Central Frontend for Segment Routing Applications

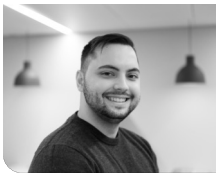
Department of Computer Science
OST – University of Applied Sciences
Campus Rapperswil-Jona

Autumn Term 2021

Author(s):	Leonard Obernhuber, Davor Gajic
Advisor:	Prof. Laurent Metzger
Co-Advisor:	Michel Bongard
Project Partner:	Cisco represented by Bruce McDougall

Central frontend for Segment Routing applications

Students



Davor Gajic

Leonard Obernhuber

Initial Situation: Many segment routing (SR) applications are developed and maintained at the INS, and each of these applications must create its own UI. The implementation of these user interfaces is never the main focus during the development, resulting in additional effort for each SR app and reimplementation of existing components without maintaining a unified look and feel for each frontend.

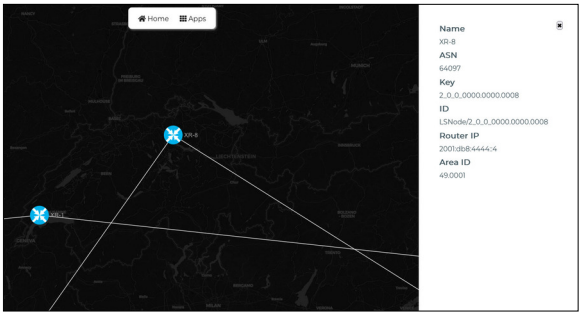
Objective: The goal is to create a central user interface that connects all existing and future SR apps. The Central Frontend should provide modularity, extensibility and reusability of the components.

In this thesis, the Central Frontend should have a landing page that draws up to 1000 SR nodes onto a map. GPU-accelerated processing is required to meet the performance requirements. The graph should cluster the nodes together into groups to maintain visibility. The nodes and edges should be interactive and show the corresponding information coming from the Jalapeno API Gateway. In addition, the Central Frontend should allow other SR applications to be launched. A mock SR App should be integrated in this thesis.

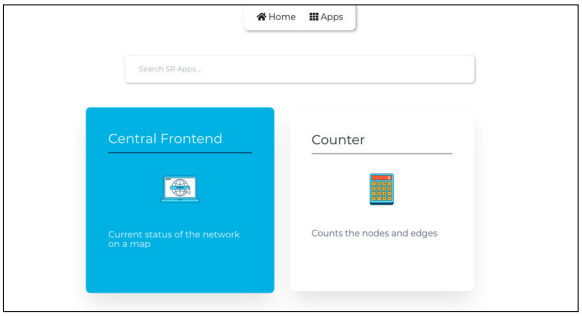
Result: It was decided to implement a ReactJS application that communicates with the Jalapeno API Gateway via gRPC-web. The sigma.js graph visualization library, which uses webGL, is used to display and render the network on a leaflet map in a performant way. TypeScript is used as the programming language.

The Central Frontend covers all mandatory features and use cases and renders up to 1000 nodes in less than two seconds. Additionally, a custom clustering algorithm groups the nodes together regionally to increase visibility. An SR app list was implemented to search and launch different applications. A node and edge counter was implemented as a mock SR app.

Node information
Own presentment



SR app list
Own presentment



Landing page with clustering
Own presentment

Examiner
Prof. Laurent Metzger

Subject Area
Internet Technologies and Applications

Project Partner
Cisco Systems Belgium De, Machelen, Belgien

1 Management Summary

Segment routing apps are developed by many different engineers. There are no requirements or guidelines on what the frontend needs to look like. Also the apps are scattered around the internal network and are only reachable over the URI of the web server which they are deployed on. This is a problem which INS employees have to deal with on a daily basis. In addition, developing a standalone user interface for each of the different segment routing applications adds a significant amount of effort in each project. Until now there was no centralised solution available in any form and it is a completely new idea with no defined approach.

So to simplify the process of adding new SR apps to the repertoire and aggregate consisting apps we developed the central frontend. It should provide a landing page which shows certain network topologies of different autonomous systems and offers the overview of all SR apps available in this network.

To achieve this goal we decided on a configuration based approach where the developer can add his application with some configurational effort. This is not final and can be changed in the bachelor thesis to be even easier to use. Also important is the fact the map on the landing page is dynamically generated and can be reused by other apps which gives the SR developer a lot of freedom.

The product allows for an easier use of the segment routing ecosystem by combining all the apps into one single frontend and give an overview over the current network topology with a clustered graphical representation on the landing page.

Contents

2 Technical Documentation

2.1 Introduction and Overview

2.1.1 Initial situation and problem definition

Currently, many different segment routing applications are developed and maintained at INS. These segment routing applications are based on segment routing technology.

Segment routing is a source-based routing protocol released in 2015 that simplifies traffic engineering and management across network domains.

Network data is collected and processed by the Cisco-developed application called Jalapeno and stored in either InfluxDB, a time series database, or ArangoDB, a graph database.

To interact with the network or Jalapeno, the INS has developed the Jalapeno API Gateway. It allows users to make simple API requests using gRPC to retrieve topology and telemetry data from Jalapeno. Every INS SR application uses the Jalapeno API Gateway as an interface to the SR network and to Jalapeno.

There are currently 3 SR applications developed/maintained by the INS:

- Analytics: Used for link quality assessment and link saturation prediction
- Service Programming: Used for service programming
- Green Routing: Used for traffic engineering and finding the most power efficient path

2.1.2 Problem description

Each SR application developed at INS needs to create and maintain its own user interface. This leads to a number of problems:

- The user interface of each application has never been the focus, resulting in a suboptimal user interface and user experience
- Each application currently builds the user interface from scratch, adding overhead to each project
- Applications do not have a consistent look and feel.
- Each application needs roughly the same UI components (e.g. drawing a network)

2.1.3 Goal

The goal is to create a central user interface that connects all existing and future SR applications.

The UI should be as modular as possible, easily extensible and provide high abstraction and reusability. For example, the application should provide the functionality to easily draw a network so that future applications can use that feature.

The UI should have a landing page that shows the network on a world map. The map is zoomable with multiple layers and should cluster current links and routers into geographical groups.

When you click on a link or node, you should get information about the object itself. Two nodes may have multiple links, so the application should be able to draw multiple links between two

nodes.

The landing page should be able to display at least 1000 nodes at a time. Ideally 100000 nodes. To help with the performance, the application should use webGL, to hardware-accelerate the rendering.

The user interface should allow the user to search and load other SR apps. For this project, a mock SR app should be integrated.

In addition, the Central Frontend should be developed cloud-natively so that the application is resilient, highly available, and easily scalable.

Optional

Filtering The application should allow the user to filter for specific properties e.g. ASN. The desired selection is either highlighted or the leftovers hidden.

Identity and Access Management Application administrators can create/update/delete users and give them permissions to different functionality and SR applications.

Additionally, you can connect the application to different identity providers through OIDC.

Responsive CSS The application can be used on mobile phones and tablets.

2.1.4 Scope and limitations

Currently, there is no way to get the coordinates of all the SR routers. The API needs to be extended for this. However, this is not in the scope of this project and we will therefore mock the coordinates and work as if the functionality is present in the API.

In this project, only a mock SR App will be ported over to use the Central Frontend. Optionally, the Service Programming SR-App could be ported over, however only if there is enough time. The remaining SR-Apps will not be ported over in this project.

2.2 Requirements

2.2.1 Actors

Since the user management is an optional use case, there is only one actor.

User The central frontend should be able to show the landing page to a user and give the possibility to switch to different SR apps.

2.2.2 Use Case Diagram

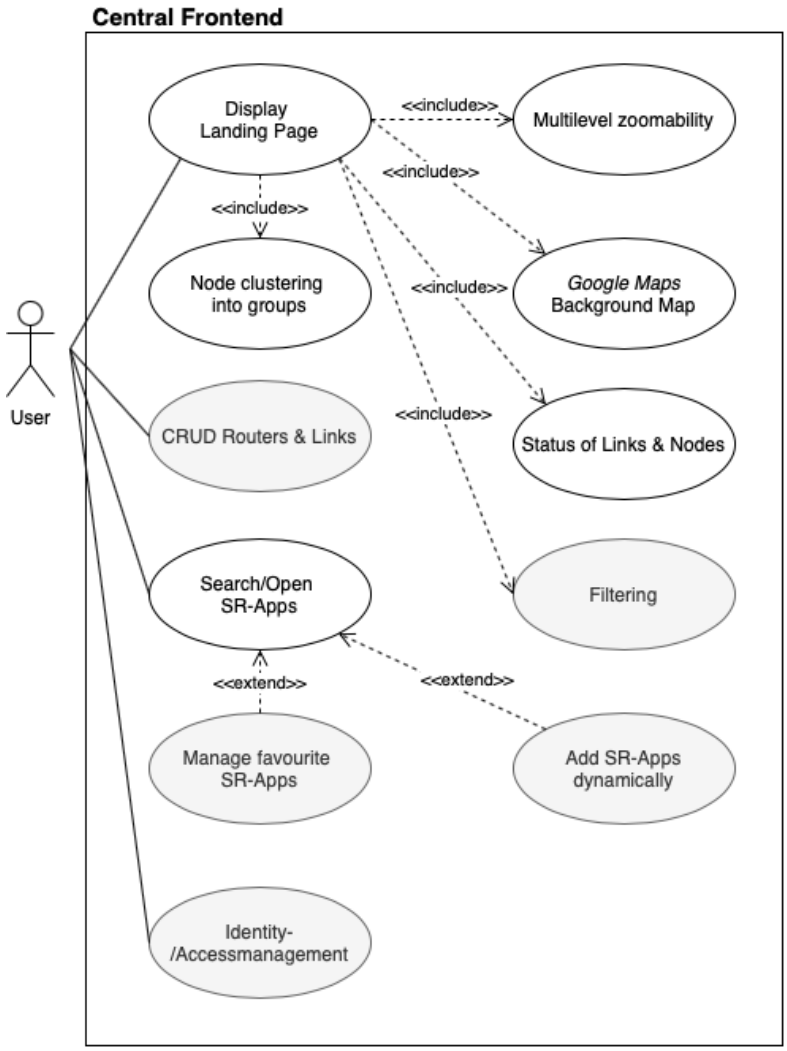


Figure 1: Use Case Diagram

2.2.3 Use Cases Brief

#	Use Case	Brief	optional /mandatory
UC01	Show Landing Page	User visits central frontend and is greeted by a landing page wich shows the current status of nodes and links in the global network topology on a map.	mandatory
UC01-1	Landing Page - Map	The background of the landing page is a map. The map is based on Google Maps or Open Street Map.	mandatory
UC01-2	Landing Page - Map Zoom	The zoom levels of the map should be adapted to the number of subnodes that need to be shown on the screen. They need to be dynamic in a way that at least 5 zoom levels can be shown.	mandatory
UC01-3	Landing Page - Status of Links and Nodes	The landing page shows the current status of the network. The links and nodes are placed onto their respective coordinates. The network is correctly drawn and can have multiple links between two nodes. The node symbol is based on the node type (e.g. router is drawn as a router, etc.).	mandatory
UC01-4	Landing Page - Clustering	The nodes which are roughly at the same location should be clustered into groups according to the zoom level. The clustering can be activated/disabled by clicking on the group.	mandatory
UC01-5	Landing Page - Filtering	The landing page allows the user to filter for certain properties (e.g. ASN). The desired selection is either highlighted or the leftover are hidden.	optional
UC02	Open other SR apps	There are many SR apps which should be selectable in a searchable menu. Central frontend should always show the option to select different SR apps in the top bar. For this project, a mock SR app is sufficient.	mandatory
UC03	Port over the Service Programming application	The current UI of the Service Programming SR-App should be ported over to use the newly developed Central Frontend	optional
UC04	Manage favourite SR apps	The user may need the same SR apps often. So a favourite bar or category in the menu could be provided.	optional
UC05	CRUD links	This functionality would allow to add/remove simulated links to the network topology. This data would come from a different SR app and would only be queried.	optional

UC06	CRUD routers	This functionality would allow to add/remove routers to the network topology. This data would come from a different SR app and would only be queried.	optional
UC07	Add SR apps dynamically	New SR-apps can be added dynamically without touching the code of central frontend and make it modular. In some cases the user only wants access to certain SR apps and not all of them. Making a "marketplace" for adding/removing them is implemented.	optional
UC08	Identity and Access Management	This functionality would allow the application to connect to an identity provider (e.g. Keycloak) and provide/allow different functionality depending on the users permissions.	optional

Table 1: Functional Use Cases

2.2.4 NFRs (Non functional requirements)

#	**Requirement**	**optional/mandatory**
NFR1	The application needs to be developed cloud-natively. Meaning it has to run in a Docker.	mandatory
NFR2	The application needs to easily scale horizontally and be able to be deployed onto any Kubernetes cluster	mandatory
NFR3	The application should display up to 1000 nodes in less than 2 seconds. This can be achieved by using a webGL framework for client side graphics assisted calculation.	mandatory
NFR4	The components used to implement the functional use cases are reusable for future SR apps.	mandatory

Table 2: Non Functional Requirements

2.3 Architecture

2.3.1 Overview

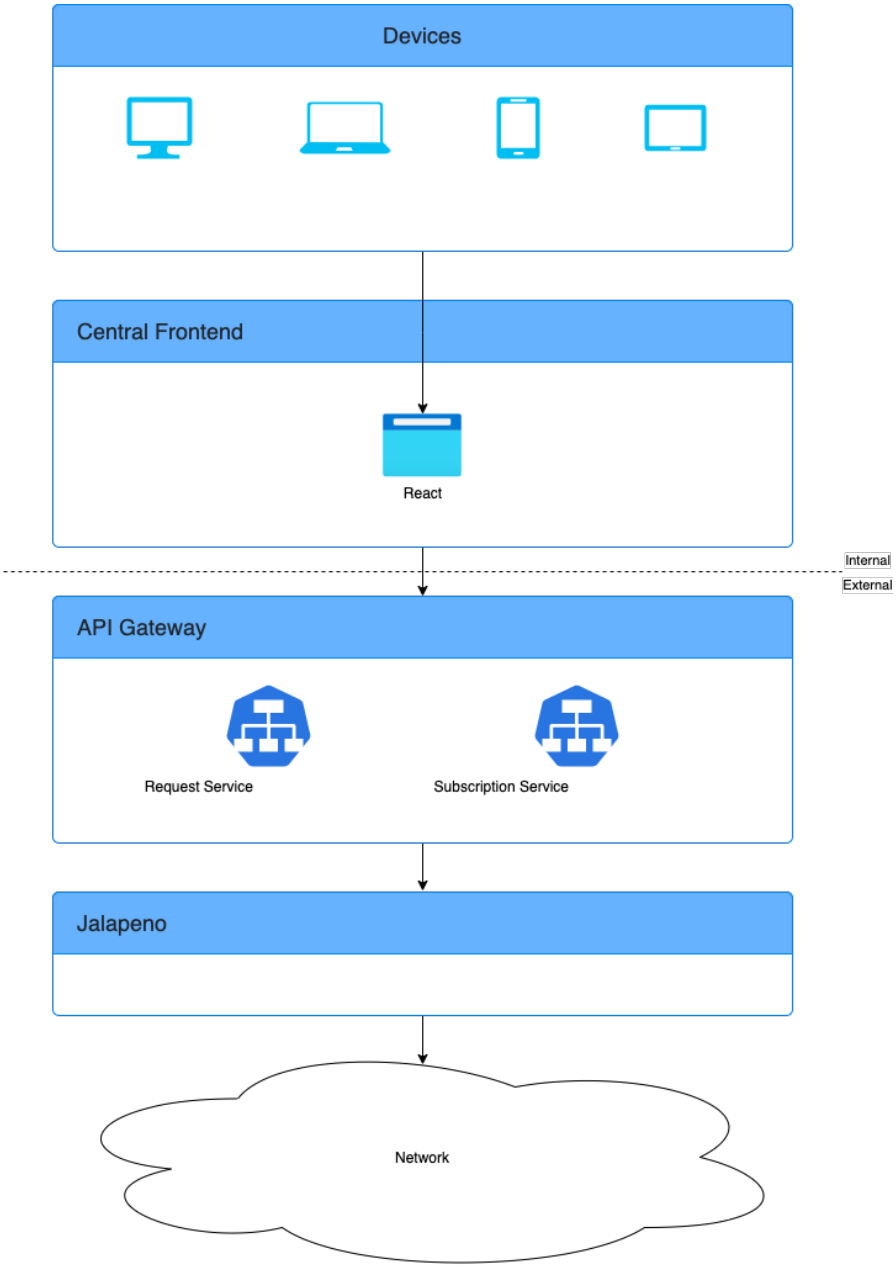


Figure 2: System Overview

2.3.2 Technologies

We are limiting our technologies to open source and freeware to enable a steady development process which is not relying on paid frameworks or infrastructure. Also further development outside of this project are made accessible in the future with this approach.

Languages

- TypeScript
- HTML5
- CSS (SASS)

Data Serialisation languages

- YAML
- Helm Templating

Libraries

- sigmaJS v2 - A JavaScript library dedicated to graph drawing
- React >= v17.0.0 - A JavaScript library for building user interfaces
- gRPC-web - A gRPC implementation for browsers

Browser Support (tested)

- Chrome v96.0.4664.110
- Firefox v95.0.1
- Safari v15.0

2.3.3 External interfaces

The Central Frontend application will be using the Jalapeno API Gateway to interact with the SR network.

The API is based on gRPC and the protobuf definitions can be found here.

The API documentation can be found here.

The Jalapeno API Gateway was extended to use the envoy proxy, allowing the Central Frontend to use the gRPC-web implementation.

2.3.4 Domain Model

We waive the Domain Model, since we do not see the added value of it.

2.3.5 UI/UX Design

Wireframes The interactive wireframes can be found here.

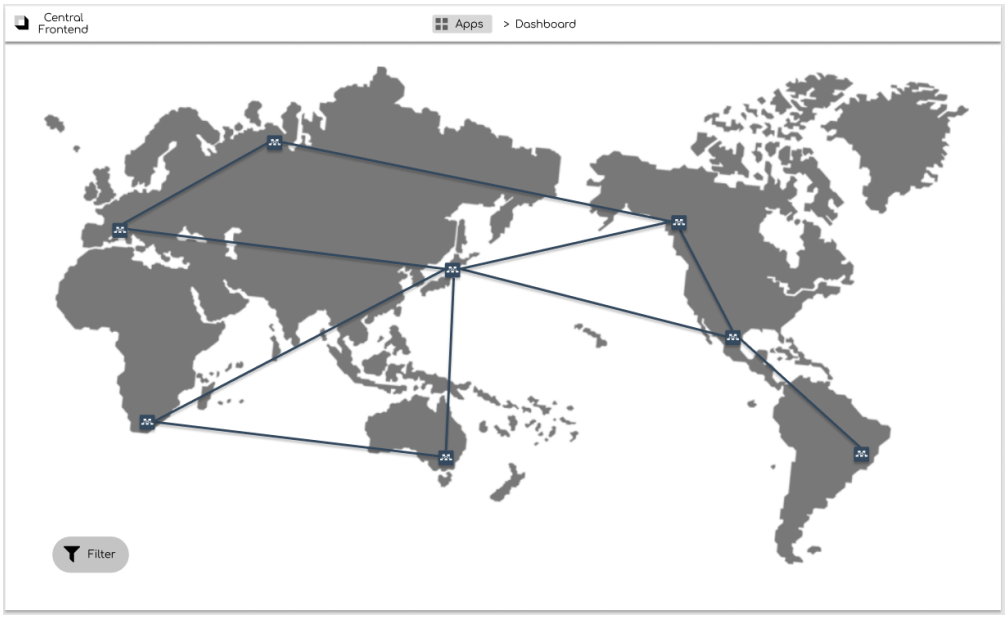


Figure 3: Dashboard

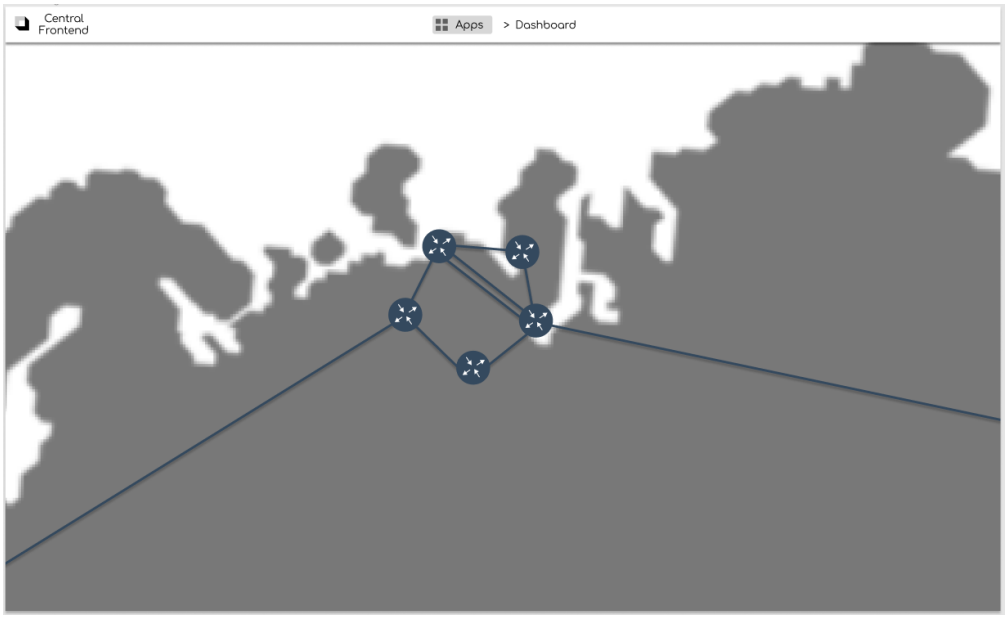


Figure 4: Clustered Nodes

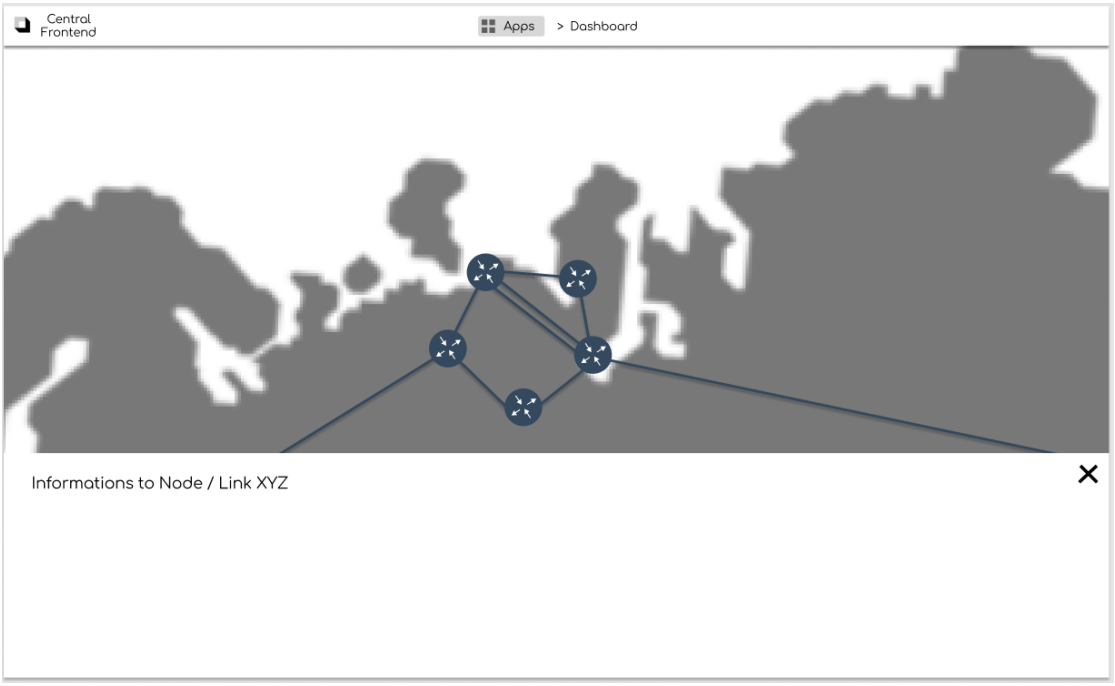


Figure 5: Node Information

Figure 6: Filtering

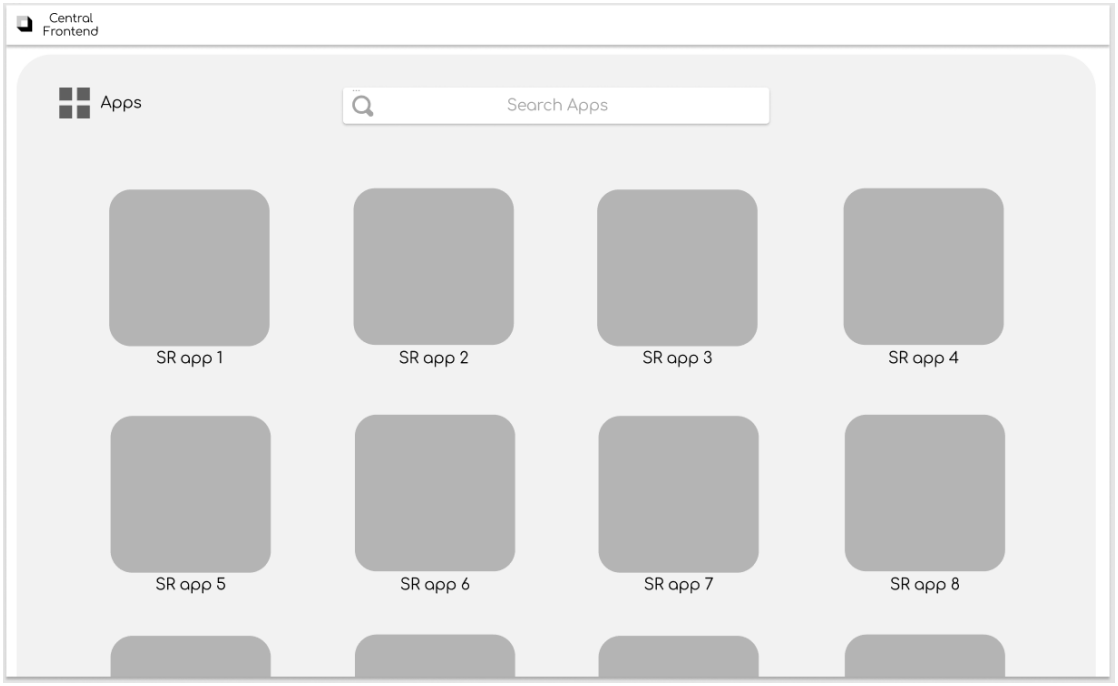


Figure 7: SR app menu

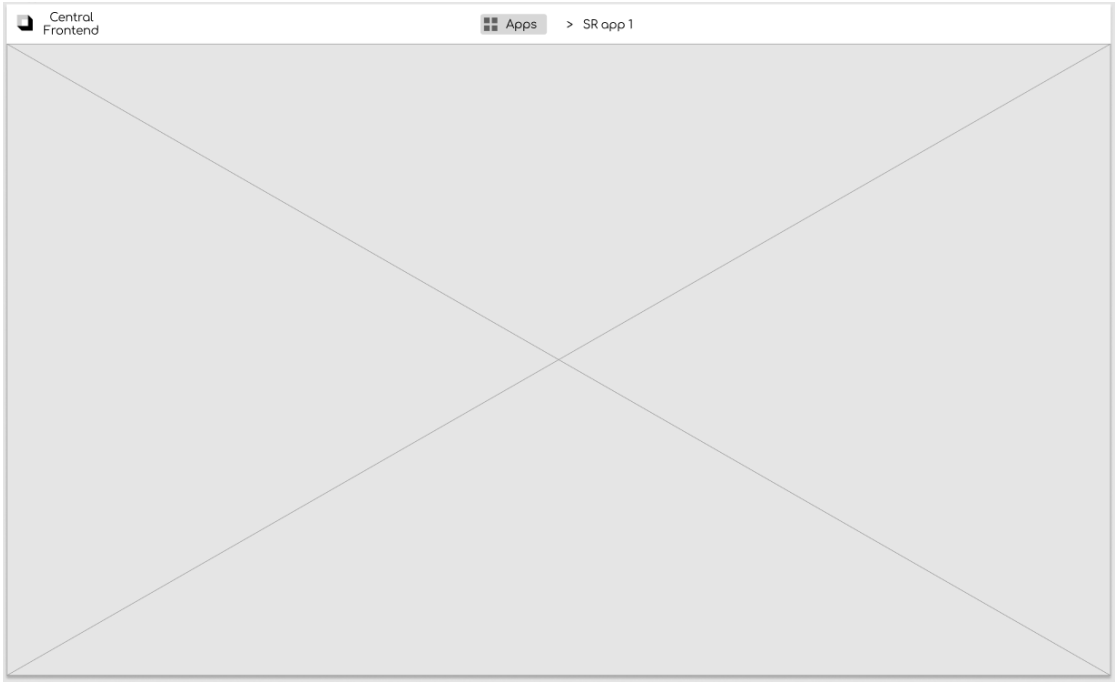


Figure 8: SR app screen

Material Design The Central Frontend will be following the Material Design guidelines.

Color Palette We decided to use the official cisco webex meetings colourpalette as it seemed suitable for our purpose. It features a main color, a shade and an accent color. This follows the material design guidelines in which those three colors are needed.

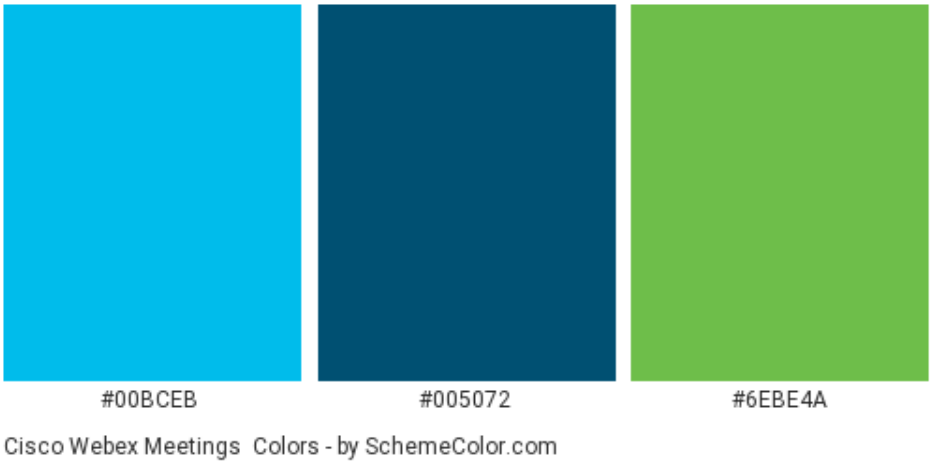


Figure 9: Color Palette

2.3.6 Deployment

The Central Frontend will be containerized and deployed to a Kubernetes cluster. To simplify the deployment, the Central Frontend will be packaged into a helm chart.

The helm chart will allow for either an ingress deployment or a service type NodePort in case the future Kubernetes cluster does not have a service type load balancer running.

All of the Central Frontend components will be running in the namespace "central-frontend".

A ConfigMap will be available to configure any parameters.

The Central Frontend will not handle any TLS termination, as this is the task of the ingress.

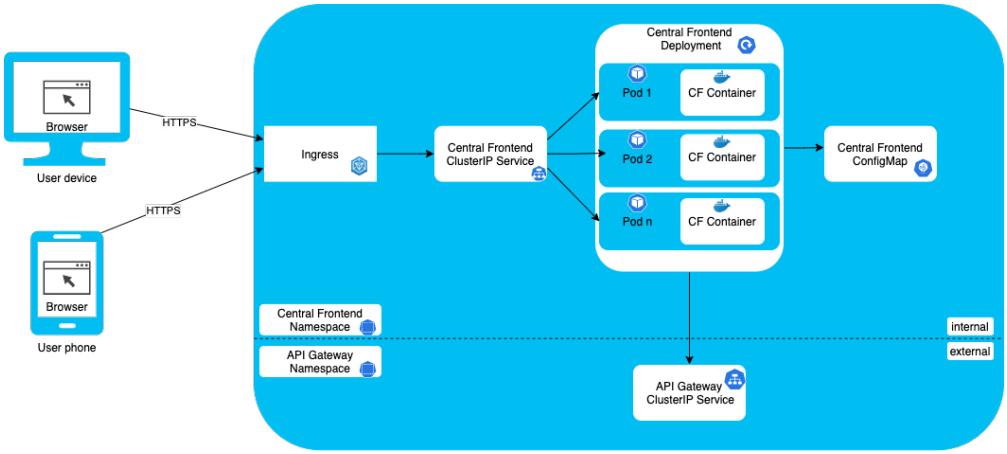


Figure 10: Deployment with Ingress

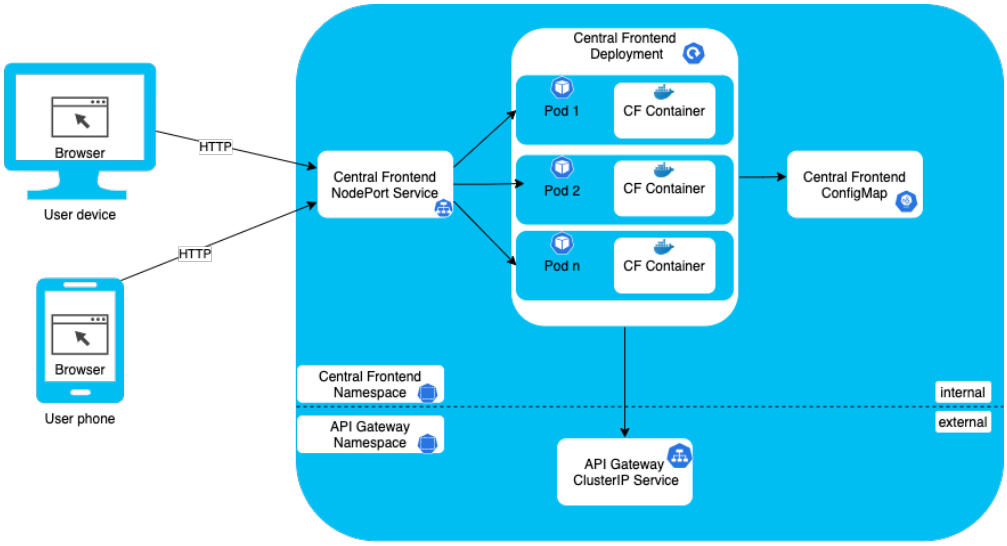


Figure 11: Deployment with NodePort

2.3.7 Infrastructure

The Central Frontend has two different enviroments where it can run:

- Server: sr-000167.network.garden, this enviroment is a real segment routing network. It contains 10 LSNodes. The only part of it that is mocked, is the ArrangoDB coordinates collection and its values.
- Server: sr-000169.network.garden, this is a mocked enviroment, which we can use to mock nodes, links, edges and coordinates in high numbers. The Jalapeno API Gateway is deployed and can be used.

2.4 Design decisions

2.4.1 Network graph visualisation framework

To be able to display a network graph you need a JavaScript visualisation framework. We decided to use the open source framework sigmaJS v2, as it provides all the necessary tooling to visualise nodes and edges in a variety of ways. It was released in the beginning of this thesis.

The reason against sigmaJS v1 is that the plugins and documentation are outdated and no longer officially supported. Additionally, the INS will not have to migrate to version 2 in the near future. This comes with a few caveats which will be described in this chapter.

The reason that we decided against other libraries like d3.js or vis.js is that sigmaJS integrates webGL natively. This means it is more performant than other SVG or canvas based rendering libraries out of the box. This is essential to the purpose of the Central Frontend application because the topologies which should be displayed can contain thousands of nodes and links.

There also were other frameworks which had webGL support, but they are not free or open source which eliminated them from the evaluation process altogether.

Scalability Tests

Network topologies can have a large number of nodes and edges that need to be displayed. This can have a big effect on performance of our product. To verify the capabilities of sigmaJS we did performance tests with varying numbers of nodes and edges both with webGL disabled and enabled.

#	No. Nodes/Edges	Time with webGL	Time without webGL
1	1000 Nodes / 0 Edges	519ms	1.91s
2	1000 Nodes / 1000 Edges	866ms	3.23s
3	10000 Nodes / 0 Edges	1.23s	5.52s
4	10000 Nodes / 10000 Edges	1.85s	10.1s

Table 3: Scalability Test

We can infer from the above tests that there will be no problem displaying thousands of nodes without waiting longer than 1-2 seconds. With webGL enabled we can satisfy the requirements of this project and could therefore use this framework performance-wise.

Advantages

SigmaJS v2 has a big community and has a high activity in its open source repository. We could establish a connection to one of the maintainers Alexis Jacomy and have direct contact with him for questions regarding feature development of the newly released version of sigmaJS.

Downsides

Because this library was released during this term it is quite new and has little plugins written for it. To our advantage we can still write our own so called programs for sigma which can add certain functionalities to the rendering process. This gives us a lot of room to customise the aspects of our graph which need to be met.

There is no support for integrating OpenStreetMap or Google Maps as a synchronised background

layer. This has to be implemented by us with no integrated support of the sigmaJS framework.

Multiple edges between nodes are possible, but are overlapping. This feature has not yet been integrated into sigmaJS v2 but will follow soon.

2.4.2 Map

Following the open source approach of this thesis we decided that Leaflet will be used instead of Google Maps. They are very similar in usage and performance and are therefore interchangeable.

There are different map faces available which change the appearance of the background map. This allows maximum customisation to our colour scheme and adds to the overall look and feel.

Instruction to move the map according to the graphs movements can be passed to the Leaflet coordinate API. This allows to synchronise all the drag and zoom events to the map and move it to the same coordinates.

2.4.3 Clustering

The network graph can contain a lot of nodes, which will eventually impair usability and functionality if too many are displayed. This is where clustering the nodes comes into play. By taking a group of nodes separated by city, country or general geolocation they can be condensed down to a region or cluster. This will greatly increase visibility and usability of the network topology map by showing reduced information when needed.

We will focus on multi-level clustering as topologies can span over the globe and therefore need multiple levels of abstraction.

This is not supported natively in sigmaJS and needs to be implemented by us. An algorithm that clusters nodes by geolocation has to be made and integrated into the graph construction. The goal is to group nodes which are geologically close together and define the center of this group to display the cluster.

Possible variations

The clustering algorithm was initially undervalued in its complexity. It is very difficult to find an appropriate clustering approach and needs to be thoroughly addressed in the bachelor thesis. There are three possible variations of clustering we discussed and evaluated. Each of them relies on fixed zoom levels which depending on how far you zoom into the map it changes from clustering to showing nodes. At the moment there are two zoom levels planned, but this can easily be expanded later on.

City based If we were to cluster all the nodes based on a list of coordinates from big cities (also possible over Leaflet API) we would cluster all the nodes into those coordinates when they are inside the desired radius. This approach can have its downsides when routers are far away from a big city as they would not get clustered.

Geographically based This approach would cluster nodes with a maximum distance bias to each other to a group and calculate the center with the mean vector function. This is the most versatile approach because every node has the possibility of being allocated to a cluster no matter how remote the node is located. The downsides to this method is that clusters can appear between region borders or in oceans if the bias is not suited for the topology.

Static Static clusters would always be displayed in the same spots on the map if any nodes were

present near the static locations. This was the easiest approach but also the least desirable. It offers the least flexibility and is not dynamic at all.

The decision on this was quite hard and we just went for the geographically based clustering approach with fixed zoom levels as it was just the most flexible one and can be built on quite well because of its dynamicity. This will also be a topic in our bachelor thesis where we invest more time in the ideal design of this problem.

2.4.4 Browse SR Apps

One of the mandatory requirements for the Central Frontend is to be able to list and launch other SR Apps and to be the main UI for all the current and future SR Apps.

There are different possible solutions to solve this requirement:

Static The easiest and most straight-forward solution is to add an SR App manually into the Central Frontend source code and use the components already implemented in it. The Central Frontend would need to be rebuilt and redeployed whenever a new SR App is added.

Dynamic While a static approach is much easier, the cleaner solution would be to let the SR Apps add their UIs dynamically to the Central Frontend.

Decision

It was deciced that as a hard requirement for this thesis, the Central Frontend should allow to add the SR Apps statically. As an optional requirement and if there is enough time left, a dynamic approach can be implemented.

2.4.5 Geo Location

As there is no way and no plan currently to add geo coordinates of nodes into the segment routing network itself, it was decided that the Central Frontend will use mocked data for the geo location of the nodes. For this, the INS team has expanded the ArrangoDB with a custom collection called `LSNode_Coordinates`. The Jalapeno API Gateway was also extended to be able to deliver the coordinates through the gRPC API.

2.5 Implementation

2.5.1 Repositories

These are all the GitLab repositories that we created and used:

- central-frontend: The main repository for the Central Frontend React app. `https://gitlab.ost.ch/ins-stud/sr-app-central-frontend/central-frontend`
- central-frontend: The repository containing the helm chart of the Central Frontend. `https://gitlab.ost.ch/ins-stud/sr-app-central-frontend/central-frontend-helm-chart`
- docs: The repo containing all the docs. `https://gitlab.ost.ch/ins-stud/sr-app-central-frontend/docs`
- gen: The repo containing the small go program used to generate json file which were uploaded to the mocked environment. `https://gitlab.ost.ch/ins-stud/sr-app-central-frontend/gen`

2.5.2 Central Frontend Overview

The Central Frontend is a React application that communicates with the Jalapeno API Gateway through gRPC-web and renders the nodes using sigmaJS v2. The sigma graph is drawn onto a Leaflet map and being clustered to make it more visible and performant. Additionally, other SR Apps that are being implemented in the Central Frontend can be searched and launched through the SR App list. The application is written in TypeScript.

2.5.3 Project Structure

central-frontend The root directory. Contains all the configuration files e.g. ESLint and prettier, the `Dockerfile`, package.json and the `public` and `src` directories.

central-frontend/public Contains all the files that the end users browser accesses directly. E.g. images, icons and the index.html file

The `index.html` contains a div with the id `root`. Into that div, the whole Central Frontend Application will be rendered.

central-frontend/src Contains the `index.tsx` and `index.css` files and the `api`, `apps`, `components` and `test` directories.

The `index.tsx` file injects the `App` component into the `root` div.

central-frontend/src/api Contains all the generated Javascript/Typescript files from the Jalapeno API Gateway proto files. Additionally, it contains the `NodeService.ts` file, which the abstraction for the Central Frontend to communicate with the Jalapeno API Gateway.

central-frontend/src/components Contains all the components that are used in the Central Frontend.

central-frontend/src/apps Contains all the SR-Apps that implemented in the Central Frontend and the `App.tsx` file that represent the `App` component.

The `App` component renders the `CentralFrontend` app-component.

2.5.4 App-Component

The app-components are React components that represent different SR Apps. The app-components use the components and the API service.

CentralFrontend

The `CentralFrontend` component renders a div with the id `container` that contains two more divs with the ids `infobar` and `sigma`.

Additionally, it renders the `SigmaMap` app-component.

Counter

The `Counter` represents a mock SR app. The counter uses the API service to count how many nodes and edges there are in the network and displays it.

2.5.5 Components

SigmaMap

The `SigmaMap` component renders the Leaflet map into the underneath the `sigma` div. When the map is rendered, it calls the `SigmaGraph` function.

For the map instance, `react-leaflet` is used to create the map. The `react-leaflet` library is used as it offers React components for Leaflet maps.

```
export const SigmaMap: React.FC = () => {
    const [map, setMap] = useState<L.Map | null>( initialState: null);
    SigmaGraph(map!).then();
    return (
        <MapContainer id="map"
                      center={[0, 0]}
                      zoom={0}
                      zoomDelta= {0.25}
                      zoomSnap= {0}
                      zoomControl={false}
                      doubleClickZoom={"center"}
                      keyboard={false}
                      whenCreated={setMap}
                      minZoom={2}
                      maxZoom={22}
        >
            <TileLayer
                attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors &copy; <a href="https://carto.com/attributions">CARTO</a>'
                url="https://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}{r}.png"
            />
        </MapContainer>
    );
};
```

Figure 12: React Leaflet

The following Leaflet settings are used for the Central Frontend:

- center = [0,0], sets the center of the map at the coordinates 0, 0
- zoom = 0, sets the intial zoom to 0
- zoomDelta = 0.25, controlls how many zoom levels to zoom in/out when zooming

- zoomSnap = 0, when set to 0, Leaflet will not snap to the zoom level
- zoomControl = false, disable the UI zoom controlls
- keyboard = false, diable the keyboard controlls
- minZoom = 2, sets the minimum zoom of the map
- maxZoom = 22, sets the maximum zoom of the map

In the title layer, we can configure which skin of the map we want to use. The Central Frontend uses the Dark Matter with label, carto.com map. More can be found here: `https://carto.com/help/building-maps/basemap-list/`

To get the map instance into the map variable, we are using the React hook `useState`. The `setMap` function sets the map into the `map` variable. When the `map` variable is instantiated and not null, `SigmaGraph` function is called is called.

SigmaGraph

The `SigmaGraph` function is the centerpiece of the landing page. It is given the `Leaflet` map instance as a parameter so it can later calculate the positions of the nodes.

As a first action, the function tries to find the `sigma` div in the page and instantiates a new MultiGraph for `sigmaJS` to later use. Once the the `sigma` div is found, the `sigmaJS` instantiated with the following settings:

```
const renderer = new Sigma(graph, container, settings: {
    stagePadding: 0,
    nodeProgramClasses: {
        image: getNodeProgramImage(),
    },
    enableEdgeClickEvents: true,
    enableEdgeHoverEvents: "debounce",
    edgeReducer(edge : string , data : Attributes ) {
        const res = { ...data };
        if (edge === hoveredEdge) res.color = "#6ebe4a";
        return res;
    },
    labelColor: { color: "#aaa" }
});
```

Figure 13: Sigma Settings

Important things to note:

- The `getNodeProgramImage` allows us later to set the custom images to our nodes.
- We enable edgeClickEvents as we want to click on edges and display information of corresponding `LSLinks`.
- The `edgeReducer` setting allows us to color the hovered edge green. We can easily extend

the `edgeReducer` to for example make the edge bigger when hovered.

Once the `sigma` instance is initialised, we get all the `LSNodes`, `LSNodeCoordinates`, `LSLinks` and `LSNodeEdges` from the Jalapeno API Gateway.

Then the nodes will be clustered together and the `renderLevel1` will be rendered. The `renderLevel1` renders clustered view of the graph, grouping the nodes into regional groups.

The `region` image is used for the cluster nodes.

Figure 14: Region

```
async function renderLevel1(renderer: Sigma, results: { centroid: number[]; elements: number[][]; }[]) {
    renderer.getGraph().clear();

    let count = 0;
    for (const result of results) {
        const x = result.centroid[0];
        const y = result.centroid[1];

        const graphProjection = latlngToGraph( coord: [x, y], renderer);

        graph.addNode( node: "area" + count,  attributes: {
            ...result,
            ...graphProjection,
            x: graphProjection.x,
            y: graphProjection.y,
            size: 10,
            type: "image",
            image: "region.png",
        });

        count++;
    }
}
```

Figure 15: Render Level 1

Depending on the zoom of the graph, if a certain threshold is crossed, the `renderLevel3` will be rendered.

`RenderLevel3` is very similar to `renderLevel1` just that the nodes are a bit smaller and use the router image.

There are currently only these two render levels implemented. These can be extended to include more render levels in the future.

Figure 16: Router

Once the nodes are rendered in the `renderLevel3`, the edges are also being rendered.

```
}).finally(
    onfinally: () => {
        getEdgesArray().then(edges => {
            edgeList = groupEdges(edges)
            for (const edge of edgeList) {
                const from = edge.getFrom();
                const to = edge.getTo();
                if (from != null && to != null) {
                    graph.addEdge(
                        edge.getFrom(),
                        edge.getTo(),
                        attributes: {
                            key: edge.getKey(),
                            multi: true,
                            color: "#fff",
                            size: 0.5,
                        }
                    );
                }
            }

        });
    }
);
}
```

Figure 17: Render Level 3

As sigmaJS does not properly render multiple edges between two nodes, as it stacks them above each other, we call the `groupEdges` functions, to filter out any multi-edges between nodes. We then render only one edge for multiple edges.

Additionally, the `SigmaGraph` function registers the `clickNode` and `clickEdge` event listeners to render `Infobar` component which shows the details of a node, a link or a multi edge and it registers the `afterRender` event listener which calls the `syncLeafletBboxWithGraphBbox` function.

```
renderer.on("clickNode", function (node) {
    const lsNode = nodeList?.find(x => x.getId() == node.node);
    if (null != lsNode) {
        ReactDOM.render(<NodeInfobar node={lsNode} showInfo={true} />, document.getElementById( elementId: "infobar"));
    }
});
```

Figure 18: Click Node

The `clickEdge` event listener differentiates between multi edges and single edges. If there are multiple edges between nodes, the event listener builds an array of `LSNodeEdges` and gives it as a prop to the the `MultiEdgeInfobar`.

```
109     renderer.on("clickEdge", ({edge}) => {
110         const label = graph.getEdgeAttribute(edge,  name: "key");
111         const lsEdge = edgeList?.find(x => x.getKey() == label);
112
113         if (lsEdge != undefined){
114             // e.g 0::1
115             const key = lsEdge.getKey()
116             // e.g 1::0
117             const inversekey = lsEdge.getTo().replace("LSNode/", "") + "::" + lsEdge.getFrom().replace("LSNode/", "")
118
119             if (multiEdgesMap.has(key) || multiEdgesMap.has(inversekey)){
120                 let otherEdges: LsNodeEdge[] | undefined = undefined;
121
122                 otherEdges = multiEdgesMap.get(key)
123
124                 const inverseEdges = multiEdgesMap.get(inversekey)
125                 if (inverseEdges != undefined){
126                     if (otherEdges == undefined) {
127                         otherEdges = inverseEdges
128                     } else {
129                         otherEdges?.concat(inverseEdges)
130                     }
131                 }
132
133                 if (otherEdges != undefined){
134                     ReactDOM.render(<MultiInfobar edges={otherEdges} showInfo={true} links={linkList}/>, document.getElementById( elementId: "infobar"));
135                 }
136             } else {
137                 const lsLink = linkList?.find(x => x.getKey() == lsEdge?.getLink())
138                 ReactDOM.render(<LinkInfobar link={lsLink} showInfo={true}/>, document.getElementById( elementId: "infobar"));
139             }
140         }
141     });
```

Figure 19: Click Edge

Clustering

The clustering algorithm was not implemented out of the box from sigma, so we had to implement it on our own. How it works is that a list of geo coordinates is passed to the clustering method as well as a bias which defines how much distance is allowed between the nodes which are in the range of a centroid coordinate of a cluster.

First all the coordinates are looped through and the distances are calculated between them. This way we can calculate the mean distance and with that the variance by squaring them and summing them up into the variance total. You get the standard deviation (1) by taking the square-root of this sum. Then by multiplying the standard deviation with the defined bias we get a threshold. This makes it controllable as this bias can be passed to the algorithm.

$$s^2 = \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2 \times bias} \tag{1}$$

Then we take a random coordinates from the given coordinate list as initial cluster centroid and iterate over all nodes and check if they are closer than the calculated threshold to this cluster. Otherwise check if next closest cluster is satisfying the threshold instead and if thats not the case create a new cluster from this coordinate. This is then repeated as long as clusters exist.

This way there are always nodes added to existing close clusters until there is a new cluster necessary. That is not the most performance optimised solution but it works quite well for the 1000 nodes we initially set as a goal.

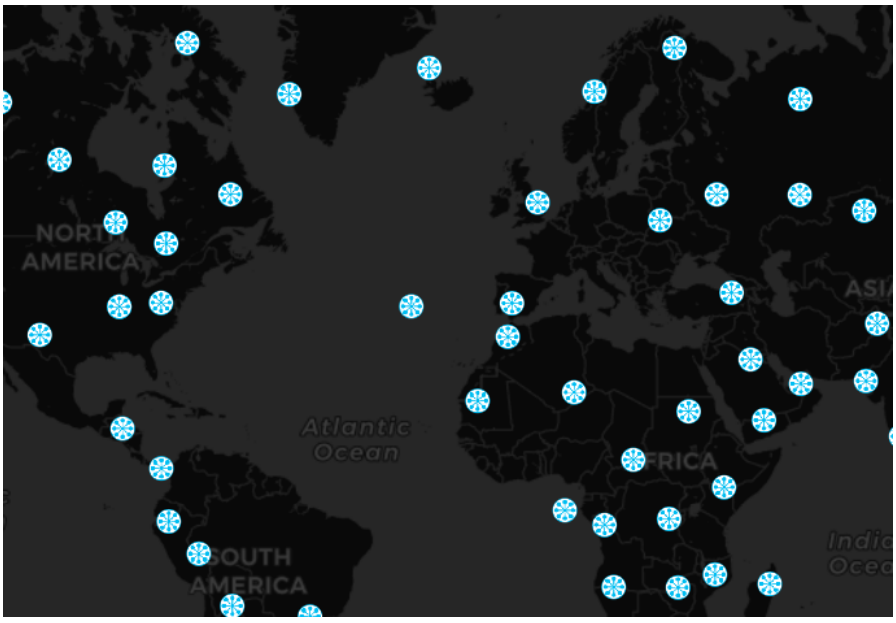


Figure 20: Clustering

Leaflet / Sigma synchronisation

The Leaflet map layer lives behind the various sigma layers and is therefore completely covered by it. This has the consequence that it cannot directly receive the `drag events` triggered from dragging around the sigma graph. Also the geo coordinates are not applicable directly to the sigma graph as it relies on x & y coordinates differing in every generated graph.

So the goal here was to calculate latitude & longitude coordinates to pixel coordinates to display them correctly on the map. This happens by taking the pixelsize of the div, displaying the graph and projecting the coordinates to this height and length. This can be done with the `project`-method that leaflet provides. The other way round this works with the `unproject`-method.

Now when you start moving and zooming the graph those events need to be transferred to the map. This works by getting the top left corner and the bottom right corner of the displayed viewport from the graph and calculating the geolocations of those points. Those will be passed to the leaflet method `fitBounds` or when zooming `flyToBounds` which then displays the map correctly according to the graph.

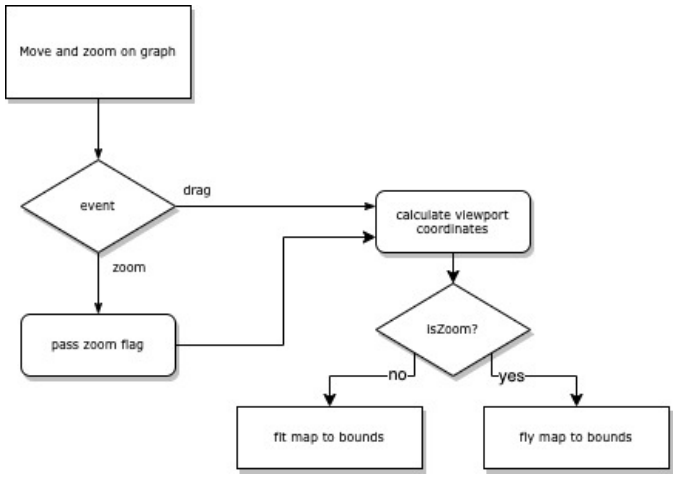


Figure 21: Map projection

Infobar

The infobar is a component which has multiple purposes and implementations. It displays node or edge information when you click on either one in the graph. This component is a simple HTML returning React component and gets rendered and data passed through on click event.

Under the directory `central-frontend/src/components/ui/infobar` all infobars and the CSS can be found.

All infobars have a `Close` button. If the button is pressed, the Infobar will hide itself.

All infobars can be extended with more topology or telemetry data during the bachelor thesis.

NodeInfobar

The `NodeInfobar` component is rendered when a LSNode is clicked in the `SigmaGraph`.

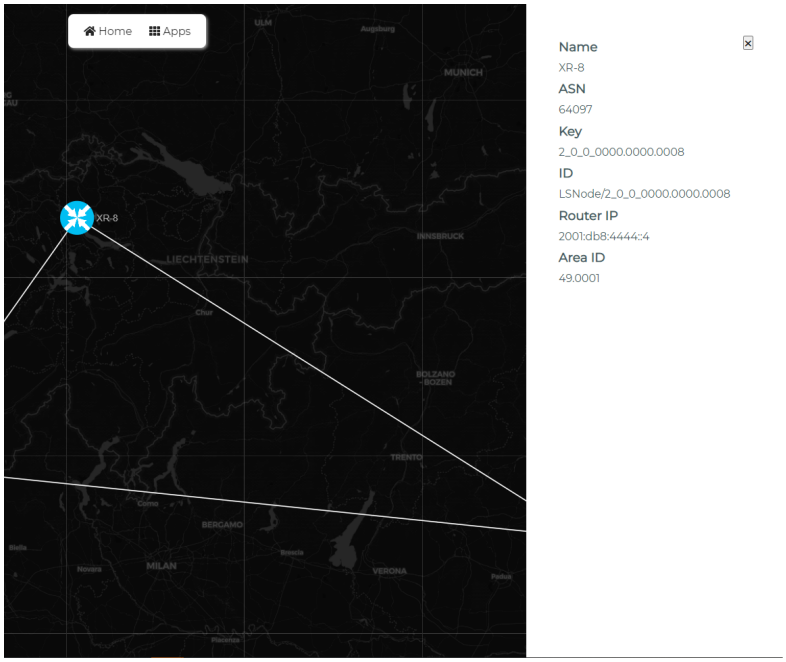


Figure 22: Infobar

```
return (
    <div>
        <button id={"close"} onClick={hideInfo}>&#10006;</button>
        <h3> Name </h3>
        <p>{node.getName()}</p>
        <h3> ASN </h3>
        <p>{node.getAsn()}</p>
        <h3> Key </h3>
        <p>{node.getKey()}</p>
        <h3> ID </h3>
        <p>{node.getId()}</p>
        <h3> Router IP </h3>
        <p>{node.getRouterIp()}</p>
        <h3> Area ID </h3>
        <p>{node.getAreaId()}</p>
    </div>
);
```

Figure 23: Node Infobar Code

LinkInfobar

The `LinkInfobar` component is rendered when a LSNodeEdge is clicked in the `SigmaGraph`.

Figure 24: Link Infobar

```
return (
    <div>
        <button id={"close"} onClick={hideInfo}>&#10006;</button>
        <h3> ID </h3>
        <p>{link.getId()}</p>
        <h3> Key </h3>
        <p>{link.getKey()}</p>
        <h3> Peer ASN </h3>
        <p>{link.getPeerAsn()}</p>
        <h3> Peer IP </h3>
        <p>{link.getPeerIp()}</p>
        <h3> Area ID </h3>
        <p>{link.getAreaId()}</p>
        <h3> Next Hop </h3>
        <p>{link.getNexthop()}</p>
    </div>
);
```

Figure 25: Link Infobar Code

MultiLinkInfobar

The `MultiLinkInfobar` component is rendered when a LSNodeEdge that has multiple edges between nodes is clicked in the `SigmaGraph`.

The `MultiLinkInfobar` differs from the the other info bars as it is given a array of `LSNodeEdges` and `LSNodeLinks`.

To give complex props to a react component we have to first define an interface.

```
interface EdgesProps{
    edges: LsNodeEdge[]
    links: LsLink[] | null
    showInfo: boolean
}
```

Figure 26: Interface

We can now give an array of `LSNodeEdges` and `LSNodeLinks` as parameters to the component.

We then iterate over the `LSNodeEdges` and show them in a list. An button will be shown, which renders the **LinkInfobar** for the corresponding **Link**.

This component is only a temporary measure to show multiple links between two nodes. When sigmaJS supports multiple edges between nodes, this feature will be removed.

```
export const MultiInfobar: React.FC<EdgesProps> = ({edges :LsNodeEdge[] , showInfo :boolean , links :LsLink[] | null }) =>{
    const infobar = document.getElementById( elementId: "infobar");
    if (infobar != null) {
        infobar.style.display = showInfo ? "block" : "none"
    }

    return (
        <div>
            <button id={"close"} onClick={hideInfo}>&#10006;</button>
            {edges.map(function (edge: LsNodeEdge, index: number){
                return (
                    <li key={index}>
                        <h3> Key </h3>
                        <p>{edge.getKey()}</p>
                        <h3> ID </h3>
                        <p>{edge.getId()}</p>
                        <h3> Link </h3>
                        <p>{edge.getLink()}</p>
                        <button onClick={() => {
                            const lsLink = links?.find(x => x.getKey() == edge.getLink())
                            ReactDOM.render(<LinkInfobar link={lsLink} showInfo={true}/>, document.getElementById( elementId: "infobar"));
                        }}>Go to Link</button>
                    </li>
                )
            })}
        </div>
    );
}
```

Figure 27: MultiLinkInfobar

SR App List

The main `App` component that lies in the structure `central-frontend/src/apps` renders the navigation bar component `NavBar` and the`CentralFrontend` component. The `CentralFrontend` component is rendered into the `content` div.

The `NavBar` component renders two buttons at the top of all pages of the Central Frontend. The `Home` button redirects the user to the landing page of the Central Frontend. The `Apps` button renders the `SrAppList` component into the `content` div.

Figure 28: Navbar

The `SrAppList` component contains a search bar and a list of cards representing an SR App. Each SR App has a title, picture and a description. The searchbar filters the card list underneath while typing and looks for a match in the title or the description. If the description is too long to fit into the card, it is cut off. When a card is being hovered, it lights up. If you click a card, the specific SR App is being rendered into the `content` div.

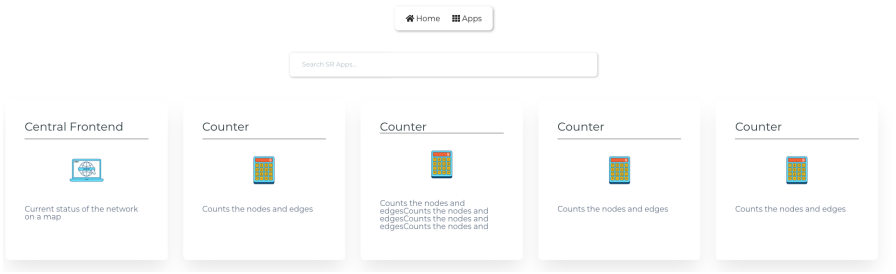


Figure 29: SR App List

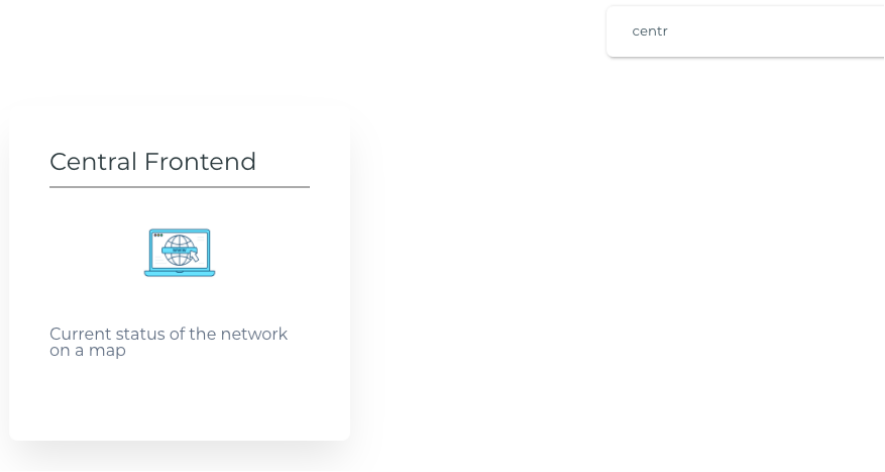


Figure 30: SR App List Search

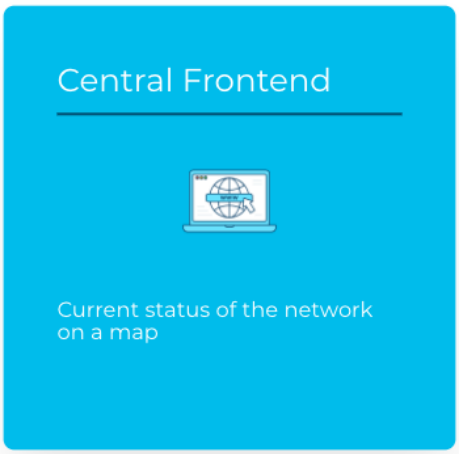


Figure 31: SR App List Hover

Add new SR App

To add a new SR App, a new folder needs to be added into the `central-frontend/src/apps` directory. In that directory we can now a new `.tsx` file.

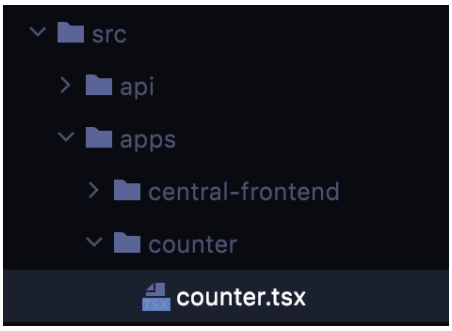


Figure 32: SR App List Directory

In the `.tsx` file, a new react functional component can now be created. This component can use all the existing components and unit of the Central Frontend. For example the `Counter` component using the Jalapeno API Gateway Service:

```tsx
import React, {useState} from "react";
import {getNodesArray} from "../../api/NodeService/NodeService";

export const Counter: React.FC = () => {
    const [node, setNode] = useState( initialState: 0);
    const [edge, setEdge] = useState( initialState: 0);

    getNodesArray().then(nodes => setNode(nodes.length))
    getNodesArray().then(edges => setEdge(edges.length))

    return(
        <div>
            <h1>I count things</h1>
            <p>Amount nodes: {node}</p>
            <p>Amount edges: {edge}</p>
        </div>
    )
}
```

Figure 33: Counter

After the component has been implemented and is ready to use, we can register it in the `SrAppList` component which can be found at `central-frontend/src/components/sr-app-list/list.tsx`.

To do this, we have to create a new instance of the `SrApp` class and add it to the `srApps` list:

```
export class SrApp {
    id: number;
    name: string;
    description: string;
    image: string;
    component: JSX.Element | null;

    constructor(id: number, name: string, description: string, image: string, component: any) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.image = image;
        this.component = component;
    }
}
```

Figure 34: SrApp Class

```
const srApps = [
    new SrApp(
        id: 0,
        name: "Central Frontend",
        description: "Current status of the network on a map",
        image: "central_frontend.png",
        <CentralFrontend />
    ),
    new SrApp(
        id: 1,
        name: "Counter",
        description: "Counts the nodes and edges",
        image: "Calculator_Outline.png",
        <Counter />
    ),
];
```

Figure 35: SrApp Class List

The picture needs to be added into the `central-frontend/public` directory.

2.5.6 Browser Events

OnClick Events

Nodes and Edges can be clicked on. This triggers the infobar with the correct information at display. This happens with a React component that returns a div which then gets added to the ReactDOM dynamically.

EdgeHover Events

Hovering over the edges between the nodes makes them visually stand out in green. This allows for better visibility of the edge you are currently on. Leaving the edge again makes it return to white color.

Drag Events

Dragging the Sigma graph produces a lot of move events which we pass through to leaflet. They are used as a trigger for moving the map in sync with the graph.

Zoom Events

Similarly to the move event zooming also produces zoom events which then get passed to the leaflet instance to move the map accordingly.

2.5.7 Similar Node Coordinates

Sometimes the rendering of the graph produces overlapping nodes because of their close proximity. For this problem we used the `graphology-layout-noverlap` framework which moves nodes with close coordinates so that they don't overlap.

```
1   noverlap.assign(graph, params: {
2      maxIterations: 50,
3      settings: {
4         ratio: 0.1,
5         margin: 0,
6      },
7   });
```

This happens iteratively until all the nodes do not overlap anymore and a maximum of 50 times. We do this just at the moment of the population of the graph directly after adding all the nodes.

2.5.8 Docker

The Central Frontend runs in an `nginx` Docker container. The Dockerfile is built up like following:

```
FROM node:13 as 🔨 build
WORKDIR /app
ENV PATH /app/node_modules/.bin:$PATH
COPY package.json ./
COPY package-lock.json ./
RUN apt-get update -y
RUN apt-get install -y python
RUN npm ci
RUN npm install react-scripts@3.4.1 -g --silent
COPY . ./
RUN npm run build

FROM nginx:stable-alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Figure 36: Dockerfile

The Central Frontend uses a multi staged Dockerfile where the first stage is used to build the react app and the second one is used to run the application in a `nginx` Docker container.

2.5.9 Helmchart

The helmchart packages the Central Frontend Docker container into a Helm chart ready to be used by any Kubernetes cluster.

```yaml
replicaCount: 1

image:
  repository: "registry.gitlab.ost.ch:45023/ins-stud/sr-app-central-frontend/central-frontend"
  pullPolicy: IfNotPresent
  tag: "v0.1.2"

imagePullSecrets:
  - name: regcred
nameOverride: ""
fullnameOverride: ""

service:
  type: NodePort
  port: 30070
```

Figure 37: Helm Chart

Currently, a NodePort service is being used as there is no ingress controller or LoadBalancer controller, so no public IP can be assigned to the Central Frontend. The port for the Central Frontend is: `30070`.

2.5.10 GitLab CI

The GitLab CI creates a new docker container on each push on a merge request and pushes that image to the GitLab container registry. On a merge onto master, a new master docker container is built and pushed. On a new tag, the commit tag is used to create a release of the latest master docker container.

The tag can then be used in the helmchart to deploy that version of the Central Frontend.

2.5.11 gen - our Go program

To generate LSNodes, LSLinks, Coordinates and Edges in high numbers, we created a small Golang program. The application uses a free API to get random city coordinates and assigns them to X amount of nodes and generates links and edges.

The API for random city coordinates can be found here: `https://api.3geonames.org/?randomland=yes`

The instructions how to use the application can be found in the README.md of the gen repo.

2.5.12 Review

Clustering

There were a lot of challenges we were being faced with. The biggest one was the clustering of the nodes in the graph. This happened to consume a lot of time in this project and we ended up with an unideal solution. Following problems arised:

- The clustering algorithm consumes a lot of computing resources as it scales with about O^2 which is not ideal for a solution that will process thousands of nodes.
- There are fixed zoom levels which won't allow for a dynamic solution in different sized graphs because they are static in a sense, that we want to cluster globally and not Switzerland-wide for example.

The mentioned problems can not be easily solved at the moment and need to be generally thought over with a newly created approach. This will be part of the bachelor thesis and will be looked at in detail in the beginning of the next term.

Map synchronisation

Also synchronisation between the graph and the map has been a challenge itself, as this comes down to understanding how the events are generated by the graph and the instructions on moving the viewport in the map are sent correctly. Following problems we discovered during implementing the synchronisation:

- Understanding how coordinates can be mapped to the viewport of sigma.
- Understanding how pixel coordinates correlate to coordinates.
- Calculate pixel coordinates according to the size of the graph.
- Fly and fit to bounds method values calculation as the viewport data from the graph were incompatible with the map viewport data.
- A bit of lag when moving because of the leaflet moving animation (cannot be set to absolute zero).

Multi edges

Currently, multi edges are not shown in the graph directly. Rather they are represented through one edge and if there is more than one edge between two nodes, the edge onClick event renders the `MultiEdgeInfobar` component which displays all edges and links.

This was done, as sadly in theory sigmaJS does support multiple edges between nodes, currently in v2 it simply overlaps each edge without the possibility to add curvature to the edges.

In sigma v1 this was possible and we hope by the beginning of the bachelor thesis, that this feature is released.

2.5.13 Improvements

Empty Latitude Longitude pairs

Sometimes in the sync from sigma to leaflet empty Latitude-Longitude pairs can be passed which then crash the program. We could not fix this error as this is something internal in the unproject method in leaflet. This needs to be addressed in the bachelors thesis. It can be avoided by not successively scrolling in and out in less than a second.

Zoom limits

There is a hard limit for zoom levels set in the map layer. This is different in Sigma. Since v2 it is not possible to set min and max zoom levels. This feature is still in development and will be added to our application as soon as it is available in the next version.

Overlapping edges

When there are multiple edges between the same node they will overlap at the moment this is also something we cannot change without a lot of effort at the moment but will be added in a future release. This will enable us to use this functionality out of the box.

Overlapping nodes

This problem has been solved with the graphology `noverlap`. It takes only the overlapping nodes and places the to a random location with a given margin if this coordinate is not already occupied by another node. It will repeat this process until no overlapping nodes are present. This is a random position and is not guaranteed to work all the time. This has to be revisited.

Docker Build Currently, the API URL is hardcoded into the APIService as the solution with the env variables did not work properly. A good improvement would be to support config hot reloading and to deploy the config in an ConfigMap onto the Kubernetes cluster.

Docker Build Currently, the API URL is hardcoded into the APIService as the solution with the env variables did not work properly. A good improvement would be to support config hot reloading and to deploy the config in an ConfigMap onto the Kubernetes cluster.

2.6 Results

This term thesis has resulted in a workable base for our bachelor thesis in which a running minimal viable product has been developed. We can verify this by looking at our requirements that have been set in the beginning.

The application has been deployed to both environments:

- `http://sr-000167.network.garden:30070/`
- `http://sr-000169.network.garden:30070/`

2.6.1 Use Cases

All the required use cases have been implemented and the optional ones had to be left out because of timely reasons. We wanted to concentrate on the important, required use cases primarily.

#	Use Case	completed
UC01	Show Landing Page	✓
UC01-1	Landing Page - Map	✓
UC01-2	Landing Page - Map Zoom	✓
UC01-3	Landing Page - Status of Links and Nodes	✓
UC01-4	Landing Page - Clustering	✓
UC01-5	Landing Page - Filtering	optional
UC02	Open other SR apps	✓
UC03	Port over the Service Programming application	optional
UC04	Manage favourite SR apps	optional
UC05	CRUD links	optional
UC06	CRUD routers	optional
UC07	Add SR apps dynamically	optional
UC08	Identity and Access Management	optional

Table 4: Use case completion

2.6.2 NFRs (Non functional requirements)

#	Requirement	completed
NFR1	The application needs to be developed cloud-natively. Meaning it has to run in a Docker.	✓
NFR2	The application needs to easily scale horizontally and be able to be deployed onto any Kubernetes cluster	✓
NFR3	The application should display up to 1000 nodes in less than 2 seconds. This can be achieved by using a webGL framework for client side graphics assisted calculation.	✓
NFR4	The components used to implement the functional use cases are reusable for future SR apps.	✓

Table 5: Non functional requirements completion

2.6.3 Speedtest

Our goal was to achieve loading times lower than 2s for a thousand nodes. The performance vastly changes with the device you are using. In the table there are the results of our speed tests:

#	**No. Nodes/Edges**	**MacBook integrated graphics**	**Computer with dedicated GPU**
1	200 Nodes / 200 Edges	246ms	243ms
2	1000 Nodes / 1000 Edges	1.3s	856ms

Table 6: Speed Test

We can infer from the above tests that the goal was reached with no problem. Even at a thousand nodes on a laptop without dedicated graphics the nodes load in 1.3s

2.6.4 Screenshots

Figure 38: Screenshot 1

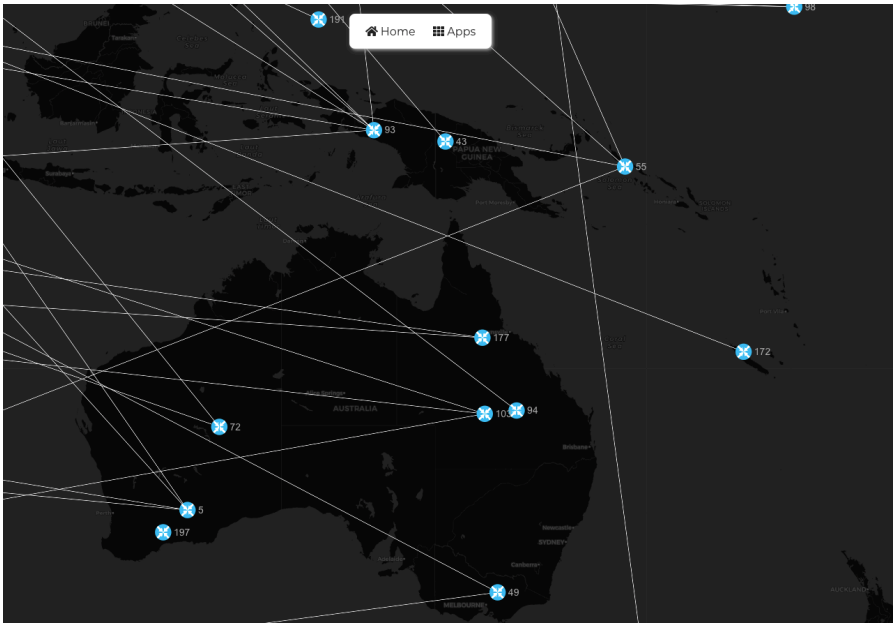


Figure 39: Screenshot 2

Figure 40: Screenshot 3

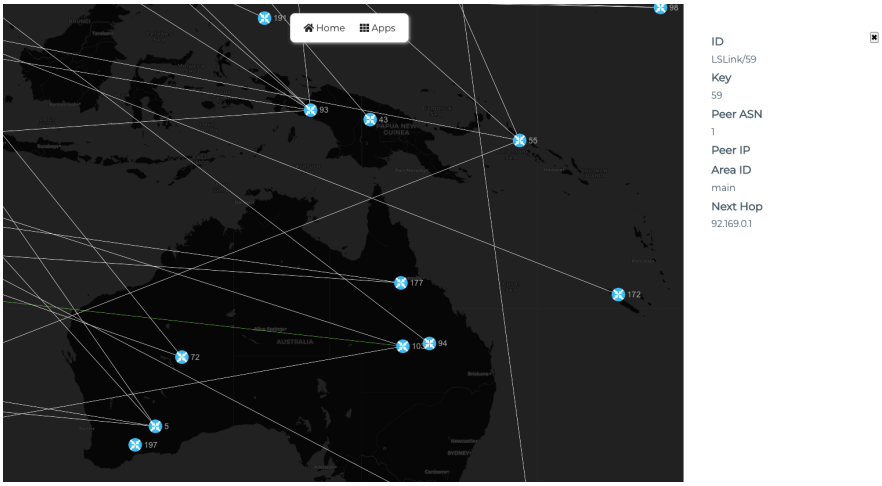


Figure 41: Screenshot 4

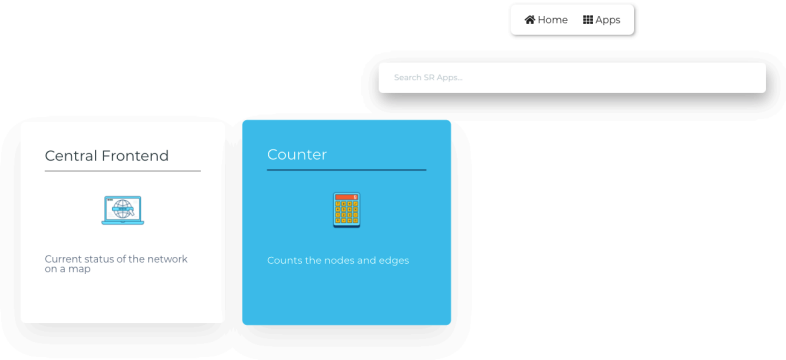


Figure 42: Screenshot 5

Figure 43: Screenshot 6

2.7 Conclusions

2.7.1 Outlook

Clustering

The focus in our bachelor thesis will be to think about the clustering algorithm as it is not ideal right now because of aforementioned reasons. A concept will be thoroughly worked out and all the learnings will be included.

SR Apps List

As we have chosen to implement the SR Apps List the static way in this thesis, we will have to re-think the approach in the bachelor thesis and solve the issue in a dynamic way.

Microfrontend approach One idea would be to create a micro frontend for each SR application and to register it in a backend where the Central Frontend can create routes to the specific SR app frontend.

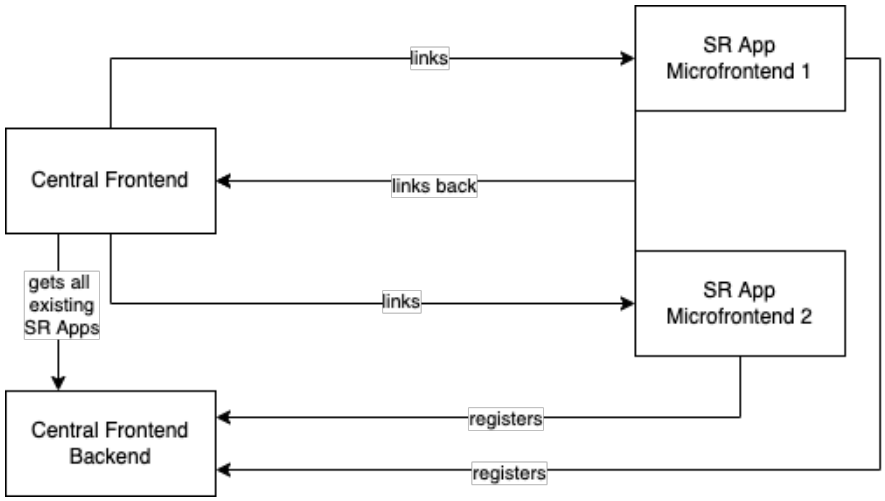


Figure 44: Microfrontend

In this approach we would need to extend the Central Frontend with a dedicated backend. In addition, as the Central Frontend would be nothing else than the first micro frontend, we would need to abstract the whole components list into an own library which all of the micro frontends can use.

Advantages of this approach would be:

- All components would be deployed separately and could scale on their own
- Unified look and feel from the components library
- Dynamic approach

Disadvantages would be:

- Complexity of the system increases

- Each SR Apps development team would need to create their own UI, still adding overhead
- The development team might need to extend components library

Frontend service approach Another idea would be to create a frontend service which would automate the idea from above. An SR App developer would give a configuration file that contains the wanted components and the service would spawn a micro frontend.

The components would still need to be extracted into a library and the SR app developers would need to extend it probably more than the first approach.

One of the disadvantages here is, that we are not sure if this is technically solvable and we should look at this during the elaboration phase of the bachelor thesis.

3 Project Documentation

3.1 Project Planning

3.1.1 Purpose

This section contains all the planning activities of the project. The project plan describes how the project will be executed, monitored, controlled, and finished.

Among other things, it breaks down the project flow, analyses the possible risks, gives an overview of the working methods, infrastructure, architecture and quality assurance.

3.1.2 Organization

People

Team The team consists of the following people:

- Davor Gajic
- Leonard Obernhuber

Role	Person
Product Owner	Davor Gajic
Developer	Davor Gajic, Leonard Obernhuber

Table 7: Roles

Product Owner The product owner leads the sprint review and is responsible for a clean, coherent and updated backlog.

Advisors

- Laurent Metzger
- Bruce McDougall
- Michel Bongard
- Julian Klaiber
- Severin Dellsperger

3.1.3 Roadmap

Dates and Figures

Start: 20.09.2021

Resources: 480h (2Members * 8ECTS * 30h)

Working days: primarily Saturday and Sunday

End: 24.12.2021 17:00

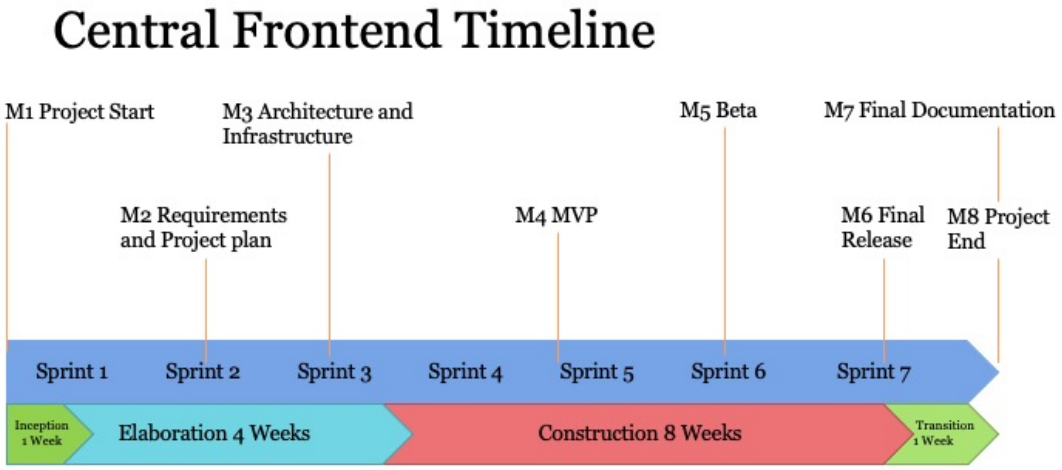


Figure 45: Timeline

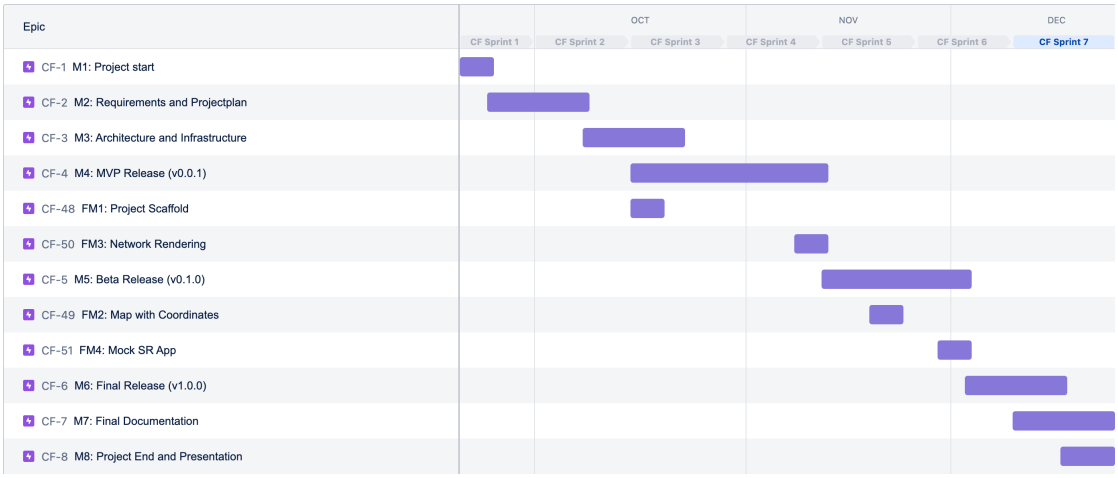


Figure 46: Roadmap

3.1.4 Milestones

There are two different types of milestones.

Phase Milestones Phase milestones are indicated with a prefix **M** and show the different phases at a specific time in the project.

- M1. Project start
  - Task description
  - Setup project tools
  - Administrative tasks
- M2: Requirements and project plan
  - Project plan documented
  - Requirements documented
  - Function and non-functional use cases
  - Use case diagram
  - Risk management
- M3: Architecture and infrastructure
  - Developer machine setup (IDE, tools)
  - CI/CD setup
  - Architecture documented
  - User stories defined for construction phase
  - Wireframes (UI mockups)
- M4: MVP Release (v0.0.1)
  - Functioning application
- M5: Beta Release (v0.1.0)
  - Functioning application
  - Required functionality implemented
- M6: Final Release (v1.0.0)
  - Execute system tests
  - Bug fixing
- M7: Final Documentation
  - Polish the existing documentation
  - Write abstract
  - Proof reading
- M8: Project End and Presentation
  - Prepare and hold presentation

  - Upload all files

Functional Milestones Functional milestones are indicated with a prefix **FM** and show the specific points when a functionality is finished.

- FM1. Project scaffold
  - React project set up
- FM2: Map with coordinates
  - The application can render a map and point to a location given the coordinates
- FM3: Network rendering
  - The application can render/draw a network given the data of the network.
- FM4: Service Programming ported
  - The complete Service Programming UI has been ported to the Central Frontend
  - It is possible to search for other SR apps

3.1.5 Project Management

Our project management is based on the method Scrum+. This combines RUP and Scrum together. This means work is split into Inception, Elaboration, Construction, followed by a short Transition phase, but still work in iterations.

3.1.6 Phases

Phase	**Duration**
Inception	1 week (7%)
Elaboration	4 weeks (29%)
Construction	8 weeks (57%)
Transition	1 week (7%)

Table 8: RUP Phases

3.1.7 Sprints

- Sprint 1 (20.09.2021 - 01.10.2021)
  - Development of the project plan
  - Deploy administrative tools
  - Define rough requirements
- Sprint 2 (01.10.2021 - 15.10.2021)
  - Finish project plan
  - Finish requirements

  - Define rough architecture
- Sprint 3 (15.10.2021 - 29.10.2021)
  - Finish architecture
  - Elaboration of UI design
  - Setup developer machines
  - Scaffold project
  - Start working on MVP
- Sprint 4 (29.10.2021 - 12.11.2021)
  - Implementation of the solution with target: MVP
  - Finish MVP
- Sprint 5 (12.11.2021 - 26.11.2021)
  - Implementation of the solution with target: Beta
- Sprint 6 (26.11.2021 - 10.12.2021)
  - Finish Beta
  - System test
  - Implementation of the solution with target: Final Release
- Sprint 7 (10.12.2021 - 24.12.2021)
  - Preparation and execution of the final presentation
  - Final work on the project and delivery of the project
  - Finish documentation

3.1.8 Meetings

Daily Daily standups are avoided since both project members are working primarily on Saturday and Sunday.

Weekly Meeting The team, the advisor and the internal co examiners(?) meet weekly on Friday 13:00-14:00 if needed. At least all 2 weeks once. The external co examiner will join monthly and then the meeting will be scheduled differently to a US friendly time. This meeting will be logged.

Sprint Retro The team meets after each Review for quick reflection of the sprint. This meeting will be held Friday evening or Saturday morning.

Sprint Review The sprint review meeting is at the end date(usually Fridays) of each sprint at 13:00-14:00. Each sprint is two weeks long. Here the team presents what they have done in the last sprint and answer any open questions.

Sprint Planning The team plans the next sprint and estimates the workloads. This happens right after the sprint retro.

3.1.9 Workloads

Jira All workloads are tracked in Jira: `https://central-frontend.atlassian.net/`

Epics Jira Epics are used as Milestones. To create Scrum Epics, a story with a [Epic] Prefix is created.

Scrum Epics Scrum Epics are a collection of other Stories.

Story A story represents a task that a team member does in the sprint. A story's estimate should never be bigger than a sprint (If it is, the team needs to split it up in the planning).

Each story is made up of:

- Title
- Description
- Estimate in hours
- Tag
- Time Tracking in minutes/hours

Description do not follow a convention. However, each Story must have a Definition of Done.

Definition of Done Definition of Done is used to specify when a Story is done from the perspective of the team.

Tasks Task is a specific task that is part of a story and only used to help the team either estimate or to get an overview. Tasks follow the conventions of the Stories.

3.1.10 Time Tracking

The time tracking is done in Jira with the Timesheets addon. Each team member books the time directly onto the ticket. The time tracking status will be shown each sprint review.

`https://central-frontend.atlassian.net/plugins/servlet/ac/com.tda.timesheet.report/main?project.key=CF&project.id=10000`

3.1.11 Risk Management

Following risks may occur in the project. They are listed in the table below 3.1.11 including damage and mitigation descriptions.

#	Title	Description	Damage in hours	Probability	Weighted damage in hours	Prevention	Behaviour	Mitigation until
R1	API Gateway unstable	The Jalapeno API gateway is unstable	20	25%	5	Use mocked data for development to ensure correctness of handling the received responses.	API calls will not be forwarded correctly which results no or empty responses.	End of Elaboration
R2	Scalability issues	The used framework for displaying the nodes is handling large number of nodes badly	50	20%	10	Do some testing to find the most suitable framework	UI will be unresponsive or data loads very slow (>2s).	End of Elaboration
R3	React unfamiliarity	Getting into React first before implementing may take some time. Also during construction time effort will be higher.	80	50%	40	Do some testing to find the most suitable framework	UI will be unresponsive or data loads very slow (>2s).	End of Elaboration
R4	Bugs in external Libraries	SigmaJS v2 just released and bugs are very likely to appear.	30	90%	27	Risk acceptance	Implementation might be halted, until bug is fixed.	End of Construction

Table 9: Risk Assessment

3.1.12 Infrastructure

Workstation developer Each developer uses his personal machine. The development environments are platform independent.

- React >= 17.0.0
- WebStorm
- LateX (TexPad)

Project Server The INS provided a test server that runs Jalapeno and the Jalapeno API gateway.

GitLab

- Version control with Git
- Time management with linked Jira
- Ticket management with linked Jira
- Release Management via GitLab CI
- Build Management via GitLab CI

CI/CD The project uses GitLab's CI/CD pipelines. GitLab uses Docker containers for this purpose.

Communication General communication will take place via Microsoft Teams. Either with direct calls or messages.

3.1.13 Quality Assurance

To ensure the highest possible quality, the following measures are taken:

Measure	Period	Target
CI/CD	With each merge request	Error prevention, improvement of code quality
Retro	Continuous	Improvement of future sprints
Documentation	Ongoing	Ensuring validity and up-to-dateness
Code reviews	On each merge request	Error prevention, improvement of code quality
Automated testing	Ongoing	Error prevention, code quality improvement
Code Linter	Ongoing	Code style compliance

Table 10: Quality Measures

Documentation The documentation is written in Latex files and stored in the docs repository on GitLab. This way, the documentation is versioned and protected from loss. As with the source code, merge requests are used to ensure a high quality standard. To ensure its correctness, it is continuously updated during the project.

Design decisions Design decisions throughout the project are iteratively documented.

Development The source code is versioned with Git and is stored on GitLab. For each workload package a new feature branch is created. Before being merged back into the master branch, the code is reviewed by another team member.

Unit Testing Unit tests are used judiciously by the developer. Unit tests are written using either Jest or React Testing library and are run on every push with the help of the CI pipelines.

Code Style Guidelines We will be following the Material Design Guidelines. If existing linters exist, they will be used and additionally executed in the CI pipelines.

Testing

Test	Description	Period
Unit	Test are written using either Jest or React Testing library. Individual components.	Started on each push from our CI pipelines.
Integration	Test are written using either Jest or React Testing library. Multiple components.	Launched on every push from our CI pipelines.
System	The system as a whole is tested manually. Test logs needed at end of construction phase.	Performed by hand by the whole team.

Table 11: Quality Measures

3.2 Project Report

Assignee	Time spent
DG Davor Gajic	216h 1m
LO Leonard Obernhuber	165h
Unassigned	106h
	487h 1m

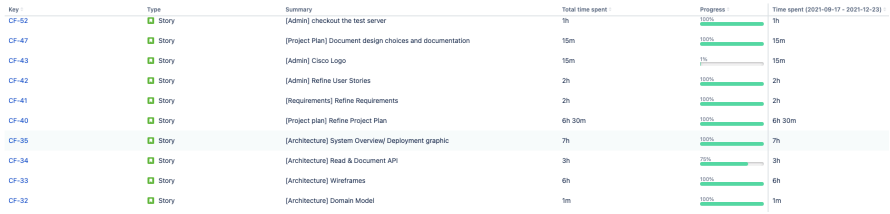
Figure 47: Hours

Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-18	Story	[Requirements] Draw Use Case Diagram	1h	85%	1h
CF-17	Story	[Requirements] Define Actors	1h	100%	1h
CF-16	Story	[Requirements] Define rough non-functional Use Cases	1h	100%	1h
CF-15	Story	[Requirements] Define rough functional Use Cases	1h	100%	1h
CF-14	Story	[Admin] Davor to get familiar with React	6h	85%	6h
CF-13	Story	[Admin] Leo to get familiar with React	20h	100%	20h
CF-12	Story	[Admin] Davor to watch SR lectures	2h	100%	2h
CF-11	Story	[Admin] Leo to watch SR lectures	2h	100%	2h
CF-10	Story	[Admin] Setup Overleaf	5h	100%	5h
CF-9	Story	[Admin] Setup Jira	3h	75%	3h

Figure 48: Hours 1

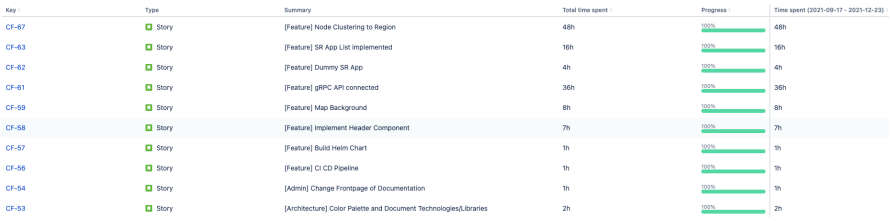
Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-28	Story	[Admin] Trash Overleaf, use Texpad and GitLab	2h	66%	2h
CF-27	Story	[Admin] Leo get familiar with Latex	1h	100%	1h
CF-26	Story	[Admin] Davor get familiar with Latex	1h 30m	100%	1h 30m
CF-25	Story	[Project Plan] Risk Management	2h	50%	2h
CF-24	Story	[Project Plan] Infrastructure & Quality Assurance	2h	100%	2h
CF-23	Story	[Project Plan] Project Management	3h	100%	3h
CF-22	Story	[Project Plan] Roadmap & Milestones	5h 30m	100%	5h 30m
CF-21	Story	[Requirements] Define (more detailed) Use Cases	8h	100%	8h
CF-20	Story	[Meeting] Weekly Meeting 1	2h	100%	2h
CF-19	Story	[Meeting] Kickoff Meeting	2h	100%	2h

Figure 49: Hours 2



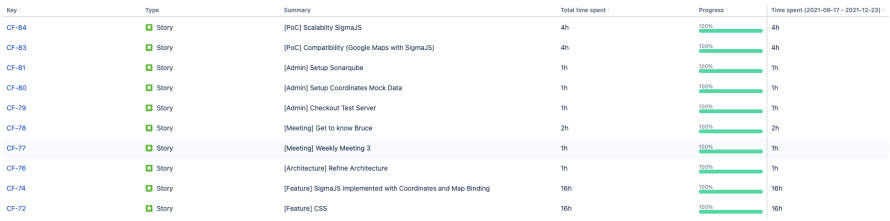
Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-52	Story	[Admin] checkout the test server	1h	100%	1h
CF-47	Story	[Project Plan] Document design choices and documentation	15m	100%	15m
CF-43	Story	[Admin] Cisco Logo	15m	1%	15m
CF-42	Story	[Admin] Refine User Stories	2h	100%	2h
CF-41	Story	[Requirements] Refine Requirements	2h	100%	2h
CF-40	Story	[Project plan] Refine Project Plan	6h 30m	100%	6h 30m
CF-35	Story	[Architecture] System Overview/ Deployment graphic	7h	100%	7h
CF-34	Story	[Architecture] Read & Document API	3h	75%	3h
CF-33	Story	[Architecture] Wireframes	6h	100%	6h
CF-32	Story	[Architecture] Domain Model	1m	100%	1m

Figure 50: Hours 3



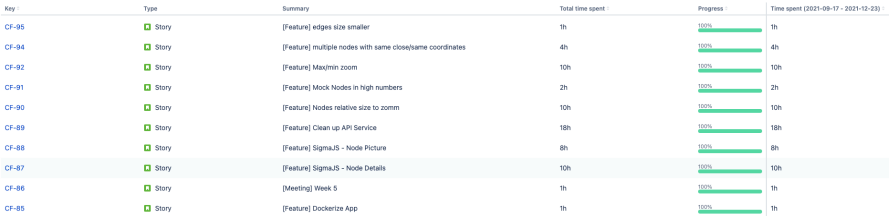
Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-67	Story	[Feature] Node Clustering to Region	48h	100%	48h
CF-63	Story	[Feature] SR App List implemented	16h	100%	16h
CF-62	Story	[Feature] Dummy SR App	4h	100%	4h
CF-61	Story	[Feature] gRPC API connected	36h	100%	36h
CF-59	Story	[Feature] Map Background	8h	100%	8h
CF-58	Story	[Feature] Implement Header Component	7h	100%	7h
CF-57	Story	[Feature] Build Helm Chart	1h	100%	1h
CF-56	Story	[Feature] CI CD Pipeline	1h	100%	1h
CF-54	Story	[Admin] Change Frontpage of Documentation	1h	100%	1h
CF-53	Story	[Architecture] Color Palette and Document Technologies/Libraries	2h	100%	2h

Figure 51: Hours 4



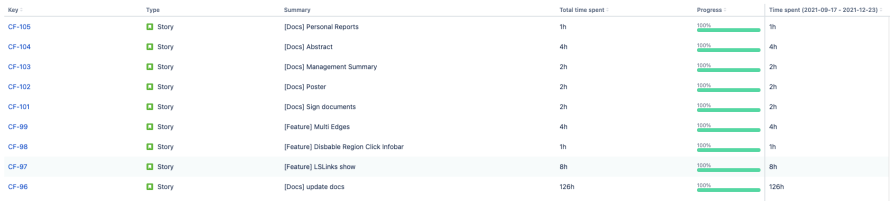
Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-84	Story	[PoC] Scalabilty SigmaJS	4h	100%	4h
CF-83	Story	[PoC] Compatibility (Google Maps with SigmaJS)	4h	100%	4h
CF-81	Story	[Admin] Setup Sonarqube	1h	100%	1h
CF-80	Story	[Admin] Setup Coordinates Mock Data	1h	100%	1h
CF-79	Story	[Admin] Checkout Test Server	1h	100%	1h
CF-78	Story	[Meeting] Get to know Bruce	2h	100%	2h
CF-77	Story	[Meeting] Weekly Meeting 3	1h	100%	1h
CF-76	Story	[Architecture] Refine Architecture	1h	100%	1h
CF-74	Story	[Feature] SigmaJS implemented with Coordinates and Map Binding	16h	100%	16h
CF-72	Story	[Feature] CSS	16h	100%	16h

Figure 52: Hours 5



Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-95	Story	[Feature] edges size smaller	1h	100%	1h
CF-94	Story	[Feature] multiple nodes with same close/same coordinates	4h	100%	4h
CF-92	Story	[Feature] Max/min zoom	10h	100%	10h
CF-91	Story	[Feature] Mock Nodes in high numbers	2h	100%	2h
CF-90	Story	[Feature] Nodes relative size to zomm	10h	100%	10h
CF-89	Story	[Feature] Clean up API Service	18h	100%	18h
CF-88	Story	[Feature] SigmaJS - Node Picture	8h	100%	8h
CF-87	Story	[Feature] SigmaJS - Node Details	10h	100%	10h
CF-86	Story	[Meeting] Week 5	1h	100%	1h
CF-85	Story	[Feature] Dockerize App	1h	100%	1h

Figure 53: Hours 6



Key	Type	Summary	Total time spent	Progress	Time spent (2021-09-17 - 2021-12-23)
CF-105	Story	[Docs] Personal Reports	1h	100%	1h
CF-104	Story	[Docs] Abstract	4h	100%	4h
CF-103	Story	[Docs] Management Summary	2h	100%	2h
CF-102	Story	[Docs] Poster	2h	100%	2h
CF-101	Story	[Docs] Sign documents	2h	100%	2h
CF-99	Story	[Feature] Multi Edges	4h	100%	4h
CF-98	Story	[Feature] Disbable Region Click Infobar	1h	100%	1h
CF-97	Story	[Feature] LSLinks show	8h	100%	8h
CF-96	Story	[Docs] update docs	126h	100%	126h

Figure 54: Hours 7

4 Appendix

4.1 Personal Reports

4.1.1 Leonard Obernhuber

This thesis has been an exciting experience. I learned a lot during those three months. I had the chance to get to know React as well as the sigmaJS visualisation framework. It was demanding to learn all those things in such a small timeframe but the result is something we can build on in the bachelor thesis which was the main goal.

The awareness of the importance of clustering the nodes has shifted a bit late in the project and has consumed a lot more time than initially thought. But it has been interesting figuring out how to best cluster the nodes to get a good overview of the network topology.

Working with sigmaJS was also challenging because there wasn't too much documentation or many examples existing as it has just been released. This happened to be a hurdle in a few milestones on the visualisation. Nevertheless I could profit a lot from this.

I have acquired some frontend software engineering knowledge, especially with CSS and React which has been very beneficial.

4.1.2 Davor Gajic

Before starting this thesis, I have never worked on real frontend or Javascript project. My only experience with frontend development has been the web modules at the Ost school. The opportunity to work with ReactJS was an exciting one.

At the start of the project, a real challenge was to find out the requirements for this thesis. As the wanted requirements were too big for only this term project, we had to scope the wanted requirements properly. This was real valuable experience for me, as scoping a project and figuring out the wanted requirements in iteration with customers is part of the industry as well.

One thing I would do differently in the Elaboration phase, is to actually make an API call with grpc-web. We assumed that the previous UI projects at the INS also used that framework, so we were surprised to learn that we needed the Envoy proxy and a lot of effort was invested into it. From now on, I will not make any assumptions in the Elaboration phase of a project. No matter how simple an API call seems, I would rather confirm it works as expected and reduce the risk that it doesn't work to zero.

Getting used to TypeScipt and React took a bit of time, but in the end it was worth it. If I ever have to build a UI myself in the future, I will use React and TypeScript gladly again.

Leaflet has been a bit of a letdown for me, as there are some bugs and technical limitations that are annoying. One of those bugs, is that there are white line that appear sometimes on Firefox and always on Google Chrome. The technical limitation is the lag while dragging the sigmaJs graph and the map. Leaflet has a bit of a delay and making the experience clunky. We sadly could not find these limitation and bugs during the Elaboration phase, as it did require the connection of Leaflet to sigma and that was part of the Construction phase. In hindsight, I might have leaned more towards google maps, however the issue there still is the potential costs.

SigmaJs still impresses me with how fast it can render 1000 nodes and the decision to use the v2 of sigma in my opinion is the right one. Even though, it made the lives of Leo and me really

hard at times. Sigma v1 has more features and a lot of addons, but seems very outdated and buggy. V2 had some bugs as well and we opened issues at GitHub and they were fixed very fast. I have to say I am impressed by the development team of sigmaJS and it was a pleasure working with their code. While not all wanted features are existing yet, e.g. max and min zoom for the sigma graph, I am confident that in future the decision to go with sigmaJS v2 over something like vis.js will pay off immensely.

The most frustrating part of this thesis was the clustering for me. It was very complex and very time consuming and in the end, the INS wanted it differently. It was a mistake to not specify the requirements of the clustering more granularly. Luckily, we will continue working on the Central Frontend for our bachelor thesis and we will have another look at the clustering topic. The current approach is not the most performant one, as usual with custom algorithms.

I am quite happy with the CSS of the application, as I am not a designer nor am I creative. For this, I have to thank Leo a lot. It was a pleasure working with him.

4.2 Meeting Minutes

4.2.1 Meeting Week 1

Project: Central Frontend
Week: 1
Date/Time: 24.09.2021 / 13:00

Participants

Gajic Davor

Obernhuber Leonard

Metzger Laurent

Bongard Michael

Dellsperger Severin

Klaiber Julian

Agenda

- Kickoff

ToDo

ToDo	Responsibility
Inform if modularity is possible with React apps	Obernhuber, Gajic
Find webGL framework for network topology display	Obernhuber, Gajic
Request access to INS infrastructure for project if needed	Obernhuber, Gajic
Make app cloud native	Obernhuber, Gajic

Table 12: TODOs Meeting W1

Next Meeting

Date: 01.10.21

Time: 13:00

Duration: 1h

4.2.2 Meeting Week 2

Project: Central Frontend
Week: 2
Date/Time: 01.10.2021 / 13:00

Participants

Gajic Davor

Obernhuber Leonard

Bongard Michael

Agenda

- Project Progress
- Sprint Review

ToDo

ToDo	Responsibility
Which cisco logo needs to be used for front page of SA?	Obernhuber, Gajic
State use cases more precisely	Obernhuber, Gajic
Do we need to write the helm charts?	Klaiber

Table 13: TODOs Meeting W2

Next Meeting

Date: 08.10.21

Time: 13:00

Duration: 1h

4.2.3 Meeting Week 3

Project: Central Frontend
Week: 3
Date/Time: 08.10.2021 / 13:00

Participants

Gajic Davor

Obernhuber Leonard

Metzger Laurent

Bongard Michael

Dellsperger Severin

Agenda

- Project Progress
- Use Case Discussion

ToDo

ToDo	Responsibility
Split UC1 into sub use cases and add AS filtering UC	Obernhuber, Gajic
State use cases more precisely	Obernhuber, Gajic
Provide GitLab runner config	Dellsperger

Table 14: TODOs Meeting W3

Next Meeting

Date: 15.10.21

Time: 13:00

Duration: 1h

4.2.4 Meeting Week 4

Project: Central Frontend
Week: 4
Date/Time: 15.10.2021 / 13:00

Participants

Gajic Davor

Obernhuber Leonard

Metzger Laurent

Bongard Michael

Agenda

- Project Progress
- End of elaboration

ToDo

ToDo	Responsibility
Ask sigma.js maintainer if beta version should be used	Obernhuber, Gajic
PoC for compatibility and scalability of sigma.js	Obernhuber, Gajic

Table 15: TODOs Meeting W4

Next Meeting

Date: 22.10.21

Time: 11:00

Duration: 1h

4.2.5 Meeting Week 5

Project: Central Frontend
Week: 5
Date/Time: 22.10.2021 / 11:00

Participants

Gajic Davor

Metzger Laurent

Bongard Michael

Dellsperger Severin

Klaiber Julian

Agenda

- Show progress

ToDo

ToDo	Responsibility

Table 16: TODOs Meeting W5

Next Meeting

Date: 29.10.21

Time: 13:00

Duration: 1h

4.2.6 Meeting Week 6

Project: Central Frontend
Week: 6
Date/Time: 29.10.2021 / 13:00

Participants

Gajic Davor

Leonard Obernhuber

Metzger Laurent

Bongard Michael

Dellsperger Severin

Klaiber Julian

Agenda

- Show progress

ToDo

ToDo	Responsibility
document design decisions	Obernhuber, Gajic
fake coordinate data	Obernhuber, Gajic

Table 17: TODOs Meeting W6

Next Meeting

Date: 05.11.21

Time: 13:00

Duration: 1h

4.2.7 Meeting Week 7

Project: Central Frontend
Week: 7
Date/Time: 05.11.2021 / 13:00

Participants

Gajic Davor

Leonard Obernhuber

Bongard Michael

Dellsperger Severin

Agenda

- Show progress

ToDo

ToDo	Responsibility

Table 18: TODOs Meeting W7

Next Meeting

Date: 12.11.21

Time: 13:00

Duration: 1h

4.2.8 Meeting Week 8

Project: Central Frontend
Week: 8
Date/Time: 12.11.2021 / 13:00

Participants

Gajic Davor

Leonard Obernhuber

Bongard Michael

Agenda

- Show progress

ToDo

ToDo	Responsibility

Table 19: TODOs Meeting W8

Next Meeting

Date: 19.11.21

Time: 13:00

Duration: 1h

4.2.9 Meeting Week 9

Project: Central Frontend
Week: 9
Date/Time: 19.11.2021 / 13:00

Participants

Gajic Davor

Leonard Obernhuber

Bongard Michael

Dellsperger Severin

Agenda

- Show progress

ToDo

ToDo	Responsibility
Address node zooming issues	Obernhuber, Gajic

Table 20: TODOs Meeting W9

Next Meeting

Date: 03.12.21

Time: 13:00

Duration: 1h

4.2.10 Meeting Week 11

Project: Central Frontend
Week: 11
Date/Time: 03.12.2021 / 13:00

Participants

Gajic Davor

Leonard Obernhuber

Metzger Laurent

Bongard Michael

Dellsperger Severin

Agenda

- Discuss clustering approach
- Show current implementation of clustering

ToDo

ToDo	Responsibility
nodes still too big, check out autoRescale option	Obernhuber, Gajic

Table 21: TODOs Meeting W11

Next Meeting

Date: 10.12.21

Time: 13:00

Duration: 1h

4.2.11 Meeting Week 12

Project: Central Frontend
Week: 12
Date/Time: 10.12.2021 / 13:00

Participants
Gajic Davor

Leonard Obernhuber

Bongard Michael

Dellsperger Severin

Agenda

- Show CSS
- Fixed clustering
- Edge onclick infobar

ToDo

ToDo	Responsibility
Open TODOs for finishing project	Obernhuber, Gajic

Table 22: TODOs Meeting W12

Next Meeting

Date: 17.12.21

Time: 13:00

Duration: 1h

4.2.12 Meeting Week 13

Project: Central Frontend
Week: 13
Date/Time: 17.12.2021 / 13:00

Participants
Gajic Davor

Leonard Obernhuber

Bongard Michael

Dellsperger Severin

Julian Klaiber

Dominique Illi

Agenda

- Show documentation
- Demo for Dominique

ToDo

ToDo	Responsibility
Write down thoughts on how to solve clustering	Obernhuber, Gajic

Table 23: TODOs Meeting W13

Next Meeting

Date: -

Time: -

Duration: -

Eigenständigkeitserklärung

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:
Rapperswil, 24.12.2021

Name, Unterschrift:

Leonard Obernhuber,

Davor Gajic

Einverständniserklärung Publikation auf eprints.hsr.ch

Θ SA
O BA

Titel der Arbeit: Central Frontend

Team: Leonard Obernhuber, Davor Gajic

Betreuer: Laurent Metzger

Wir sind mit der Publikation unserer Arbeit auf eprints.hsr.ch einverstanden, sofern für diese Arbeit keine Geheimhaltungsvereinbarung unterzeichnet wurde.
Nach Bekanntgabe der Note haben wir die Möglichkeit innert 14 Tagen Einsprache zu erheben und das Einverständnis zur Publikation der Arbeit auf eprints.hsr.ch zurückzuziehen. In diesem Falle wird nur der Abstract publiziert.

Rapperswil,

Name(n)	Unterschrift(en)
Leonard Obernhuber, Davor Gajic	

1. Vereinbarung

1.1.1 1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit Central Frontend von Leonard Obernhuber und Davor Gajic unter der Betreuung von Laurent Metzger geregelt.

1.1.2 2. Urheberrecht

Die Urheberrechte stehen der Studentin / dem Studenten zu.

1.1.3 3. Verwendung

Die Ergebnisse der Arbeit dürfen sowohl von der Studentin / dem Studenten, von der OST wie von Cisco nach Abschluss der Arbeit verwendet und weiterentwickelt werden

1.1.4 Beilage/n:

Rapperswil, den 24.12.2021 .....
Die Studentin/der Student

Rapperswil, den..... .....
Der Betreuer / die Betreuerin der Studienarbeit

2. Vereinbarung

Ohne anderslautende Vereinbarungen stehen die Schutzrechte und das Know-how an der Studienarbeit oder Bachelorarbeit (nachfolgend ‚Arbeit' genannt) und an der in diesem Rahmen geschaffenen Güter, wie Software, sowohl dem Rechtsträger der OST Ostschweizer Fachhochschule, dem für die Arbeit verantwortlichen Professoren sowie dem Verfasser der Arbeit resp. Entwickler der in diesem Rahmen geschaffenen Güter, wie Software, zu.

Die genannten Parteien übertragen sich gegenseitig nicht exklusiv, jedoch unentgeltlich, weltweit, sachlich und zeitlich unbeschränkt die jeweiligen Schutzrechte und das Know-how an der Arbeit und an der in diesem Rahmen geschaffenen Güter, wie Software, einschliesslich dem Recht zur Weiterübertragung, ab. Entsprechend steht es jeder Partei zu, sämtliche Schutzrechte an der Arbeit resp. an der in diesem Rahmen geschaffenen Güter, wie Software, beliebig weltweit, zeitlich und sachlich unbeschränkt zu verwerten. Darunter fällt namentlich aber nicht abschliessend das Recht zur Lizenzierung in jeder Art, Umfang und Form, das Recht zur Bearbeitung und damit zur Nutzung z. B. der Software oder Komponenten hiervon als Grundlage eines neuen schutzfähigen Guts. Die Parteien erklären sich gegenseitig den Verzicht auf Namensnennung bei der Verwertung der Schutzrechte und des Know-how durch eine oder mehrere Parteien gemeinsam und stimmen namentlich zu, dass jede Partei allein unter ihrem eigenem Namen die Schutzrechte resp. das Know-how verwertet. Die vorliegende gegenseitige unentgeltliche Übertragung der Schutzrechte resp. des Know-how bezieht sich auch auf Verwertungsarten, welche heute noch nicht bekannt sind.

Rapperswil, den 24.12.2021 .....
Die Studentin/der Student

Rapperswil, den..... .....
Der Betreuer / die Betreuerin der Studienarbeit

Glossary

LsEdge

Is a type which represents an Edge in a Segment Routing network 81

LsLink

Is a type which represents a Link in a Segment Routing network 81

LsNode

Is a type which represents a Node in a Segment Routing network 81

Scrum+

Is a markup language specially suited for scientific documents 54, 81

SR application

Is an application that communicates with a Segment Routing network 6, 81

5 List of Figures

6 List of Tables