TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Hendrik Indrikson 164719IAPB

# UXP DIRECTORY KASUTAJALIIDESELE ANDMEID VISUALISEERIVA TÖÖRIISTA LOOMINE

Bakalaureusetöö

Juhendaja: Inna Švartsman
Magistrikraad

Tallinn 2020

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Karl-Hendrik Indrikson

30.03.2020

# Annotatsioon

Käesoleva töö eesmärgiks on luua tööriist Cybernetica AS *Unified eXchange Platform* (UXP) Directory süsteemi kasutajaliidesele, mis visualiseeriks andmevahetust X-Tee laadses süsteemis. Visualisatsioon toimub kliendirakenduses (*front-end*) ning peab kuvama süsteemi seisu nii võrgustikukaardina (*network graph*), kui soojuskaardi (*heat map*) kujul. Visualisatsioonile lisanduvad kasutaja poolt konfigureeritavad sisendid, mille alusel visualisatsioon kohandub. Nendeks on süsteemi kindel alamhulk, kuvatavate liikmete arv, detailsuse tase, kuvatavad identifikaatorid ning UXP metateenused.

Andmete visualisatsioon peab toimuma kliendirakenduses (*front-end*) ning töötama Angular 8 raamistikuga. Visualisatsioon peab muutuma sõltuvalt sisenditest ja olema võimalikult vähekoormav kliendi arvutile.

Töö kvaliteeti ning nõuetele vastavust kontrollis ettevõtte Cybernetica AS testija. Samuti toimusid ettevõttes iganädalased koosolekud, mis projekti kulgemist ning töö nõudeid mõjutasid.

Töö tulemuseks on UXP Directory kasutajaliidese uus funktsionaalsus, millega UXP liikmete andmevahetust visualiseerida.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 5 peatükki, 9 joonist, 1 tabeli.

# Abstract

Development of a data visualization tool for the UXP Directory user interface

The goal of this thesis is to create a tool for Cybernetica AS' *Unified eXchange Platform* (UXP) Directory product that would visualize data exchange in a system similar to the *X-Road*. The visualization will take place in the client-side application (front-end) and must display the system in both network graph and heat map forms. The visualization will be accompanied by configurable inputs which the displayed result will adapt to. These inputs are a certain observable subset of the system, the number of members displayed, the level of detail, the displayed identifier type and whether UXP metaservices are included in the visualization.

The data visualization must be implemented on the client-side (*front-end*) and be compatible with the Angular 8 framework. The visualization must change according to the user inputs and be as little stressful to the user's computer as possible.

The quality and compliance to the project requirements will be reviewed by a tester from Cybernetica AS. There were also weekly project meetings held that influenced the requirements and course of work for this project.

The result of this thesis is a new functionality to the UXP Directory user interface that displays data exchange between UXP members.

The thesis is in Estonian and contains 35 pages of text, 5 chapters, 9 figures, 1 table.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| API | *Application Programming Interface*, rakendusliides |
| CSS | *Cascading Style Sheets*, programmeerimiskeel |
| REST | *Representational State Transfer*, tarkvara arhitektuur |
| SASS | *Sassy CSS*, programmeerimiskeel |
| SCSS | *Syntactically Awesome Style Sheets*, programmeerimiskeel |
| UXP | *Unified eXchange Platform*, Cybernetica AS andmevahetusplatvorm |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Cybernetica UXP (*Unified eXhange Platform*) on Eesti X-Teele sarnase arhitektuuriga turvalise andmevahetuse platvorm, mida eksporditakse paljude välisriikide valitsustele ning suurematele organisatsioonidele.

UXP Directory on selle süsteemi üks komponent. See on avalikuks kasutuseks mõeldud rakendus, mis kuvab andmeid ühe UXP installatsiooni kohta. Selle abil saab sirvida kõiki UXP liikmeid, nende alamsüsteeme ning pakutavaid teenuseid. UXP Directory peamine eesmärk on paremini näidata süsteemi kui tervikut nii selle haldajatele, liikmetele kui ka huvitatud osapooltele.

Kõikidest andmepäringutest UXP süsteemis jääb maha jälg ning kuigi kõik need andmed on salvestatud, ei eksisteeri hetkel funktsionaalsust nende andmete visualiseerimiseks ega sirvimiseks. Otsitakse lahendust, mis annaks kiire ning arusaadava ülevaate nendest andmetest.

Käesoleva lõputöö eesmärgiks on uuendada UXP Directory kliendirakendust funktsionaalsusega, mis lubaks visualiseerida andmevahetust UXP süsteemis nii võrgustikukaardi kui ka soojuskaardi kujul.

Nii UXP Directory süsteemi kood kui ka käesoleva lõputöö väljund on Cybernetica AS omand ning seda ei avalikustata antud lõputöös terviklikult. Selle asemel kirjeldan põhjalikult kasutatavaid tehnoloogiaid, arendusprotsessi ning toon selgitustega näiteid loodud funktsionaalsuse eripäradest.

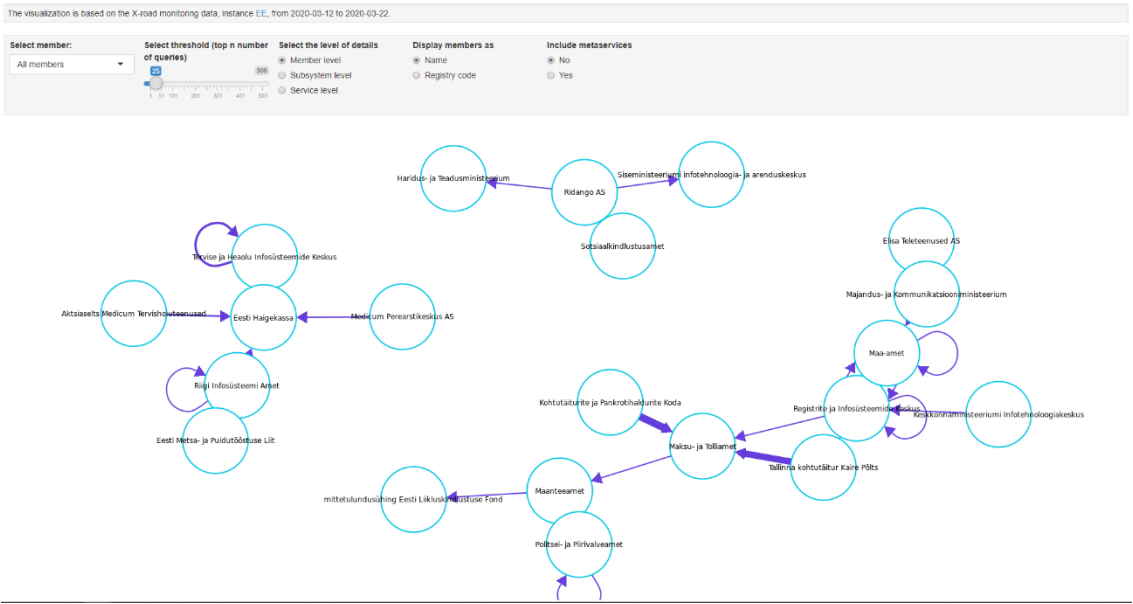
# 2 Taustauuring

Selles peatükis analüüsitakse olemasolevaid lahendusi sarnasele probleemile ning tuuakse välja nende positiivsed ja negatiivsed aspektid.

## 2.1 X-Tee visualiseerija

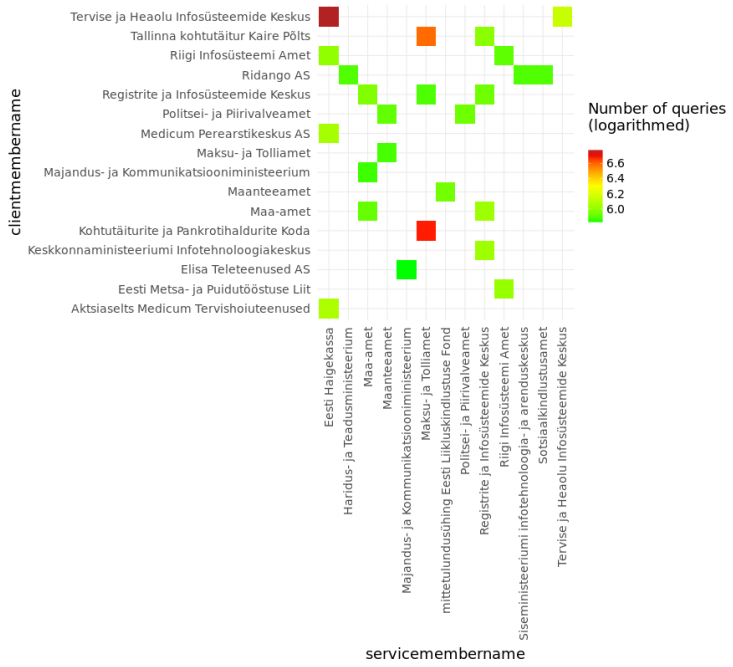
Hetkel eksisteerib sarnane lahendus Eesti X-Tee poolt, mis visualiseerib samuti andmevahetusel salvestatud andmeid ning pakub konfigureeritavaid sisendeid, mille põhjal skeemid muutuvad. X-Tee visualiseerija [1] näitab andmevahetust kõikide X-Tees kaasatud organisatsioonide vahel ning seda samuti võrgustikukaardi ja soojuskaardi kujul.

Joonis 1 näitab X-Tee visualiseerija võrgustikukaardi kasutajaliidest.




Joonis 1. X-Tee visualiseerija võrgustikukaardi kasutajaliides

Joonis 2 näitab X-Tee visualiseerija soojuskaardi kasutajaliidest.




Joonis 2. X-Tee visualiseerija soojuskaardi kasutajaliides

X-Tee visualiseerija on arendatud Tarkvara Tehnoloogia Arenduskeskus OÜ (STACC) poolt ning koostab antud skeemid serverirakenduses (*back-end*). Kliendirakenduses kuvatakse skeemid pildifailina, mida saab alla laadida.

Järgmisena on välja toodud selle lahenduse eelised:

- Väikese liikmete arvu puhul on skeemid lihtsasti loetavad.
- Skeemid annavad hea ülevaate süsteemi suurusest ning organisatsioonidevahelisest infovahetusest.
- Skeemid kuvatakse pildifailina, mida on lihtne alla laadida.

Järgmisena on välja toodud selle lahenduse puudused:

- Skeemid on saadaval ainult pildi kujul ning ei ole interaktiivsed.
- Kui kuvatavate liikmete arvu piisavalt suurendada muutuvad skeemid loetamatuks.

# 3 Analüüs

Selles peatükis kirjeldatakse süsteemi funktsionaalseid ja mittefunktsionaalseid nõudeid ning võrreldakse erinevaid kliendirakenduse teeke, millega antud töö eesmärk täita. Funktsionaalsed nõuded määravad süsteemi täpsed eesmärgid, mis peavad olema realiseeritud. Need defineeritakse kindlate funktsioonide või käitumiste abil. Mittefunktsionaalsed nõuded täpsustavad töökäiku, sätestavad nõuded, mille abil tehtava töö kvaliteeti hinnata ning defineerivad taustategevused, mis ei ole otseselt seotud süsteemi kindla ülesande täitmisega. Lõputöö funktsionaalsed ning mittefunktsionaalsed nõuded sätestati ettevõttes toimunud koosolekute käigus.

## 3.1 Funktsionaalsed nõuded

Järgmisena on välja toodud loodava kliendirakenduse funktsionaalsed nõuded:

- Visualisatsioon peab kuvama süsteemi seisu võrgustikukaardina.
- Võrgustikukaardil kuvatakse UXP liikmeid, alamsüsteeme või teenuseid (edaspidi üksus) mummudena, mille peal on kuvatud selle üksuse identifikaator.
- Võrgustikukaardil on andmeid vahetavad üksused ühendatud joonega.
- Võrgustikukaardil üksusi ühendavate joonte otstes paiknevad nooled, mis näitavad andmepäringu tegemise suunda.
- Võrgustikukaardil kuvatavate üksuste mummu suurus sõltub vahetatud päringute hulgast.
- Võrgustikukaardil üksusi ühendava joone paksus sõltub nende osapoolte vahetatud päringute hulgast.
- Kliendirakendus peab kuvama süsteemi seisu soojuskaardina.
- Soojuskaarti kuvatakse ruudustikuna, mille horisontaal- ja vertikaaltelgedel asuvad infot vahetavate osapoolte identifikaatorid.
- Soojuskaardi ruudustikul märgitakse horisontaal- ja vertikaaltelgede ristumiskohtades vastavate osapoolte andmevahetuse hulk kindla värviga.
- Soojuskaardil oleneb osapoolte ristumiskohas kuvatav värv vahetatud päringute hulgast.
- Kliendirakendusele lisanduvad konfigureeritavad sisendid, milleks on:

    - süsteemi kindel alamhulk (päritakse andmeid vaid kindla UXP liiget puudutava andmevahetuse kohta)
    - kuvatavate liikmete hulk
    - süsteemi detailsuse tase (kuvatakse kas UXP liikmeid, alamsüsteeme või teenuseid)
    - kuvatav identifikaatori tüüp (kuvatakse kas UXP liikme identifikaatorit või seda iseloomustavat nime)
    - võimalus kaasata UXP meta-päringuid.
- Kui kasutaja muudab eelnevalt kirjeldatud sisendeid, siis peab uuenema ka visualisatsioon.

## 3.2 Mittefunktsionaalsed nõuded

Järgmisena on välja toodud loodava kliendirakenduse mittefunktsionaalsed nõuded:

- Kliendirakenduse uus funktsionaalsus peab visuaalselt sobituma senise UXP Directory kliendirakenduse disainiga. Sellest nõudest tulenevad ka järgmised mittefunktsionaalsed nõuded.
- Loodavad sisendid peavad kasutama Angular Material teegi elemente.
- Kasutatavad ikoonid peavad pärinema *Material Design* [2] ressursside hulgast.

## 3.3 Tehnoloogiate taustauuring

Järgnevalt on võrreldud võimalikke kliendirakenduse teeke, millega töö eesmärk täita.

### 3.3.1 D3.js

D3.js on andmetel põhinevate dokumentide manipuleerimiseks loodud JavaScript teek [3]. D3.js on vaadeldud valikutest kõige suurema funktsionaalsusega teek, mille abil saab luua lugematul arvul unikaalseid graafe. D3.js võimaldab realiseerida nii võrgustikukaardi kui soojuskaardi. D3.js on ka kõige suurema õppimiskõveraga teek, mille kasutama õppimine võtaks arvatavasti kuid. Samuti on D3.js rajatud keelele JavaSript ning selle Angularis kasutamiseks tuleks käsitsi defineerida kõik teegis kasutatavad TypeScript tüübid. D3.js oleks parimat kvaliteeti pakkuv lahendus, kuid kuna autor on Angulari arendusega tegelenud vaid paar kuud, siis oleks otstarbekas keskenduda lihtsamini implementeeritavatele valikutele.

### 3.3.2 ngx-charts

ngx-charts on Swimlane poolt loodud graafide teek Angularile, mis pakub sobivat lahendust soojuskaardi realiseerimiseks. ngx-charts on rajatud kasutama D3.js teegi funktsionaalsust. Kuigi teek kasutab elementide kuvamiseks Angulari funktsionaalsust, kasutatakse graafi simuleerimist puudutavate matemaatiliste arvutuste läbiviimiseks D3.js teegi funktsionaalsust [4]. Teek pakub ka graafi spetsiifika kohandamiseks piisavalt funktsionaalsusi ning tundub soojuskaardi implementeerimiseks olema ideaalne valik.

### 3.3.3 ngx-graph

ngx-graph on samuti Swimlane poolt loodud graafide teek Angularile, mille abil saab luua erinevaid visuaalselt informeerivaid graafe. Nagu ngx-charts, on ngx-graph rajatud D3.js teegile ning pakub ka mõnda Swimlane poolt modifitseeritud malli graafide joonistamiseks. ngx-graph võtab enda peale D3.js keerulise ülesehituse ning spetsiifika ja teeb selle muutmise enda poolt pakutavate sisenditega lihtsaks. Samuti on teek defineerinud suure osa D3.js funktsionaalsusest Typescript'is, lihtsustades selle kasutamist Angularis.

### 3.3.4 Sigma.js

Sigma on JavaScript'i teek, mis keskendub graafide joonistamisele [5]. Sigma.js abil oleks antud projektis võimalik realiseerida võrgustikukaarti, kuna see spetsialiseerub üksuste ning nende vaheliste ühenduste kujutamisele. Sigma.js pakub ka palju sisseehitatud interaktiivsuse elemente, milleks on graafi elemendi kohal hiirega peatudes selle elemendi kohta info kuvamine, elemendi peale vajutades vaid selle elemendiga seotud info esile tõstmine ning graafil sisse ja välja suumimine. Sigma.js peamiseks puudujäägiks on see, et see on JavaScript'i teek ning Angularis kasutamiseks peab selle funktsionaalsuse defineerima TypeScript'is. Samuti kuvab Sigma.js üksusi vaid sisendinfo põhiselt ning ei genereeri neile ise asukohti. Seega tuleks asukohtade genereerimise funktsionaalsus implementeerida kas serverirakenduses, või leida teine teek, mis seda Sigma.js eest kliendirakenduses teeks.

### 3.3.5 Kokkuvõte

Järgnevalt esitan eelnevalt mainitud tehnoloogiaid võrdleva tabeli (Tabel 1).

Tabel 1. Visualisatsiooniks kasutatavate teekide võrdlus

| | **D3.js** | **ngx-charts** | **ngx-graph** | **Sigma.js** |
|---|---|---|---|---|
| **Võimaldab realiseerida võrgustikukaarti** | Jah | Ei | Jah | Jah |
| **Võimaldab konfigureerida võrgustikukaarti vastavalt töö nõuetele** | Jah | Ei | Jah | Jah |
| **Võimaldab realiseerida soojuskaarti** | Jah | Jah | Ei | Ei |
| **Võimaldab konfigureerida soojuskaarti vastavalt töö nõuetele** | Jah | Jah | Ei | Ei |
| **Teegi programmeerimiskeel** | JavaScript | TypeScript | TypeScript | JavaScript |
| **Teegi õppimine on autorile varasema kogemusega võrreldes jõukohane** | Ei | Jah | Jah | Jah |
| **Teegi implementatsioon on ajamahukas** | Jah | Ei | Ei | Jah |

Tabelist järeldub, et kõik kirjeldatud teegid on võimelised realiseerima kas võrgustikukaarti või soojuskaarti vastavalt töö funktsionaalsetele ja mittefunktsionaalsetele nõuetele. Antud valikutest pakub kõige rohkem funktsionaalsust ning konfigureeritavust D3.js teek, kuid selle lahenduse kasutama õppimine ja implementeerimine võtaks ka kõige rohkem aega. Et JavaScript'is loodud teeke Angularis kasutada tuleb kõik neis kasutatavad tüübid defineerida ka TypeScript'is. Olles Angulari rakenduste arendusega tegelenud vaid lühikest aega, eelistasin TypeScript teeke. Sigma.js vajab võrgustikukaardi korrektselt kuvamiseks üksuste asukohtade manuaalset genereerimist. Seega on see ainuke teek, mis oma ülesannete täitmiseks lisafunktsionaalsusi vajab. Antud tabeli omaduste põhjal valisin graafide realiseerimiseks ngx-charts ja ngx-graph teegid. Võrgustikukaardi funktsionaalsus implementeeritakse kasutades ngx-graph teeki ning soojuskaardi funktsionaalsus ngx-charts teegi abil.

# 4 Arendus

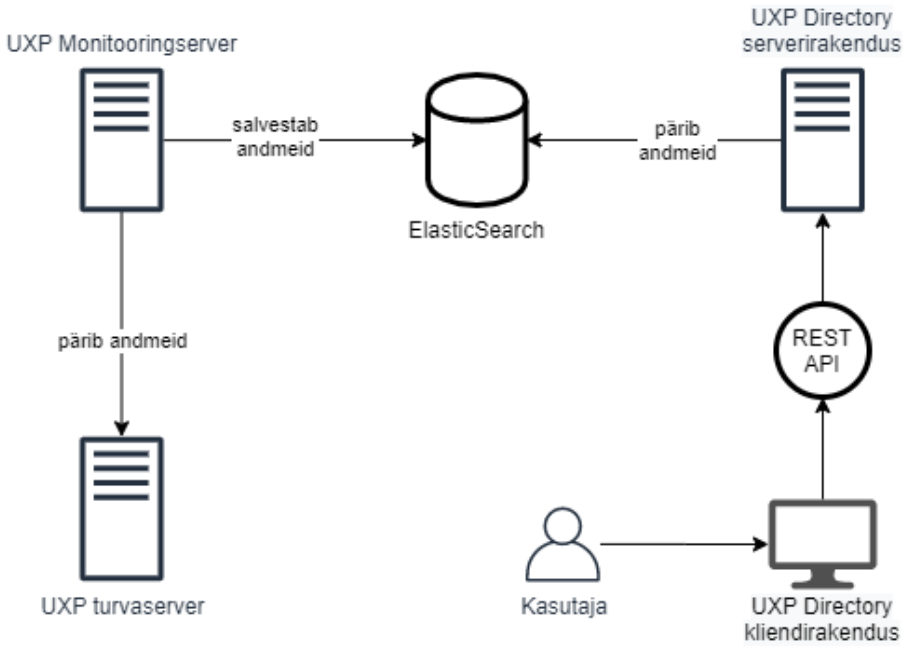
Selles peatükis kirjeldatakse loodava funktsionaalsuse arhitektuuri, kasutatud tehnoloogiaid, arenduskäiku ning arendusest tingitud järeldusi.

## 4.1 UXP Directory Arhitektuur

Cybernetica AS poolt arendatav Unified eXchange Platform (UXP) on süsteem, kus registreeritud liikmed saavad turvaliselt andmeid vahetada. Süsteemiga liitunud organisatsioonid registreeritakse UXP liikmeteks ning saavad deklareerida alamsüsteeme ning teenuseid, mis andmeid jagavad või kasutavad. UXP süsteemis käib andmevahetus läbi turvaserverite. Turvaserverid salvestavad infot UXP liikmete, alamsüsteemide ning teenuste kohta ja kontrollivad andmevahetusel osapoolte ligipääsuõigusi. Turvaserveritelt pärivad kindlate intervallidega andmeid monitooringserverid, mis selle info omakorda ElasticSearch tarkvarasse edastavad.

UXP Directory on süsteem, mis pärib ElasticSearch tarkvarast andmeid ning kuvab infot ühe UXP installatsiooni kohta.

Järgnevalt on kujutatud UXP Directory andmevooskeem (Joonis 3).




Joonis 3. UXP Directory andmevooskeem

## 4.2 Serverirakendus

UXP Directory serverirakendus on loodud keeles Java ning kasutab kliendirakendusega suhtlemiseks JAX-RS ehk *Java API for RESTful Web Services* (Java REST API veebiteenustele) teeki.

Andmed, mille põhjal visualisatsiooni genereerida, pärib UXP Directory kliendirakendus serverirakenduselt, kasutades selleks REST (*Representational State Transfer*) API-t (*Application Programming Interface*). Seejärel pärib serverirakendus ElasticSearch süsteemist uusimad UXP andmed ning genereerib nendest visualisatsioonis kasutatavad andmed. Serverirakendus saadab kliendirakenduse päringule vaste ning kasutades ngx-graph ja ngx-charts teeke genereeritakse nendest visualisatsioonid.

## 4.3 Kliendirakendus

### 4.3.1 Node Package Manager (NPM)

Node Package Manager (NPM) on internetipõhine Node.js projektide hoidla ning haldusvahend, mis abistab projekte teekide ning raamistike installimisega ning versiooni- ja sõltuvuste haldusega [6].

### 4.3.2 Angular 8

Angular on kliendirakenduse disainiraamistik ning arendusplatvorm, mis lihtsustab veebiarendust ning keskendub dünaamiliste *single-page* kliendirakenduste ehitamisele [7]. Angular kasutab rakenduste loomiseks HTMLi ning TypeScript'i ning on kättesaadav läbi NPMi.

Angulari rakenduse arhitektuur on rajatud moodulitele ning komponentidele. Moodulid pakuvad komponentide kompileerimiseks vajalikku konteksti ning komponendid defineerivad vaated, millega rakenduse kasutaja suhtleb. Moodulid on rakenduse alamosad, milles on defineeritud mingi hulk funktsionaalsust, ning mis tihti üksteisest sõltuvad.

Üks Angulari põhimõtteid on *lazy-loading* (laisk laadimine). Selle põhimõtte järgi laetakse mooduleid osade kaupa, ainult siis, kui neid parajasti kasutajale tarvis on. Seega

defineeritakse moodulid võimalikult väikestena, et kliendirakendus peaks alati võimalikult vähe andmeid pärima ja rakendus laeks uusi funktsionaalsuseid kiiremini.

Komponendid on defineeritud TypeScript klassidena, mis sisaldavad nii andmeid kui talitusloogikat, ning on seotud HTML malliga, mida veebilehel kuvatakse. Mallid koosnevad nii tava HTML koodist, kui Angulari spetsiifilisest süntaksist, mis võib HTML elemente enne nende kuvamist muuta. Mallides saab HTML elemente siduda komponendi andmete ja loogikaga. Nii vahetatakse komponendi ja vaate vahel andmeid. Komponendi mallis kasutatakse ka viiteid teistele Angulari komponentidele, mille rakendus reaalajas vastavate elementidega asendab. Üks Angulari eripärasusi on ka see, et komponendid saavad läbi malli oma alam- või ülemkomponentidega andmeid vahetada. Tänu sellele ei pea igas komponendis uuesti samu andmeid pärima.

Angulari rakenduse arhitektuuris on olulisel kohal ka teenused (*services*). Kogu loogika, mis ei ole seotud spetsiifilise vaatega ning mida tahetakse jagada komponentide vahel, defineeritakse teenusena. Komponendid defineerivad neile vajalikud teenused ning rakendus süstib (*inject*) need komponendi külge, kui need vaatesse tulevad. Teenused on staatilised ning lasevad komponentidel vahetada infot ja välja kutsuda keerulisemat või taaskasutatavat loogikat.

Komponendis saab ka defineerida vaid sellele komponendile rakenduvad stiilid. Komponendile võib lisada SASS (*Syntactically Awesome Style Sheets*), SCSS (*Sassy CSS*) või tava CSS (*Cascading Style Sheets*) stiilikirjeid, mis kompileeritakse rakenduse ehitamisel CSS-ks ning paigutatakse omakorda JavaScript'i failidesse. Angulari rakendusel on nii globaalne stiilifail, mille kirjeid automaatselt kuvatakse, kui ka komponendipõhised stiilid, mis lisatakse dokumendile vaid selleks ajaks, kui komponent on vaates.

### 4.3.3 TypeScript

TypeScript on JavaScriptile rajatud uue süntaksiga programmeerimiskeel, mida kasutatakse veebiarenduses ning mis kompileerub JavaScript'iks [8]. TypeScript on Angularis eelistatud programmeerimiskeel ning JavaScript'iga võrreldes seisneb TypeScript'i eripära selles, et kõik arenduses kasutatavad muutujad on defineeritud kindlate tüüpidega. See lihtsustab suurte projektide haldamist ning kindlustab rakenduses vahetatud andmete vastavust defineeritule. TypeScript on avatud lähtekoodiga tarkvara,

millega saab luua ka JavaScript rakendusi nii serveri- kui kliendirakendustele ning seda haldab Microsoft.

### 4.3.4 SCSS

SCSS (*Sassy CSS*) on kirjeldav programmeerimiskeel, mis lisab veebilehel kuvatavatele HTML elementidele nende kujunduse. SCSS on laiendus tüüpilisele CSS keelele, mis pakub tavalisest keelest rohkem funktsionaalsusi ning mugavust. Kuigi SCSS ja CSS on sarnased, ei saa SCSS-i veebilehel kuvada. Selleks tuleb see CSS-ks kompileerida. Kuna SCSS on CSS keele laiendus, siis on kõik CSS kood korrektne SCSS kood, kuid mitte vastupidi [9].

Järgnevalt loetlen peamised SCSS lisafunktsionaalsused võrreldes CSS-ga:

- SCSS süntaks lubab stiilielemendi sisse lisada ka teisi stiilielemente. Nii saab lihtsamini defineerida spetsiifilisi stiilijuhte.
- SCSS lubab teisi SCSS faile importida.
- SCSS lubab stiiliatribuute muutujatesse salvestada ning neid teistes stiilifailides kasutada. Nii saab näiteks kliendirakendusele luua lihtsa värviskeemi, millele elementide värvimisel viidata.

Angular projekti ehitades kompileeritakse SCSS CSS-ks ning kuigi projektis defineeritud globaalset stiilifaili hoitakse eraldiseisva CSS failina, salvestatakse komponendipõhised SCSS stiilielemendid JavaScript failidesse, mis seotakse veebilehega vaid selleks perioodiks kui komponent on vaates.

Sarnane alternatiiv SCSS-le on SASS (*Syntactically Awesome Style Sheets*). SASS lubab samuti stiilielemente üksteise sisse lisada, kuid selle süntaks erineb SCSS-st ja CSS-st selle võrra, et loogsulgude asemel viidatakse elementide konteksti algusele ja lõpule kindla arvu tühikutega, sarnaselt Python'ile. SCSS ja SASS koodi saab mõlemat pidi ümber kompileerida ning Angular lubab kasutada nii SCSS, SASS kui ka CSS keeli.

### 4.3.5 NGX-Translate

NGX-Translate on Angularile loodud teek [10], mis lihtsustab kliendirakenduses tekstide tõlkimist. NGX-Translate abil saab defineerida JSON formaadis tõlkefailid, milles seotakse tõlgitav tekst kindla võtmega. Seejärel asendatakse Angulari komponendi mallis

staatilised tekstilõigud samade võtmetega ning rakenduse töö käigus asendab NGX-Translate teek komponendi vaatesse ilmudes võtmed otsesteks väärtusteks.

Iga tõlgitava keele jaoks koostatakse uus JSON fail, kuhu samade võtmetega uued tõlked salvestatakse.

Kui kasutajaliideses toimub keele vahetus, laeb teek uue tõlkefaili ning asendab kõik tõlgitavad tekstid veebilehel uute väärtustega.

### 4.3.6 Swimlane ngx-charts

ngx-charts on Swimlane poolt arendatav avatud lähtekoodiga teek, mis on ehitatud D3.js teegi peale ning pakub graafide visualiseerimise funktsionaalsust. ngx-charts abil on võimalik visualiseerida suurel hulgal unikaalseid graafe, mis on teegi abil konfigureeritavad. ngx-charts on loodud Angulari raamistiku jaoks ning on kättesaadav läbi Node Package Manager'i.

### 4.3.7 Swimlane ngx-graph

ngx-graph on Swimlane poolt arendatav avatud lähtekoodiga teek, mis on ehitatud D3.js teegi peale ning pakub graafide visualiseerimise funktsionaalsust. Teek keskendub informatiivsete graafide visualiseerimisele ning pakub nende konfigureerimiseks hulganisti sisendeid. ngx-graph on loodud Angulari raamistiku jaoks ning on kättesaadav läbi Node Package Manager'i.

## 4.4 Arendusprotsess

### 4.4.1 Kliendirakenduse täiendamine

Kliendirakenduse uus funktsionaalsus koosneb kolmest Angulari komponendist:

- Võrgustikukaardi komponent (NetworkMapComponent), milles kuvatakse võrgustikukaardi visualisatsioon.
- Soojuskaardi komponent (HeatMapComponent), milles kuvatakse soojuskaardi visualisatsioon.
- Visualisatsiooni vaheleht (VisualizationToolComponent), milles kuvatakse terve visualisatsioonirakendus. Antud komponendis kuvatakse ka visualisatsiooni komponendid ja kasutaja poolt konfigureeritavad sisendid.

UXP Directory Angulari projektis kasutatakse Angulari moodulite laadimiseks *lazy-loading* [11] põhimõtet. Kuna visualisatsiooniga seotud komponendid on ülejäänud kliendirakendusega võrreldes mahukamad ning vajavad kasutaja arvutilt visualisatsiooni genereerimiseks suuremat võimsust kui teised komponendid, lõin neile uue mooduli InstanceModule, mida Angular vaid uuele vahelehele navigeerides laeb. Selle mooduli peamiseks vaateks on visualisatsiooni vahelehe komponent ning vaheleht hakkab asuma uuel Angulari suunal (*route*) */visualization*.

Järgmiseks lõin visualisatsiooni vahelehe komponendi (VisualizationToolComponent), lisasin selle InstanceModule mooduli deklareeritud komponentide nimekirja ning määrasin ta mooduli peamiseks suunaks (*route*). Seega kuvatakse antud komponent peale InstanceModule mooduli laadimist automaatselt. Seejärel koostan komponendis kuvatava HTML koodi ning sean üles komponendisisese infoliikumise.

Komponendis kuvatakse viite kasutaja poolt konfigureeritavat sisendit. Nendeks on:

- *Select* tüüpi sisend, kust saab valida kindla süsteemis oleva UXP liikme, et visualisatsioon vaid sellega relevantset infot kuvaks.
- Horisontaalne kerimisriba, mille abil saab valida visualisatsioonis kuvatavate UXP liikmete arvu. Mida suurem on sisendi väärtus, seda rohkem liikmeid ja nendevahelisi seoseid kuvatakse.
- Kolme valikuga sisendkast, milles määratakse, kas visualisatsioon kuvab UXP liikmeid, alamsüsteeme või teenuseid.
- Kahe valikuga sisendkast, milles määratakse, kas visualisatsioonis kuvatakse liikmete kohal nende UXP siseseid nimesid või teenuskoode.
- Kahe valikuga sisendkast, milles määratakse, kas visualisatsioonis kajastatakse ka UXP meta-teenuseid.

Kõik antud sisendid on implementeeritud kasutades Angular Material [12] elemente.

Iga sisendi HTML elemendile lisasin Angulari *change* (muutus) sündmuse, mis tuvastab, kui sisendi väärtus muutub. Sel juhul saadetakse serverirakendusele uus päring, kus on kajastatud sisendite muutus. Vaste saades uuendatakse visualisatsioone, kusjuures antud päringu vaste sisaldab nii võrgustikukaardi kui soojuskaardi kuvamiseks vajalikku informatsiooni. Seega uuenevad ühe päringu tulemusena mõlemad visualisatsioonid.

Järgmiseks lõin konteineri, milles visualisatsioone kuvama hakatakse, ning sisendi, kus saab valida võrgustikukaardi ja soojuskaardi kuvamise vahel. See sisend on seotud muutujaga, mis kuvab antud ajahetkel kas võrgustikukaardi või soojuskaardi komponendi.

Seejärel lõin võrgustikukaardi ja soojuskaardi komponendid ning asetasin nad eelnevalt kirjeldatud konteineri sisse. Nende külge lisasin Angulari ngIf teegi, mis kuvab antud elementi ainult siis, kui teegi taga olev väide vastab tõele. ngIf teek seotakse eelnevalt kirjeldatud sisendiga ning selle väärtusest olenevalt kuvab Angular vaid hetkel valitud visualisatsiooni. Samuti lisasin mõlemad loodud komponendid visualisatsiooni moodulis deklareeritud komponentide alla, et Angular neid korrektselt laadida oskaks.

### 4.4.2 Võrgustikukaardi implementeerimine

Antud projektis on võrgustikukaardi visualisatsioon implementeeritud kasutades ngx-graph teeki. ngx-graph defineeritakse koodis samanimelise HTML elemendi abil. Elemendile antakse sisenditena kaasa visualisatsiooni graafi tipud (*nodes*) ning servad (*edges*).

Järgnevalt on kujutatud võrgustikukaardi andmemudel TypeScript liidestena (Joonis 4).

```
export interface GraphData {
  nodes: GraphNode[];
  edges: GraphEdge[];
}

export interface GraphNode {
  id: string;
  label: string;
}

export interface GraphEdge {
  id: string;
  source: string;
  target: string;
  count: number;
  size: number;
}
```

Joonis 4. Võrgustikukaardi andmemudel TypeScript’is

Liides (*interface*) GraphData (graafi andmed) on kogum graafi tippudest (*nodes*) ning servadest (*edges*). Väli *nodes* on kuvatavate tippude massiiv (*array*) ning väli *edges* servade massiiv.

Liides GraphNode (graafi tipp) iseloomustab graafi tippusid. Väli id on väärtus, mis määratakse visualisatsioonis kuvatava HTML elemendi id-ks ning label on väärtus, mis kuvatakse visualisatsioonis tipu kohale, et element oleks tuvastatav.

Liides GraphEdge (graafi serv) iseloomustab graafi servi. Väli id on väärtus, mis määratakse visualisatsioonis kuvatava HTML elemendi id-ks. Väljad *source* (algus) ja *target* (sihtpunkt) viitavad serva alguses ja lõpus paiknevate tippude id-dele. Nende abil tuvastab ngx-graph teek vastavad tipud ning liigutab serva nende asukohtade järgi.

ngx-graph pakub ka paljusid teisi sisendeid, millest projektis on kasutatud järgmiseid:

- draggingEnabled – *boolean* (*true/false*) väärtus, mille abil otsustab teek kas kasutaja saab graafil hiirega tõmmates tippe liigutada või mitte. Ettevõttes toimunud koosolekute käigus otsustati see funktsionaalsus välja jätta, kuna tippe liigutades hakkaks kliendi arvuti uuesti graafi simuleerima ning see võib suure arvu elementide puhul kliendi arvuti võimsust suuresti mõjutada.
- animate – *boolean* (*true/false*) väärtus, mille abil otsustab visualisatsioon, kas animatsioone kasutada või mitte. Kliendi arvuti võimsuse kokkuhoiu mõttes otsustasin animatsioonid ära jätta.
- layout – ngx-graph Layout objekt tüüpi väärtus, mis juhib graafi simuleerimist tuginedes erinevatele objekti atribuutidele. Leidsin, et antud projekti jaoks on ngx-graph tavapärane Layout ebasobiv selle mitme omapärasuse tõttu. Järgnevalt toon välja tavapärase Layout objekti ebasobivad omadused:
    - Graafi simuleerides kasutatakse tehis-füüsikajõudusid, nagu gravitatsioon ning põrkamine, et tippe ning servi üksteisest sõltuvalt liikuma panna. Tahtsin neid jõudusid muuta, et graafi oma nägemuse järgi modifitseerida.
    - Tahtsin muuta servade pikkusi ning nende alg- ja lõppkoordinaatide arvutamist. Tavakonfiguratsiooni kohaselt proovitakse kõiki servi sama pikaks venitada, kuid juhul, kui tippude endi suurused ei ole staatilised, peaks serva pikkus tippude suurustest sõltuma. Samuti tõmmatakse serv tavakonfiguratsiooni põhiselt ühe tipu keskpunktist teise tipu keskpunkti, kuid kuna soovin serva lõppu lisada noole, mis näitab andmepäringu tegemise suunda, peab serva lõpp-punkt asuma tipu ääre vastas.

- Tahtsin servadele lisada erijuhu situatsiooniks, kus serva alg-ja lõpp-punkt on samad. Sellisel juhul kuvataks tavalise joone asemel kaar tipust iseendasse.
- Tahtsin kuvada vaid visualisatsiooni lõppseisu, jättes vahele visualisatsiooni animeerimise. Vastasel juhul võiks suure kuvatavate elementide arvu korral visualisatsiooni simuleerimine kasutaja arvutile liialt koormav olla.

Enda poolt modifitseeritava Layout tüüpi objekti loomiseks lõin TypeScript faili modified-force-layout.ts ning kasutasin selle malliks ngx-graph poolt kasutatavat D3-force [13] Layout objekti. Lisasin võrgustikukaardi komponendi viite, et ngx-graph teek kasutaks simulatsioonis antud objekti.

ngx-graph kasutab graafi simuleerimiseks D3.js teeki. D3.js asetab kõik simulatsioonis kuvatavad tipud algselt vaatevälja keskpunkti ning rakendab neil erinevaid jõudusid, et genereerida graafi lõpp-pilt. Antud jõududest on tähtsaimaks gravitatsioonijõud ning põrkejõud.

Graafile rakendatud gravitatsioonijõud võib elemente kas üksteisest eemale paisata või lähemale tuua. Gravitatsioonijõu funktsioon ning selle tugevus määratakse arvulise väärtusega. Positiivse väärtuse puhul tõmbuvad kuvatavad elemendid kokku ning negatiivse väärtuse puhul paiskuvad üksteisest eemale. Mida suurem on sisestatud väärtus, seda tugevamalt jõudu rakendatakse. Juhul kui väärtus on 0, ei rakendata elementidele gravitatsioonijõudu. ngx-graph teegi poolt kasutatavas d3-force mallis on gravitatsioonijõu vaikeväärtuseks arv -30. Sellise väärtusega rakendatakse graafil jõudu, mis elemendid üksteisest nõrgalt eemale tõukab. Kuna UXP süsteemis on tüüpiliselt palju kuvatavaid liikmeid, mis moodustavad suuremad andmevahetuse klastrid, tuleb graafile rakendada suuremat jõudu. Selleks testisin graafil visuaalselt hinnates erinevaid jõudusid vahemikus -30 kuni -2000. Tulemustest järeldasin, et UXP Directory kontekstis on gravitatsioonijõule sobivaks vahemikuks -500 kuni -800. Väärtused üle -500 rakendasid graafile suure arvu elementide korral liiga väikest jõudu, et suuri elementide kogumeid üksteisest lahku paisata ning väärtused alla -800 rakendasid nii väikse kui suure elementide arvu juures graafile üleliigset jõudu, muutes graafi üldpildi liiga hõredaks. Graafi visuaalse testimise tulemustest sõltuvalt otsustasin kasutada väärtust -600, kuna see andis parima tulemuse nii väikse kui suure arvu elementide korral.

Graafile rakendatud põrkejõud surub põrkuvad elemendid üksteisest eemale. Põrkejõu väärtus jääb vahemikku 0 kuni 1. Väärtuse 1 puhul ei lasta põrkuvatel elementidel üksteise sisse jääda ning väärtuse 0 puhul ei takistata elemente põrkumast. Põrkejõudu rakendatakse elementidele ka kindlas raadiuses nende ümber, millest võib tekkida probleem juhul, kui elementidel ei ole ruumi üksteisest eemalduda. Tüüpiliselt soovitatakse kasutada väärtust, mis jääb antud vahemiku keskele. Sellisel juhul ei takistata põrkumist üleliigselt ning graafi lõpp-pilt näeb välja loomulik. Seega otsustasin põrkejõuks kasutada väärtust 0,5.

Samuti lisasin simulatsioonis kasutatavale Layout objektile funktsiooni, millega arvutada graafi servade pikkused. Serva pikkus peaks sõltuma tema tippude suurustest. Tipu minimaalne suurus on 30 pikslit ning suureneb vastavalt sellele kui palju tipp andmeid vahetab.

Järgnevalt toon välja peamised servade tüübid, mida visualisatsioonis kohatakse.

- Ühe tipu raadius on 30 pikslit. Antud juhul on arvatavasti tegemist graafi lõpp-punktiga, millel on vaid üks serv. Sellised servad tuleks hoida graafi ruumi kokkuhoiuks võimalikult lühikesed, et väiksed tipud paikneksid suuremate tippude lähedal.
- Tegu ei ole esimese juhuga ning kahest tipust suurema raadius on alla 50 piksli. Antud juhul omavad mõlemad tipud eeldatavasti enda küljes graafi lõpp-punkte. Võttes arvesse eelmise juhu, kus graafi lõpp-punktid kuvatakse teisele tippudele võimalikult lähedale, peaks antud juhul tipud üksteisest varasemast kaugemal asetsema, et nende lõpp-punktide paigutamise jaoks oleks piisavalt ruumi.
- Tegu ei ole varasemate juhtudega ning kahest tipust suurema raadius on alla 100 piksli. Antud juhul on tegu vähemalt ühe suure tipuga, mis tuleks ülejäänud tippudest veel kaugemale kuvada. Eeldatavasti on antud tipul palju teisi servi.
- Tegu ei ole ühegi varasema juhuga. Seega on tegu vähemalt ühe väga suure tipuga ning serv peaks veel pikem olema.

Antud probleemi lahendamiseks lõin funktsiooni getDistance, mis arvutab serva pikkuse sõltuvalt tippude suurusest ning nende servade arvust (Lisa 1). Pikkus sõltus kolmest parameetrist: algpunkti raadius, suurim servade arv ning suurima tipu raadius. Esimene nendest on staatiline. Teise parameetri jaoks leitakse, kumb tipp omab rohkem servi, ning

sellest sõltuvalt pikendatakse serva. Kolmanda parameetri jaoks leitakse varasemalt kirjeldatud sobilik tippude raadiuste vahemik ning pikendatakse serva selle võrra. Kusjuures, kui eeldatakse, et tegemist on graafi lõpp-punktiga, ei lisata servale teist parameetrit, vaid serv tehakse võimalikult väike.

Lõpuks lisasin enda Layout tüüpi objektile funktsiooni mapEdge, mis arvutab servadele uue alg- ja lõpp-punkti (Lisa 2). Tüüpiliselt joonistatakse serv ühe tipu keskpunktist teise tipu keskpunkti. Selline lahendus ei rahuldaks käesoleva lõputöö nõudeid, kuna servade lõppudesse tuleb kuvada andmevahetuse suunda iseloomustavad nooled. Juhul kui noolepea kuvatakse tipu keskpunkti, ei oleks ta graafil nähtav. Seega kasutasin trigonomeetria reegleid, et tippude koordinaatide ning raadiuste alusel leida lõpp-punkt, mis asetseb tipu ääre vastas.

Edasi lisasin võrgustikukaardi komponendile tippude ning servade kuvamismallid. Need defineeritakse komponendi HTML mallis ng-template elementidena ning nende järgi kuvatakse iga tipp ning serv, mis rakenduses vaatesse ilmub. Tippudele määratakse mallis enda sisendandmetest sõltuvalt raadius ning nende kohal kuvatav identifikaator. Serva mallis määratakse talle kuju. Juhul, kui serva alg- ja lõppkoordinaatidel asuvad erinevad tipud, tuleb serv kuvada joonena, mille otsa lisatakse ka noolepea kujutis. Vastasel juhul kuvatakse serv ellipsina, millest jääb tipu tagant paistma vaid kaar. Samuti määratakse servale kuvatava joone või ellipsi paksus.

Samuti lisasin tippudele sündmuse, mis tuvastab, kui kasutaja oma hiire antud tipu kohale asetab. Antud juhul peaks visualisatsioonis interaktiivsuse mõttes esile tõstma vaid selle tipuga seotud tipud ning servad. Selleks sidusin vastava sündmuse tuvastaja uue funktsiooniga highlight (Lisa 3). Funktsioon saab sisendiks tipu, mille kohal kiir asetseb, ning filtreerib D3.js funktsionaalsusi kasutades välja kõik selle tipuga seotud servad. Järgmisena leitakse kõik vastavate servade otstes paiknevad tipud. Seejärel alandatakse kõikide tippude ja servade, mida leitud nimekirjas pole, *opacity* (läbipaistvus) atribuudi väärtust oluliselt. Nii jäävad erksalt paistma vaid algse tipuga seotud visualisatsiooni elemendid.

### 4.4.3 Soojuskaardi implementeerimine

Soojuskaart implementeeriti, kasutades ngx-charts teeki, mis on samuti loodud Swimlane poolt. ngx-charts defineeritakse koodis samanimelise HTML elemendi abil. Elemendile antakse sisenditena kaasa visualisatsioonis kuvatavad objektid.

Järgnevalt on kujutatud soojuskaardi andmemudel TypeScript liidestena (Joonis 5).

```
export interface HeatGraphData {
    sources: HeatGraphNode[];
    targetCount: number;
}

export interface HeatGraphNode {
    name: string;
    series: HeatGraphSeries[];
}

export interface HeatGraphSeries {
    name: string;
    value: number;
}
```

Joonis 5. Soojuskaardi andmemudel TypeScript'is

Soojuskaardi liidese *HeatGraphData* (soojuskaardi andmed) väli *sources* (allikad) viitab soojuskaardi horisontaalteljel kuvatavatele üksustele ning väli *targetCount* (sihtide arv) graafi vertikaalteljel paiknevate elementide arvule. Viimast kasutatakse manuaalselt soojuskaardile õige vertikaalkõrguse arvutamiseks, kuna ngx-charts teek seda automaatselt ei tee.

Soojuskaardi visualisatsioon kuvatakse Angulari komponendis HeatMapComponent kasutades ngx-charts teeki. See kuvab sisendandmetel põhinevat ruudustiku. Antud ruudustiku horisontaalteljel asetsevad liidese HeatGraphNode kujul UXP liikmed, alamsüsteemid või teenused. Ruudustiku vertikaalteljel asetsevad kõikide horisontaalteljel kuvatavate üksusega andmeid vahetanud UXP liikmed, alamsüsteemid või teenused liidese HeatGraphSeries kujul. Nii horisontaal- kui vertikaalteljel asetsevad üksused on unikaalsed ning ei kordu ühel teljel mitu korda. Ruudustiku telgedel kuvatakse ka vastavate osapoolte UXP identifikaatorid ning telgede ristumiskohtadel kuvatakse vastavate UXP liikmete, alamsüsteemide või teenuste andmevahetust iseloomustav üksus.

Soojuskaardil kuvatavate üksuste värv oleneb antud osapoolte vahetatud päringute logaritmilisest arvust. Kuvatavad värvitoonid varieeruvad roheliste, kollaste ning punaste toonide vahel, kusjuures väiksele päringute arvule vastavad rohelised toonid ning suurele päringute arvule punased toonid. Kui antud osapooled päringuid vahetanud ei ole, kuvatakse soojuskaardil halli värvi ruut. Elementide värvimise funktsionaalsus on ngx-charts teeki sisse ehitatud ning selle kasutamiseks peab vaid määrama soovitud värvispektri.

Samuti kasutasin ngx-charts teegi poolt sisseehitatud legendi funktsiooni, mis kuvab visualisatsiooni kõrvale antud värviskeemi ning eri värvitoonidele vastavad sisendandmete väärtused. Käesolevas lõputöös kuvatakse legend visualisatsiooni alla.

Kui asetada hiir soojuskaardi mingile elemendile, siis ilmub selle kohale objekti iseloomustav märgis. Selles kuvatakse mõlema infot vahetava osapoole identifikaatorid ning nende poolt vahetatud päringute kogus (Lisa 4). Kuna UXP süsteemis võidakse soojuskaardil kuvada nii UXP liikmeid, alamsüsteeme kui teenuseid, võib antud osapoolte identifikaator olla mitmerealine. Et seda märgisel tasakaalustada, lõin ma märgise genereerimiseks eraldi funktsiooni formatTooltip (Lisa 5). See funktsioon koostab üksuste identifikaatorite põhjal uue mitmerealise teksti, kus igal real kuvatavad identifikaatorid on ümbritsetud õige arvu tühikutega. Antud tekst kuvatakse elemendi märgisel *monospace* tüüpi fondiga, kus iga täht, kaasa arvatud tühik, kuvatakse sama laiusega. Nii näeb üksusi iseloomustav märgis alati korrektne välja.

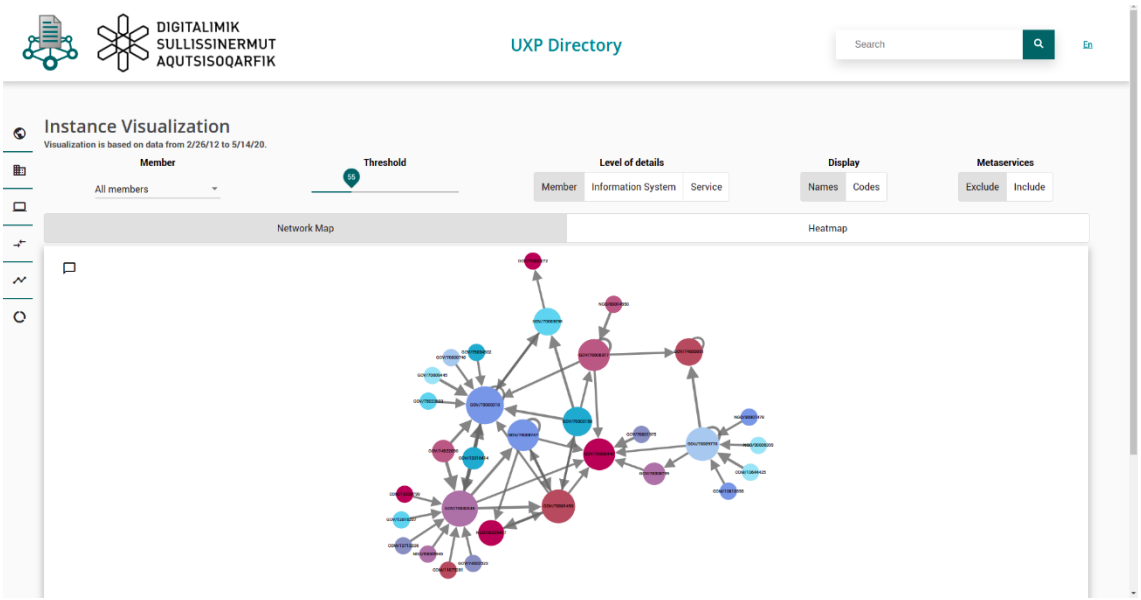
ngx-charts teek defineeritakse staatilise vaateväljaga, mille laius ja kõrgus on kindlalt määratud. See tähendab, et juhul, kui graafil kuvatavate elementide arv muutub, ei muudeta graafi laiust ega kõrgust, vaid surutakse graafil kuvatavaid elemente kokku. Antud juhul võidakse graafil kuvada nii palju elemente, et need kokku surudes tervenisti ära kaovad. Selle parandamiseks lisasin soojuskaardi komponendile sisendi *graphHeight*, mida peale igat visualisatsiooni andmepäringut uuendatakse. See sunnib ka soojuskaarti vastavalt kuvatavate elementide arvule vertikaalselt suurenema või.

## 4.5 Tulemused

Arengukäigu tulemusena valmis UXP Directory kliendirakendusele uus funktsionaalsus, millega saab UXP süsteemi andmeid visualiseerida nii võrgustikukaardi kui soojuskaardi

kujul. Kõik antud peatükis kuvatavad visualisatsioonid kasutavad sisenditeks Eesti X-Tee vabaandmeid, mis on kohandatud vastavalt võrgustikukaardi ning soojuskaardi andmemudelitele (Joonis 4 ja Joonis 5).

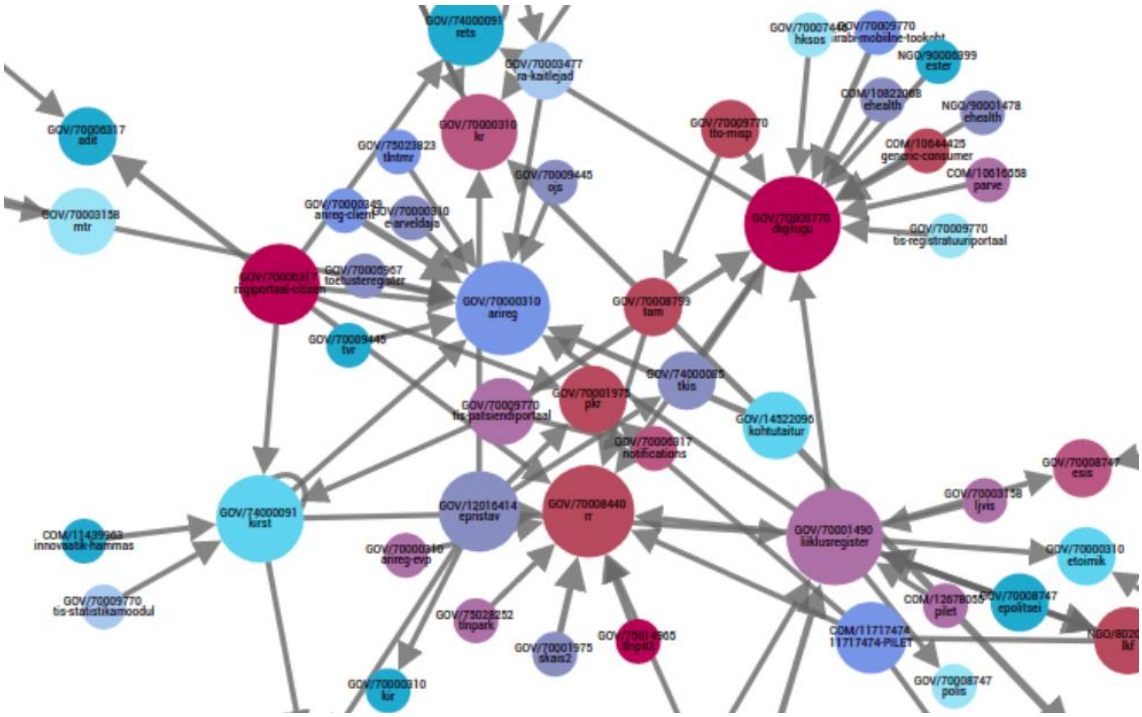
Lisafunktsionaalsuse lõplik vaade veebilehel näeb välja järgnevalt (Joonis 6).




Joonis 6. UXP Directory visualization vaheleht

Joonisel on süsteemi liikmed, alamsüsteemid ning teenused (edaspidi üksused) kuvatud mummudena, mille kohal paikneb nende identifikaator. Üksustevahelised jooned näitavad antud osapoolte andmevahetust ning nool joone otsas andmepäringu tegemise suunda. Et visualiseeritud andmetest paremat ülevaadet saada, tuleks detailsuse taseme sisendiks valida *Information System* (alamsüsteemid) või *Service* (teenused). Nendes vaadetes on üksuse kohal kuvatav identifikaator mitmerealine ning paremini mõistetav.

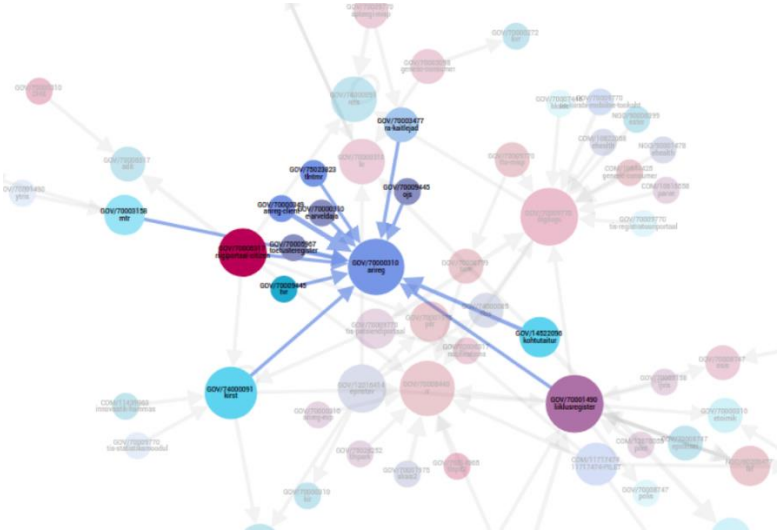
Järgnev joonis (Joonis 7) kuvab võrgustikukaardi visualisatsiooni alamsüsteemide detailsuse tasemega.




Joonis 7. UXP Directory võrgustikukaardi vaade teenuste detailitasemega

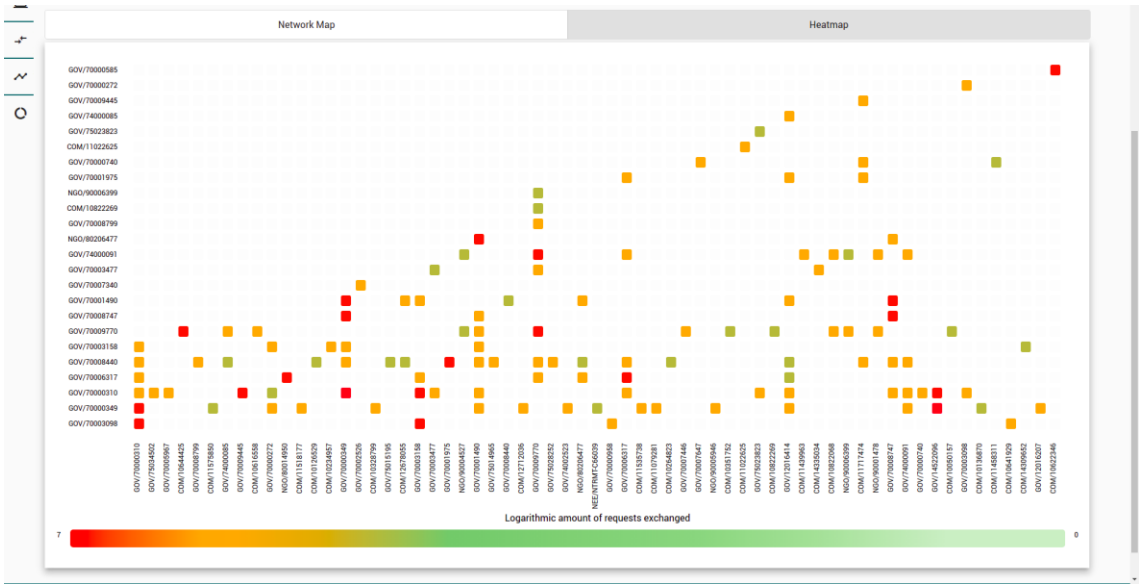
Antud joonisel on näha erinevate X-Tee infosüsteemide omavahelist andmevahetust. Üksuste kohal on kuvatud nii nende liikme identifikaator kui alamsüsteemi identifikaator.

Võrgustikukaardi visualisatsioonile lisasin ka funktsionaalsuse, mis lubab kasutajal näha vaid kindlat üksust puudutavat andmevahetust. Lahendus toimib nii, et kui kasutaja oma hiire mingi üksuse kohale asetab, muudetakse kõik üksused ning ühendused, mis antud üksusega seotud pole, vähem nähtavaks. Antud lahendus näeb välja järgnevalt (Joonis 8).



Joonis 8. Võrgustikukaardi interaktiivsuse näide

Visualisatsiooni soojuskaardi vaade näeb välja järmiselt (Joonis 9).




Joonis 9. UXP Directory soojuskaardi visualisatsioon

Soojuskaardil kuvatavad ruudud viitavad tema horisontaal- ja vertikaaltelgedel asetsevate üksuste andmevahetusele. Üksuse värv oleneb vahetatud andmetepäringute logaritmilisest hulgast. Soojuskaardi all on ka kuvatud värvispektrit iseloomustav legend.

## 4.6 Testimine

Koostatud lahendust testis Cybernetica AS töötaja. Testimisel tuli välja, et võrgustikukaardil kuvatavad servad näevad korrektsed välja vaid siis, kui nad on kindla paksusega. Kuna serva lõpus paiknev noolepea suureneb ja kahaneb serva paksusest sõltuvalt, muutub ta väga peenikese serva puhul pea nähtamatuks ning võib liiga paksu serva korral paisuda suuremaks kui serva lõpus paiknev tipp.

Testimise tagasiside põhjal täiendasin lahendust, lisades serva laiusele miinimum- ja maksimumpiirid.

## 4.7 Järeldused

Kuigi koostatud lahendus vastab kõigile jaotises 3.1 ning jaotises 3.2 kirjeldatud nõuetele, sai arenduse käigus selgeks, et algselt valitud teegid ei olnud antud lahenduse jaoks optimaalsed. Nagu juba analüüsi käigus välja tõin, sobiks antud nõuetega lahenduse jaoks kõige paremini D3.js teek, kuid tuginedes ajalimiitidele ning antud teegiga seotud

teadmiste puudumisele, tegin valiku kasutada ngx-graph ja ngx-charts teeke. Antud teegid kasutavad osa D3.js funktsionaalsusest ning pakuvad seda Angularis. Kuna mu eesmärgiks oli ka muuta spetsiifilisi visualisatsiooni genereerimise parameetreid ning funktsioone, pidin ka rohkesti muutma D3.js teegi spetsiifikat läbi ngx-charts ja ngx-graph teekide. Seda tehes sain aru, et teen rohkem samme ümber valitud teekide, kui neid kasutades. Kuigi antud teegid on D3.js funktsionaalsuse Angularis lihtsasti kättesaadavaks teinud, oleks D3.js otsene kasutamine viinud kvaliteetsema lahenduseni.

Lõputöö vältel õppisin kasutama uusi teeke ning sain teada, kuidas JavaScript teekide spetsiifikat TypeScript keeles defineerida nii, et neid ka Angularis kasutada saaks. Samuti õppisin paremini hindama ülesanneteks kasutatavaid tarkvarasid, sõltuvalt enda oskustest. Õppisin ka palju uut UXP süsteemi toimimise kohta ning värskendasin oma kooliaegseid trigonomeetria ning graafidega seotud matemaatikateadmisi.

# 5 Kokkuvõte

Cybernetica AS *Unified eXhange Platform* (UXP) on Eesti X-Teele sarnase arhitektuuriga turvalise andmevahetuse platvorm, mida eksporditakse paljude välisriikide valitsustele ning suurematele organisatsioonidele.

UXP Directory on antud süsteemi üks komponentidest, millega saab sirvida UXP liikmeid, alamsüsteeme ning teenuseid. Praeguseni ei ole UXP Directory süsteemil olnud funktsionaalsust salvestatud statistikaandmeid visuaalselt kuvada.

Käesoleva lõputöö eesmärgiks oli luua tööriist Cybernetica AS UXP Directory süsteemi kasutajaliidesele, mis visualiseeriks andmevahetust UXP süsteemis. Lahendus pidi visualiseerima süsteemi seisu nii võrgustikukaardi kui soojuskaardi kujul, et anda kiire ülevaade UXP süsteemist nii süsteemi haldajatele, liikmetele kui huvitatud osapooltele.

Eesmärgi saavutamiseks analüüsisin sarnaseid lahendusi, kirjeldasin loodava funktsionaalsuse nõuded, võrdlesin nende põhjal teeke, millega eesmärk täita, ning koostasin lahenduse, mis täidaks kõiki lõputöö nõudeid.

Koostatud lahenduse võrgustikukaardi funktsionaalsus realiseeriti kasutades ngx-graph teeki. ngx-graph on Angularile loodud graafide visualiseerimist võimaldav teek.

Lahenduse soojuskaardi funktsionaalsus realiseeriti kasutades ngx-charts teeki. ngx-charts on Angularile loodud graafide visualiseerimist võimaldav teek.

Lõputöö tulemuseks on UXP Directory kliendirakenduse uus funktsionaalsus, mille abil saab UXP süsteemi seisu visualiseerida nii võrgustikukaardi kui ka soojuskaardi kujul. Lõputöö väljund vastas ettevõtte poolt sätestatud nõuetele ning läbis ka ettevõtte töötaja poolt läbiviidud testimise.

Kasutatud kirjandus


[1] Tarkvara Tehnoloogia Arenduskeskus OÜ (STACC), „X-road members networking visualizaton,“ [Võrgumaterjal]. Available: https://logs.x-tee.ee/visualizer/EE/. [Kasutatud 06 05 2020].

[2] Google, „Material Design,“ [Võrgumaterjal]. Available: https://material.io/. [Kasutatud 05 05 2020].

[3] M. Bostock, „D3.js - Data Driven Documents,“ [Võrgumaterjal]. Available: https://d3js.org/. [Kasutatud 05 05 2020].

[4] Swimlane, „Github - swimlane/ngx-charts,“ [Võrgumaterjal]. Available: https://github.com/swimlane/ngx-charts. [Kasutatud 05 05 2020].

[5] A. Jacomy ja G. Plique, „Sigma js,“ [Võrgumaterjal]. Available: http://sigmajs.org/. [Kasutatud 05 05 2020].

[6] OpenJS Foundation, „Node.js,“ [Võrgumaterjal]. Available: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/. [Kasutatud 05 05 2020].

[7] Google, „Angular - Introduction to Angular,“ [Võrgumaterjal]. Available: https://angular.io/guide/architecture. [Kasutatud 06 05 2020].

[8] Microsoft, „Typescript - JavaScript that scales.,“ [Võrgumaterjal]. Available: https://www.typescriptlang.org/. [Kasutatud 06 05 2020].

[9] H. Catlin, N. Weizenbaum, C. Eppstein ja J. Anne, „Sass: Syntax,“ [Võrgumaterjal]. Available: https://sass-lang.com/documentation/syntax. [Kasutatud 06 05 2020].

[10] NGX-Translate, „NGX-Translate: The i18n library for Angular 2+,“ [Võrgumaterjal]. Available: http://www.ngx-translate.com/. [Kasutatud 17 05 2020].

[11] Google, „Angular - lazy loading ngmodules,“ [Võrgumaterjal]. Available: https://angular.io/guide/lazy-loading-ngmodules. [Kasutatud 14 05 2020].

[12] Google, „Angular Material,“ [Võrgumaterjal]. Available: https://material.angular.io/. [Kasutatud 05 05 2020].

[13] M. Bostock, V. Asturiano, T. MacWright, E. Jewett, S. Haroz, N. Watson, M. Stubbs, A. Velikiy, J. Heer, P. Rivière ja Devinsuit, „Github - d3-force,“ [Võrgumaterjal]. Available: https://github.com/d3/d3-force. [Kasutatud 14 05 2020].

# Lisa 1 – Funktsioon getDistance

```
getDistance(edge) {
  const sourceSize = parseInt(edge.source.size, 10);
  const targetSize = parseInt(edge.target.size, 10);

  const attached = Math.max(parseInt(edge.source.count, 10), parseInt(edge.target.count, 10));
  let attachedPadding = Math.floor(attached / 10) * 50;

  let sizePadding = 0;

  if (sourceSize < 30 || targetSize < 30) {
    sizePadding =  70;
    attachedPadding = 0;
  } else if (Math.max(sourceSize, targetSize) < 50) {
    sizePadding = 150;
  } else if (Math.max(sourceSize, targetSize) < 100) {
    sizePadding = 250;
  } else {
    sizePadding = 350;
  }

  return sourceSize + sizePadding + attachedPadding;
}
```

# Lisa 2 – Funktsioon mapEdge

```
mapEdge(edge) {
  // Get object relevant data
  const x1 = toD3Node(edge.source).x;
  const y1 = toD3Node(edge.source).y;
  const x2 = toD3Node(edge.target).x;
  const y2 = toD3Node(edge.target).y;

  // If link is to self create an arc
  if (edge.source === edge.target) {
    const xRotation = -45;
    const largeArc = 1;
    const sweep = 1;
    const drx = parseInt(edge.source.size, 10) / 2 + 5;
    const dry = parseInt(edge.source.size, 10) / 2;

    return ({
        ...edge,
        curve: `M${x1},${y1}A${drx},${dry} ${xRotation},${largeArc},${sweep} ${x1 + 1},${y1 + 1}`,
        source: toD3Node(edge.source).id,
        target: toD3Node(edge.target).id,
        points: [{ x: x1, y: y1 }, { x: x2, y: y2 }]
    });
  }

  const targetPadding = parseInt(edge.size, 10) * 5 - 0.7;
  const l = Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
  const r1 = parseInt(toD3Node(edge.source).size, 10) - 1;
  const r2 = parseInt(toD3Node(edge.target).size, 10) + targetPadding;

  const sinA = Math.abs(y2 - y1) / l;
  const cosA = Math.abs(x2 - x1) / l;

  // const x1i = x1;
  // const y1i = y1;
  // Links from center of source (above) or links from source edge (below - visible whitespace when dragging).
  const x1i = r1 == null ? x1 :
                x1 + (((x1 < x2 && y1 < y2) || (x1 < x2 && y1 > y2)) ? cosA * r1 : cosA * r1 * -1);
  const y1i = r1 == null ? y1 :
                y1 + (((x1 < x2 && y1 < y2) || (x1 > x2 && y1 < y2)) ? sinA * r1 : sinA * r1 * -1);
  const x2i = r2 == null ? x2 :
                x2 + (((x1 > x2 && y1 < y2) || (x1 > x2 && y1 > y2)) ? cosA * r2 : cosA * r2 * -1);
  const y2i = r2 == null ? y2 :
                y2 + (((x1 > x2 && y1 > y2) || (x1 < x2 && y1 > y2)) ? sinA * r2 : sinA * r2 * -1);

  return ({
      ...edge,
      source: toD3Node(edge.source).id,
      target: toD3Node(edge.target).id,
      points: [{ x: x1i, y: y1i }, { x: x2i, y: y2i }]
  });
}
```

# Lisa 3 – Funktsioon highlight

```
highlight(node) {
  const allNodes = [];
  const connectingNodes = [];
  d3.selectAll('.edge')
    .data(this.graph.edges)
    .filter((d) => {
      if (!allNodes.includes(d.source)) { allNodes.push(d.source); }
      if (!allNodes.includes(d.target)) { allNodes.push(d.target); }
      if (d.source === node.id || d.target === node.id) {
        if (!connectingNodes.includes(d.source)) { connectingNodes.push(d.source); }
        if (!connectingNodes.includes(d.target)) { connectingNodes.push(d.target); }
        return true;
      }
      return false;
    })
    .style('stroke', node.data.color)
    .style('opacity', '1.0')
    .select('.line')
    .attr('marker-end', 'url(' + node.data.color + '-arrow)');
  d3.selectAll('.edge')
    .data(this.graph.edges)
    .filter((d) => {
      return !(d.source === node.id || d.target === node.id);
    })
    .style('opacity', '0.1')
    .select('.line')
    .attr('marker-end', 'url(#inactive-arrow)');
  d3.selectAll('g.node-group')
    .data(this.graph.nodes)
    .filter((d) => {
      return allNodes.filter((e) => !connectingNodes.includes(e)).includes(d.id);
    }).style('opacity', '0.3');
}
```

# Lisa 4 – Soojuskaardi märgise näide

# Lisa 5 – Funktsioon formatTooltip

```
formatTooltip(model): string {
  // Splits both identifiers by rows
  const s1 = model.series.split(/(?:\r\n|\r|\n)/g);
  const s2 = model.name.split(/(?:\r\n|\r|\n)/g);

  // Find amount of rows for both members
  const lv1 = s1.length;
  const lv2 = s2.length;

  // Find longest row of both members
  let lh1 = 0;
  s1.forEach(row => {
    if (row.length > lh1) { lh1 = row.length; }
  });
  let lh2 = 0;
  s2.forEach(row => {
    if (row.length > lh2) { lh2 = row.length; }
  });

  const result: string[] = new Array();

  // Get biggest amount of rows and longest row for padding
  const rows = Math.max(lv1, lv2);
  const maxLength = Math.max(lh1, lh2);

  // Modify and combine each result row by taking (and padding) the current line of Member1 + separator + Member2
  for (let i = 0; i < rows; i++) {
    const p1 = s1.length > i ? this.formatCenter(s1[i], maxLength) : ''.padEnd(maxLength, ' ');
    const p2 = this.getConnector(i, rows);
    const p3 = s2.length > i ? this.formatCenter(s2[i], maxLength) : ''.padEnd(maxLength, ' ');
    result.push(p1 + p2 + p3);
  }

  return result.join('<br>');
}

formatCenter(element: string, padAmount: number): string {
  return element.padStart(element.length + Math.floor((padAmount - element.length) / 2), ' ').padEnd(padAmount, ' ');
}

getConnector(row, maxRows): string {
  return Math.floor(maxRows / 2) === row ? ' --> ' : '     ';
}
```