potentially causing congestion.
- Secondly, after a network topology change, each router must perform recalculations. As a result, network packets may get lost or fail to transport.

In the past, good network architecture, capacity planning, and protocol enhancements could have shortened the convergence time. However, in each network, quick rerouting was not always achievable. As a result, completely new approaches were required.

1.2.2 Segment Routing

Segment routing wants to solve many of the previously described problems with a new concept. The following sections provide the necessary basics about segment routing that need to be understood in the context of this thesis. It also contains chapters on the basic segment routing features and concepts that underlie this project. In addition, it introduces some terms used in segment routing, which will be used further in this project work. It also has some advantages over traditional routing concepts and lists why this approach will be used frequently in the future.

1.2.2.1 Concept

Segment Routing is a brand-new routing method based on the source routing concept. The initial router determines the route the packet should take. This decision might be based on a number of configurable attributes or rules. After the path has been defined, the path is expressed by including instructions in the packet header, known as segments. A segment can represent many instructions. The network packet header contains a list of instructions (segment list) that ensures the packet follows the path that the source node intended. The packet can then be sent across the network after being built with its segment list. Each node examines the packet's outermost portion and follows its instructions. Because each intermediate node knows exactly which instruction to follow if a packet containing a specified section is received, the notion works. The outermost segment is deleted after an instruction is completed, and the next segment is evaluated and processed. For each segment in the segment list, the mechanism is repeated. After all segments of the segment list have been removed, the packet should have arrived at its intended destination, if the segment list was followed as intended. This also indicates that all scheduled instructions were processed and executed successfully.

1.2.2.2 Example

The example below in Figure Part A 1.3 should clear things up. Two packets, purple and pink, from different source networks and types must be forwarded to network A behind router R3.

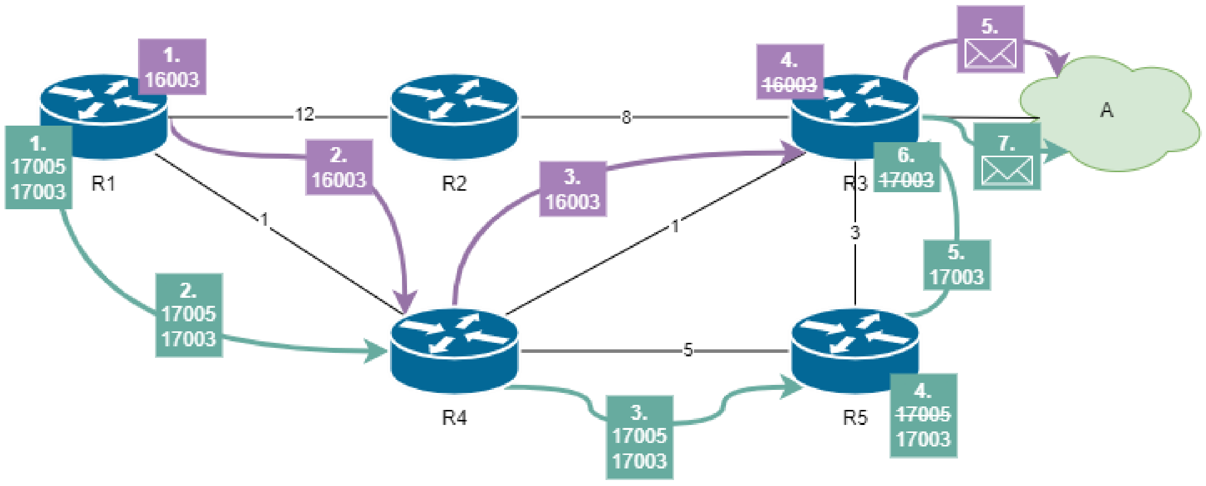


Figure Part A 1.3: Concept of Segment Routing [2]

The purple packet indicates significant business traffic that must be transmitted to the target network via the quickest path possible. As a result, the source node R1 adds the segment 16003, which instructs all intermediate routers to send the packet to router R3 using the shortest path. After the packet has allocated its segment list, which in this example is made up of one instruction, the packet will deliver the shortest path to R3. The router R4 is the initial link in the chain. This router examines the segment, identifies the instruction, and passes it on to router R3. The destination router, R3, examines the outermost segment and recognizes that the packet is for itself. It can also delete the outermost packet and send it to the end destination since the packet has no further segments assigned to it. The empty segment list, as previously stated, verifies that the packet was routed appropriately.

Voice traffic is represented by the pink packet, which must be routed over the most sensitive delay traffic path. This path is different from the shortest path in this scenario. The router R1 adds two segments to the packet, 17005 and 17003, to express this path. Two new instructions have been added to the segment list. They can be translated as "send the packet to R5 on the least delay path possible" and “Forward the packet to R3 on the least delay path", respectively. As a result, these two segments can express the whole path from source router to destination router.

The packet can be transmitted on the specified path after the segment list is appended to the packet header. R4 is the first intermediate node once more. R4 examines segment 17005 and determines that the packet has to be transmitted to R5. R5 pops the outermost segment after receiving the packet since the first segment of the route has been completed. The packet is then forwarded to the target router R3 through the path with the smallest delay. R3 examines the top segment, interprets its orders, and strips down the last section. The last step is to forward the packet to the intended recipient. As a result, the packet was transmitted from source to destination along the path with the shortest delay.

1.2.2.3 Terminology

The terminology is based on the RFC standard RFC8402 - Segment Routing Architecture [3] and is kept as simple as possible while still containing the important information.

Node

A Node is a device that understands and participates in the segment routing protocol's operations. This is usually a router, but it may also be other devices that can direct a packet through the network based on segments.

In Figure Part A 1.3, for example, are the Router R1-R5 Nodes.

Segment

An instruction is expressed by a segment. The context of such an instruction might be topological or service based. A segment can have either a local or a global significance. Within a limited topological context, a segment can mean "advance the packet on a certain outgoing interface". In a global topological context, a segment might be thought as "forwarding the packet on a defined path through the network". This path then might follow specific metrics, such as the delay, IGP, or Traffic Engineering (TE) metric.

In Figure Part A 1.3, 16003, 17005, 17003 are segments.

Segment Identifier

A Segment Identifier, often known as a Segment ID or SID, is a number that acts as a segment's unique ID.

Segment List

The segment list is a list of distinct segments that expresses the set of instructions that need to be followed to traverse across the segment routing domain. It is made up of one or more parts.

In Figure Part A 1.3 you can find a segment list in both routes, green and purple. It is in the coloured boxes.

Prefix Segment

A prefix segment, also known as an IGP-Prefix segment, refers to the instruction to forward the packet through a calculated path to the node that has the prefix with the interior gateway protocol advertised. The route may be calculated using a variety of algorithms, each of which uses distinct metrics/characteristics to do so.

For example, in Figure Part A 1.3, 16003 or 17003 may relate to Network A's prefix if it was promoted from router 3 using IGP and alternative methods. The first algorithm was used to compute the shortest IGP total weight path in this case. The overall path with the smallest delay was calculated using the second algorithm, which was linked to segment 17003.

Segment Routing Domain

The segment routing domain is the collection of devices that engage in the segment routing protocol actively.

In Figure Part A 1.3 the segment routing domain is the set containing the nodes R1-R5.

Ingress

The Ingress, also known as the ingress router, is the nearest node to the source. The interface between external (customer) networks and the segment routing domain is this unique device. This router receives a certain packet, determines its path, and appends the segment list to the packet header. Policies that impact how packets are routed over the network must be written to the ingress. Provider edges are routers that perform the ingress function. As a result, several routers in the network might act as ingress nodes.

In Figure Part A 1.3 the Router R1 is the ingress node for both packet flows.

Egress

The egress, also known as the egress router, is the nearest node to the destination. It is the last node in the segment routing domain and the final segment will lead up to this node. As a result, this router must strip down the last segment and forward the packet to the target endpoint, which is tied to a certain packet flow.

In Figure Part A 1.3 the Router R3 is the egress router for both packet flows.

1.2.2.4 Advantages

State in the Packet

Segment routing presents a number of novel ways that alter certain fundamental concepts: Previously, each router was responsible for maintaining the state. As a result of these methods, several protocols were developed to ensure that per-path information was communicated between nodes. This might quickly result in a network that is too complex and has too much overhead.
In segment routing, the whole path may now be specified as a segment list within the packet header. As a result, the network nodes must simply obey and carry out these instructions. Following that, nodes are no longer required to keep state information.
This fundamental alteration has several benefits. First, because the protocol for sharing per-flow information can be eliminated, it gets a lot easier. In addition, the network implementation becomes easy to comprehend, which simplifies troubleshooting and component replacement. Second, the entire path is decided at the ingress. This approach enables the introduction of whole new ways to direct traffic across the network, which are both simple and scalable.

Traffic Engineering

Segment routing improves traffic engineering (TE), which is the particular guiding of packets. TE is simplified and becomes more powerful than ever before since the ingress router has complete control over which way should be chosen. With the advancements achieved in the new technique, packet steering is also improved. Not only may topological choices be made today, but services can also be included in the packet flow. Furthermore, criteria can be programmed, such as avoiding a certain path or service. Following the new methods adds additional levels of scalability and capability to the system. On top of that, networks become more intelligent and may interact more closely with applications.

Convergence

We already covered how traditional methodologies struggled with convergence and re-routing. As a result, segment routing was used to address and enhance this issue. In the traditional system, all routers were involved in computing new pathways and keeping per-flow information, but this is no longer the case in segment routing. As shown in Figure Part A 1.4, if a path is broken, the avoidance router must detect it and notify the ingress. The ingress can then swiftly adapt and redirect the packet to the target through the backup path. This strategy is known as Topology Independent Loop-Free Alternate (TI-LFA), and despite not being covered in this project thesis, it is an important benefit of segment routing.

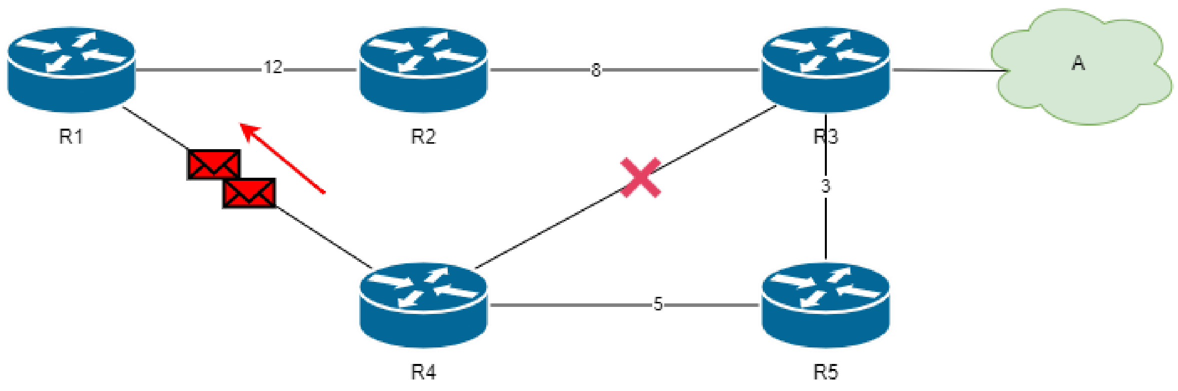


Figure Part A 1.4: Convergence in Segment Routing [2]

1.2.3 SR-Apps

Soon after the first draft of segment routing was released, the Institute for Network Solutions started developing apps which use the new possibilities segment routing opens. Most of these new apps are based around the premise that you can change the path packets use based on some metrics or characteristics of the network. Another common factor is the requirement, that all apps must be cloud native and easily scalable in a Kubernetes cluster.

1.2.4 Telemetry data

Our app relies heavily on the collection and usage of telemetry data. In our case we focus on the ASR9000 routers of Cisco with the IOS-XR software which makes selecting the needed resources very convenient. The concept Cisco uses for the selection of sensor data is called yang models [4] and is a descriptive way to group sensor data and request them over a path. If the path can be resolved, it delivers one or more groups of sensor data in a pre-configured time interval.

As an example, we want to receive the data to power consumption. For the ASR9000 router with version 7.4.1 we need the path **Cisco-IOS-XR-sysadmin-asr9k-envmon-ui.** This path contains not only power data but also some other sensor groups we do not need. Therefore, we specify the container, which is a different name for group in the yang world, we want. A container can contain other containers. In our case we want the container **oper** which is contained in the container **environment** to get our power data. The whole path we need to configure is **Cisco-IOS-XR-sysadmin-asr9k-envmon-ui:environment/oper**. The router knows now exactly what we want and starts delivering the configured telemetry data to the defined address in the network. There the data can be collected and stored in a time series database, which is optimized for telemetry data. [5]

1.2.5 Jalapeño

In the previous chapter we talked about telemetry data which gets sent to a specific address, where it is collected, processed and stored. One software which can handle this is Jalapeño from Cisco. It consists of a InfluxDB time series database and a GraphDB~~,~~ to store the topology data, called ArangoDB.

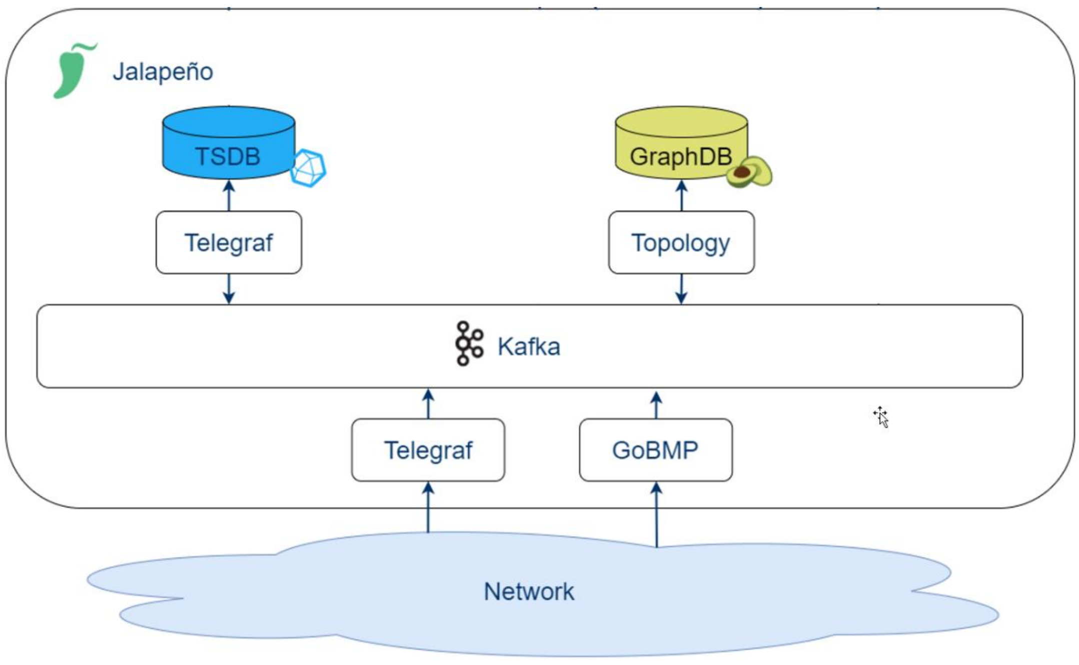


Figure Part A 1.5: Overview of Jalapeño [6]

In Figure Part A 1.5 you can see that data is delivered from the network to two processors. For the telemetry data, Telegraf is used which has a plugin from Cisco to decode the data received from the routers. For the topology data, Jalapeño uses GoBMP to process the received data. Both processors publish their results to Kafka, which in turn provides topics which consumers can subscribe to. One of the consumers is another Telegraf processor who prepares the data for the InfluxDB. Another is the Topology processor which does the same for the ArangoDB.

1.2.6 Jalapeño API Gateway

Once the data is stored in both databases, there needs to be a way to access said data. To help with this, the INS started developing the Jalapeño API Gateway which provides a well-defined interface for SR-Apps to access the collected data. The API Gateway provides two main services. One is the request service, through which one can request data on demand. The other is the subscription service which one can subscribe itself to the service to get notified on topology changes.

This way, SR-Apps can request time ranges from the time series database and get notified on topology changes to recalculate the path based on the new information.

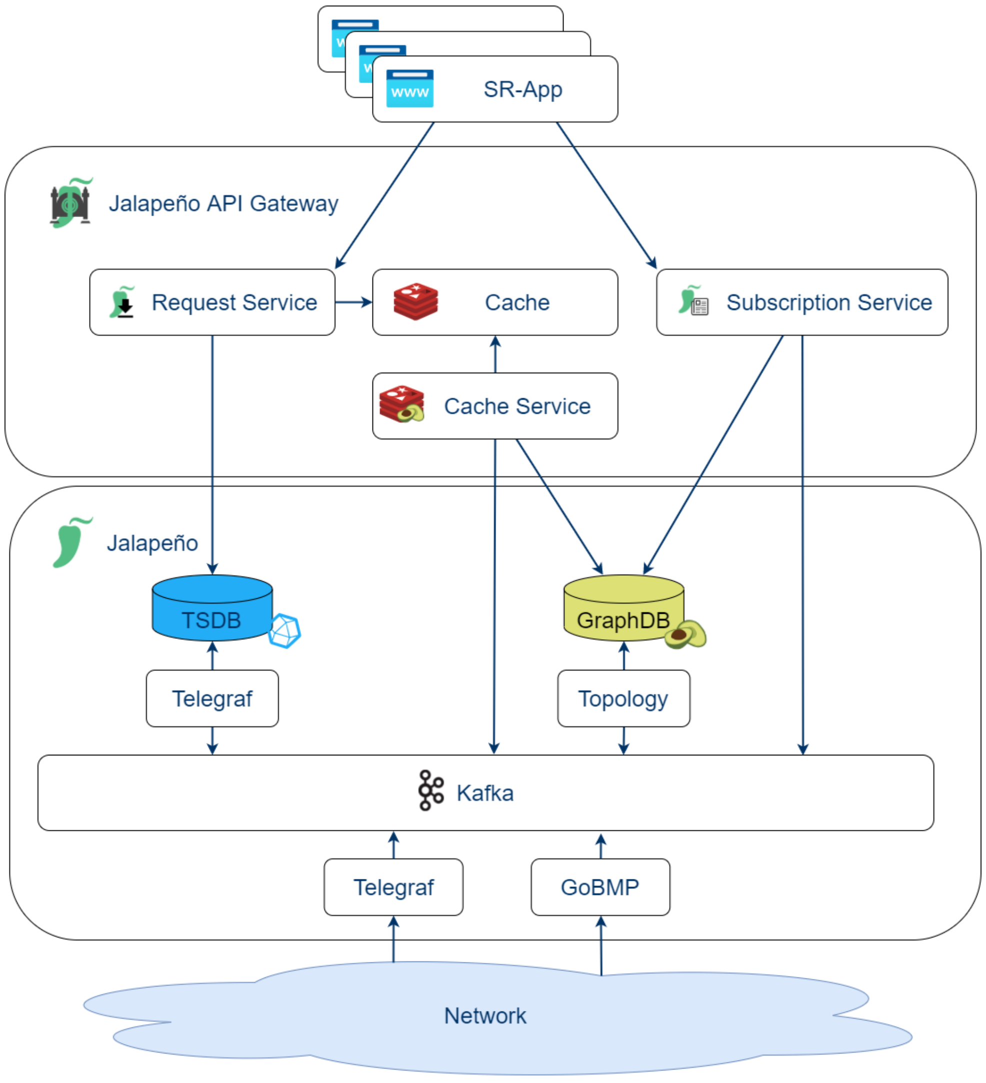


Figure Part A 1.6: Jalapeño with API Gateway [6]

1.3 Goals and Tasks

1.3.1 Problem

In our world, where we work against climate change, every part counts. Especially if this part contributes around 6% of the total global CO_2 emissions. We are talking about the steady growing backbone networks handling our everyday traffic. With around 1.4Gt CO_2 per year, it is worth investigating potential improvements [7]. One idea which came up, was to manage traffic over these networks with the new-found possibilities of segment routing. This way one could analyse the power consumption of routers in a network over a certain timespan and then route the traffic over those routers which require the least amount of electrical energy in relation to their throughput. Generally, it can be defined that newer is better. If you compare a router from a decade ago with e new router of today, there is a 95% decrease in consumed power per Gbps [8]. With this in mind, it should be possible to find the most efficient router per Gbps and route the big heaps of traffic over these routers. In the best case you would take the source of the energy into consideration, prioritizing routers with electricity from a renewable source.

1.3.2 Solution

In our thesis we want to tackle a part of this problem. Namely the part where we analyse the power consumption of routers to create a green index based on the findings. For this we need to analyse how power consumption of a router behaves under load. If we see that the power to Gbps ratio increases or decreases, we need to take that into account. It is also useful to know over what timespan we need to average the data to create stable paths but still get a useful result.

In a first step, we need to know each node with its corresponding average of power consumption over a pre-defined time period. In a second step we set this calculated value on all incoming links of each node. This allows us to calculate the green route on these values and therefore to calculate the path with the lowest power consumption.

To be able to calculate these values, we need an interface to the Jalapeño API Gateway which provides the topology data as well as the telemetry data with the actual values of the used amount of electrical energy.

All this is in vain if you cannot access the produced data. Therefore, we will also need an API which allows us to output and use the calculated greenest path. It should be possible to request a calculation of a path between two nodes.

Conclusion

In the beginning, research is necessary to give us enough information to collect useful data which we then can use in our calculations.

2 Results and Discussion

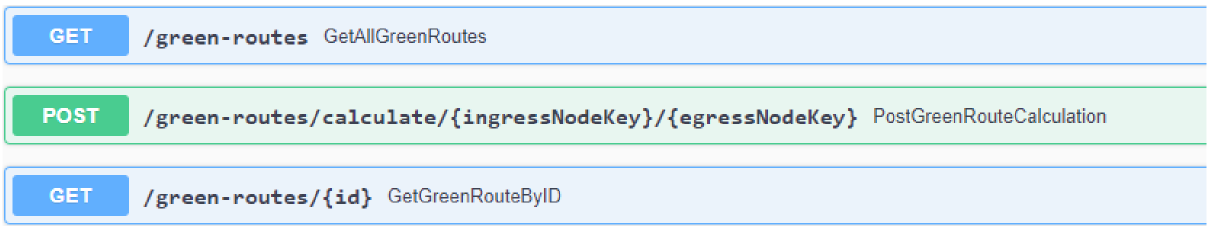
This chapter outlines the results of our thesis. We showcase everything we produced and also give insight into the hidden recognitions we achieved. In the chapter Accomplishments we explain what we have achieved. And in the chapter Implementation we describe how the results have been realized.

2.1 Accomplishments

During the development of this thesis, a backend application and a definition for a green index was developed and defined. This was the foundation for the greenest path calculation between a specified ingress and an egress node through a REST-API. All use cases for the prototype were fulfilled and one additional use case could also be fulfilled.

2.1.1 CRUD greenest route

A user can request the calculation of a new green route via a HTTP POST method to the API. They can specify the ingress and egress node key to start a new calculation. In turn, they will receive a JSON object with the green route containing the overall metric as well as all the links, which together build the route. Additionally, the user can request all previous calculations of all green routes or a specific green route by either its id or by its corresponding ingress or egress node, each identified by their respective the node key.


GET `/green-routes` GetAllGreenRoutes

POST `/green-routes/calculate/{ingressNodeKey}/{egressNodeKey}` PostGreenRouteCalculation

GET `/green-routes/{id}` GetGreenRouteByID

Figure Part A 2.1: Endpoint green-routes

2.1.2 Calculate paths

The backend receives the request to calculate a new green route. To perform the calculation, it first needs to synchronize the topology and load the telemetry data. As soon as the synchronization is completed the calculation is started, which provides an existing Dijkstra algorithm with the necessary data, including all the nodes as well as the corresponding links. Subsequently, the calculation is started, and the result returned to the user.

2.1.3 Get ecological metrics of nodes

After the data from Jalapeño has been converted into entities, it will calculate the average power consumption over the last 10 minutes for each node. This timespan can be overridden environment variables. The calculated value will then be set on the incoming link, which enables us to use the Dijkstra shortest path algorithm over the power consumption metric.

We decided that for this project thesis we will only consider the power consumption without any other metrics, such as throughput. Future ideas regarding which additional metrics we could consider are described in chapter Next Steps.

2.1.4 Get structured data

After the topology data and the telemetry data has been retrieved from the Jalapeño API Gateway service, we need to convert the data into our internal entity types. The resulting data is stored in the database for further use. As part of this thesis, the synchronization is executed manually on demand to keep the solution simpler in comparison to an implementation of the subscription service. In the future, it is planned to use the subscription service from the Jalapeño API Gateway to get notified on topology changes and only perform a synchronization if something has changed. This process is described in more detail in chapter Next Steps.

2.1.5 Define stable route

To prevent routes from changing too quickly, we introduced an environment variable which controls the interval in which the telemetry data is obtained and over which the average power consumption is calculated. The current pre-defined value for this variable is 10 minutes. This way, spikes in power consumption are not going to have a big impact and thus the long-term value provides the more interesting metric of the node.

2.1.6 Gather statistics

For researching purposes, it is of interest to have some statistical data over the course of the power consumption of a node or an entire network. This was solved with the introduction of the node power consumption entity, which stores all the calculated averages of all nodes with the timestamp of its calculation.

2.1.7 Miscellaneous

To be able to write unit tests we had to mock the repository and the synchronization part of the service layer. This was solved with an interface for the repository which contained all the methods which had to be mocked. This method also allowed the use of a fail repository to return specific errors the API should handle. This increased the code coverage noticeably and led to an overall very solid test coverage based on the view of application functionalities and error handling and prevention.

Additionally, we created a separate tool to generate telemetry messages based on the yang models we expect. This was necessary because we had no access to real telemetry data. The tool is named “Jalapeño Yang Model Mocker”, is written in Golang to keep the technology stack simple and works from an experimental point of view. However, it needs some improvements if we want to use it with our app further. Currently, the generated message is sent to Jalapeño where it is processed and stored in the InfluxDB.

2.2 Implementation

In this section we will explain how we achieved our solution. Additionally, we want to show the key decisions and thoughts we made to realise the final product.

2.2.1 Architecture

Two major components influenced the way we designed the architecture. One component was the requirement to build a cloud native application and the other was the decision to follow a domain driven design principle.

To be able to deploy the app in the Kubernetes cluster of the INS, it was necessary to develop the app according to the Twelve-Factors methodology. [9] This led to the decision to not use a cache to reduce complexity and to only store the topology in the database. Essentially, this means that every instance of the app shares the same data without synchronization.

The decision to base our architecture around the domain driven design principle was, that Jonas had already gained some experience in developing APIs following the DDD principle. The key takeaways are that it helps keeping a clean project structure as well as understanding the underlying business problems and converting them into business logic. This was very important for us, due to the complexity of segment routing, and we needed a way to turn the theoretical concepts into well defined data structures. The implementation of entities which were used by the repository layer and the internal logic of the service layer provided a solid base on which the whole business logic could be built on. The use of data transfer objects (DTO) for the incoming and outgoing data of the API provides a clear separation between internal and external properties. It guarantees that only the data we want leaves the API to avoid exposing private data or data structures that do not match our API specification.

We also wanted the possibility to replace the database with a document store or a different kind of database. For this reason, among others, we introduced a repository layer which handles the database specific queries. This strategy also made it possible to easily mock the whole repository layer for integration tests, without having to set up an in-memory database.

2.2.2 Backend

The backend is the most important component and handles all the requests, network synchronizations as well as the path calculations. It is responsible for the input validation and returns standardized and encapsulated responses for each case.

For the API we used the Gin Web Framework [10], which is currently the most popular web framework for Go. [11] It claims to be up to 40 times faster than its competitors. The deciding factors were the relatively big user base, that it is being maintained actively and that it is very performant.

All endpoints are defined in the *controller* package. During the start-up of our application all endpoints are registered in the *api* package in Gin to be able to handle incoming requests with its API router. The endpoints listed below are provided by our app.

The base endpoint returns a greeting message and the URL to the generated OpenAPI specification with Swagger. Due to the fact that the OpenAPI specification 3 is not officially offered yet by Go, the version 2 of the specification was used.

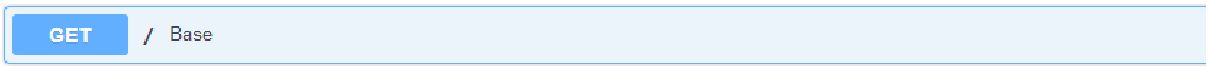
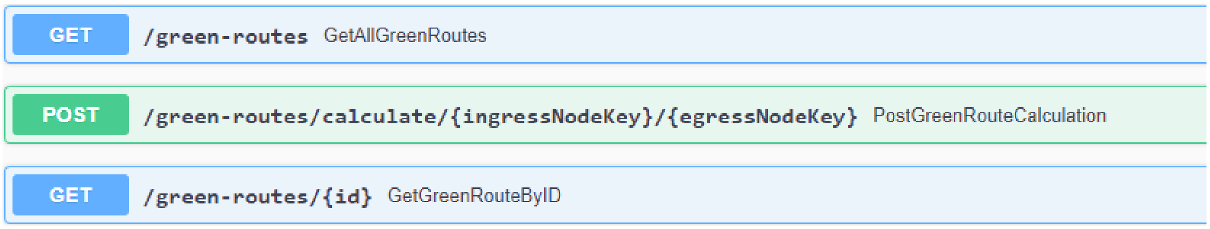

GET / Base

Figure Part A 2.2: Endpoint base

The green route endpoints provide the possibility to calculate a new green route by providing both the ingress node key and the egress node key. If some calculations have been requested, it is possible to get the previously created green routes after an initial request was made.



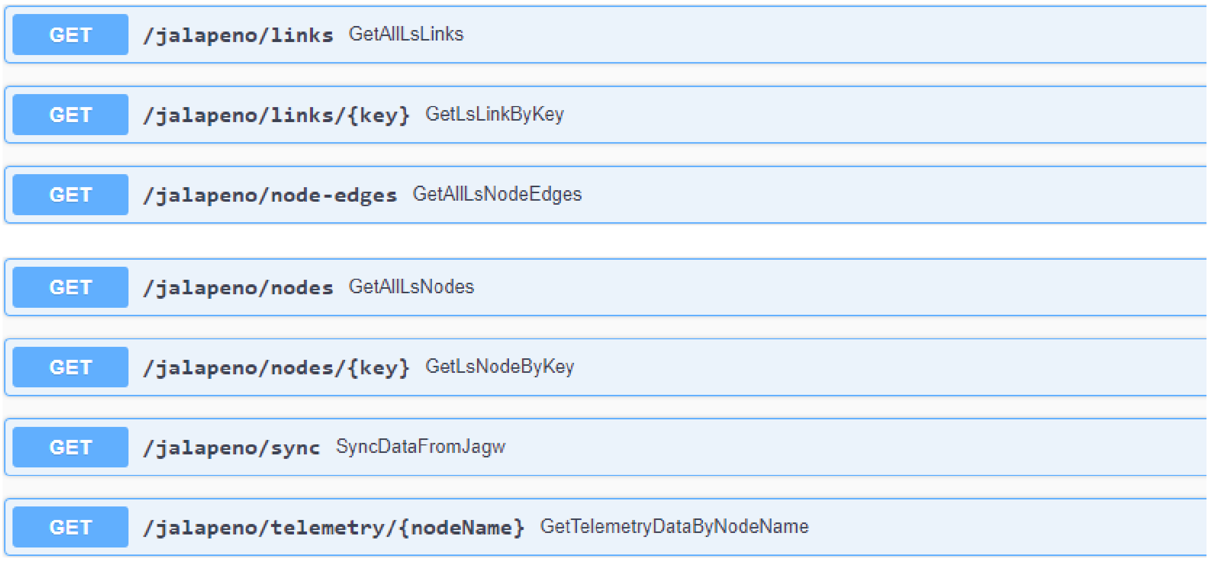
GET /green-routes GetAllGreenRoutes

POST /green-routes/calculate/{ingressNodeKey}/{egressNodeKey} PostGreenRouteCalculation

GET /green-routes/{id} GetGreenRouteByID

Figure Part A 2.3: Endpoint green-routes

The Jalapeño endpoints provide an easy way to get a list of either all nodes or all links from the attached network to Jalapeño. In addition, a new synchronization of the topology data can be requested. Furthermore, it is possible to request telemetry data of a specific node, by providing the respective node name.



GET /jalapeno/links GetAllLsLinks

GET /jalapeno/links/{key} GetLsLinkByKey

GET /jalapeno/node-edges GetAllLsNodeEdges

GET /jalapeno/nodes GetAllLsNodes

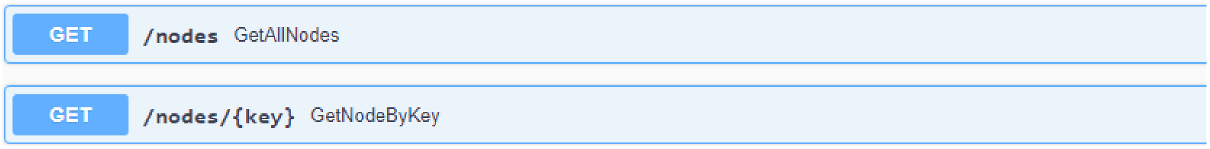
GET /jalapeno/nodes/{key} GetLsNodeByKey

GET /jalapeno/sync SyncDataFromJagw

GET /jalapeno/telemetry/{nodeName} GetTelemetryDataByNodeName

Figure Part A 2.4: Endpoint Jalapeño

The node endpoint is for statistical purposes and allows the consumer to access the calculated power consumptions over time in form of a history table.



GET /nodes GetAllNodes

GET /nodes/{key} GetNodeByKey

Figure Part A 2.5: Endpoint nodes

After the controller has received the request, it calls the corresponding function in the `service` package. Here lies the whole business logic and the logic for the synchronization with the Jalapeño API Gateway. The calculation, which is also done in the service layer, is described in more detail in the next chapter.

If any data from the database is needed, the corresponding function in the `repository` package is called. In this package the queries are defined and executed via the OR mapper Gorm. We used Gorm because it is the most widely used OR mapper for Go and is being maintained actively.

Another important aspect is the fact that it provides a good documentation on its features and usage. [12]

The Twelve-Factor methodology requires the app to implement log levels. This responsibility is handled by the library zap. The decision to use zap as the log manager for our app was based on two key factors. The first one being that this library has been used by many other developers and projects and is being maintained actively. The second highly prioritized factor was, that the performance is significantly better compared to similar libraries in Go. The used log level is configurable in the environment file. The initialization of the logger is done in the logger package which itself is part of the infrastructure package.

2.2.3 Calculation

The calculation of the greenest route takes place in the service package since it is part of the business logic. Before the calculation is performed, the topology is synchronized to ensure that the proper data is used. The calculation initially loads all the nodes and logical links from the database.

We decided against an own implementation of a Dijkstra shortest path algorithm, as there are already various implementations available, which are both more mature and have been compared against each other regarding performance. We therefore decided to use the Dijkstra implementation from RyanCarrier. [13] It is very fast and saved us a lot of time needed to improve the surrounding logic to provide the Dijkstra with the correct data. In a first step, all nodes get fed into the algorithm. Then, in a second step, all the links with the “from node”, the “to node” and the power consumption are fed into the algorithm. If this was successful, a function with the ingress and egress node as parameters can be called to retrieve the path with the lowest power consumption.

The result the database and then be converted to a DTO to be returned over the API.

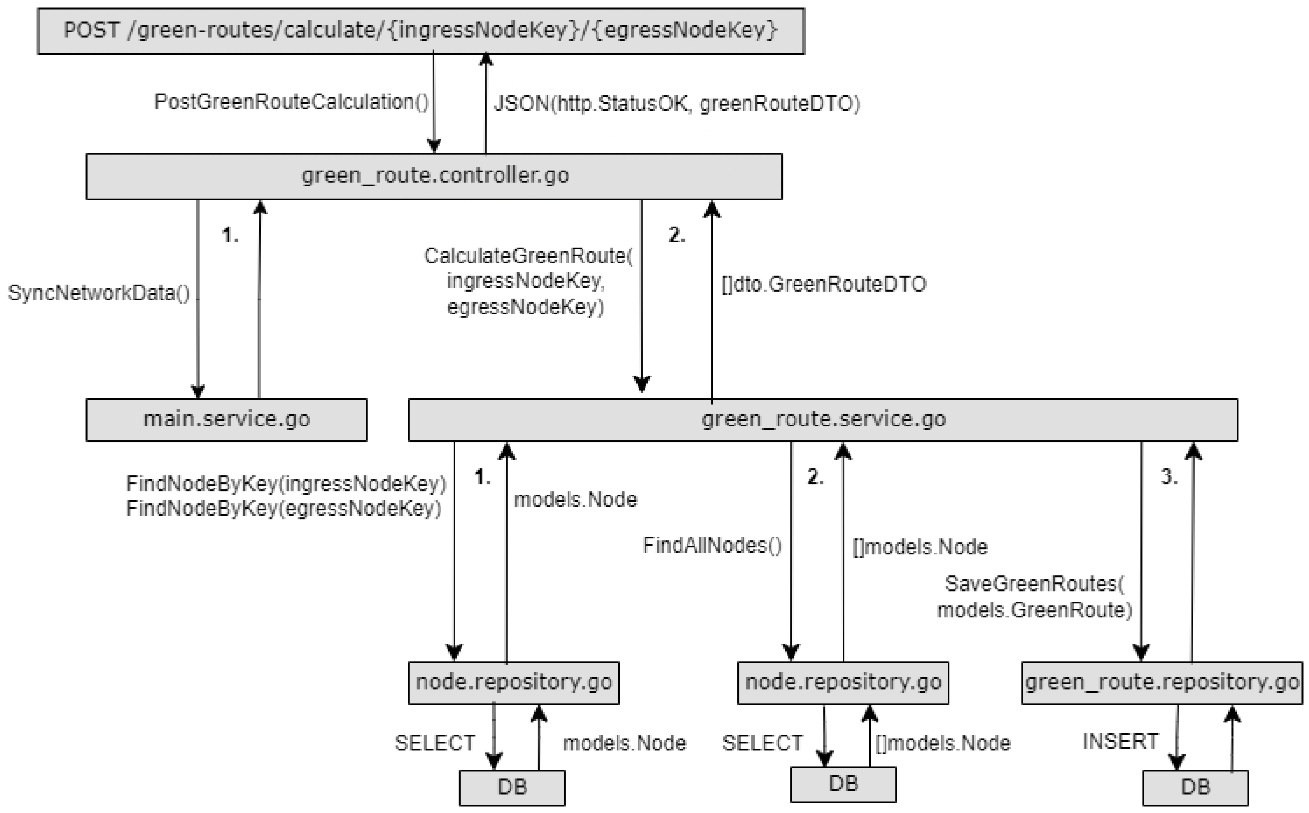

POST /green-routes/calculate/{ingressNodeKey}/{egressNodeKey}
PostGreenRouteCalculation()
JSON(http.StatusOK, greenRouteDTO)
green_route.controller.go
1.
SyncNetworkData()
2.
CalculateGreenRoute(ingressNodeKey, egressNodeKey)
[]dto.GreenRouteDTO
main.service.go
green_route.service.go
FindNodeByKey(ingressNodeKey)
FindNodeByKey(egressNodeKey)
1.
models.Node
2.
FindAllNodes()
[]models.Node
3.
SaveGreenRoutes(models.GreenRoute)
node.repository.go
node.repository.go
green_route.repository.go
SELECT
models.Node
SELECT
[]models.Node
INSERT
DB
DB
DB

Figure Part A 2.6: Application flow of green route calculation

2.2.4 Testing

Testing was a time-intensive part of our project, especially in regard to automated integration tests. We decided against test driven development, due to the lack of possibilities to mock the database and the Jalapeño API Gateway in early phases of the project. Without these preconditions fulfilled, it was not possible to write integration tests in an efficient manner. We however manged to mock the repository layer as well as the part of the service layer which receives data from the Jalapeño API Gateway. This way we reached a test coverage of over 60%. To get a higher coverage we would have had to mock the database itself. This idea was explored at one point, but it turned out to be very tedious with the provided tools hard to maintain. We also decided against a in memory database for testing, since it would have added even more complexity to the development setup and the CI pipeline.

3 Conclusion

We examine our work in depth and critically in the following chapter. We also take a look at the optional use cases that have not been completed and discuss why they were not completed within the scope of this project. This is followed by a discussion section in which suggested improvements to the project's next phase are discussed. This part should be motivating and informative about potential adjustments and features that could be addressed in a subsequent project.

3.1 Requirements revisited

In this chapter, we go through each use case and explain if it was fulfilled or not and why. The mandatory use cases are marked with a blue colour ✓. And the optional use cases are marked with a brown colour ✓.

UC01: CRUD greenest route ✓

"As a User, I can get the greenest route over an API, so that I can lower my energy consumption."

This use case is fulfilled completely. It is possible to access all the nodes, links and node-edges over a REST API. It is possible to trigger, process and store a synchronization with Jalapeño API Gateway. Most importantly, a calculation can be started, by providing an egress and ingress node, to receive the path with all the links and hops.

UC02: Calculate paths ✓

"The Green SR-App can calculate the greenest (most energy efficient) route through a network."

This use case was completed during the project thesis. It is possible to calculate the greenest path through a network with the two parameters ingress node and egress node. The calculation is quite fast and manages to find a path in a network with 1000 nodes and 25000 links in less than 2.5 seconds on average.

We have some improvements planned in the Outlook about this use case.

UC03: Get ecological metrics of nodes ✓

"The algorithm needs to know the ecological aspect of all the nodes to be able to calculate the most ecological path."

This use case was the foundation for the previous use case. Without it, it would have been impossible to calculate a green route. Therefore, we can say that it is fulfilled as well. The calculation of the metrics happens during the synchronization and currently takes the last 10 minutes of telemetry data into account to calculate the average.

This value could change in the future and is configurable via environment variables. There currently is a limit to the time range caused by transfer limitations of gRPC. The amount of data that needs to be sent over the network is too big for a single message and would have to be split up into smaller parts, which is currently not possible on either side. Neither the Green SR-App nor the Jalapeño API Gateway can handle such a requirement.

UC04: Get structured data ✓

“The algorithm needs the necessary fields from the Jalapeño API Gateway to calculate the most efficient route.”

This use case is also fulfilled, because otherwise the previous use case would not have been possible. It is possible to synchronize the whole topology and all telemetry data and store the results in a database. The synchronization with the Jalapeño API Gateway is quite performant and manages to sync a network with 7000 nodes and 100’000 links in 10s. The necessary data for this test was provided directly via a data file and not the network itself.

Currently the synchronization gets started and synced on demand over the API or if a calculation is started. This is not necessary, because the Jalapeño API Gateway offers a subscription service, which sends live notifications if the topology changes. This allows us to synchronize only when it is necessary because of a change in the topology.

UC05: Define stable route ✓

“As a User, I expect that the calculated most efficient path stays stable for the duration I define.”

This use case is also fulfilled. It is currently solved by setting the time range for the obtained telemetry data, which then forms the base for the average calculation. It is possible that this range will increase when we have the ability to observe the power consumption of a network over time.

UC06: View routes ✗

“As a User, I want a simple web interface to see the chosen path and be able to recalculate the routes.”

This additional use case could not be fulfilled, because the time needed would have exceeded the time budget, and we prioritized investing more time improving the performance and quality of our product over implementing the feature to visually see the generated graph.

This will definitely change in the future. We plan to use the newly developed frontend of the INS, which allows to display graphs in a network. This was also realised in a project thesis that was done in parallel to ours.

UC07: Login ✗

"As a User, I want to prevent unauthorized access to the application and only allow a restricted group to access the settings."

This use case was not fulfilled, since there are already enough security measures to protect the app and its API from malicious intent. This justifies that we do not invest time for a login. We will re-evaluate this decision in the bachelor thesis and implement this use case if we consider it to be necessary.

UC08: Gather statistics ✓

"As a User, I want to create statistical reports to visualise certain patterns of the system over a timeframe."

This additional use case was fulfilled. It was solved by storing the calculated metrics of each node, for every calculation. In addition, a history of power consumption values from each individual node is stored per synchronization. This way it is possible to observe the change over time of the power consumption.

It is possible that in the future, additional metrics will follow.

3.2 Learnings

In this chapter we reflect and talk about the things we would do differently if we had to do it again.

A major challenge was the lack of access to real telemetry data. This forced us to implement our own ways to generate the needed telemetry data for testing purposes. We invested a lot of time into replicating the messages, the router sends to Jalapeño. We would rather have spent this time on the application internal mocking to enable us to write unit tests faster and therefore find errors quicker. The time spent on the external telemetry data generator was not in vain though, if it turns out that we cannot work with the data Swisscom provides over their routers, we can re-utilize this method instead.

The runners for the CI/CD pipelines from the INS work with Kaniko images and not with Docker. This led to some confusion at first, as we set up our pipeline. We were not accustomed to Kaniko with its possibilities and characteristics and thus created some potentially suboptimal behaviour which we still do not fully understand. We managed to fill up the whole diskspace with our generated images which prevented us from using the pipeline for a few days until it was cleaned out again. We do not know if we could have prevented this issue, if we invested some more time into learning more about the characteristics of Kaniko. It is also possible that some faults were made while configuring the runners by the INS, but this was not evaluated further.

Furthermore, we could have benefitted from multistage builds, which allow for the reuse of already built components to save time and space. This was taken into consideration, but due to the unknown amount of effort required to implement this strategy we decided to postpone this to the bachelor thesis.

3.3 Next Steps

In this chapter we will show the points, which could be improved in the actual implementation. In the outlook we will take a look at the features we want to add or improve and give an insight into what we have planned.

3.3.1 Improvements

In the current state of the app, we synchronize the whole topology with every executed calculation. This is the biggest part of the whole request and is redundant if the topology has not changed in the meantime. The Jalapeño API Gateway offers a subscription service which sends a notification if the topology has changed. This is also a greener solution as it leaves out a calculation heavy part and only initiates it when necessary.

Right now, it is only possible to get one route at a time even if there are multiple equally green paths. This prevents that the traffic could be split up over multiple paths to keep the overall load lower. Therefore, we want to implement the possibility to calculate and return multiple paths~~,~~ if there are more than one equally green paths.

We already have a sufficient error handling practice but there are still some areas with potential for improvements. The sooner we improve this, the more it will assist us in our further development.

We still have some couplings between our packages which we possibly could decrease. This also helps with further development and saves us time in the long run. This is also something we intend to address early on.

In the pipeline there is potential for improvement, such as using multistage builds and caching to improve the build times as well as finding a solution to prevent the filling of the disk space with old images.

And last but not least we need to deploy our app in the Kubernetes environment.

3.3.2 Outlook

Right now, we do not have a SR-App because the defining part of an SR-App is to tell routers with the segment lists added to the packages, what they need to do. Therefore, it will be the first goal in the bachelor thesis to implement segments and to configure the routers to handle these segments.

We also want to improve the algorithm with the addition of the throughput metrics which we want to combine with the power metrics to detect the more efficient routers. If we only take the power consumption into account, an old router with a low throughput will be preferred over a new router with a very high throughput and a much better Gbps per watt metric.

In a second step we want to include the possibility of utilizing more kinds of metrics in the calculation. Maybe even the source of the electricity. If we could favour routers which are powered with renewable energies, this would be a big step forward.

B. Project Documentation

Change history

Version	Date	Changes	Responsible
1.0	10.10.2021	Finished the initial project plan.	Jonas H. and Pascal S.
1.1	17.10.2021	Finished the initial requirements specification.	Jonas H. and Pascal S.
1.2	22.10.2021	Converted risk table to hours and percentage values.	Jonas H.
1.3	31.10.2021	Finished development concept and most parts of architecture and design specifications. Revalidated and updated risks and risk graph.	Jonas H. and Pascal S.
1.4	21.11.2021	Risk Ri9 is considered as occurred. Risks Ri6 and Ri7 have occurred in a trivial way. Risks Ri1 and Ri8 have been eliminated or can no longer occur.	Jonas H.
1.5	19.12.2021	Backend architecture, Domain Model and 12-Factor Methodology updated to the latest findings and optimizations.	Jonas H. and Pascal S.
1.6	22.12.2021	Create API Documentation, entity relationship diagram and SonarQube Quality Gate Status parts.	Jonas H. and Pascal S.
1.7	24.12.2021	Final small changes due to proofreading everything.	Jonas H. and Pascal S.

1 Requirement specification

This chapter contains all the requirement specifications which were defined in the inception phase and in the beginning of the elaboration phase of our project. We defined the functional requirements in the form of use cases and user stories. Non-functional requirements are defined later in the chapter.

1.1 Use cases

We divided the use cases and user stories into prototype use cases and additional use cases.

🟢 The green colour marks the use cases of the prototype.

🟠 The orange colour marks the additional use cases.

1.1.1 Actors

Actor	Description
Anonymous	Anonymous describes an unauthorized entity who has read access but does not have the privilege to change any settings.
User	The user is an authorized entity and has read as well as write access. Users can change the settings for the algorithm.
Jalapeño API Gateway	The API Gateway is the provider of all the data the application needs. It can notify the subscribed services whenever the topology has changed.

Table Part B 1.1: Actors

1.1.2 Use case diagram

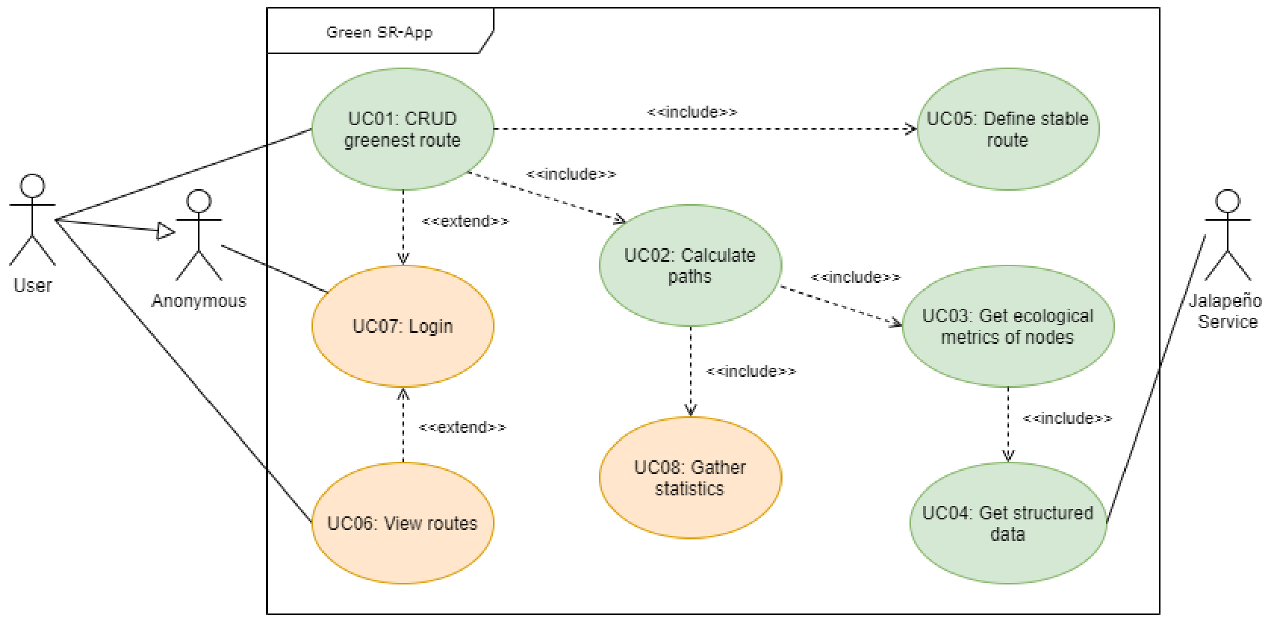


Figure Part B 1.1: Use case diagram

1.1.3 Use case description

For each use case we defined a user story. The user stories follow the Connextra template, stated below [14, pp. 205-222].

"As a <role> I can <capability>, so that <receive benefit>" [14].

Where the “so that” clause is optional and is only added if it gives an additional value. Thanks to the template all user stories have the same format and therefore provide a quick overview in an easy-to-read format.

For the more complex use cases we decided to use fully dressed use cases according to Larman [15]. This format was also taught at the University of Applied Science of Eastern Switzerland. We only used the fields considered to provide additional value. Larman describes the following fields which we omitted:
Scope, Level, Success Guarantee, Special Requirements, Technology and Data Variations List and Miscellaneous.

UC01: CRUD greenest route

“As a User, I can get the greenest route over an API, so that I can lower my energy consumption.”

Primary Actor	User
Overview	The User can access the Green SR-App over an API where he inputs the source node and destination node and receives the greenest route.
Stakeholders and Interests	The User wants an easy access over an API.
Preconditions	The SR-App is running. The Jalapeño API Gateway is delivering data.
Main Success Scenario	1. The User wants to know the greenest path and makes a request with the inputs source and destination. 2. The application validates the inputs and starts the calculation. 3. If a path has been found it returns the path.
Extensions	2. The User inputs one or two nodes which cannot be found in the network. 1. The User wants to know the greenest path and makes a request with the inputs source and destination. 2. The application validates the inputs but cannot find the nodes. 3. It returns an error code with the information that the input nodes do not exist.

	3. The application is not able to calculate a path. 1. 1 and 2 are the same as the main success scenario. 3. The application is for some reason not able to calculate the greenest path and returns this reason as an error code.
Frequency of Occurrence	As often as required

Table Part B 1.2: UC01: CRUD greenest route - Fully dressed use case

UC02: Calculate paths

“The Green SR-App can calculate the greenest (most energy efficient) route through a network.”

Overview	The path calculations take the two parameters source and destination and calculate the most energy efficient path between those two nodes.
Stakeholders and Interests	User wants to use the most energy efficient route.
Preconditions	The necessary data has been delivered from the Jalapeño API Gateway.
Main Success Scenario	1. The two parameters have been received. 2. Based on Dijkstra's shortest path algorithm[16], one or more greenest paths are calculated. How exactly the algorithm will work is part of our research. 3. If only one path has been found, it will be returned.
Extensions	3. If multiple equivalent paths have been found, the application takes the best path defined by metrics. 4. If there are still multiple remaining paths with the same speed, all the paths will be returned with the intention to spread the load over all of them.
Frequency of Occurrence	The User defines the frequency.

Table Part B 1.3: UC02: Calculate paths - Fully dressed use case

UC03: Get ecological metrics of nodes

“The algorithm needs to know the ecological aspect of all the nodes to be able to calculate the most ecological path.”

Overview	For us to be able to calculate the lowest power consumption, we need to know how big the energy consumption is for each node.

Stakeholders and Interests	It is of interest for the User to know how much power a router consumes under a certain load.
Main Success Scenario	We can get the current power consumption for each node and can calculate the power consumption under load.
Frequency of Occurrence	As often as required.

Table Part B 1.4: UC03: Get ecological metrics of nodes - Fully dressed use case

UC04: Get structured data

"The algorithm needs the necessary fields from the Jalapeño API Gateway to calculate the most efficient route."

Primary Actor	Jalapeño API Gateway
Overview	Get all the metrics over the Jalapeño API Gateway to calculate the best route.
Main Success Scenario	All the data has been sent and the backend is able to calculate the power consumptions of the different routes.
Frequency of Occurrence	As soon as data is received.

Table Part B 1.5: UC04: Structure data - Fully dressed use case

UC05: Define stable route

"As a User, I expect that the calculated most efficient path stays stable for the duration I define."

Primary Actor	User
Overview	The User can define the frequency at which the paths get recalculated. The data is collected continuously but the paths will be calculated after the defined time has passed.
Stakeholders and Interests	The User needs to rely on a certain stability of the routes. Additionally, if the paths get changed to often, the overhead will be too much.
Preconditions	The frequency needs to be set. The data needs to be ready for consumption.
Main Success Scenario	1. After an event has been triggered the calculation of the greenest path will be executed. 2. The result will be logged into a database or caching service.

	3. If the User defined time has passed or a certain threshold is breached the active path will be updated.
Frequency of Occurrence	The calculation will repeat after the defined duration.

Table Part B 1.6: UC05: Define stable route - Fully dressed use case

UC06: View routes

“As a User, I want a simple web interface to see the chosen path and be able to recalculate the routes.”

Primary Actor	User
Overview	A web interface for the User to calculate the path, see the results and configure the algorithm.
Stakeholders and Interests	The User will be happy to see the algorithm in action.
Preconditions	The MVP is completed.
Main Success Scenario	The User has a web interface to access the Green SR-App.

Table Part B 1.7: UC06: View routes - Fully dressed use case

UC07: Login

“As a User, I want to prevent unauthorized access to the application and only allow a restricted group to access the settings.”

Primary Actor	User
Overview	To make sure that only authorized Users can access the app and its settings, a login is necessary.
Stakeholders and Interests	The login makes sure that only the Users, which the client has defined, gain access to the application. This prevents tampering with the settings by unauthorized persons.
Preconditions	The web interface is finished and working.
Main Success Scenario	It is only possible to access the application via the login. The settings are protected.

Table Part B 1.8: UC07: Login - Fully dressed use case

UC08: Gather statistics

"As a User, I want to create statistical reports to visualise certain patterns of the system over a timeframe."

Primary Actor	User
Overview	The User can create reports based on the data the Green SR-App gathers. This enables him to observe the change of paths over the day or how much power the system consumes at all times.
Stakeholders and Interests	The User can make decisions based on the results of the reports.
Preconditions	To draw a graph with the variety, the path calculation must have been run a few times.
Main Success Scenario	1. Every few seconds or minutes the greenest path is calculated and stored in the database. 2. The telemetry data is continuously collected. The service reduces it to the relevant fields and enriches the data with some calculated information before it is also written to the database or the caching service.
Frequency of Occurrence	Multiple times per defined number of seconds or minutes.

Table Part B 1.9: UC08: Gather statistics - Fully dressed use case

1.2 Non-Functional Requirements

The non-functional requirements were defined according to the FURPS+ model. [15, pp. 56-57]

1.2.1 Functionality

1.2.1.1 Security

The application will not be accessible from the internet and is therefore protected by the existing security measures from the private network. It will not be necessary to implement additional security measures.

1.2.1.2 Interoperability

The Green SR-App needs to work together with the Jalapeño API Gateway which provides the necessary data for the calculations. The latest version available will be used as long as the version upgrades do not imply any limitations. If such limitations are found, the previously used version will be fixed fur further development and use.

1.2.1.3 Accuracy

The application should return the path with the lowest power consumption with an accuracy of 99%.

1.2.2 Usability

1.2.2.1 Understandability

The user should understand how the path was calculated. This will be achieved with statistics over the network.

1.2.2.2 Operability

The system should behave in a predictable and secure way to prevent invalid configurations which could lead to a breakdown of the network.

1.2.3 Reliability

1.2.3.1 Availability

The system should be available 99% of the time.
This requirement is tried to be achieved in the sense of this term project, but it is not mandatory, because it is influenced by too many factors that cannot be directly controlled or influenced.

1.2.3.2 Recoverability

The application should be developed with the Cloud Native Standard[9] in mind, which includes a simple redeployment.

1.2.3.3 Fault Tolerance

To detect a faulty router sending data, we implement boundaries to detect data which would lead to an invalid path, according to “Patterns for Fault Tolerant Software”[17, p. 8]. Additionally, with multiple reflector routers, we can compare the results of the router to detect faulty data.

1.2.4 Performance

1.2.4.1 Capacity

The application should be able to calculate paths in a network of up to 1000 nodes.

1.2.4.2 Time behaviour
A new calculation of the greenest path should not take more than 10 seconds.

1.2.5 Scalability
The backend should be scalable and always fit to the according environment need. If a lot of requests are made, an additional backend-pod should be started.

1.2.6 Maintainability
1.2.6.1 Analysability
It should be possible to change the log level of the application. If the log level is set to debug mode, the application should write information about important events to the standard output.

2 Project Management

2.1 Used Methods

We based our process around the Rational Unified Process (RUP) with the four phases: inception, elaboration, construction, and transition. The inception phase is about finding out what to build and performing initial research. The elaboration phase is understanding the requirements in detail, perform additional research work and defining the software architecture. The construction phase is about the implementation of the defined requirements. Finally, the transition phase is about validating and releasing the project results.

In addition, we will proceed with Scrum and thus iterations. The combination of RUP and Scrum is called Scrum+ at the Eastern University of Applied Sciences and is supposed to combine advantages of the agile approach through Scrum with the classic approach of phases and milestones to best plan and execute the project. We will also use a Scrum Board to track the state of each work item.

2.2 Organization

2.2.1 Project internal

Name	Job	Email
Pascal Schlumpf	Software Developer	pascal.schlumpf@ost.ch
Jonas Hauser	Software Developer	jonas.hauser@ost.ch

Table Part B 2.1: Project internal organization

2.2.2 Project external

Name	Position	Email
Prof. Laurent Metzger	Supervisor	laurent.metzger@ost.ch
Severin Dellsperger	Co-Supervisor	severing.dellsperger@ost.ch
Julian Klaiber	Co-Supervisor	julian.klaiber@ost.ch
Michel Bongard	Developer of the Jalapeño API Gateway (INS contractor)	michel.bongard@ost.ch
Francois Clad	Cisco Systems liason	fclad@cisco.com

Table Part B 2.2: Project external organization

2.3 Scheduling

The semester project is marked with eight ECTS and therefore requires approximately 240 hours of study per person. Since we carry out this project in pairs, this results in a targeted total effort of about 480 hours. As we work according to an agile setup, we will use all the time we approximated.

The project kick-off meeting took place relatively late in the first week of the semester. Due to the relatively late start in the first week of the semester, the project was compressed to 13 weeks with an average weekly working time of 18.5 hours per student, so that working time does not have to be planned and made up afterwards.

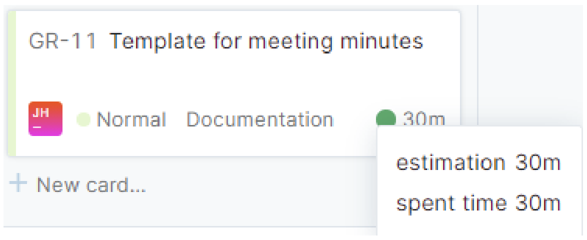
2.3.1 Iterations / Sprints

We normally work in two-week sprints except in the elaboration phase where we have a three-week sprint because this seems to be the most reasonable approach. Longer sprints are suboptimal due to the limited project duration of fourteen weeks, and shorter iterations would tend to lead to too much management overhead.

The sprints start on Monday and end on Sunday unless the team defines a different time slot due to absences or work limitations.

2.3.2 Estimation and Time Spent

In sprint planning, we estimate the workload for each work item and record it on the tickets in our YouTrack tool.
The time spent is recorded on the tickets according to the categories in the timetable chapter and provides a transparent overview of how many hours were spent on which categories of work and tasks.

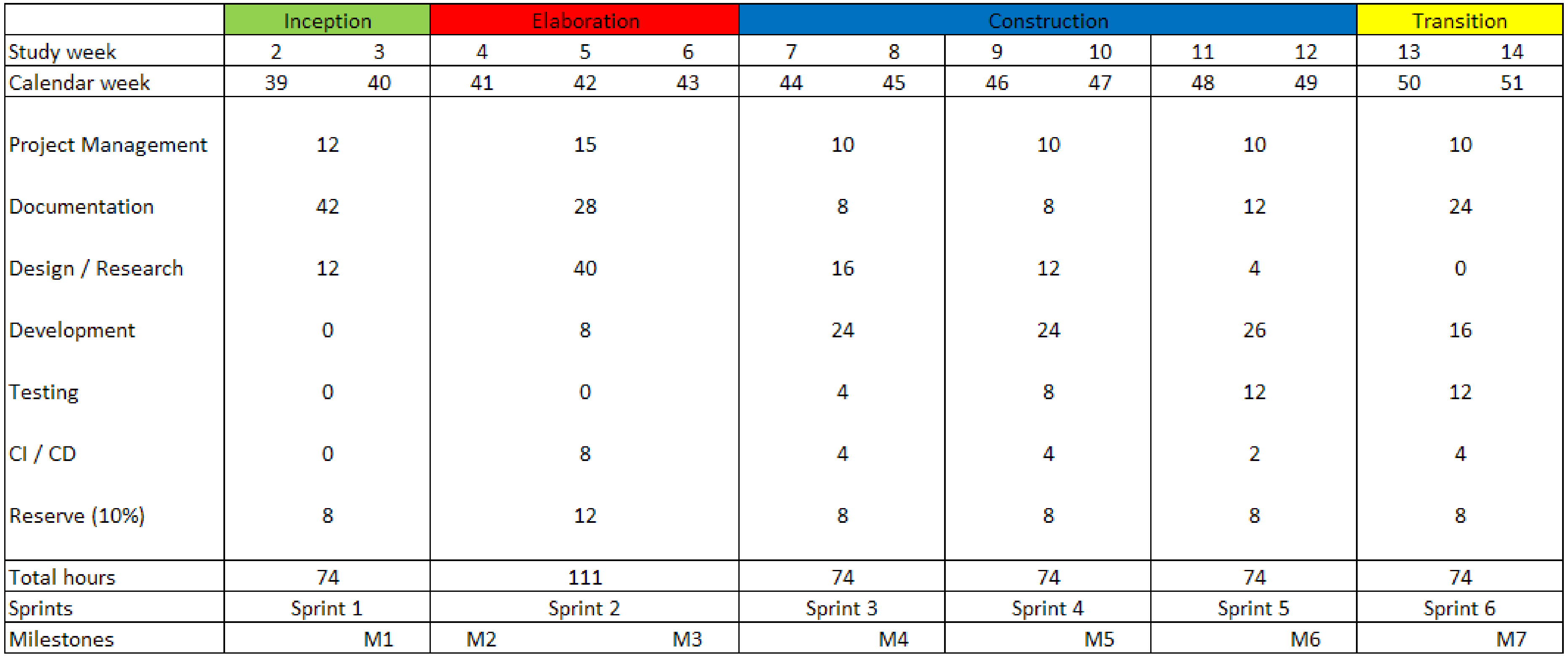


estimation 30m
spent time 30m

Figure Part B 2.1: YouTrack estimation and time spent

2.3.3 Timetable

Based on the nature of the project, seven categories were defined for tasks to be completed. For each category, a rough time estimate was made to get an overview, which can then be tracked on YouTrack based on the time booked. The milestones mentioned are defined and described in the milestones chapter.



	Inception		Elaboration			Construction						Transition	
Study week	2	3	4	5	6	7	8	9	10	11	12	13	14
Calendar week	39	40	41	42	43	44	45	46	47	48	49	50	51
Project Management	12		15			10		10		10		10	
Documentation	42		28			8		8		12		24	
Design / Research	12		40			16		12		4		0	
Development	0		8			24		24		26		16	
Testing	0		0			4		8		12		12	
CI / CD	0		8			4		4		2		4	
Reserve (10%)	8		12			8		8		8		8	
Total hours	74		111			74		74		74		74	
Sprints	Sprint 1		Sprint 2			Sprint 3		Sprint 4		Sprint 5		Sprint 6	
Milestones		M1	M2		M3		M4		M5		M6		M7

Figure Part B 2.2: Timetable

2.3.3.1 Project management time composition

Description	Effort per person (hours)	Total effort (hours)
Sprint Refinement and Planning	1.25	2.5
Sprint Review	0.5	1
Sprint Retrospective	0.25	0.5
Meeting with Supervisors	2	4
Administration and further Meetings	1	2
Total per two-week sprint	**5**	**10**
Total per three-week sprint	**7.5**	**15**

Table Part B 2.3: Project management time composition

2.4 Milestones

In the next table, all seven milestones are defined, and the associated main objectives are described. They were created in the Inception phase, which means that there may still be changes due to the requirements still to be worked out up to M2 and the ongoing Elaboration phase, which mainly includes extended research. The changes are shown in a change history located in the beginning of the project management part.

ID	Date	Title	Goals
M1	10.10.2021	End of Inception	Creation of the project plan with the parts project management and development concepts. Setup of all project management related tools (YouTrack, MS Teams and MS OneNote). Creation of a documentation template with the planned parts and chapters.
M2	17.10.2021	Requirements	Finalization of the requirements in form of fully dressed use cases, functional and non-functional requirements including a use case diagram.
M3	31.10.2021	End of Elaboration	Focus on research, capturing results and decision making on the following topics: • Router telemetry data mocking • How to send data from the router deflectors to the Jalapeño API Gateway • Jalapeño API Gateway itself • Existing demo SR-App • Definition of the green index • Proof of concept for the most ecological path calculation algorithm

			The future architecture and design of the Green SR-App is defined as an initial theoretical proof of concept. General definition of the development concept. The initial project base for the development is created and the environment is set up including the first version of the CI/CD pipelines.
M4	14.11.2021	Data collector / processor prototype	Use of an existing or self-developed and working prototype of a telemetry data collector and processor via the Jalapeño API Gateway. The results are documented.
M5	28.11.2021	Path calculator CRUD backend	A developed and working prototype of the CRUD backend of the Green SR-App including the implementation of the greenest path calculation algorithm as prototype. The results are documented.
M6	12.12.2021	Green SR-App beta release	Consolidation of the data collector / processor and path calculator CRUD backend to the final Green SR-App. General improvements, refinements and finalization of the Green SR-App for a beta release.
M7	24.12.2021	Project delivery	The main goal is to finish the project and deliver all products which were constructed. The documentation must be fully completed and in a finalized state.

Table Part B 2.4: Milestones

2.5 Responsibilities

Both students are ultimately equally responsible, and both have equal decision-making rights in all parts of the project. However, in order to define the division of work more clearly and be more efficient, we have assigned focuses to each of us. Both have been equally involved in the overall development.

Jonas’ primary focus entails the project management, such as the coordination of meetings, and infrastructure related work, including the work with GitLab CI, Docker and the cloud-native application environment. In addition, he is the primary idea carrier for the architecture and design of the application, because he has much experience in the development of web applications including API’s.
Pascal's focus is on the requirements engineering for functional and non-functional requirements

and definition of implementation of our applications. He is the main actor in the field of research on our topics and uses this knowledge in the implementation of mock services or applications.

Both students have been involved in all parts of the project and medium to large decisions have always been made bilaterally by both parties.

2.6 Meetings

Due to the ongoing Corona pandemic and the fact that both students study part-time, the meetings are conducted online via Microsoft Teams.

2.6.1 With Supervisors

Regular weekly meetings with one or more supervisors place on Thursday at 10 a.m. to discuss the current work progress, to clarify questions and to solve problems. Occasionally, the industry partners of the work are also present to gain insight and to place requests.
If necessary, additional meetings are held with INS staff on an individual basis.

2.6.2 Scrum Meetings

The team generally finishes the sprints and the associated milestones at the end of the week. Accordingly, the respective Scrum meetings are held on Sunday evening.

Scrum Event	Timeboxing
Sprint Review	0.50 hours*
Sprint Retrospective	0.25 hours*
Backlog Refinement and Sprint Planning	1.25 hour*

Table Part B 2.5: Scrum Meetings

*The timeboxing for longer sprints as in the elaboration phase are multiplied according to the additional weeks added. In the case of a three-week sprint this means a multiplicator with one and a half.

2.7 Risk Management

In this area of project management, risks to the project are listed and classified. Preventive and reactive measures are evaluated for each risk. The risks are re-evaluated in each sprint and, if necessary, reclassified or deleted if risks have been eliminated.

The maximum damage potential is roughly estimated in hours based on ordinary working days (eight hours). A team member will generally work based on workload and sprint planning between 16 and 21 hours per week. Rough percent values for the probability of occurrence are used.

In the following list, all identified risks are described and evaluated with a weighed damage potential result in hours. The weighed damage potential is always rounded to the next full number.

ID	Description	Damage potential	Probability of occurrence	Weighted damage potential
~~**Ri1**~~	The newly developed Jalapeño API Gateway from INS will not be ready in time with all functionalities needed.	~~63 hours~~	~~30 percent~~ ~~20 percent~~	~~**19 hours**~~ ~~**13 hours**~~
Ri2	The Jalapeño API Gateway from INS has unexpected breaking changes.	21 hours	5 percent	**1 hours**
Ri3	The Jalapeño API Gateway from INS has unexpected bugs or limited use of functionalities.	~~63 hours~~ 42 hours	~~40 percent~~ 20 percent	~~**25 hours**~~ **8 hours**
Ri4	Connecting to the necessary network components at the INS (and partly OST) is not possible.	~~105 hours~~ 84 hours	~~20 percent~~ 15 percent	~~**21 hours**~~ **13 hours**
Ri5	A student absence due to illness (most likely due to the Coronavirus) or an accident.	42 hours	20 percent	**8 hours**
Ri6	The lack of knowledge in the still unfamiliar technologies used is so high that the project is severely delayed.	~~84 hours~~ 63 hours	~~30 percent~~ 20 percent	~~**25 hours**~~ **13 hours**
Ri7	The used platforms, tools or frameworks do not work properly as expected and are limiting the work progress.	~~63 hours~~ 42 hours	~~20 percent~~ 10 percent	~~**13 hours**~~ **4 hours**
~~**Ri8**~~	Bad project planning and organization of the project.	~~42 hours~~	~~10 percent~~ ~~5 percent~~	~~**4 hours**~~ ~~**2 hours**~~
Ri9	It is not possible for us to obtain the necessary data when we need them, because we are not provided with any physical devices.	21 hours	~~60 percent~~ 70 percent	~~**13 hours**~~ **15 hours**

Table Part B 2.6: Risk management

2.7.1 Risk Overview

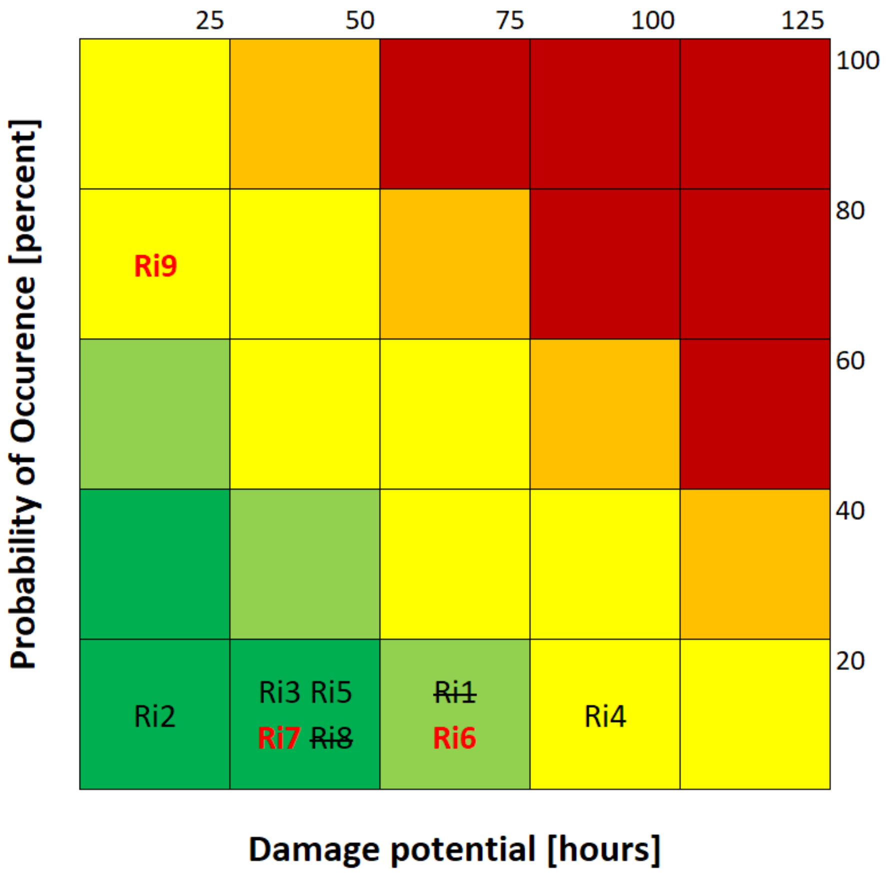


Figure Part B 2.3: Risk overview graph

2.7.2 Dealing with risks

ID	Prevention	Behaviour upon occurrence
Ri1	The status is regularly discussed with the developer of the API at the INS. The developer has already announced the completion in like two up to three weeks at the beginning of the term project.	If there is indeed such a large delay in finishing of the Gateway, the data will be temporarily mocked completely in the data collector of the SR-App.
Ri2	Again, close contact is maintained with the developer to prevent such a scenario. In addition, a strict versioning is required in order to be able to deploy an older version in case of need.	If an irreparable breaking change occurs, an attempt is made to put an older version of the gateway into operation. If this is not possible, the data will be mocked as it is described in Ri1.
Ri3	Here, the close contact with the developer is also a preventive measure to be able to fix occurring bugs as quickly as possible.	First and foremost, attempts are made to bypass the error without further ado or not to use this part with the error at present. In case of urgency, time is invested by team

		members from this project to support solving the bugs in the API together with the INS.
Ri4	Known tools for accessing network devices are used and, if defined, those recommended by the INS or the OST.	It is possible to get to the equipment on site by a more direct route. If necessary, all data can also be provided by a second virtual network or the data can be specified statically on the application side.
Ri5	The students strictly follow the prescribed measures to prevent infection with the coronavirus. The hygienic actions are implemented in the best possible way.	If a student is absent, the next steps in the project will be discussed with the supervisors and the project will either be shortened or rescheduled based on the amount of time lost.
Ri6	An early start is made on acquiring knowledge in all the technologies used. Unsolvable ambiguities are discussed with the co-supervisors at an early stage, because they already have sound experience in all technologies. Due to the iterative approach, more time can be calculated for the knowledge development per sprint.	In case of even small delays in the progress of the work due to knowledge gaps, measures are taken to solve the problem before major delays can occur. If necessary, students can invest additional time for knowledge gaps outside of the project, because the learning process in this profession is part of it.
Ri7	Well-known tools are used like JetBrains IDE's, GitLab, YouTrack and Microsoft products. Unknown technologies like Python and Django are learned and explored early on. Both students are well versed in development and its core components. For GitLab CI we used our own runners from INS, because the standard runners in the engineering project crashed again and again and had to less performance.	In case of complete failure of tools and platforms in general are switched to alternatives. This is especially meant for used platforms. The other technologies and frameworks used are very widespread and it is therefore extremely unlikely that problems occur there.
Ri8	A clear structure for planning based on experiences from previous projects and guidelines is used for this work from the beginning. Work steps and planned activities are discussed and validated in regular steps with the supervisors. Profound experiences in project management from the working environment at the employers of the students are included.	The project plan is adjusted in case of emergency and completely revalidated so that the project can get back on track. The organization is constantly improved as soon as individual difficulties arise, both internally and externally.

	Students conduct a Sprint Review and Retrospective in each Sprint to address issues and continuously improve in organization and execution.	
Ri9	It is currently being clarified with Swisscom whether they have a corresponding laboratory with real routers.	We mock the data as close to the real system as possible and simulate their behaviour.

Table Part B 2.7: Dealing with risks

2.7.3 Occurred risks

The occurred risks are marked with the font colour red in the risk overview above.

ID	Extent of occurrence	Reaction and further steps	Estimated damage
Ri6	Online communities exist for our used technologies and libraries/extensions, but still the experience levels are relatively low. The implementation of the Domain Driven Design, some 12-Factor Methodology related parts, testing and all parts of mocking took longer than initially planned.	Due to the high experiences in software development and especially in web development by the authors, the early difficulties could be absorbed relatively well. The software could be completed successfully without any relevant limitations or a time delay.	16 hours[1]
Ri7	During the project, it became constantly apparent that the used version control and CI/CD system GitLab, which is provided by the OST, shows strong performance reductions at certain times of the day. In some cases, these performance degradations even led to situations in which the loading times were so long that active use was unthinkable.	The work on times of the day with performance issues were avoided as much as possible after the finding of the consistence in the performance reductions. In addition, towards the end of the construction phase, the CI/CD pipeline was completely recreated locally with Docker so that work could be done more efficiently without the dependence on GitLab.	8 hours[1]
Ri9	Completely occurred, because the Swisscom Lab was not finished converting to the	After entry, the estimated time was put into implementing mocking services. The data was faked as	18 hours[1]

[1] Estimation based on the risk related time booking on YouTrack work items and noticeable time implications in other work items.

Jalapeño needs in time and therefore not usable for our application.	deep down in the application layers as possible. The mocking could additionally be used to perform real integration tests with big data sets.

Table Part B 2.8: Occured risks

2.8 Tooling

The tool YouTrack from the manufacturer JetBrains is used for coordination of tasks. This tool had already proven itself in other projects and offers exactly all necessary functionalities. The platform is used to plan, record the status of the work via a scrum board, and to book working time. A short overview of the scrum board in action is shown in the figure below.

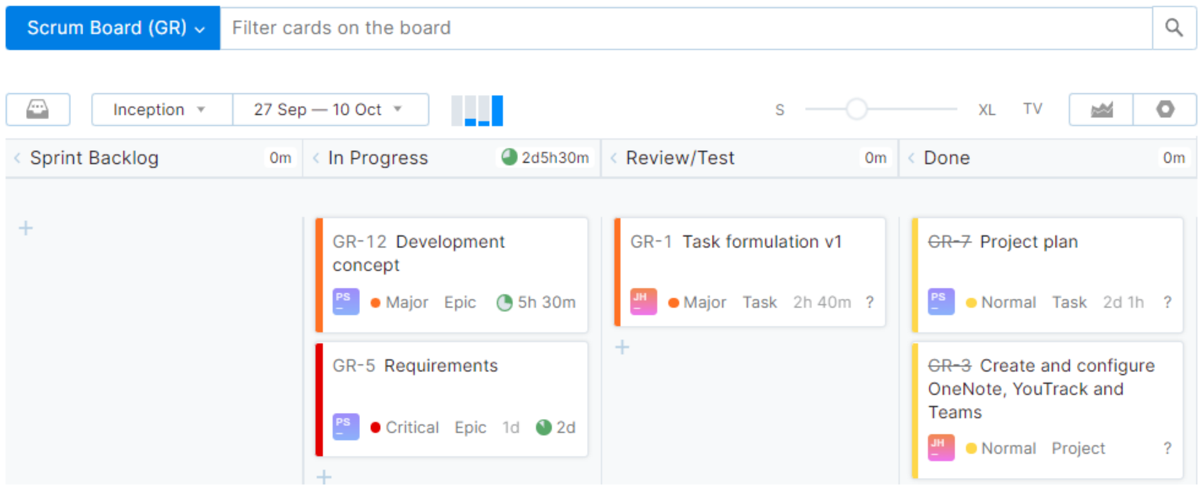


Figure Part B 2.4: YouTrack Scrum Board

Microsoft Teams is used to communicate both within the team and with the external parties involved. The storage capability by Microsoft OneDrive is used to securely store all data produced. The team uses Microsoft OneNote to write quick notes, meetings minutes, prototype documentation parts, and store other information that is not directly included in the documentation.

Finally, both team members are equipped with several personal and very powerful computers with the necessary peripherals for professional work.

2.9 Meeting minutes

Brief minutes are taken for each meeting with the following content. The date is not in the content because it is already present in the chapter title and the location is always online via Microsoft Teams as already described earlier:

- Time
- Participants
- Agenda
- Notes on each agenda item discussed

If necessary for a meeting, the content is adjusted to provide the best possible benefit for the reader.
The minutes are not sent to all participants after each meeting, as agreed with the supervisors. However, everyone may request see the protocols at any time during the project.

3 Development

3.1 Version Management

For the administration of the source code and the project documentation we use GitLab provided by the OST, including the provided continuous integration solution.

We have defined various guidelines for dealing with Git to achieve a professional and simple cooperation in the software development process in the team. See Code reviews

The branches are created and used with the concepts and tooling of Git Flow (AVH Edition)[18]. We use feature, bugfix, release and hotfix branches with the two main branches develop for the development environment and master for the production environment. We do not need support branches in this project. All branches are named in kebab-case, unless the release and hotfix branches, which are written in semantic version numbers without a prefix.

All commits must follow the Conventional Commits[19] guidelines. This ensures that the commit messages are uniform, describe the extensions and adjustments exactly and the change history is quickly traceable.

3.2 Principles

To ensure maintainability and quality of the code basis, common software engineering principles are always considered. These include as a basis KISS, YAGNI, DRY, BDUF and S.O.L.I.D. Further we adhere to the clean code principles.

3.3 Quality

3.3.1 Definition of Done

We work with different DoD's for development tasks. For quality assurance, the issues can be populated a corresponding DoD. The Definition of Done serves the developers as a guide for the implementation. We define standardized DoD's for the backend and frontend. These can be supplemented individually for each issue.

3.3.1.1 Backend

- Coding Conventions were respected
- No linter errors
- New potential unit tests introduced[2]
- All unit tests passed successfully
- Continuous Integration went through without errors
- All findings from the code review have been fixed
- The documentation was expanded or adapted if necessary[2]

3.3.1.2 Frontend

- Coding Conventions were respected
- Continuous Integration went through without errors
- All findings from the code review have been fixed

[2] The developer of the new or adapted code is responsible to decide if unit test and documentation changes or expansions are necessary directly.

- The documentation was expanded or adapted if necessary[2]

3.3.2 Code reviews

No changes are incorporated into the Git repositories, which have not been validated and confirmed at least by the other team member in a review. Exceptions to this are trivial configuration work and bug fixes, which must be introduced immediately.
We always create pull requests that are checked by the other team member. We always work with support branches and only after the code review, the merge to a main development branch is performed.

3.3.3 Testing

Unit tests are used as a first test strategy for our backend application to be developed and its potential surrounding systems. Unit tests are a common way to test functionalities of an application with different scenarios and can easily be included in the Continuous Integration (see chapter Continuous Integration and Deployment).
If a frontend is developed, only manual frontend tests are performed by the developers, because automated tests are not worthwhile for a potential frontend of this small size and necessity.

As a second test strategy, a complete system test will be performed near to the end of the construction phase and recorded in the appendix of this document.

In addition to the two test strategies, manual tests are also carried out repeatedly during development in order to guarantee the correct functioning of the automated tests themselves.

3.3.4 Static code analysis

In order to be able to statically check our code, a SonarQube instance was setup on our server at the INS. SonarQube can analyse static code from many common programming languages and detect errors, weaknesses as well as ugliness's and visualize them on a dashboard.
This tool will also consolidate in our Continuous Integration system so that all code is automatically analysed during the development workflow.

The code metrics we use and our goal for each metric are described in the next chapter.

3.3.4.1 Code Metrics

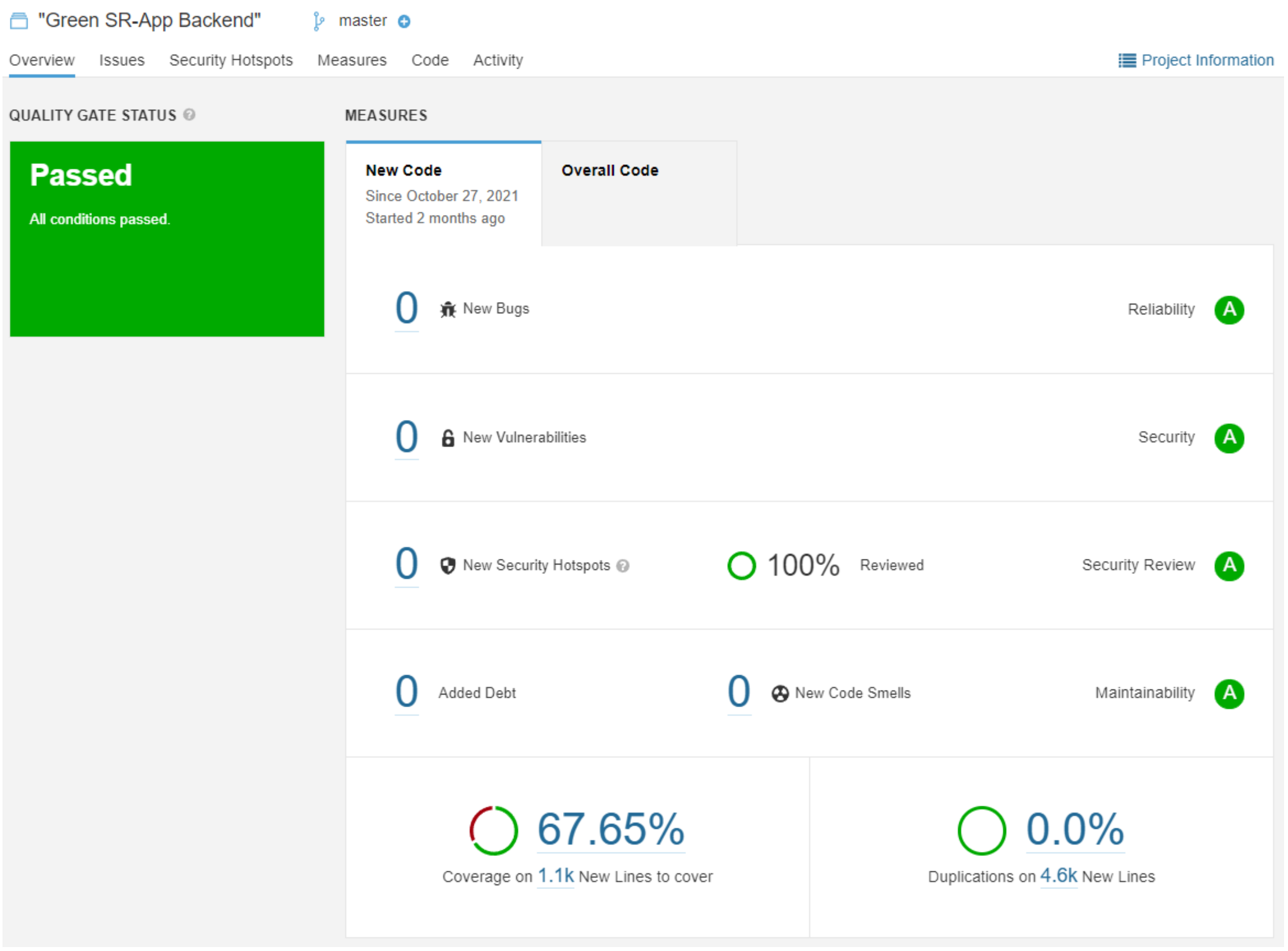
By choosing to use SonarQube as a static code analysis tool, we limit ourselves to the most important metrics that are used by this tool. The main metrics are explained below and our goals for each are defined as well.

Name	**Explanation**	**Goal**
Bugs	Bugs are errors in the code that can cause the application to stop working.	0
Vulnerabilities	An application vulnerability is a flaw or weakness in the system that can be exploited to jeopardize the program's security.	0

Security Hotspots	Code which needs manual checks to ensure that there are no security flaws.	0
Code Smells	Parts in the code which are hard read and understand.	0
Test coverage	Defines in percent how much code is tested by unit tests.	min. 80%
Duplications	Defines in percent how much lines of code are duplicated over the whole application.	max. 1%

Table Part B 3.1: SonarQube code metrics

The next screenshot shows the final Quality Gate status on the platform. SonarQube has been reconfigured so that a code coverage value of 60 percent is accepted and marked as “Passed”. This has the origin that due to missing real data from Jalapeño and the attached network, single mocking’s had to be implemented on the service application layer. In addition, for the integration tests, the deepest application layer, namely the repository layer, also had to be mocked so that the tests could run through all application layers without a real or in-memory database.


"Green SR-App Backend" master

Overview Issues Security Hotspots Measures Code Activity Project Information

QUALITY GATE STATUS

Passed
All conditions passed.

MEASURES

New Code
Since October 27, 2021
Started 2 months ago

Overall Code

0 New Bugs — Reliability A

0 New Vulnerabilities — Security A

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

0 Added Debt — 0 New Code Smells — Maintainability A

67.65% Coverage on 1.1k New Lines to cover

0.0% Duplications on 4.6k New Lines

Figure Part B 3.1: SonarQube Guality Gate Status

3.3.5 Coding Conventions

We use the coding guidelines of the CockroachDB which describes a wide variety of different guidelines for style, performance and best practices. [20]

To ensure good code quality, we rely on two tools. One is golangci-lint [21] which applies a large set of rules through different single linters every time a file is saved and also enforces these rules when committing with Git. We have some control over which rules are applied and in what manner. We can add linters if we think they help improve the code quality and we can remove linters if we think that the overhead is too much or if the linter runs into problems.

The second tool is gofmt [22], which also formats the entire document according to standard formatting rules every time it is saved. This tool ensures that the code is easy to read, which leads to potential less bugs.

3.3.6 Continuous Integration and Deployment

For the complete CI/CD we use the GitLab integrated CI/CD feature with custom runners from the INS to have better performance and all access needed to the infrastructure.

The pipeline is separated into three Docker image strategies. For linting, testing and SonarQube scanner there is a build only image used with the source code and configs included. The two other strategies are for development and production deployment builds only and do not include source code (only the executable) and have their specific environment configs included in the lightest way possible.

In the following table all stages are shown chronologically and described.

Stage	Description	Fail level	Execution Target
build	Builds the backend based on the build only image[3] with optional use of the cached Docker image with tag "develop" and tags it with the latest commit hash.	Build failure	Every commit
lint	Executes the complete linting for the build image and displays the results.	Linting errors (not warnings or information's)	Develop, feature, bugfix, release, hotfix and merge requests
test	Executes all unit tests from the backend.	If at least one test fails	Develop, feature, bugfix, release, hotfix and merge requests
sonar-scanner	Executes the SonarQube check through sonar-scanner.	If a code metric is not met as expected (described in chapter code metrics)	Develop, feature, bugfix, release, hotfix and merge requests
build-dev*	Builds and tags the development deployment build image[4] with "develop".	No directly associated fail level	Develop
build-prod*	Builds and tags the production deployment build image[4] with "latest" and the specific version for the application (semantic version).	No directly associated fail level	Master (implicit only tags)
deploy-dev*	Deploys the development deployment image to the development environment server.	If deployment is not successful	Develop

[3] Dockerfile name: «Dockerfile»
[4] Dockerfile name: «Dockerfile.deploy»

deploy-prod*	Deploys the production deployment image to the production environment server.	If deployment is not successful	Master (implicit only tags)

Table Part B 3.2: CI/CD pipeline stages

*Due to prioritization on more important topics, the deployment stages in the context of this work were waived in the construction phase with the supervisors approval.

3.4 Error Handling

3.4.1 Input validation and output sanitization

In order to generally avoid considered errors inside the application, all data objects for creation or editing via the API endpoints are validated with the help of clearly defined data fields and data types on predefined data transfer objects, or in short DTO's. The API works for ingress and egress data only with DTO's and so entities[5] will strictly never leave the application. This prevents also exposing or allowing incoming data, which is not intended to leave the API to the consumer.

3.4.2 HTTP status

Whenever possible, the backend responds with a suitable HTTP status code with an error message if this is available. Special errors in exceptional cases are handled by default with the code 500 Internal Server Error.
The application itself only crashes completely in case of fatal errors. All other errors that are not explicitly handled and the so-called Go panics are implicitly handled by the Gin Web Framework without resulting in crashing the application. These are then also handled with the standard Internal Server Error code.
In the next table, a simple example is shown in reference to the previously mentioned points.

Request	**Response Header**	**Response Body**

GET /node/1	HTTP **404 Not Found**
	
	Content-Type: application/json
	Content-Length: 28
		{
		"**error**": "element(s) not found"
		"**identifier**": "example"
		}

Table Part B 3.3: HTTP Status example

3.4.3 Logging

To meet the eleventh point of the Twelve-Factor Methodology and to have a well-founded logging possibility, an extended logger was implemented instead of Go's own and registered on all other included frameworks and libraries as the main logger. This logger supports the following seven log levels, listed from the most serious to the most marginal level: fatal, panic, dpanic, error, warn, info, debug. As also described in the previous chapter HTTP status, only fatal errors will cause the application to stop. All other log types will not stop the backends runtime and are handled through the Gin Web Framework. The log service writes its output directly to the stdout event stream of the

[5] Entities are also called models and they define how the data must be stored on the database.

operating system it runs on. The level used can be set at application startup via the environment variables.

3.5 Environment

The IDE's from JetBrains are used with additional plugins from the JetBrains marketplace depending on the needs by the developer itself. Depending on the technology used, the appropriate IDE from JetBrains is used. GoLand IDE is the product which is used for Golang development.
In addition, all required services, such as databases, are provided with Docker and defined in the repositories via docker compose files, so that a fast setting up of the development environment on any device can be made possible.

The development is supported by the version control system remote server GitLab, which also offers additional functionalities. Additional functionalities used are GitLab CI/CD, merge requests and integrations with Kubernetes and YouTrack (described in chapter tooling).

4 Domain Analysis

4.1 Domain Model Diagram

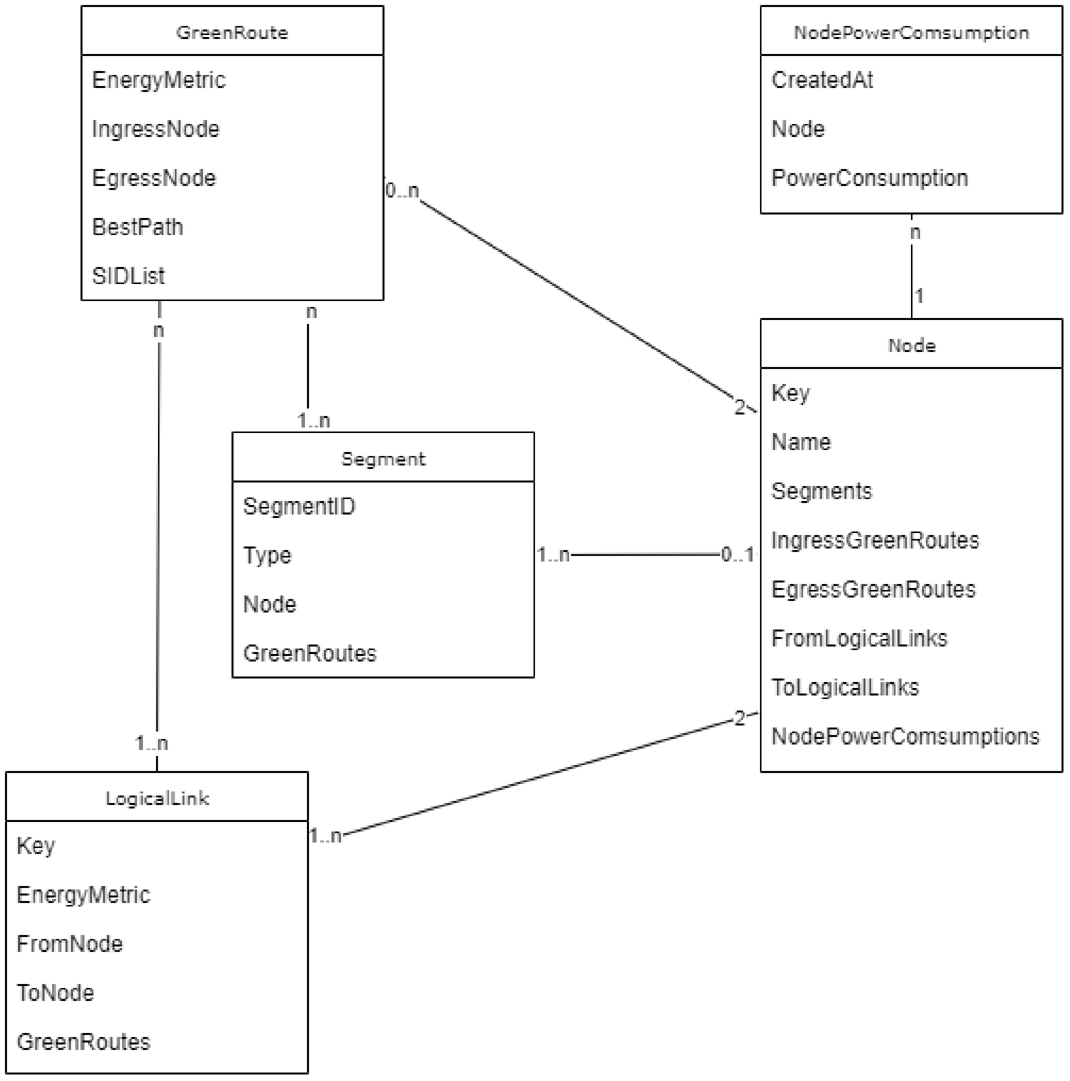


Figure Part B 4.1: Domain Model

4.2 Domain Model Explanation

4.2.1 Node

A Node is an ASR 9000 router in the segment routing domain which actively participates in segment routing. For this thesis we limit ourselves only to the ASR 9000 router from Cisco with the IOS-XR software.

The router has several logical links connected. In segment routing the source Node is called ingress. This is the point where the packets enter the segment routing domain. The other end is called egress and this is the point where the packets leave the domain again. It is also the destination Node.

Because of the attributes we explained above, an ingress and egress Node must be included in the GreenRoute to know the start and the end of the path.

4.2.2 NodePowerConsumption
Due to the additional use case UC08, gather statistics, it is necessary to not only store the current node power consumptions, but also to keep a history of the values synced.
Each Node has a history of power consumptions associated. The power consumption with the latest CreatedAt timestamp is the most recent synced value from the network.

4.2.3 Segment
An essential part of segment routing is the Segment itself. Each Segment describes a different operation, telling a Node which action it has to execute when a packet with a certain Segment is received. Depending on the context the Segments can lead to different actions. A Segment belongs to a particular Node and may be available in zero or more GreenRoutes.

4.2.4 LogicalLink
A LogicalLink is a connection between two Nodes. Relevant information is stored on such a type of link, for example in which direction the data traffic flows and what metrics are present on it. LogicalLinks is an important part of the green route calculation and can therefore be found in the GreenRoute.

4.2.5 GreenRoute
A GreenRoute contains the greenest path as well as the Segment list or so called SIDList.

The greenest path describes the path which consumes the least amount of energy and packets should therefore follow this defined route. It needs at least one LogicalLink to be a working result.

To achieve a correct list of Segments, it needs at least one Segment in the GreenRoute, if segment routing is enabled. Normally there will be more than one Segment in the SIDList.

5 Architecture and design specifications

5.1 System Overview

To give the readers a visual overview over the existing software and the newly developed app, several diagrams according to the C4 model [23] were created.

Because the API Gateway notifies the Green SR-App on topology changes and new telemetry data The Green SR-App does not need a special cache to handle the current data. It is possible to request the needed data and process it just in time. This way it is possible for multiple instances to have the same results for the calculation because all instances request and get the same data.

The following C4 system context diagram shows the user, as well as the Green SR-App in development and the Jalapeño API Gateway which accesses Jalapeño from Cisco Systems.

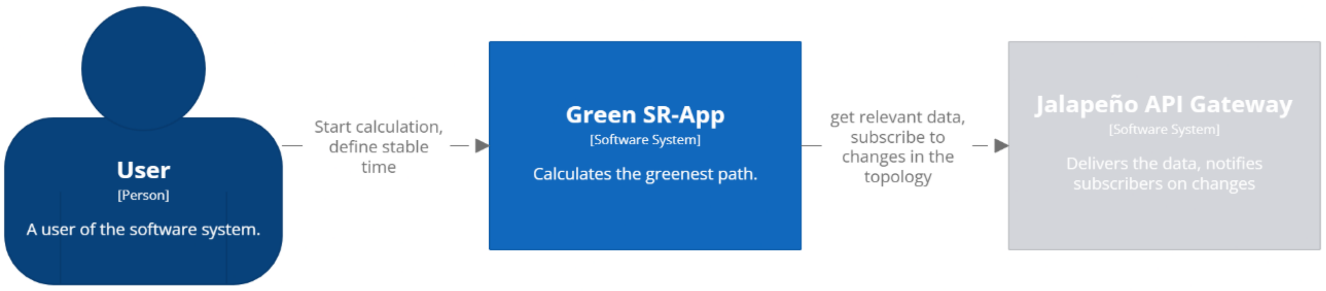

User
[Person]
A user of the software system.
Start calculation, define stable time
Green SR-App
[Software System]
Calculates the greenest path.
get relevant data, subscribe to changes in the topology
Jalapeño API Gateway
[Software System]
Delivers the data, notifies subscribers on changes

Figure Part B 5.1: C4 model System Overview

The following C4 container diagram shows how the different containers interact with each other and which protocols are used. In addition, the diagram shows how the user communicates with the software system. The external System Jalapeño API Gateway is not visualized in detail, since we only use the provided services of the gateway.

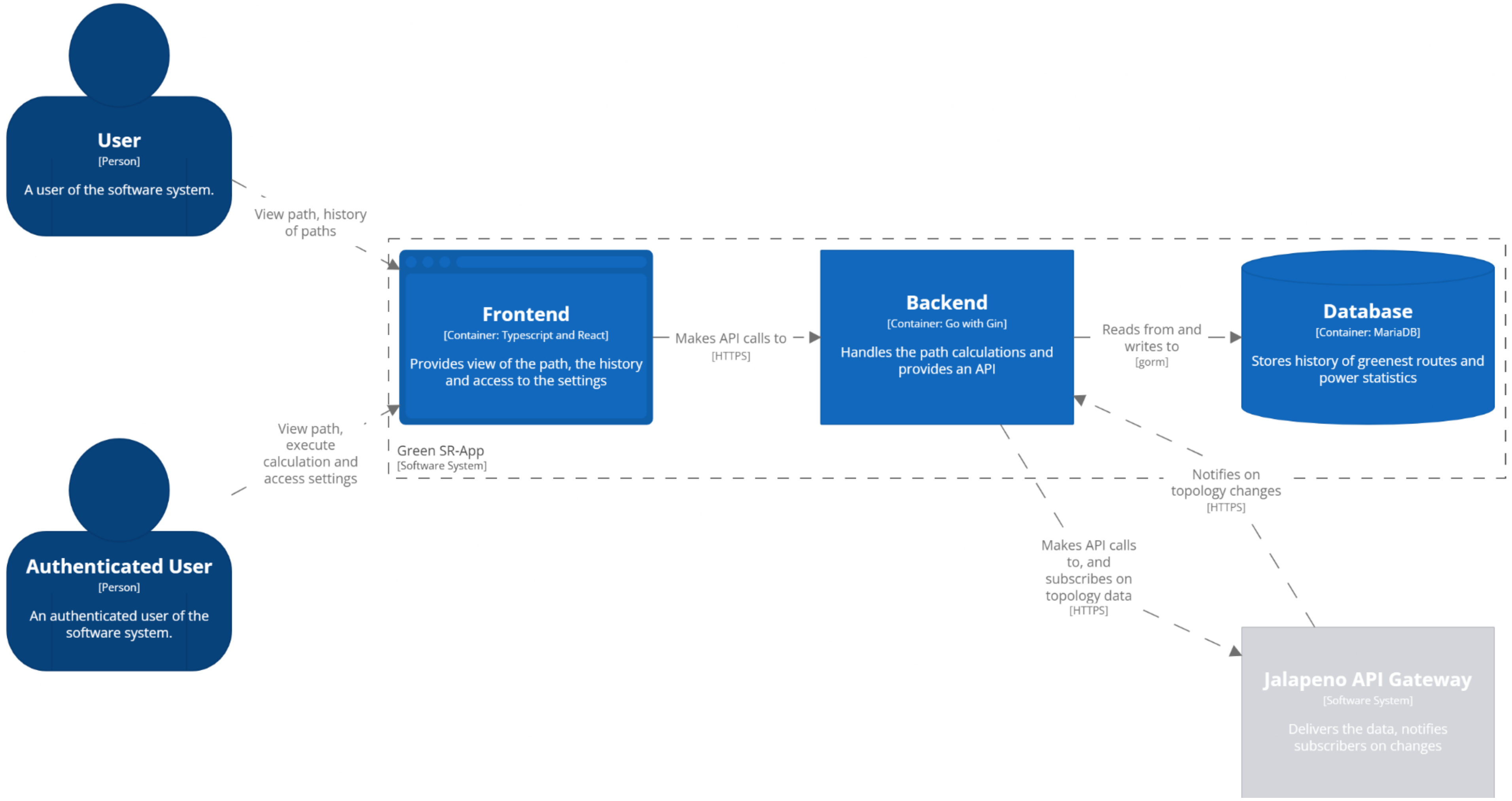

User
[Person]
A user of the software system.
View path, history of paths
Authenticated User
[Person]
An authenticated user of the software system.
View path, execute calculation and access settings
Frontend
[Container: Typescript and React]
Provides view of the path, the history and access to the settings
Makes API calls to
[HTTPS]
Backend
[Container: Go with Gin]
Handles the path calculations and provides an API
Reads from and writes to
[gorm]
Database
[Container: MariaDB]
Stores history of greenest routes and power statistics
Green SR-App
[Software System]
Notifies on topology changes
[HTTPS]
Makes API calls to, and subscribes on topology data
[HTTPS]
Jalapeno API Gateway
[Software System]
Delivers the data, notifies subscribers on changes

Figure Part B 5.2: C4 container System Overview

5.1.1 Jalapeño API Gateway

The Jalapeño API Gateway enables the Backend to access the stored date of Jalapeño in an easy and standardized way over a gRPC API. Additionally, the Gateway API notifies the Backend on changes in the topology or on new telemetry data. This way the application can react on changes and can calculate a new path if necessary.

Jalapeño is a software from Cisco which collects and processes telemetry data and topology data of the connected network and stores them in an InfluxDB for telemetry data and in an ArangoDB for topology data.

5.1.2 Frontend

A user who uses the Green SR-App will access the app via the frontend, in which it is possible to view the chosen route, which represents the most energy efficient way. It is also possible to trigger a new calculation if necessary. In the settings there will be an option through which the user can define the duration in which the route does not change under the condition that the topology stays the same.

This part is optional and will only be part of this thesis if there is enough time left after the mandatory use cases have been fulfilled.

5.1.3 Backend

The backend is the heart piece of the whole application. Here, the data from the API Gateway is evaluated and prepared for path calculation. When the data is in the required form, the path calculation is executed, and the most efficient path is evaluated. The result will be sent to the frontend and will just be stored in the database for later statistical analysis.

The path calculation gets triggered when a change in the topology occurred or when the defined duration has passed. To keep energy costs low, the calculation is only executed when needed. To get a reliable reading of the power consumption of the routers the application takes the average power consumption over 24 hours. If the available time range is smaller than 24 hours, the application calculates the average over the available time range. If possible, we want to know how much energy a router needs to process a certain number of packages. This way more efficient routers will be considered.

5.1.4 Database

In the Database the calculated routes will be stored for future statistical analysis. The application also stores other potentially interesting metrics to create reports on. The reports will not be part of this thesis. It was decided to use a MariaDB because it offers the possibility to store graph data as well as relational data. It works well with go and it is possible to run it in memory if we need the performance boost. This way we can store the chosen route as a graph and the power data in a relational table.

5.2 Twelve-Factor App Methodology

For the application to be truly cloud native standard compliant, it will be built using the methods of the 12-factor methodology. [9]
The entire application with its environment and services is analysed step by step using this methodology in the following subchapters.

5.2.1 Codebase

"One codebase tracked in revision control, many deploys" [9]

GitLab is used as the version control system, which also allows the direct use of an integration for CI/CD and connections to YouTrack.
The backend and the potential frontend are created in individual repositories with completely seperated code and have deployments to a strictly separated development and production environment on a server with different accessing definitions (i.e. different URLs). We will not use a direct staging environment because based on our concepts, the development environment behaves similarly.

The backend and the frontend could each be verified completely independently using the 12-factor methodologies.

Evaluation: fulfilled

5.2.2 Dependencies

"Explicitly declare and isolate dependencies" [9]

The application consists classically of a frontend and a backend, which serves as an API. It is not yet completely clear whether there is enough time for a frontend as we are working in iterations and therefore only the backend is described here.

Go implemented with version 1.12 a new dependency management system, which describes every dependency used with their peer dependencies in a file called go.mod. This file also includes definitions for which version should be used including for the peer dependencies.
With version 1.12 also the possibility for modules was introduced which we use in the backend to separate the code clearly and in a structured way. Each package includes apart from the domain driven design.

Evaluation: fulfilled

5.2.3 Config

"Store config in the environment" [9]

Environment variables are used to define all the static configurations for the application for the backend and the potential frontend.
They are set locally by the developer based on default environment files per possible environment and injected in the pipeline from the environment configuration for the pipeline with additional sensitive config variables provided by GitLab.

The library GoDotEnv is used to fully reach this in the Golang context. Every dynamic variable which differs between environments is set in a specific environment file, e.g. in the file .env-local for local development usage. The application loads the environment variables on startup which then can be consumed by the application.

Evaluation: fulfilled

5.2.4 Backing services

"Treat backing services as attached resources" [9]

The MariaDB is connected via URL which is composed of different values from the environment file. It is possible to change the underlying database as long as it is still a MySQL database provider. It is even possible to change the whole repository layer to use a different kind of database, if necessary. If you want to use a different MySQL provider, no changes to the code are required.

Evaluation: fulfilled

5.2.5 Build, release, run

"Strictly separate build and run stages" [9]

The backend CI/CD has eight clearly separated stages defined. They are fully described in the chapter continuous integration and deployment. In summary, there is one build stage for checking purposes, three quality stages composed by linting, unit testing and static code analysis, two build stages for development and production and finally two deployment stages for each environment.
The pipeline clearly separates build, release and run from each other and traversing the path in the other direction is not possible or not foreseen after following the correct workflow.

A clearly defined path is achieved through the three major steps GitLab, GitLab CI and Kubernetes deployment. One is for version control and the other ones for continuous integration and deployment. The running part then takes place on the server with Kubernetes.

Evaluation: fulfilled

5.2.6 Processes

"Execute the app as one or more stateless processes" [9]

The app must act stateless. This means that it must not save any states at runtime, except for a short period of time, such as during further processing of data. However, data stored for a short time must never be intended for a future request. All data to be stored for a long time must be stored in associated services, such as databases.

The app is completely stateless since we always calculate based on the telemetry information which is currently available from Jalapeño. This could be one or more telemetry entries based on the specified time range or the network uptime until now. All data intended for long-term storage use is stored in the relational and synchronized database. Short-term data is only stored on a per-request basis.

Evaluation: fulfilled

5.2.7 Port binding

"Export services via port binding" [9]

Each service offered must be mandatorily bound to a port. It must not build on runtime injection from a webserver. The web app must provide HTTP as a service by binding to a port and listen to ingress traffic on that port.

This is achieved in Go by direct port binding, which is also fully provided also by the Gin Web Framework. The application only requires one port for handling HTTP traffic.

Evaluation: fulfilled

5.2.8 Concurrency

"Scale out via the process model" [9]

Each running program is represented by one or more processes. Web applications have a variety of different forms of process execution as a possibility.

The computation-heavy work, consisting of the synchronization of Jalapeño and the subsequent computation of the green path, was achieved by a parallelization with semaphores and mutexes. Go has inherently very powerful parallelization capabilities. All other parts of the application are already fast enough without parallelization, even with very large amounts of data.
Go also inherently offers vertical scaling of the processors, which is also compatible with the Green SR-App.

Evaluation: fulfilled

5.2.9 Disposability

"Maximize robustness with fast startup and graceful shutdown" [9]

The app processes should be disposable. This means that they must be able to be started and stopped in very short time. By defining this short time period, it is meant for only a few seconds and not longer. Additionally, it must be possible to gracefully turn off the application.

The processes were designed to be very sleek and lightweight, allowing for a very quick startup and shutdown. However, it is not ensured that ongoing requests and calculations have been completed when a shutdown request is incoming.

Evaluation: partly fulfilled

5.2.10 Dev/prod parity

"Keep development, staging, and production as similar as possible" [9]

This point requires that the environments used should be as similar as possible. In this project, the environments development and production are used.

In the next table, you can see the comparison of traditional developed apps to twelve factor apps with an additional assessment for the Green SR-App developed in this term.[6]

[6] The idea and parts of the content for this table was taken from the 12-Factor Methodology website [9].

	Traditional app	Twelve-factor app	Green SR-App
Time between deploys	Weeks	Hours	Hours to days
Code authors versus code deployers	Different people	Same people	Same people
Development versus production environment	Divergent	As similar as possible	As similar as possible

Table Part B 5.1: Dev/prod parity comparison

The dependent services, such as databases, are kept the same and the build differences are reduced to a minimum.

Evaluation: fulfilled

5.2.11 Logs

"Treat logs as event streams" [9]

A twelve-factor app should never itself manage the forwarding or storage of generated logs. Every running process must send its logs as event streams directly to stdout without any intermediate steps.

The developed app writes all logging information directly to the output stream (including stdout and stderr) without any intermediate steps. It is possible to set the log level via the environment variables.

Evaluation: fulfilled

5.2.12 Admin processes

"Run admin/management tasks as one-off processes" [9]

Here it is about single tasks to be executed, which are needed on the productive system, for example to start database migrations. These tasks must always be executed on systems that are as similar as possible.

By using containers for all application parts, such tasks are not directly necessary. In the event of changes or adaptions, the container is easily redeployed. Migration scripts are sometimes necessary for certain services used, but these are always tested first in the development system that is as similar as possible to the production environment.

Evaluation: fulfilled

5.3 Technologies

Component	Technologies and Frameworks	Libraries/Extensions
Backend	• Golang • Gin Web Framework • Jalapeño API Gateway	• RyanCarrier/dijkstra • Gin-contrib/zap • Jinzhu/Copier • Joho/GoDotEnv • Stretchr/Testify • Swaggo • Zap • gRPC • Zapgorm2
Database	• MariaDB	• Gorm • Gorm MySQL driver
Development support	• Golang • Docker and Docker Compose	• Go run, build, fmt, test, clean and mod • GoDotEnv • Golangci-lint • Swaggo

Table Part B 5.2: Technologies overview

5.3.1 Programming Language

It was primarily decided on three possible programming languages for this term project. These include Node.JS, Python and Golang. Each of these three programming languages offers its own advantages for the project as well as from the perspective of the INS and the team members. Node.JS is very well known by the team members, whereas the know-how on Python and Golang is very basic. Python as well as a little less Golang have already been used in the INS.
Tailored to this project and the associated calculation of paths in networks, Go is in comparison up to three times as fast as Node.JS and up to eight times as fast as Python [24].
The Jalapeño system from Cisco and the Jalapeño API Gateway of the INS use Go as programming language.

Due to the very good performance and the widespread use in peripheral systems, the decision was made in favour of Go, although this means increased effort for the project members to become familiar with it. The performance and extensive use in existing systems were conscious considered as major factors.

5.3.2 Web Framework

To facilitate the handling of an API for developers, an additional framework is used. The choice went to the most popular web framework for Go with the largest online community to support it.

Gin Web Framework is a framework for Go, which follows a Martini like API, but is up to 40 times faster than Martini itself. Gin is suitable for API's where performance plays a big role.
In addition, this web framework also offers middleware’s and supports a crash-free API. It can

therefore catch and recover from panics that occur in the Go context. Thus, the server always remains available. In addition to the functionalities already mentioned, Gin can also easily validate JSON and improve routes [10].

5.3.3 Storage

5.3.3.1 Long-term

5.3.3.1.1 Relational or document oriented

In order to ensure more flexibility during the development for the storage of the calculated paths and the related statistics, a relational database is used. After this term project and the experiences made, it must be revalidated whether a document store database could be used for simplification and additional performance increasement.

5.3.3.1.2 Database management system

The relational database services PostgreSQL, MySQL and MariaDB were included for detailed selection.
Golang supports all three in somewhat equal measure, so this criterion can be excluded. PostgreSQL is widely used at the OST. MySQL, on the other hand, has no direct advantages over PostgreSQL for this project, except for the downside that it is used less. MariaDB, on the other hand, offers more database engines than MySQL and PostgreSQL, including a secondary engine for Graph DBMS, which can be an advantage when storing paths in networks. MariaDB is also a very modern and high-performance database system for small to medium-sized databases [25].
Due to the described advantages and disadvantages MariaDB is used for this project.

5.3.3.1.3 Object relation mapper

In order to ensure a clean transition from the application to the database and to automate database migrations, an ORM system must also be used. Here, too, an ORM library was chosen that is most widely used in the Go world and supports our relational database technology.
Gorm is the most popular ORM system for GO, which also supports MySQL and therefore also MariaDB [12].

5.3.3.2 Short-term

Consideration was given to whether additional short-term storage could provide a direct benefit. It is possible to provide the calculated paths faster in the cache and to keep telemetry data for the calculation in the cache for a short time.
Since Jalapeño writes the telemetry data directly to the TSDB at a predefined interval and the Jalapeño API gateway can deliver either a live subscript or a query request for a defined period, no dedicated short-term storage is required on the Green SR-App.

This term project is only about providing the interface, relating the data, and calculating the paths, but not yet about setting the new paths on the routers. Because of this, a cache for the calculations brings no significant benefit from this point of view.
As a result of the main facts described before, no dedicated cache is used in this project.

5.3.4 Development support

5.3.4.1 Linter's runner

For go, there are many different linters, all of which cover certain parts or some of which are for a single purpose. So that not all of them must be integrated and executed individually, the golangci-lint Go linters runner is used in this project. This linters runner offers the possibility to execute all available linters and to configure them individually if needed [21].
All go own linters and recommended linters are used at least. The exact list of the used linters and checkers can be seen in the appendix (chapter Golangci-lint linters).

5.3.4.2 Testing

Since the execution of tests is not carried out via the linter's runner, the unit tests must be executed separately. This is achieved by the command "go test" included in Go.

In addition, a library called Testify is used to simplify the definition of unit tests. Testify mainly offers the advantage of wounding assertion checks, which are widely used in unit tests [26].

5.4 Backend Architecture

This chapter contains an overview over the backend architecture and shows the connections between the used components. We differentiate between the frontend, which we do not cover in more detail, the backend with the whole business logic and API's and the database.

We follow the domain driven development concept with the three layers application, domain model and infrastructure. Where in the application layer we find our controllers which form the API. In the domain we manage the models in different entities files as well as the data transfer objects (DTO). And finally, we have the infrastructure layer where we have the services and the repository.

Additionally, we placed the logger and the environment (config) also in the infrastructure layer. We have a go specific Cmd container which is the entry point into the application.

The common container handles migrations and seeds which are not used during the normal runtime but rather to handle changes in the database and to provide a set of default data in the database.

The frontend receives all its data over the API. The API gets the necessary data via the data transfer objects from the services. The services get the data over the database handler which manages the database connection. All config data is provided over the environment files and all container which use config data therefore access the environment container. The same goes for the logger. All containers which need to log information access the logger.

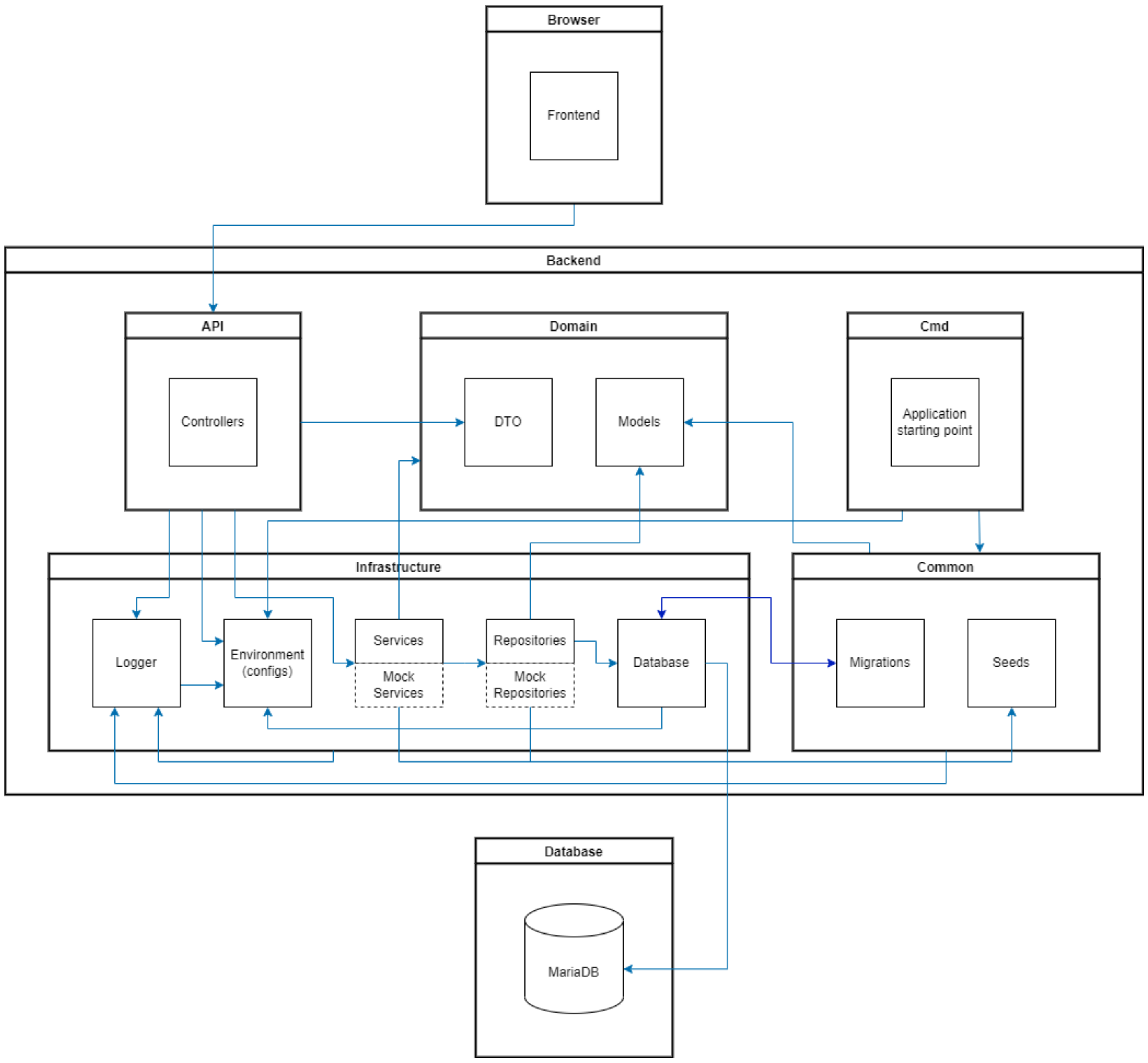


Figure Part B 5.3: Backend Architecture Overview

5.4.1 Backend

The Green SR-App backend consists of the API, domain, infrastructure and common sub-modules. Each of these sub-modules has at least one, but usually several packages in it. In the following chapters the sub-modules including their packages are described.

5.4.1.1 API

The sub-module API consists of only one package, in which all controllers of the application are defined as a single file per available endpoint.

The controllers are the first point of contact for an incoming request and therefore also define how the endpoints look like and what functionalities and operations they offer. In addition, they are responsible for defining the input and output and for validating the input. HTTP error codes are also defined here based on the results from the services and other individual factors.

5.4.1.2 Domain

In the sub-module domain at least two packages are to be found and optionally even more, which are not yet to be defined in this high-level architecture and are created during the development.

The package DTO contains all data transfer objects which are used by the controllers and the services as transfer data objects. These DTO’s are simply said the objects that live between the layers and they transport their produced data between each other.

Models is the package that contains various entities. Entities define what the data for the persistent storage should look like. They also implicitly define how the database schema should look like.

5.4.1.3 Infrastructure

In the infrastructure sub-module, most of the packages explained so far are present. In the setup there are five packages, most of which are only intended for the internal application part.

The most important package called Services contains the entire business logic for each functionality. The services receive requests from the controller and returns the computed result. This practically always includes communication with the repositories in order to access the persistent data. They also very often do not pass on the data directly but add the desired business logic including various needed calculations. The individually used Mock Services include the same logic than the normal services, but simply with realistic mock data.

Probably the second most important package in this sub-module besides the services is the repositories package. The repositories take care of the direct database queries. They are defined by default via the ORM library and offer the most important CRUD queries to the database by default without any additional definition. As soon as non-standard defined queries are needed, the additional repository definition files with raw queries or a query language are created. The individually used Mock Repositories include the same logic than the normal repositories, but simply with realistic mock data.

In the third package called Database the connection to the database is initiated and set up. It makes an active database connection available to the repositories.

The fourth package in the bundle is called Environment and takes care of loading and providing the environment variables defined in an environment file or directly from the environment variables of the system.

Finally, the package logger is a small heart of the application. It takes care of the complete logging of the application into the stdout and the stderr. It offers different log levels for different purposes and environments. This level can be set via an environment variable.

5.4.1.4 Cmd

This sub-module does not have a directly existing package in it. Because of the chosen programming language Go it is common to place the start of the application in a package called Cmd. There is no further program logic in this package and so it is often called “main”-function in other programming languages.

5.4.1.5 Common

In this container we place all the packages which do not belong into the DDD context and are not used in the normal runtime. The migration handles changes of the database schema and the seed populates the database with initial data or test data in a test or development environment. It is possible that additional packages will follow.

5.4.2 Database

This container houses the database which is a relational MariaDB. It stores all the calculated paths as well as other statistical data. It gets accessed via gorm which is a ORM library for go which is located in the repositories package.

5.5 REST API

The API was built according to standards and best practices known in API development and does not deviate from them. These standards and best practices are well known by both developers and therefore no further external source was consulted to fulfil the API definition.
It was a major concern at development time to make the API endpoints as feasible and simple as possible.

Special mention should be made about the definition of the POST request for the calculation of the greenest path. It was intentionally decided by the developers to specify the resource to be created in the path of the request. The alternative passing of the information via the body was not chosen because the data to be created is fully calculated by the backend and not by user creating the request.

5.5.1 API responses

In the following table all possible responses of the API are defined in detail. In the default success scenario, the requested resources are sent as a DTO object(s) back or in specific cases an empty response is also valid.
Following individual DTO objects are provided:

- GreenRouteDTO
- NodeDTO
- NodePowerConsumptionDTO
- LogicalLinkDTO

For more detailed information the source code of the Green SR-App backend should be consolidated.

Definition	Status type	HTTP status code	Response object with fields
Default	Success	200	Individual DTO (object) or empty response
Base	Success	200	BaseResponse • Message[7] (string) OpenAPIDocs (string)
Telemetry data	Success	200	TelemetryDataResponse • FakedData (bool) • TelemetryData (array of EnvmonUI) ○ Time (time) PowerConsumed (integer)
Not found	Failure	404	NotFoundResponse • Error[8] (string) Identifier (string)
Internal Server Error	Failure	500	ErrorResponse • Error (string)
Not Implemented	Failure	501	ErrorResponse • Error[9] (string)

Table Part B 5.3: API responses

[7] Response String: «Hello from the Green-SR App API»
[8] Response string: «element(s) not found»
[9] Response string: «Not supported due to FAKE_ALL_DATA activated»

5.6 Entity relationship diagram

This following entity relationship diagram, or ERD, was generated with MySQL Workbench 8.0 CE based on the realized database schema. More information about the ERD display style by MySQL Workbench can be obtained from the referenced source. [27]

For readability reasons, all table names, fields, field data types and foreign key contraints are displayed, but not the indexes on the individual tables.

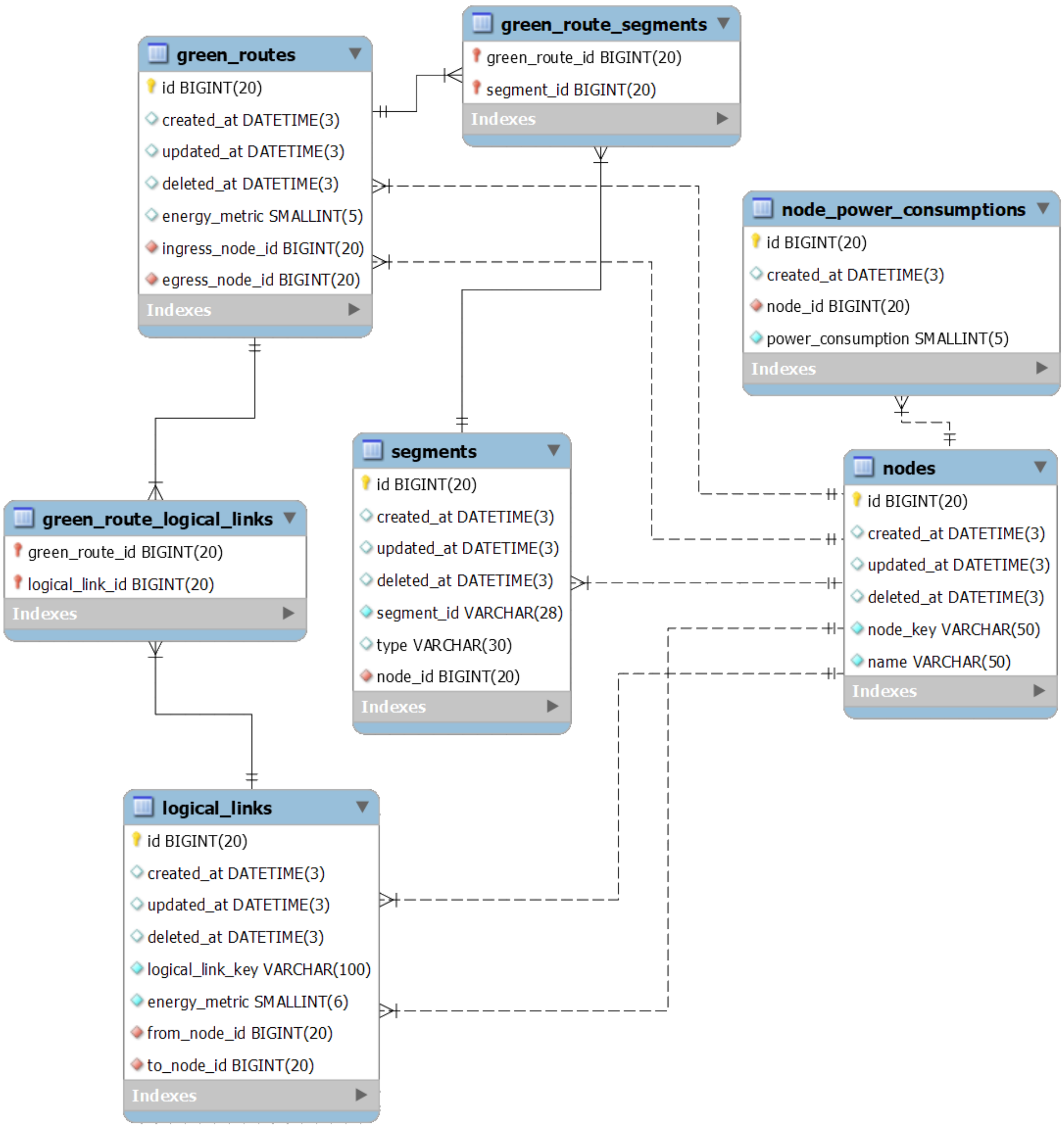


Figure Part B 5.4: Entity relationship diagram

5.7 Infrastructure

5.7.1 INS Lab

A server was set up in the INS network using Kubernetes, which provides all the necessary services for this project through various Kubernetes pods. As explained in the Technical Report, the following environment has been set up:

- Webserver (Nginx)
- Jalapeño API Gateway
    - Request and Subscription Service
    - Cache and Cache Service (Redis)
- Jalapeño
    - Time Series Database (InfluxDB)
    - Graph Database (ArangoDB)
    - Telegraf ingress
    - Gobmp ingress
    - Event streaming (Kafka)
- SonarQube
    - Web Platform
    - Relational Database (PostgreSQL)

The latest versions of the new Green SR-App are also always deployed to this server.

In order to obtain the router (Cisco ASR 9000) configuration and data relevant to this study, a virtual lab was set up by the INS. All devices are accessible by SSH over Port 22. The lab consists of the following virtual devices in the next table.

Name	Type	Intended use
XR-1	Router	Base ASR 9000 like router that routes traffic on the network with segment routing.
XR-2	Router	Base ASR 9000 like router that routes traffic on the network with segment routing.
XR-3	Router	Base ASR 9000 like router that routes traffic on the network with segment routing.
XR-4	Router reflector	Special ASR 9000 like router reflector which listens on all other router updates. Main data source for Jalapeño.
XR-5	Router reflector	Special ASR 9000 like router reflector which listens on all other router updates. Main data source for Jalapeño.
XR-6	Router	Base ASR 9000 like router that routes traffic on the network with segment routing.
XR-7	Router	Base ASR 9000 like router that routes traffic on the network with segment routing.

XR-8	Router	Base ASR 9000 like router that routes traffic on the network with segment routing.
Cust-A-ZRH	Customer Network entry	Sample private customer network on one side of the whole network.
Cust-B-ZRH	Customer Network entry	Sample private customer network on one side of the whole network.
Cust-A-BSL	Customer Network entry	Sample private customer network on one side of the whole network.
Cust-B-BSL	Customer Network entry	Sample private customer network on one side of the whole network.
Cust-A-ZRH-PC1	Customer PC (Ubuntu)	Computer within the customer's private network. Its main purpose is to generate traffic on the network.
Cust-B-ZRH-PC1	Customer PC (Ubuntu)	Computer within the customer's private network. Its main purpose is to generate traffic on the network.
Cust-A-BSL-PC1	Customer PC (Ubuntu)	Computer within the customer's private network. Its main purpose is to generate traffic on the network.
Cust-B-BSL-PC1	Customer PC (Ubuntu)	Computer within the customer's private network. Its main purpose is to generate traffic on the network.

Table Part B 5.4: INS virtual lab devices

It turned out that the virtual routers did not have any sensor data to power consumption. This limited our possibilities in terms research and testing of our app.

The next picture shows the exact composition of all devices and their network configuration.

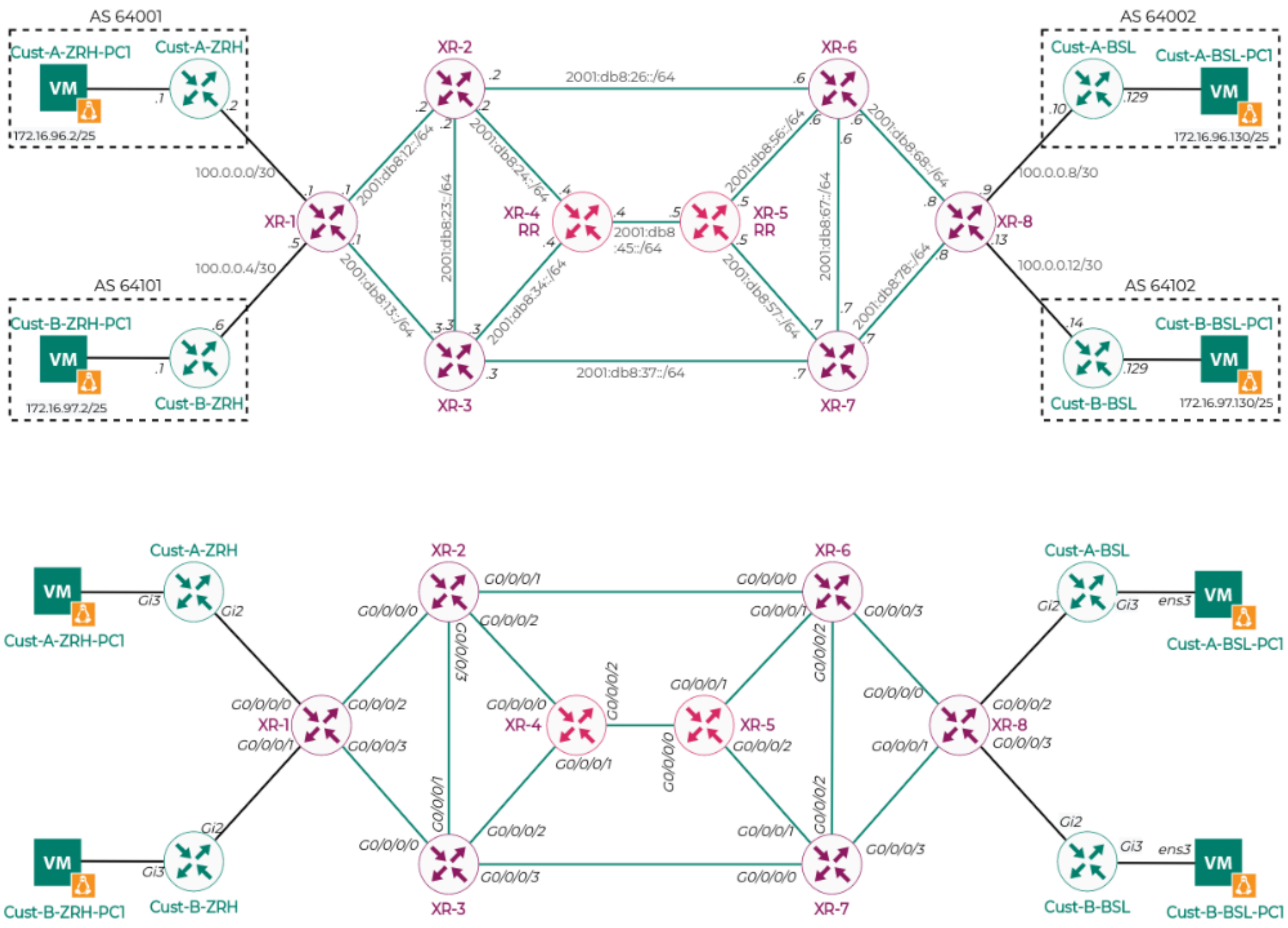


Figure Part B 5.5: INS virtual lab overview

5.7.2 Swisscom Lab

The INS had several exchanges with Swisscom to get a realistic test network with physical data, because the virtual network from INS, as pointed out in the previous chapter, cannot implement the Yang Models we need to have power consumption telemetry data. At the first contact with Swisscom in the early construction phase of the project, we were there ourselves for an introduction. The test network looked promising, but unfortunately it has not been converted until then and therefore couldn’t be used for our Jalapeño instance.

During the project construction phase, this conversion could not be completed. This is the reason why the final product of this work defines and generates all data in the lowest application layer itself.

As long as the Swisscom lab is not fully compatible with our needs, it isn’t described further in the context of this project documentation.

6 Declaration of independence

I hereby declare,

- that I have carried out the present work myself and without outside help, except for that which is explicitly mentioned in the assignment or agreed upon in writing with the supervisor.
- that I have mentioned all sources used and cited them correctly according to common scientific rules of citation.
- that I have not used any material protected by copyright in this work in an unauthorized way.

Pascal Schlumpf

Place and Date:
Rapperswil,
23.12.2021

Signature:
P. Schlumpf

Jonas Hauser

Place and Date:
Rorschach,
23.12.2021

Signature:
[illegible]

7 Rights of use

7.1 Agreement

7.1.1 Subject of the agreement

This agreement regulates the rights over the use and further development of the results of the student research project Green Routing by Jonas Hauser and Pascal Schlumpf under the supervision of Prof. Laurent Metzger.

7.1.2 Copyrights

The student is entitled to the copyrights.

7.1.3 Usage

The results of the work may be used and further developed by both students, the OST and Cisco Systems after completion of the work.

Student	**Student**	**Supervisor**
Pascal Schlumpf	**Jonas Hauser**	**Prof. Laurent Metzger**
Place and Date:	Place and Date:	Place and Date:
Rapperswil, 23.12.2021	Rorschach, 23.12.2021	Rapperswil, 23.12.2021
Signature:	Signature:	Signature:
[illegible]	[illegible]	[illegible]

List of Figures

List of Tables

Glossary

ArangoDB	ArangoDB is a free and native open-source database system with multiple models. It supports three data models (key/value, documents and graphs) with a database core and a unified query language called AQL (ArangoDB Query Language).
Container	Processes running on the host system or hypervisor, but in a strictly delimited context. Often used in the context of Docker or Kubernetes.
Database Management System	A software system that facilitates the creation and maintenance and use of an electronic database
Data transfer object	A data transfer object (DTO) is an object that carries data between processes and application layers.
Dijkstra	Mathematical algorithm that solves the shortest path problem for a defined starting to endpoint in a graph full of interlinked nodes.
Docker	Docker is free software for isolating applications using container virtualization.
Domain Driven Design	Domain Driven Design is an approach to modelling complex software. The modelling of the software is significantly influenced by the technicalities to be implemented in the application domain.
ECTS	European Credit Transfer System to accumulate study achievements.
Engineering Project	A project that took place the semester before this Term Project as practice.
Gbps	Gigabits per second
Gin	Gin Web Framework
GoBMP	Is basically an implementation of Open BMP (RFC 7854) protocol's collector in Golang.
Golang	A statically typed, compiled programming language designed at Google
Gorm	Object Relation Mapper for Go
Graph Database	A graph database is a database that uses graphs to represent and store heavily interconnected information.
gRPC	gRPC is a modern open-source high performance Remote Procedure Call (RPC) framework that can run in different environments. It can efficiently connect services for multiple purposes.

Interior Gateway Protocol	A distance vector routing protocol produced by Cisco.
InfluxDB	InfluxDB is an open-source database management system, specifically for time series concepted. It is developed and distributed by the company InfluxData.
Insomnia	Simple and open-source API Client. Insomnia is an alternative for the well-known tool Postman, which is also an API client.
Jalapeño	System developed by Cisco, which collects and processes data from attached networks, including telemetry data.
JetBrains YouTrack	Issue tracking and project management system from the manufacturer JetBrains. Alternative for other brands like Atlassian Jira.
Kafka	Kafka, developed by Apache, is an open-source distributed event streaming platform used for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.
Kaniko	Kaniko is a tool to build container images from a Dockerfile
Kubernetes	Is a professional open-source system for automating the deployment, scaling and management of container applications.
Martini API	Martini creates standards-compliant APIs with native OpenAPI 3.0 support to improve API discoverability and management. Basically, it is a set of tools to build and consume web APIs for this standard.
Merge Requests	Also known as Pull Request. Workflow in a version control system to make source code changes and review them.
Metric	A standard of measurement
Network Topology	Describes the arrangement of systems on a computer network.
Object-relational mapping	Object-relational mapping (ORM) is a programming technique for converting data between incompatible type systems using object-oriented programming languages.
OpenAPI specification	Defines a standard interface to RESTful APIs for both humans and computers
Panic	In Golang, panic is just like an exception in other known languages. It arises at runtime. In other words, panic means an unexpected condition occurred in a Go program due to which the execution of the program is terminated.
Protocol	A protocol is a standard set of rules that allow electronic devices to communicate with each other.

Rational Unified Process	Procedure model for software development projects divided in four main phases.
Redis	Is an open source, in-memory data structure store, used as a database, cache and message broker.
Scrum Board	Is one of the tools used when applying the Scrum project method. Basically, a board filled with work items.
Standard Error Stream	Via standard error a program can output error data via error stream. This is often used to display error logs in a command line interface.
Standard Output Stream	Via standard output a program can output data via data stream. This is often used to display logs in a command line interface.
Static Code Analysis	Static code analysis (SAST) is a static software testing procedure performed at translation time of software. The source code is subjected to a series of formal checks that can detect certain types of errors.
Swagger	Widely used documentation method and user interface for Web API documentations. Supports multiple versions of the OpenAPI Standard.
Telegraf	Telegraf is a plugin-driven server agent for collecting and sending metrics and events from databases, systems, and sensors.
Time Series Database	A time series database (TSDB) is a database optimized for storing and analysing time series such as sensor or telemetry data.
Traffic Engineering	Technique used to control and steer traffic to optimize the network utilization and performance.

Acronyms

BDUF	Big Design Up Front
DBMS	Database management system Glossary: Database management system
DDD	Domain driven design Glossary: Domain driven design
DRY	Don’t repeat yourself
DTO	Data transfer object Glossary: Data transfer object
Go	Golang Glossary: Golang
IGP	Interior Gateway Protocol Glossary: Interior Gateway Protocol
KISS	Keep it simple stupid
OR	Object relation
ORM	Object-relational mapping Glossary: Object-relational mapping
OST	Short form for the Eastern University of Applied Sciences
PR / MR	Pull Request / Merge Request
RUP	Rational Unified Process Glossary: Rational Unified Process
S.O.L.I.D	S - Single-responsibility principle O - Open-closed principle L - Liskov substitution principle I - Interface segregation principle D - Dependency Inversion Principle
SAST	Static Application Security Testing Glossary: Static Code Analysis
SR	Segment Routing
SR-App	Segment Routing Application
Stderr	Standard Error Stream Glossary: Standard Error Stream

Stdout	Standard Output Stream Glossary: Standard Output Stream
TE	Traffic Engineering Glossary: Traffic Engineering
TSDB	Time Series Database Glossary: Time Series Database
YAGNI	You aren’t gonna need it

References

[1] T. Keary, *Types of Routing Protocols – The Ultimate Guide.* [Online]. Available: https://www.comparitech.com/net-admin/routing-protocol-types-guide/ (accessed: Dec. 19 2021).

[2] S. Dellsperger and J. Kleiber, “Service Chaining Path Calculation,” Project Thesis, Departement of Computer Science, OST - University of Applied Sciences, Campus Rapperswil-Jona, 2020.

[3] *Segment Routing Architecture*, IETF RFC8402, Internet Engineering Task Force (IETF). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8402

[4] YangModels, *YangModels.* [Online]. Available: https://github.com/YangModels/yang (accessed: Dec. 21 2021).

[5] F. Cuiller, *IOS-XR power consumption monitoring: an ephemeral telemetry stack use case.* [Online]. Available: https://xrdocs.io/telemetry/tutorials/ios-xr-telemetry-power-consumption-docker-compose/ (accessed: Oct. 2 2021).

[6] M. Bongard, *Jalapeño API Gateway: A simple, light-weight, cloud-native API Gateway for Jalapeño.* [Online]. Available: https://jalapeno-api-gateway.github.io/jagw-docs (accessed: Dec. 21 2021).

[7] L. Metzger, *New Semester Thesis - SR-App: Green Routing.* [Online]. Available: https://www.segment-routing.ch/articles/article-20210823-01/ (accessed: Dec. 21 2021).

[8] Cisco SP Routing Team, *Optimize Power Consumption.* [Online]. Available: https://xrdocs.io/asr9k/blogs/2018-09-06-power/#:~:text=In%20less%20than,on%20power%20savings (accessed: Dec. 21 2021).

[9] A. Wiggins, *The Twelve-Factor App.* [Online]. Available: https://12factor.net/ (accessed: Sep. 29 2021).

[10] Gin Team, *Gin Web Framework.* [Online]. Available: https://gin-gonic.com/ (accessed: Oct. 31 2021).

[11] mingrammer, *Top Go Web Frameworks.* [Online]. Available: https://github.com/mingrammer/go-web-framework-stars (accessed: Dec. 23 2021).

[12] Jinzhu, *The fantastic ORM library for Golang.* [Online]. Available: https://gorm.io/ (accessed: Oct. 31 2021).

[13] R. Carrier, *dijkstra.* [Online]. Available: https://github.com/RyanCarrier/dijkstra

[14] REFSQ; International Working Conference on Requirements Engineering: Foundation for Software Quality, *Requirements engineering: foundation for software quality: 22nd International Working Conference, REFSQ 2016, Gothenburg, Sweden, March 14-17, 2016 : proceedings*. Cham, s.l.: Springer International Publishing, 2016.

[15] C. Larman, *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development,* 3rd ed. Upper Saddle River, NJ: Pearson; Prentice Hall, 2005.

[16] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959, doi: 10.1007/BF01386390.

[17] R. S. Hanmer and R. Hanmer, *Patterns for Fault Tolerant Software,* 1st ed. s.l.: Wiley, 2013.

[18] D. Vincent, *A successful Git branching model.* [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/ (accessed: Oct. 12 2021).

[19] *Conventional Commits.* [Online]. Available: https://www.conventionalcommits.org/en/v1.0.0/ (accessed: Oct. 12 2021).

[20] Raphael 'kena' Poss, *Go (Golang) conding guidelines.* [Online]. Available: https://wiki.crdb.io/wiki/spaces/CRDB/pages/181371303/Go+Golang+coding+guidelines (accessed: Nov. 1 2021).

[21] *Golangci-lint is a Go linters aggregator.* [Online]. Available: https://golangci-lint.run/ (accessed: Oct. 31 2021).

[22] *Code formatting and naming convention tools in Golang.* [Online]. Available: https://www.golangprograms.com/code-formatting-and-naming-conventions-in-golang.html (accessed: Oct. 6 2021).

[23] S. Brown, *C4 Model.* [Online]. Available: https://c4model.com/ (accessed: Sep. 29 2021).

[24] H. Cheung, *Shortest Path Algorithm Revisit with Golang.* [Online]. Available: https://www.e-tinkers.com/2019/06/shortest-path-algorithm-revisit-with-golang/ (accessed: Oct. 31 2021).

[25] solid IT gmbh, *Vergleich der Systemeigenschaften MariaDB vs. PostgreSQL.* [Online]. Available: https://db-engines.com/de/system/MariaDB%3BPostgreSQL (accessed: Oct. 31 2021).

[26] Stretchr, Inc., *Testify - Thou Shalt Write Tests.* [Online]. Available: https://github.com/stretchr/testify (accessed: Oct. 31 2021).

[27] Oracle Corporation, *MySQL Workbench 8.0 CE.* [Online]. Available: https://www.mysql.com/de/products/workbench/ (accessed: Dec. 22 2021).

C. Appendix

1 System Test

The system test was done manually with the open source API client Insomnia[10] during the end of the construction phase of the project. During development, requests were continuously created for all API endpoints and methods, which are fully suitable for system test requirements.
Prerequisite for the test is that the entire Jalapeño system and especially the Jalapeño API Gateway request service are working, and the network data is consistent. Without this prerequisite a system test is not feasible.
The test was performed on a powerful machine of one of the developers. The application ran the same as it would if it were deployed on a real server environment with a connection to the Jalapeño API gateway in the provided INS Lab. An exception for that are the non-functional requirement tests, which are performed using a big data set of mock data loaded from a network specification file. All telemetry data for the power consumptions of the nodes were faked due to no real data being accessible during the time this system test took place.

1.1 Based on functional requirements

The system tests for functional requirements are based on use cases defined in the chapter Use Cases in the project documentation (part B).

1.1.1 Jalapeño data

ID	Request	Expected result	Fulfilled
TF-1	GET /jalapeno/sync	Base tests are JD-1-A up to D. Processing and storing data: • Calculating average power consumption for every node • Setting node links based on node edges • Storing all received and processed data into the database **Http status code**: 200 **Body**: empty	Yes
TF-1-A	GET /jalapeno/nodes	Receiving all nodes from Jalapeño **Http status code**: 200 **Body**: nodes array	Yes
TF-1-B	GET /jalapeno/links	Receiving all links from Jalapeño	Yes

[10] Website: https://insomnia.rest/

		Http status code: 200 **Body**: links array	
TF-1-C	GET /jalapeno/node-edges	Receiving all node edges from Jalapeño **Http status code**: 200 **Body**: node edges array	Yes
TF-1-D	GET /jalapeno/telemetry/*:name* Parameter *name* with a specific node name identifier	Receiving power consumption telemetry data from Jalapeño about the node name given	Yes

1.1.2 Green routes

ID	Request	Expected result	Fulfilled
TF-2	GET /green-routes	All previously calculated green routes for all nodes available **Http status code**: 200 **Body**: green routes array	Yes
TF-3	GET /green-routes?*:nodeKey* Query *nodeKey* with a specific node key identifier	All previously calculated green routes for the specific node (ingress or egress) **Http status code**: 200 **Body**: green routes array	Yes
TF-4	GET /green-routes/*:id* Parameter *id* with a specific previously calculated green route identifier	Get calculated green route by a specific identifier **Http status code**: 200 **Body**: green route	Yes
TF-5	GET /green-routes/calculate/ *:ingressNodeKey*/*:egressNodeKey* Parameters with specific node key identifiers	Receiving and processing Jalapeño network data as described in T-1 and calculating the greenest route based on the new data. The green route must be defined based on an average power consumption and not a single power consumption snapshot. **Http status code**: 200 **Body**: green route	Yes

1.1.3 Statistic purposes

ID	Request	Expected result	Fulfilled
TF-6	GET /nodes	All previously received nodes from Jalapeño with a power consumption history **Http status code**: 200 **Body**: nodes array	Yes
TF-7	GET /nodes/*:nodeKey* Parameter *nodeKey* for a specific node key identifier	Get received node from Jalapeño with a power consumption history **Http status code**: 200 **Body**: node	Yes

1.2 Based on non-functional requirements

The system tests for non-functional requirements are based on the definition in the chapter Non-Functional Requirements in the project documentation (part B).

1.2.1 Functionality

There are no tests needed based on the definition for the topic's security and interoperability (defined in chapter Functionality).

ID	Based on	Conclusion
TNF-1	Accuracy	It is not possible to test the accuracy since the power consumption telemetry data had to be faked for this system test.

1.2.2 Usability

ID	Based on	Expected behaviour and possibilities	Fulfilled
TNF-2	Understandability	The green route and node objects provide data about the power consumption of link between nodes involved and a cumulated value for the whole route path. It is possible to see a power consumption history of the nodes.	Yes
TNF-3	Operability	No possibility to test this now.	-

1.2.3 Reliability

ID	Based on	Conclusion
TNF-4	Availability	No possibility to test this now.
TNF-5	Recoverability	No possibility to test this now.
TNF-6	Fault Tolerance	Not implemented in the scope of the term project.

1.2.4 Performance

ID	Based on	Expected behaviour and possibilities	Fulfilled
TNF-7	Capacity	The application can handle a green route calculation for a network with 1000 nodes.	Yes
TNF-8	Time behaviour	The application should perform a complete Jalapeño data sync and a calculation for a green path in less than 10 seconds. The reached value is about 2.5 seconds with 1000 nodes on powerful desktop systems inside the docker build environment. Prerequisite is, that the Jalapeño API Gateway is running in normal workload and normal response times.	Yes

1.2.5 Scalability

This could not be tested since the project team (including supervisors) decided to omit a deployment in the scope of this term project.

1.2.6 Maintainability

ID	Based on	Expected behaviour and possibilities	Fulfilled
TNF-9	Analysability	The application must have multiple log levels which could be set over the environment variables. The log events must vary as defined and available.	Yes

2 Meeting minutes

23.09.2021, Project Kick-Off

Time 10:00 to 11:00

Participants

- Prof. Laurent Metzger
- Severin Dellsperger
- Julian Klaiber
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Project Kick-Off prepared by the supervisors

Notes on each agenda item discussed

1. The Thesis was presented and various key points on existing system informations were given. The main expectations and requirements for the project are the following:
   - To be able to calculate greenest path (at least theoretically), lowest energy consumption
   - How to bring the data from the network into Jalapeno with own or extended processor
   - Find and compare existing products
   - How to do it so that the paths are stable (no flapping)
   - How to collect and store the telemetry data
   - Write simple application to display the data
   - Existing Jalapeno API Gateway under construction by Michel Bongrad
   - Application should be cloud native with Kubernetes environment
   - Optional nice visualization (frontend)
   - Processor and Jalapeno are written in GoLang
   - Backend in Python and frontend in React
   - IOS XR and Cisco ASR 9000 routers used
   - Cisco liason is Francois Clad, a research engineer and graph theory expert
   - Requirements definition as Use Cases in the next weeks

30.09.2021, Weekly exchange

Time 10:00 to 11:00

Participants

- Prof. Laurent Metzger
- Severin Dellsperger
- Julian Klaiber
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Discuss task formulation
2. Access to server, network and Jalapeno
3. Update on current work progress and planned organization of the project
4. Working time of each participant and stakeholder of this term project
5. Open questions
    a. How to define the rout lifetime (topic flapping)
    b. Must the app also configure the paths or only calculate them
    c. How to generate traffic in a network to analyse the telemetry data

Notes on each agenda item discussed

1. Severin agrees with the task formulation and finds it good as it is. Laurent has not yet had time to look at it.
2. They will order a VPN access for us to the INS network. They also will setup a virtual lab with routers and virtual machines and a working Jalapeno instance deployed on our server.
3. We showed the current decisions and progress and told everyone that we will organise this project like the Engineering Project we've had last semester. Everything looks fine till now.
4. Pascal and Jonas are normally working at their school days (Tuesday and Wednesday) and on Sunday for this project. Julian and Severin are working only on Thursday and Friday for the INS, but they try to support us also on other weekdays. Michel is available every weekday.
5. Answers
    a. A compromise must be found here. Possibly changed every 10 minutes, but individually adaptable and set by the customer.
    b. Only calculation of the paths is necessary for this term project.
    c. We can use the tool IPerf for that.

07.10.2021, Weekly exchange

Time	10:00 to 11:00
Participants	• Severin Dellsperger • Jonas Hauser • Pascal Schlumpf

Agenda

1. Discussion of the use cases suggestion
2. Introduction to our server with Kubernetes
3. Introduction to the virtual lab
4. No sensors and physical data on the virtual routers
5. Open questions
    a. Should we create our own SonarQube instance?
    b. How to access the Grafana instance?
    c. Status about the Jalapeno API Gateways and the Jalapeno instance?
    d. What are the Non-Functional requirements?

Notes on each agenda item discussed

1. The use cases are looking good as far.
    a. The naming's are not optimal and UC04 should be reconsidered.
    b. A use case should also start with a verb all the time.
    c. We also must describe how use cases work.
    d. We will provide a new use case suggestion until the next day to Severin. He will discuss them with the other supervisors.
2. Access to every service on the server with the server address and the port. Severin sent us multiple information's about the server and Kubernetes and how to use them.
3. Severin sent us various information during the introduction that is beyond the scope of these minutes.
4. We need to configure the telemetry models on the routers properly to implement real mocked data.
5. Answers
    a. The INS will setup an instance of SonarQube for us on our server.
    b. Server address and port 30300.
    c. Both are deployed on our server since this morning.
    d. Our application should work properly up to 1000 nodes (routers). We will discuss this again separately in the next weekly exchange together with the other supervisors to finalize those requirements.

11.10.2021, Use cases exchange

Time	10:00 to 10:30
Participants	• Prof. Laurent Metzger • Severin Dellsperger • Jonas Hauser • Pascal Schlumpf

Agenda

1. Discussion about the adjusted use cases

Notes on each agenda item discussed

1. In general, the use cases are too technical (everybody should understand them). The main success scenarios are mostly to fixed on one solution. A new use case for statistics (variance of routes) must be created additionally. Some further points must be changed according to the following notes:
    a. UC2: Actor is missing, do not write "I can". Main success scenario not scaled not enough. Also add an alternative scenario if possible.
    b. UC3: Title is confusing. Get ecological metric of nodes not very understandable. Generalize the main success scenario. The preconditions are not optimal.
    c. UC4: The preconditions are also not optimal here.

d. UC5: Overview; recalculation constantly, set paths frequently. Statement for Stakeholders and interests are to vague described (only overhead)
e. Everything else is fine and mostly the content is good defined.

14.10.2021, Weekly exchange

Time	09:00 to 10:00
Participants	• Prof. Laurent Metzger (only first 15 minutes) • Severin Dellsperger • Julian Klaiber • Michel Bongrad • Jonas Hauser • Pascal Schlumpf

Agenda

1. Info: Please cancel meetings early if participation is not possible to better plan our work
2. Discussion of the revised use cases
3. Discuss NF-Requirements again
4. Presentation of project plan and general documentation
5. Problem from Jalapeno API GW:
    a. Topology data works, but telemetry data does not
    b. Further procedure/solution?
6. Exchange about programming languages and frameworks
7. Questions

Notes on each agenda item discussed

1. Acknowledged
2. Use Cases everything io (revise red marked parts in the documentation, afterwards finished)
3. NF requirements are good (revise red marked parts in the documentation, afterwards finished)
4. Specify damage potential in a damage time unit. “How many hours it could cost in the worst case?” Do it in the same course as revalidation of risks. Question “Can we start the construction phase?” should be clear after that. All the risks should be near the green marked fields in the risk graph
    a. Otherwise, everything is fine. They are very satisfied with the documentation till now.
5. We stay with SRv6 (IPv6) because it is the future. Adaptations on the Jalapeno API Gateway will be implemented iteratively by Michel starting next week. We need to look at the telemetry models as soon as possible and afterwards talk to Michel.
6. Processor must be written in Golang.
    a. SR-App could also be Go, but it doesn't have to be.
    b. Go does not know generics so far
7. No explicit questions this week

21.10.2021, Weekly exchange

Time	10:00 to 11:00
Participants	• Prof. Laurent Metzger • Severin Dellsperger • Julian Klaiber • Michel Bongrad • Jonas Hauser • Pascal Schlumpf

Agenda

1. Decision for data mocking of Router telemetry data due to not existing physical devices
   a. Generator script/app simulating multiple routers with simple telemetry metrics
2. No data collector and data processor needed in planned setup
3. Ecological aspect: path with summed power consumption is basically already the greenest one, or not?
   a. Potential metrics for defining the green index
      i. Pure power consumption summation (per example 400 Watt)
      ii. Router efficiency index (example with 3 value scale: low, medium and high)
4. No data found for load to power efficiency ratio while research
   a. Real world research on this is not possible currently because no physical devices are available
   b. Hypothesize based on marginal information to start with?
5. Server availability and performance is not very good now
   a. It was very slow on 20.10.21 and completely not usable after 21:30
6. Questions
   a. When will the exchange with Francoise take place?
7. Show shortly the current work progress
8. General feedback round from supervisors to students and vice versa

Notes on each agenda item discussed

1. Mocking to be followed up
   a. Swisscom will give us access to their lab where they have several physical IOS XR routers, date not yet known
   b. Supervisors are looking to get an example from Swisscom or Cisco of a response using the Yang model
2. We definitely won't build a separate data collector and processor for this term project
   a. We need to look at telegraf to identify where the messages timestamp is coming from
3. Ecological aspects
   a. Power origin would be interesting but not feasible
   b. Only wattage is possible and sufficient for the term project
   c. Link has taken value from next hop
   d. Use only one type of router (ASR 9000)
   e. Look at one period and look at average

4. Nothing to say here due to decisions before
5. They will try to fix the problem on the server as soon as possible
6. Laurent is still in contact with Francoise and will give us an update as fast as possible
7. They are happy with the work progress, nothing more to say here
8. Most important feedbacks
    a. Certain things were taken for granted in the beginning, which has not been for us since we are software engineers and had no contact to network related topics in the last couple of years
    b. We had some difficulties in the beginning, but these were tackled professionally and so everything is now running well
    c. They like our professional approach and organization and think that they too can learn something from us during this time
    d. In conclusion, everyone is very happy with how it works currently

26.10.2021, Exchange with Swisscom

Participants
- Prof. Laurent Metzger
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Project presentation for Francoise
2. Update Exchange Swisscom

Notes on each agenda item discussed

1. Presentation for Francois
    a. Is a researcher
    b. Wants to see research aspects
        i. Power Consumption
        ii. Efficiency Index
    c. Just briefly about how we organize
    d. How to make routing greener?
    e. What do we use Jalapeno for?
2. Swisscom
    a. They do an onboarding (pre-production, 10 routers)
        i. We get a VPN and a Linux machine
        ii. We also have the possibility to simulate data
    b. Switching to SR throughout Swisscom, strategy busy
        i. 2 out of 20 networks are already on SR-MPLS
        ii. All are already configured, but not yet activated
3. Configuration aspect only plays a role when we are going to configure routes ourselves.

28.10.2021, Weekly Exchange

Participants
- Francois Clad (only for the presentation)
- Prof. Laurent Metzger
- Severin Dellsperger

- Julian Klaiber
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Presentation term project to Cisco partner
    a. Introduction with the Jalapeño API Gateway diagram
    b. Telemetry data, existing yang model on Cisco Router for Power consumption
        i. Cisco-IOS-XR-sysadmin-envmon-ui:environment/oper
    c. No new data collector or processor needed
        i. Direct subscription through Jalapeno API GW to Time Series DB
    d. Ecological aspects of routing: Path with summed power consumption is basically already the greenest, this is enough for the term project context
        i. Outlook: there is another potential metric beside the pur power consumption (e.g. 400 Wats)
            1. "Efficiency-Index" (Relation throughput to power consumption)
            2. Downside different routers
            3. Example: Generation 4 of ASR 9000 routers compared to Generation 1 are up to 95% more efficient
        ii. Outside context for term project:
            1. Router environment, power source (green or not), cooling, geographical location
    e. Until now no information found on the load to power efficiency ratio for those routers
        i. Research on this was not possible until now because we do not have physical devices available
        ii. We will become access to Swisscom Lab with real routers in the next couple of weeks
    f. Short overlook to use cases diagram
    g. Show project milestones shortly
    h. Current status: In the end of elaboration phase
        i. Technology decisions
            1. Why did we select Go as our language?
                1. It is very performant
                2. There are already a number of Djikstra implementations
            2. Our thoughts to caching
            3. Persistent database for long term statistics
        ii. Architecture and System Overview under construction
    i. Questions / Discussion
        . https://www.segment-routing.ch/articles/article-20210823-01/
    j. Closing presentation, switching to discussion with supervisors (project internal)
2. Architecture
    a. Do we want to implement a cache Yes or No?
    b. Based on 12 Factor Methodology we would need to
    c. https://12factor.net/de/processes#:~:text=Der%20RAM%20oder,und%20Dateisystem)%20l%C3%B6schen.
3. Can we use the GitLab runner from INS
4. Current work progress / Outlook

Notes on each agenda item discussed

04.11.2021, Weekly Exchange

Participants

- Prof. Laurent Metzger
- Severin Dellsperger
- Julian Klaiber
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Elaboration phase completed
    a. Was close at the end, but well reached
    b. From now on reduce some time because elaboration needed a little more than originally planned
    c. A lot of research done
    d. Especially a lot in mocking
        i. Estimated remaining effort approx. 6h
2. Elaboration documentation
    a. Development concept
        i. How we use Version Management
        ii. Our use of Definition of Done
        iii. Our inputs to Unit tests
            1. Seems to be very hard with GO
            2. 80% will be hard
        iv. SonarQube
        v. Describe steps of CI/CD shortly
            1. The migration to Kaniko is planned
    b. Domain Analysis (Pascal)
    c. Architecture and Design specifications
        i. System Overview (Pascal)
    d. Work in progress (Prototype) of the 12 Factor Methodology
    e. Technologies
        i. Go as the programming language
        ii. Only MariaDB and no cache as persistence store
        iii. The use of a linter
    f. Backend Architecture
        i. We decided to follow Domain Driven Design principles
        ii. Some Go specific informations
        iii. We have no Architecture for the frontend planned currently
1. Current Works / Planned for Early Construction phase
    a. The introduction to the Swisscom Lab
    b. Maybe we are going to complete the telemetry data mocking
    c. Base Project improvements
    d. ORM
    e. Controllers
    f. Entities
    g. CI/CD Pipeline to Kaniko
    h. Finish 12 Factor Methodology

2. Outlook
    a. Project is fully on track
    b. Time saved due to processor, but time needed for mocking
    c. Start with implementation of the calculation in middle construction phase
3. Maybe insight into the code?
4. Michel Update regarding API GW adjustments
    a. Need the changes on 15.10 at the earliest

Notes on each agenda item discussed

1. GreenRoute and GreenRouteCalculation
2. The team expressed interest in the domain driven design approach
3. Using scratch container in a Multistage build
4. View environment file solution
5. All good, except that multistage Docker files should be used (example sent)

18.11.2021, Weekly Exchange

Participants

- Prof. Laurent Metzger
- Severin Dellsperger
- Julian Klaiber
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Status Green SR-App
    a. DDD Implementation
    b. Logging and environment variables v2
    c. ORM / Database
    d. Jalapeno API GW Service v1
    e. Network sample (Database data)
    f. Entity to DTO conversion
    g. API definition v1 and demonstration
    h. Current:
        i. Implementation algorithm v1
        i. Database Mocking and more Unit Tests
        ii. API definition v2
        iii. Migrations and Seeds
        iv. Swagger
2. Status Mocking (Pascal)
3. In general
    a. Everything is going according to plan so far
4. Outlook
    a. Swisscom Lab Jalapeno Integration
    b. Algorithm v1
    c. Prototype Green SR-App
    d. Fine-tuning and documentation

Notes on each agenda item discussed

1. DTO is good, maybe a little overkill, but separation makes sense
2. Mixture of integration test and module test
3. Michel adds a security mechanism to avoid overloading the gateway
4. For the algorithm should be a first version presentable next week
5. There is an interest in the Mocker, but current focus on Swisscom
6. The possibility to configure grpc on the router will be discussed on Friday

25.11.2021, Weekly Exchange

Participants

- Severin Dellsperger
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. UUID as identifiers
   a. ... bad performance for relational databases (native use as string)
   b. ... id and uuid in database with uuid_hash for index is complicated and an overhead
   c. ... the benefit in our use case for this project is not big enough
   d. ... the solution with UUID_TO_BIN is a to big overhead compared to the benefit
2. The first version of the Jalapeño API Gateway data synchronizer and processor is still in progress
3. Some insights into the first version of the shortest path algorithm
4. The OST GitLab now much faster after the software update
5. Outlook
   a. Current Sprint
      i. Finish Jagw data processing and storing
      ii. Repository Mocking for Testing
      iii. Integration & Unit Tests
      iv. Finish API definition
   b. End Construction
      i. Database Migrations & Seeds improvement
      ii. Swagger
      iii. Refactoring / Improvements / Cleanup
      iv. Maybe deployment
   c. Transition
      i. Focus on documentation
      ii. No more big things in the code or otherwise in the products

Notes on each agenda item discussed

1. They also don't see a use case for UUID
2. Why is there a LogicalLink entitiy and why do we not work with the provided LsLink structs.
   a. The LogicalLink entitiy has more information and is needed for mapping with the nodes
0. The LsEdge table on the ArangoDB takes a long time to update.
   a. This can lead to inconsistencies
   b. It is also possible to match the links and the nodes via the igpRouterId

1. To view the mocker telegraf logs use the following command “kubectl logs influx-0 -n jalapeno –follow”
2. We need to explain why we have implemented the algorithm this way
    a. Why did we decide to not implement Dijkstra ourselves?
    b. Looks good otherwise
3. Would the concurrent querying of telemetry data also be possible?
4. A deployment is not necessary for the project thesis, as the data basis is also not available
    a. Maybe this will be possible in the bachelor thesis

09.12.2021, Weekly Echange

Participants
- Prof. Laurent Metzger (arrived 25mins later)
- Severin Dellsperger
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Green SR app completion underway
    a. The API is now final
    b. Our implementation of Swagger
    c. Some information to the nodes history
    d. How we parallelized our prototype
    e. Some insights into our testing
        i. We show how we mocked the whole repository layer
        ii. With the mocked repository layer it was possible to write integration tests
    f. Our use of migrations
    g. How we implemented seeds
    h. We did some restructuring of our code base
2. Update Mocker @Pascal
3. The OST GitLab is in a very poor condition
    a. It is often the case in the evening that timeouts occur when the platform is used, because the requests take so long
4. Some clarification questions regarding the final submission of the project thesis
    a. Many informations can be found in the documentation
    b. Do we need to create a poster for the project thesis? What is the expected form of this poster?
5. Outlook
    a. We are working on some optimizations to meet our nonfunctional requirements (1000 nodes in a network)
    b. We are also working on the final code polish
    c. Finish of the backend
    d. The transition phase is only for documentation and the final submission

Notes on each agenda item discussed

1. SR-APP
    a. No comments were given
    b. Is good. Returns only GET and no POST requests
        i. Why is green-routes/calculate not a POST, this would be cleaner
    c. For Insomnia the config can be loaded to create all endpoints

        i. For BA SR should also work -> SR workshop should be possible
    d. We are still on the analysis of our performance problems
    e. They would still have scripts for the Jalapeno Mocking
        i. We have already solved this ourselves
    f. No remarks were given
    g. Code coverage of 80% may not be achieved
        i. But this is not severe
    h. It is important that we document well what we have achieved and what belongs to the product
2. Will no longer be treated in the project thesis
    a. Status now is not yet complete but would not need much more time
3. They have no control over GitLab and the situation does not improve
    a. Maybe a CRON job runs at 10 o'clock, maybe they can postpone it to a later time
4. It is desired that it be done according to requirements
    a. No remarks to this topic
    b. Poster should be made as an exercise for the bachelor thesis. It can then also be taken over for the bachelor thesis
        i. It's enough if it's digital
    c. How do they plan to correct the thesis?
        i. Described in the guide in Chapter 6
    d. Project thesis will be published in a magazine
5. There will be an appointment about the status of the project thesis
6. The team stays tuned with Swisscon and telemetry data

16.12.2021, Weekly Exchange

Participants

- Prof. Laurent Metzger
- Severin Dellsperger
- Michel Bongrad
- Jonas Hauser
- Pascal Schlumpf

Agenda

1. Software finished with Version 0.2.0
    a. Some remarks to the functional requirements testing
        i. All mandatory use cases achieved
        ii. From the optional use cases only the "gather statistics" use case was achieved
            1. It provides a green routes history
            2. Also the nodes history & nodes power consumption history will be saved per request
        iii. Optional target "Login" and "Frontend" made no sense for the project thesis.
    b. Information regarding our nonfunctional requirements testing with a big amount of data
        i. We have faked all data on lowest layer
        ii. We tested our software with 986 nodes and 24'852 links and 10 telemetry data entries per node
        iii. We created a local Docker environment where we copied the pipeline steps (go alpine image as base)

    - iv. The network synchronization, processing and storing into the database takes less than 2.0 seconds.
    - v. If we only look at the green route calculation, we measured a time of less than 0.5 seconds
  - c. nonfunctional requirements testing with topology data of the virtual network where we only had to fake the telemetry data
    - i. The nodes and the logical links were loaded from the Jalapeño API Gateway. The telemetry data needed to be faked
    - ii. We tested our software with 8 nodes and 26 links and 10 telemetry data entries per node. The whole network synchronization, processing and storing into the database took less than 0.5 seconds which is strongly dependent on the connection to the Jalapeño API Gateway. The minimum value measured was 0.07 seconds
    - iii. If we only look at the green route calculation, we measured a time of less than 0.05 seconds
  - d. Nothing implemented in terms of segments, but basis is optimally prepared for it

1. Miscellaneous for finish
   - a. Software License? Closed Source
   - b. Submission of reports digitally or also printed?
   - c. Poster definitely only digital
2. Feedback round on the student thesis / exchange project (optional)
3. Outlook

Notes on each agenda item discussed

1. Finish
   1. Deployment will not be done in the SA
      - a. Post parameters in the body
        - i. No need to adapt for SA
        - ii. Explain why we implemented it with url parameters
      - b. Mocker should also be handed in
      - c. SID List are only implemented in the BA. But the foundations have been laid
      - d. Stable paths can be defined but will be revised in the BA
2. Various
   - a. No license required. Use rights of use the latest, as already done
   - b. Submission in the team as Zip in a folder Submissions
   - c. Poster only digital
   - d. For presentation is expected about 20m, stand, challenges as we solved it, review positive and negative points

3 Task formulation

3.1 Supervisors

The Green Routing student research project is being conducted in partnership with Cisco Systems.

3.1.1 Supervisor

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

3.1.2 Co-Supervisors

Severin Dellsperger, Institute for Networked Solutions (INS), severin.dellsperger@ost.ch
Julian Klaiber, Institute for Networked Solutions (INS), julian.klaiber@ost.ch

3.1.3 Partner from Cisco Systems

Cisco Systems represented by Francois Clad

3.2 Initial situation

Ecological aspects play more and more a role in all areas of science nowadays. It is tried in each area not only to get the best for the consumer, but also to include aspects to further improve the ecological aspects.
The Internet today consumes a lot of electricity and is therefore partly responsible for our huge Co2 emissions around the world. Routers often run all around the clock, consuming a significant amount of power due to potential inefficiencies at higher workload.
In today's approach, the fastest available path from A to B is usually chosen in a network. However, with today's availability of high bandwidth, other metrics can now be used to select routing in a network. This is where the green routing approach comes into play. It should be made possible to find the most ecological route by including different metrics and information from network components, especially routers.

3.3 Expectations and goals

The main goal is to make research on the idea for green routing and use or develop a data processor and algorithm(s) for the most ecological paths for routing inside a network.

Expected minimum work results:

- A result for how to become the green index.
- It must be analysed through research if there is an existing well-suited processor for processing the data from routers in our use case. There are the possibilities to use an existing one or develop an own.
- The implementation of a data processor based on the decision from the work result listed before.
- One or more algorithms to calculate the most ecological routing path in a network (theoretical or implemented as a Segment Routing application Backend).

Optional work results:

- Development of a simple Frontend Application to visualize the calculated paths.

Additional goals for success:

- All calculated paths must be stable. This means no flapping between paths should occur.
- The newly developed Cisco Jalapeño API by Michel Bongard must be used.
- The data processor must be written in GoLang and the Backend in Python or another used language in the technology stack. This is the current used Stack for the Green Routing API and frameworks inside the Institute of Network Solutions and Cisco Systems.
- The optional Frontend must be written in React. This must be defined in the project planning phase in the first couple of weeks of this project.
- Infrastructure should base on Kubernetes and all apps must be developed for cloud native environments.

As usual for software projects, the newly developed code should follow best practices known in software engineering. It must be extendable and well designed.

3.4 Definition of implementation

Meetings between students and supervisors are held every week. Further meetings are held at the request of the students. The students will take minutes from each meeting and send them to the supervisors in the same week where the meeting was held. Decisions should be clear and comprehensible in the minutes.
The work progress should be transparent and continuous. The time invested must be roughly tracked.
The task formulation is created by the students and confirmed by the supervisors. Requirements Engineering is done by the persons implementing the student research project.
A project plan must be created in the first weeks of the project with milestones and a definition for a Minimal Viable Product and Optional Features.

3.5 Documentation

For this student research project is a documentation necessary based on the regulations of the department of computer science at the OST university of applied sciences. All documents should be fully finished on the time of submission. Since this project is of international interest, it should be written in English. All other not relevant papers for the public could be written in English, but it is not mandatory to do so.

3.6 Important dates

KW 39, Start of student research project
KW 51, Submission of student research project

3.7 Additional notes

This assignment was created based on the oral assignments by the supervisors and confirmed by them after the written definition by the involved students.

4 Project related configurations

4.1 Golangci-lint linters

- goconst
- gocritic
- gofmt
- goimports
- gomnd
- Gocyclo
- goprintffuncname
- gosec
- gosimple
- govet
- bodyclose
- deadcode
- depguard
- dogsled
- dupl
- errcheck
- errorlint
- exportloopref
- exhaustive
- ineffassign
- misspell
- nakedret
- prealloc
- predeclared
- revive
- staticcheck
- structcheck
- stylecheck
- thelper
- tparallel
- typecheck
- unconvert
- unparam
- unused
- varcheck