Universitat de Girona
Escola Politècnica Superior

Projecte fi de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Desenvolupament de noves característiques de visualització de xarxes en el simulador del grup Comunicacions i Sistemes Distribuïts

Document: Memòria

Alumne: Marc Cosgaya Capel

Tutor: Jose Luis Marzo Lazaro
Departament: Arquitectura i Tecnologia de Computadors
Àrea: Informàtica

Convocatòria (mes/any): Juny 2022

Universitat de Girona
Escola Politècnica Superior

Projecte Fi de Grau

Desenvolupament de noves característiques de visualització de xarxes en el simulador del grup Comunicacions i Sistemes Distribuïts

Autor:
Marc Cosgaya Capel

Juny 2022

Grau en Enginyeria Informàtica

Tutors:
Jose Luis Marzo Lazaro

Resum

Al grup de recerca de Comunicacions i Sistemes Distribuïts de la UdG es fa servir l'eina web *Network Research Simulator 2* (*NRS2*) per a l'anàlisi, càlcul de mètriques i visualització de xarxes en forma de graf. Aquest treball se centra en el desenvolupament de noves característiques del visualitzador de xarxes de l'*NRS2*. Això amb l'objectiu de millorar l'experiència de l'usuari i dotar-lo d'eines per a una millor anàlisi dels grafs.

El simulador està programat en *React* i *Node.js*, dues eines de l'entorn de JavaScript. El simulador té tres parts: *frontend*, servidor APP principal i servidors de computació. El primer gestiona la interfície, el segon gestiona les crides HTTP i el tercer executa les computacions de les mètriques. El *frontend* té un visualitzador de grafs. Aquest té 3 modes: 2D, 3D i 2D amb mapa. El Projecte se centra principalment a millorar aquests modes. Els grafs que fa servir el simulador s'obtenen de *The Internet Topology Zoo* i d'*ABM* (abm.cat).

Durant l'elaboració del Projecte s'especifiquen quatre objectius que s'han d'assolir:

- Filtres i coloració: Implementar el sistema de filtres de l'*NRS* en *React.js*. A més, afegir un sistema de coloració de vèrtexs que permetin visualitzar un atribut d'aquests, per exemple la centralitat, en una gradació de colors. Així mateix, un altre mètode de coloració pot ser ressaltar els nodes (vèrtexs) que compleixin una condició. Ara bé, la coloració no només ha de pintar els vèrtexs, sinó que també les arestes. En aquest segon cas és interessant pintar les arestes segons els atributs, si la diferència dels atributs dels nodes es correspon amb la direcció, etc. També és interessant l'opció de representar-ho amb puntets que es mouen a diferents velocitats o amb l'amplada de l'aresta en si.

- 3D Fix: Implementar la visualització 3D amb posicions fixes. El visualitzador en tres dimensions actual només mostra el graf amb un sistema de forces, sense tenir en compte les coordenades reals dels nodes. O sigui, s'ha de poder apreciar la distribució espacial dels nodes. A més a més, s'ha de poder fer servir un *slider* per accentuar més o menys la nova coordenada. Un altre aspecte interessant és la possibilitat de representar com a tercera dimensió qualsevol atribut dels nodes amb un selector i no necessàriament la z. Per exemple, la centralitat de cada node o el seu grau nodal.

- Multicapa: Implementar l'agregació de múltiples grafs en un de sol. Això es pot aconseguir afegint un nou atribut que indiqui el nom del graf inicial del

qual prové el node o aresta. Aquesta característica es pot combinar amb els filtres per poder distingir les múltiples capes i com interactuen entre elles.

- API d'Elevacions: Implementació d'una *API* d'elevacions. Això es deu al fet que la majoria de xarxes no tenen un atribut d'elevació. Aquesta *API* permet obtenir l'elevació d'un node a partir de les seves coordenades de geolocalització, latitud i longitud. Això s'aconsegueix amb un mapa d'alçades.

Per a cada objectiu primer s'analitzen les parts implicades del codi. Després s'analitza quina és la idea principal del que s'ha de desenvolupar. Més tard es desenvolupa una primera versió funcional. Finalment es depura aquesta versió i es van afegint les necessitats que especifica el grup fins que s'obté la versió final.

Al grup de recerca es treballa amb l'eina GitLab per tenir organitzat el desenvolupament del codi, amb un repositori per a cada part del simulador. S'ha fet una branca diferent per a cada objectiu.

Aquest Projecte implica la modificació i creació de nous components de *React.js* al *frontend*. Aquests sempre treballen amb les mateixes dades del servidor APP i només es limiten a canviar la manera en què es visualitzen. També es modifica, en menor grau, el servidor principal. En cap cas es modifica el servidor de computació, que queda fora de l'abast del Projecte.

Agraïments

Vull donar gràcies al grup de Comunicacions i Sistemes Distribuïts per acollir-me durant la meva EEL. Sense ells aquest Projecte no hauria estat possible. Vull agrair en especial a en David Martínez Álvarez, professor associat del departament i membre del grup, que m'ha fet de *senior* durant el Projecte i m'ha ajudat a resoldre dubtes.

Índex

Índex de figures

Capítol 1

Context

1.1 Comunicacions i Sistemes Distribuïts

El grup de recerca de Comunicacions i Sistemes Distribuïts[1][2], d'acrònim *BCDS*, forma part del Departament d'Arquitectura i Tecnologia de Computadors de la Universitat de Girona està ubicat a l'Edifici IV de la Politècnica, a Montilivi.

El grup de recerca s'especialitza en la robustesa de xarxes i en aprenentatge enriquit amb tecnologia. Més enllà de la recerca en xarxes de computadors, també s'hi treballa en l'àmbit de les xarxes d'aigua. Hi formen part professors del departament, estudiants de Grau, Màster i Doctorat. El grup té més de vint anys d'història i s'hi han dut a terme múltiples projectes. Per exemple, en els últims anys s'ha estat col·laborant amb l'Institut Català de Recerca de l'Aigua (ICRA[3]) en el projecte *CLEaN-TOUR* sobre el tractament d'aigües residuals.

Els professors que hi formen part són l'Antonio Bueno, l'Eusebi Calle (cap del grup de recerca), en Josep Lluís Marzo (tutor d'aquest Projecte), en Lluís Fàbrega, en Pere Vilà (cap de Departament), en Ramon Fabregat i en Teo Jové.

Al grup es treballa amb una instància local de *GitLab*[4], un servidor de *Git*[5] que permet treballar en equip.

[1] bcds.udg.edu
[2] www.udg.edu/ca/investiga/la-recerca-a-la-udg/grups-de-recerca/detall?grup=GRCT0040
[3] icra.cat
[4] about.gitlab.com
[5] git-scm.com

1.2 Network Research Simulator

A Comunicacions i Sistemes Distribuïts es fa servir l'eina de visualització *NRS*[6]. Més concretament l'*NRS2*[7], que és una millora basada en *Node.js*[8]. L'*NRS* i l'*NRS2* són simuladors de xarxes. Per tant, serveixen per analitzar, calcular mètriques i visualitzar xarxes en forma de graf. Tanmateix, no tots els grafs són de xarxes de comunicacions, també es treballa amb xarxes d'aigua. De fet, tot el que pugui ser modelat com a graf es pot fer servir en els simuladors.

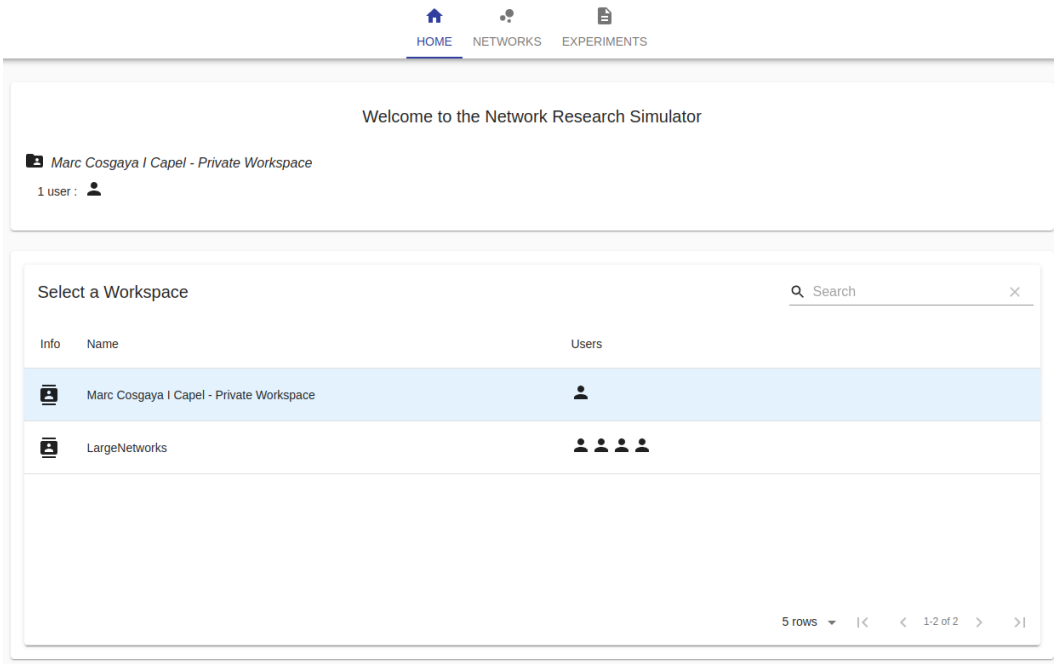

Figura 1.1: Menú principal de l'*NRS2*.

L'*NRS2* s'ha dissenyat modularment amb la finalitat d'assolir escalabilitat. La primera versió és del maig del 2020. El simulador consisteix en tres parts: la interfície gràfica, el servidor NRS APP o principal i els servidors de computació. La figura 1.2 resumeix l'arquitectura.

La interfície gràfica (*frontend*) està programada amb *React*[9], una biblioteca de *JavaScript*, per sobre de *Node.js*. Això permet definir els diferents components per separat i ajuntar-los en una *single-page application*. A més a més, la interfície fa servir llibreries com *material-ui*[10], un *framework* per a components de *React*. El

[6] nrs.udg.edu
[7] nrs2.udg.edu
[8] nodejs.org
[9] reactjs.org
[10] mui.com

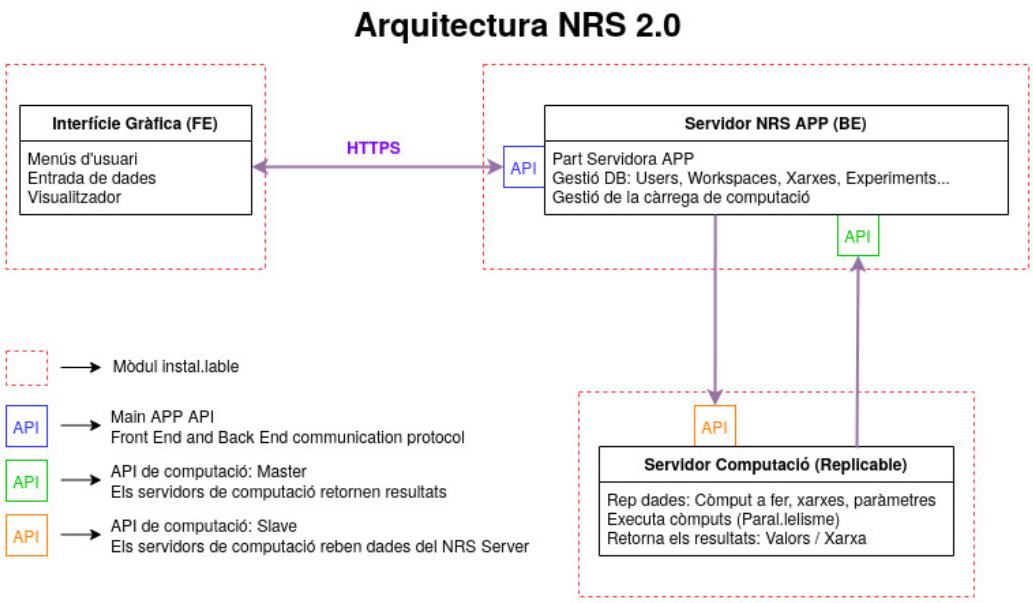

Figura 1.2: Arquitectura de l'*NRS2*.

component en què em centraré en aquest Projecte és el visualitzador de xarxes. Aquest permet visualitzar grafs en 2D, 3D o sobreposats en un mapa.

El servidor principal o NRS APP (*backend*) és el nucli del simulador. Gestiona la lògica de les computacions i serveix les peticions de la interfície gràfica i del servidor de computació. El servidor fa servir *MongoDB*[11], una base de dades *NoSQL* (figura 1.3). Així mateix, els grafs es guarden internament en format *graphml*[12] (variant de *XML*), encara que es poden pujar en altres formats. Per a la gestió de crides *HTTP* utilitza *Express*[13].

Els servidors de computació executen les computacions definides pel nucli. Per exemple, el càlcul de les mètriques d'un graf. Reben els paràmetres i els executen amb *R*. Finalment, retornen el resultat fent una crida *HTTP* al nucli. En la computació es poden afegir nous mòduls d'*R* que, juntament amb la definició ("recepta") en el nucli, permeten fer nous càlculs amb grafs sense haver de modificar el codi font.

[11] mongodb.com
[12] graphml.graphdrawing.org
[13] expressjs.com

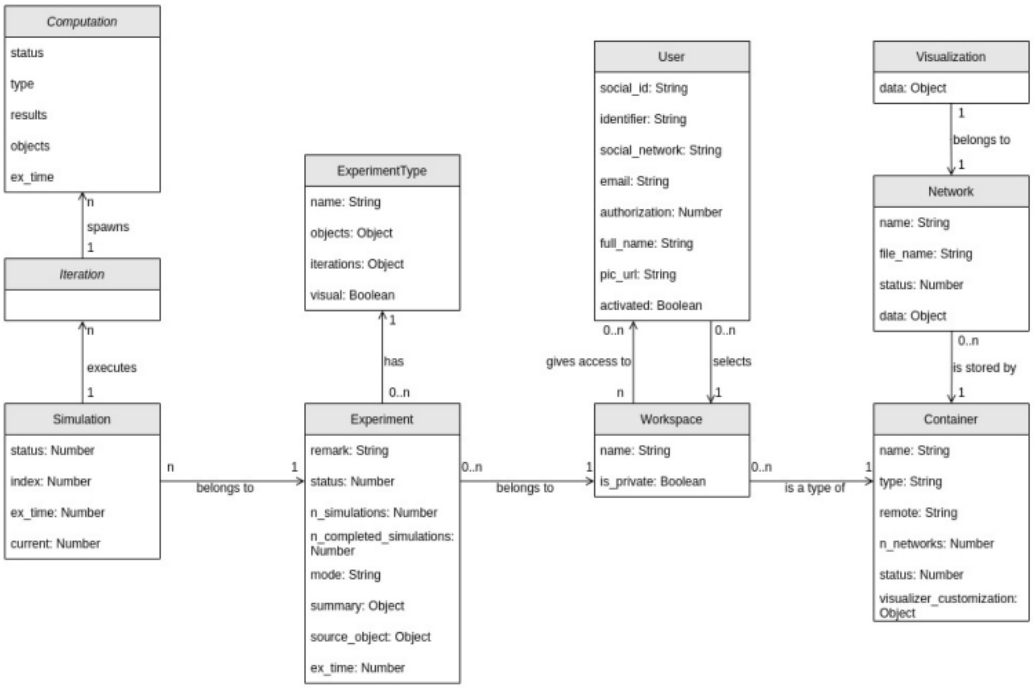

Figura 1.3: Aproximació relacional de com és la base de dades de l'*NRS2*.

Capítol 2

Objectiu

2.1 Motivació

He escollit fer aquest Projecte principalment pel meu interès en la representació de dades sobre mapes. A més a més, aquest m'ha permès combinar la feina anterior amb el grup de recerca, a causa de la meva EEL prèvia amb el grup, amb un treball final més complet. De fet, des que vaig entrar al grup que ja es parlava de millorar el visualitzador de l'*NRS2* perquè era més limitat que el del seu predecessor.

Una altra utilitat del projecte és la millora de la visualització de xarxes complexes, siguin d'aigües o de telecomunicacions. Per això penso que aquest Projecte és una gran oportunitat per millorar la capacitat d'anàlisi de xarxes del grup de recerca.

A més a més, com que ja estava familiaritzat amb *React* he pogut centrar-me en el desenvolupament del Projecte i no tant en aprendre un nou entorn de treball. Això sí, durant la programació de les característiques he hagut de fer petites modificacions al servidor principal, que està programat amb *Nodejs*.

2.2 Objectius

A l'*NRS2* es vol implementar característiques del visualitzador que ja estaven presents en el de l'*NRS*, però que encara no hi són en la nova versió. Així mateix, es vol afegir noves característiques que millorin l'experiència amb el nou simulador.

Durant l'elaboració del Projecte s'haurà de fer modificacions tant en el *frontend* com en el *backend*. El servidor de computació no es modificarà, ja que queda fora de l'abast del que es vol millorar en la visualització.

Es poden distingir diversos objectius.

2.2.1 Filtres i Coloració

Per començar, implementar el sistema de filtres de l'*NRS* en *React.js*. A més, afegir un sistema de coloració de vèrtexs que permetin visualitzar un atribut d'aquests, per exemple la centralitat, en una gradació de colors. Així mateix, un altre mètode de coloració pot ser ressaltar els nodes (vèrtexs) que compleixin una condició.

Ara bé, la coloració no només ha de pintar els vèrtexs, sinó que també les arestes. En aquest segon cas és interessant pintar les arestes segons els atributs, si la diferència dels atributs dels nodes es correspon amb la direcció, etc. També és interessant l'opció de representar-ho amb puntets que es mouen a diferents velocitats o amb l'amplada de l'aresta en si.

2.2.2 3D Fix

Tot seguit, implementar la visualització 3D amb posicions fixes. El visualitzador en tres dimensions actual només mostra el graf amb un sistema de forces, sense tenir en compte les coordenades reals dels nodes. O sigui, s'ha de poder apreciar la distribució espacial dels nodes.

A més a més, s'ha de poder fer servir un *slider* per accentuar més o menys la nova coordenada. Un altre aspecte interessant és la possibilitat de representar com a tercera dimensió qualsevol atribut dels nodes amb un selector i no necessàriament la z. Per exemple, la centralitat de cada node o el seu grau nodal.

2.2.3 Multicapa

En tercer lloc, implementar l'agregació de múltiples grafs en un de sol. Això es pot aconseguir afegint un nou atribut que indiqui el nom del graf inicial del qual prové el node o aresta. Aquesta característica es pot combinar amb els filtres per poder distingir les múltiples capes i com interactuen entre elles.

2.2.4 API d'Elevacions

Finalment, una de les parts en què també m'he hagut de centrar ha sigut la implementació d'una *API* d'elevacions. Això es deu al fet que la majoria de xarxes no tenen un atribut d'elevació. Aquesta *API* permet obtenir l'elevació d'un node a partir de les seves coordenades de geolocalització, latitud i longitud. Això s'aconsegueix amb un mapa d'alçades.

Capítol 3

Metodologia

3.1 Estructura

He volgut assignar una branca del *GitLab* a cadascun dels objectius que he esmentat anteriorment. Utilitzar un sistema de control de versions és útil per tenir un registre de què s'ha fet i quan. Els apartats del desenvolupament de cada objectiu segueixen aquesta estructura:

1. Analitzar les parts implicades del codi. Sobretot, entendre com interactuen entre elles.
2. Determinar quina ha de ser l'essència de la nova característica. És a dir, quina és la idea principal.
3. Desenvolupar les característiques i obtenir una primera versió funcional.
4. Resoldre problemes i decidir com solucionar-los durant el desenvolupament de la versió completa.
5. Un cop el *merge request* està fet, comparar les característiques de la nova versió amb les de la original.
6. Identificar les possibles millores que queden fora de l'abast d'aquest Projecte.

A més a més, durant aquest procés es van fent tutories per acabar d'especificar les necessitats. Això implica tornar a modificar el codi i la depuració de nous *bugs*.

3.2 Conjunts de Dades

En aquest Projecte utilitzo grafs com a conjunts de dades per comprovar el correcte funcionament del Visualitzador. La major part dels grafs per defecte que fa servir l'*NRS2* provenen de *The Internet Topology Zoo*[1][2], un recull de topologies de xarxa

[1] topology-zoo.org

[2] versió arxivada: web.archive.org/web/20220224141613/www.topology-zoo.org/dataset.html

de proveïdors d'Internet d'arreu del món.

A més a més, per a aquest Treball també faig servir el graf de la Xarxa d'Aigües de Girona[3]. Aquesta representa la xarxa de sanejament de Girona amb nodes representant els dipòsits i les arestes representants les canonades

3.3 Eines de Desenvolupament

Durant el Projecte s'utilitzen eines per al desenvolupament de les noves característiques.

Per començar, es fa servir *Visual Studio Code*[4], desenvolupat per *Microsoft*, com a *IDE*[5]. Aquest té suport per a múltiples llenguatges i té un sistema d'extensions[6] molt complet.

Segon, s'utilitza les *DevTools*[7] del navegador Chromium[8], navegador *FOSS*[9] desenvolupat per *Google* que forma la base de *Google Chrome*[10]. S'ha escollit aquest perquè és la base de molts navegadors[11].

Finalment, es fa servir el *CLI*[12] de *Git*[13] juntament amb el *GitLab* del grup, que és on hi ha guardats els repositoris dels components de l'*NRS* i *NRS2*.

[3] Proveït per www.abm.cat
[4] code.visualstudio.com
[5] www.redhat.com/en/topics/middleware/what-is-ide
[6] code.visualstudio.com/docs/editor/extension-marketplace
[7] developer.chrome.com/docs/devtools
[8] www.chromium.org/Home
[9] itsfoss.com/what-is-foss
[10] www.google.com/chrome
[11] en.wikipedia.org/wiki/Chromium_(web_browser)#Browsers_based_on_Chromium
[12] git-scm.com/docs/gitcli
[13] www.git-scm.com

Capítol 4

Planificació

S'ha fet servir un diagrama Gantt per representar la planificació que s'ha seguit durant aquest Projecte. Per a l'elaboració d'aquest diagrama ha sigut útil la utilització del GitLab per veure els *commits* de la branca de cadascun dels objectius. S'ha fet servir l'eina *TeamGantt*[1] per generar el diagrama. S'ha separat els filtres i coloració dels nodes de la coloració de les arestes, encara que tots dos estan a la mateixa branca.

S'ha tingut en compte 4 hores per dia, de dilluns a divendres. Això amb l'excepció de menys de la meitat de la coloració d'arestes i el 3D fix a causa de la simultaneïtat. En aquest cas el temps ha sigut 2 hores per dia per a cada branca. També hi ha l'excepció de l'API d'elevacions, que també ha sigut 2 hores per dia perquè és previ a l'inici del termini del Projecte. Cal tenir en compte que les hores del diagrama són una estimació màxima, ja que durant l'elaboració d'aquest Projecte també s'estaven fent altres projectes del grup. Per tant, l'estimació és de **486 hores**.

[1] www.teamgantt.com

Planificació PFG	**486h**	**100%**
Desenvolupament	**398h**	**100%**
Filtres i Coloració (Nodes)	236	100%
Coloració (Arestes)	40	100%
3D Fix	28	100%
Multicapa	20	100%
API d'Elevacions	74	100%
Memòria	**88h**	**100%**
Memòria	88	100%

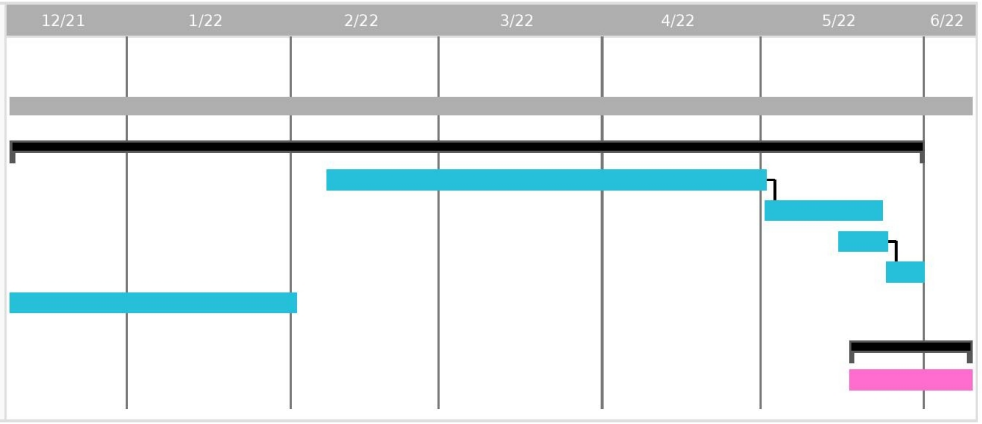

Figura 4.1: Diagrama Gantt de la distribució d'hores del Projecte.

Capítol 5

Elaboració

5.1 Filtres i Coloració

5.1.1 Situació Actual

El component *Visualizer* obté les dades a partir d'una crida al nucli. Aquestes dades són l'objecte *visualization*, que conté informació sobre els nodes, arestes, si el graf és dirigit, les claus (propietats) dels nodes, etc. Aquest objecte és el que després es passa com a paràmetre a cadascun dels modes de visualització.

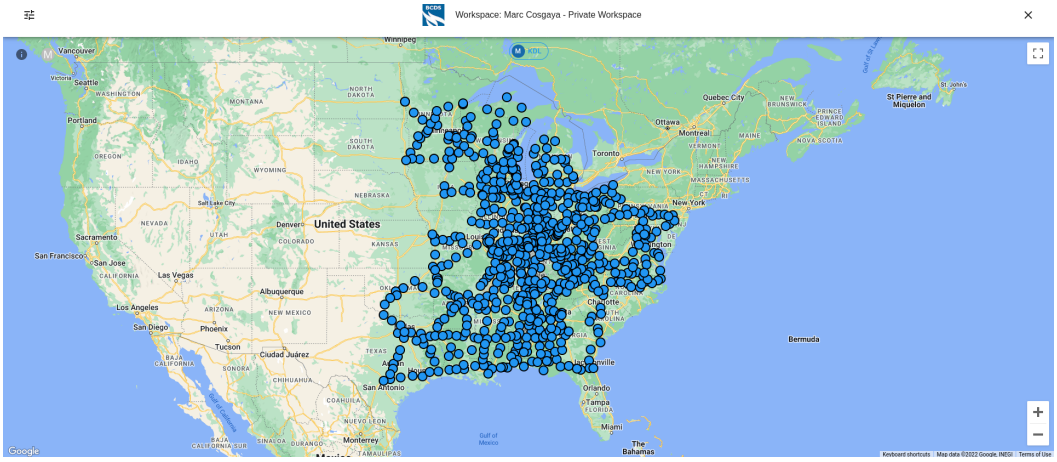

Figura 5.1: *Visualizer* actual.

El *Visualizer* té un altre objecte *customization*, que fa referència a les opcions de personalització de la mida i el color dels nodes i de les arestes. D'aquest objecte no hi faré canvis perquè la nova funcionalitat treballarà ignorant la personalització que hi hagi posat l'usuari.

Cada mode de visualització; 2D, 3D i Mapa; té el seu component; *VisualizeForceMap2D*, *VisualizeForceMap3D* i *VisualizeWorldMap*. A dins de cada un d'aquests components es genera, fent servir *visualization* com a paràmetre, l'objecte *elementsMap*, que és un mapa (clau, valor) que serveix per iterar sobre la informació dels nodes i arestes, tot mantenint l'ordre. Aquest és útil per obtenir ràpidament dades dels nodes a cada mode de visualització.

Es fa servir *react-force-graph*[1] (que fa servir *ThreeJS*[2]), cas 2D i 3D, i la llibreria *google-map-react*[3], cas mapa, per mostrar el graf en el visualitzador. Aquestes llibreries permeten definir components amb opcions de personalització, útils per modificar l'aspecte o el comportament de què s'està mostrant per pantalla.

Finalment, el visualitzador també fa servir altres components com el *VisualizerModeSpeedDial*, el *VisualizerCustomizationDrawer* i el *VisualizerNetworkInfo*. Tots tres conformen els menús i botons que permeten canviar el mode, personalitzar i mostrar la informació de la visualització, respectivament. Les noves funcionalitats les he implementades en l'últim d'aquests components.

5.1.2 Desenvolupament

5.1.2.1 Filtres

La idea principal del filtre és modificar l'objecte *visualization*, treient elements, amb uns criteris. Per aconseguir això, he fet una distinció entre *visualization*, l'objecte per defecte, i *visualizationCurrent*, l'objecte actual. D'aquesta manera s'agafa *visualization* i es genera un nou *visualizationCurrent* a cada nova execució dels filtres. Després cada mode genera el seu *elementsMap* a partir d'aquest nou objecte.

Per començar, he afegit un nou panell a *VisualizerNetworkInfo* pels nodes (figura 5.2). La meva idea inicial era tenir una llista d'*inputs* amb botons per afegir i esborrar, on cada filtre seria un *input* de text del format "mínim <= atribut del node <= màxim". Però veient que parsejar aquest *input* seria complicat, he optat per programar un component *NetworkFilter* amb diversos *inputs* a dins (apèndix A.1).

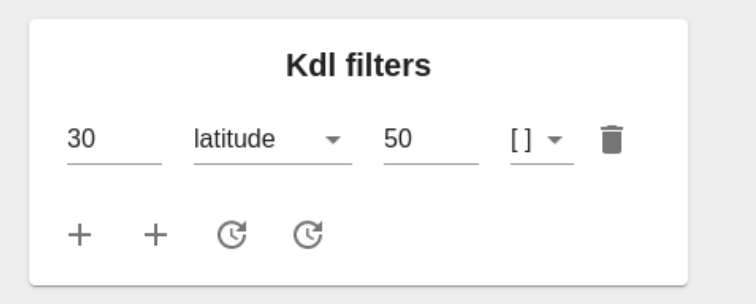

Figura 5.2: El nou menú de filtres.

[1] github.com/vasturiano/react-force-graph
[2] threejs.org
[3] github.com/google-map-react/google-map-react

Tot seguit, he decidit afegir l'opció d'escollir el tipus d'interval del valor de la selecció per a cada filtre: obert, tancat o semiobert. Aquesta opció queda implementada amb un selector. A més a més, he inclòs el tipus igualtat per mostrar elements concrets.

Tercer, he inclòs botons per seleccionar l'operador per agregar múltiples filtres (figura 5.6): *AND*, *OR*, *NAND*, *NOR* i *true*. Aquest últim ignora tots els filtres.

Per acabar, explico les funcions que he hagut d'implementar a *VisualizerNetworkInfo*. La primera és *filtersFromDom*, que genera la llista d'objectes a partir de l'objecte *DOM*, element dins la jerarquia del document *HTML*, de la llista de filtres. Una altra és *changeCurrentVisualization*, que itera sobre tots els nodes de *visualization*, treu els que no compleixen els filtres i actualitza *currentVisualization*. La tercera, *reloadVisualization*, s'encarrega de cridar les dues anteriors quan l'usuari prem el botó de recarregar del menú.

5.1.2.2 Coloració

La idea principal de la coloració és modificar *visualization* actual afegint nous atributs a nodes o arestes que indiquin que aquests s'han de pintar. En aquest cas no es fa servir la visualització per defecte perquè la coloració s'aplica després d'aplicar els filtres. En altres paraules, la coloració modifica el *visualizationCurrent* que han generat prèviament els filtres. D'aquesta manera només es treballa amb els elements que es mostraran i s'evita fer operacions innecessàries. Tanmateix, els atributs propis dels nodes que depenen dels seus veïns, per exemple el grau nodal, no s'alteren.

Per començar, he modificat el panell existent dels nodes perquè permeti la coloració i també he afegit un nou menú per a la coloració de les arestes. Aquest segon utilitza el nou component *EdgeColoration* (apèndix A.2). Encara que també hauria pogut implementar la filtració de les arestes, no vaig trobar la utilitat d'ocultar només les arestes perquè aquestes ja s'oculten quan un dels seus nodes ho fan.

Segon, en el cas dels nodes he fet una distinció entre coloració per gradació, relacionada amb els filtres de tipus interval, i una coloració per ressalt, relacionada amb els filtres de tipus igualtat. Això es deu al fet que he volgut reutilitzar el mateix panell dels filtres per afegir la coloració. Tanmateix, en el cas de les arestes la coloració té dos modes: propietat i direcció. El primer fa una gradació segons una propietat de les arestes. El segon pinta en verd o vermell si la direcció de l'aresta es correspon amb la diferència d'una propietat dels nodes d'aquesta.

Tercer, a la coloració dels nodes he volgut implementar la possibilitat de combinar gradació amb ressalt (figura 5.3). Per tant, he afegit *inputs* de tipus *checkbox* per al ressalt i de tipus *radio* per a la gradació. El primer tipus permet ressaltar múltiples

nodes mentre que el segon permet fer una gradació de només un atribut. S'ha de tenir en compte que fer una doble gradació no té sentit perquè s'haurien de barrejar colors i en complicaria la interpretació. La gradació es guarda com a percentatge del valor de l'atribut respecte al mínim i al màxim, una normalització *min-max*, i el ressalt com a variable binària.

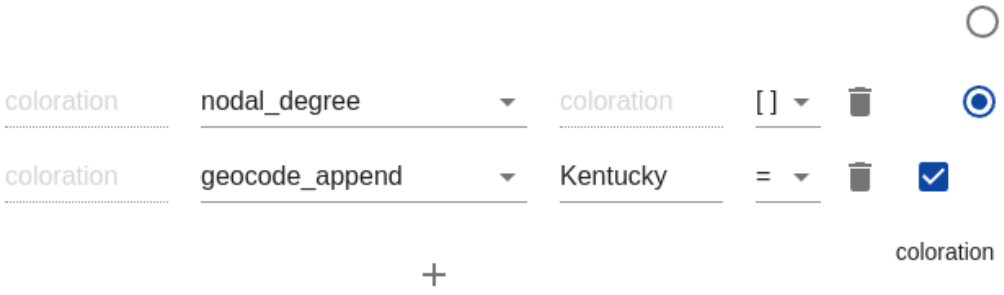

Figura 5.3: Selectors *checkbox* i selectors *radio* a la dreta dels filtres.

Quart, he fet que durant la gradació en fer *hover* en un node es pinti en verd o vermell els nodes veïns que estan per sota (valor menor) o per sobre (valor major), respectivament, del primer (figura 5.4). Això pot ser útil per veure millor les petites diferències que són difícils d'apreciar si hi ha molts valors possibles.

Figura 5.4: Un node veí per sobre i dos per sota.

Cinquè, les funcions principals de *VisualizerNetworkInfo* que implementen la coloració són *handleFilterSelect*, *updateColorationFromIndexes* i *reloadEdgeColoration*. La primera gestiona la selecció de la coloració dels nodes. La segona actualitza els nodes de *visualization* amb els índexs de coloració seleccionats. La tercera actualitza les arestes de la visualització. També he fet servir una funció[4] que converteix

[4] gist.github.com/mlocati/7210513

un percentatge en un color dins d'una gradació entre verd i vermell.

Sisè, a la coloració de les arestes hi he afegit els següents modes:

- Coloració a partir de l'atribut de l'aresta. Com més gran el valor més vermell.
- Amplada de l'aresta a partir del seu atribut. És més ample si el valor és més gran.
- Més o menys punts de flux segons l'atribut. Té més punts i es mouen més ràpidament per un valor més gran.
- Color vermell o verd si els atributs dels nodes origen i destí es corresponen amb el sentit de l'aresta. Aquest mode només està disponible si el graf és dirigit.

També hi he afegit l'opció d'invertir el color, amplada, punts i sentit.

Per acabar, he afegit la possibilitat de bloquejar el menú d'informació per així poder copiar els valors dels atributs (figura 5.5) i enganxar-los en els filtres. Aquesta característica és útil amb els filtres de tipus interval tancat.

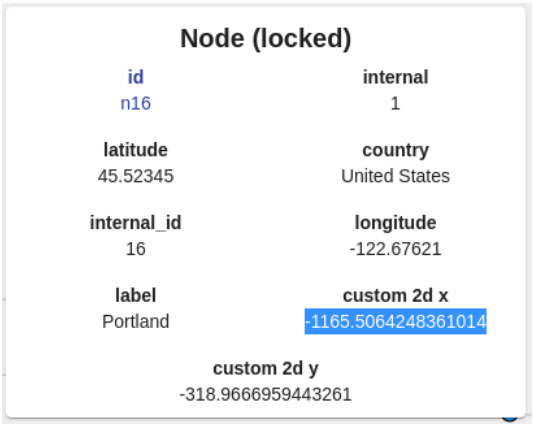

Figura 5.5: Ara els valors del menú es poden seleccionar.

5.1.3 Problemes i Decisions

Primer, per què fer servir un botó que recarrega la visualització? Perquè modificar *visualization* cada vegada que s'escriu un caràcter en els *inputs* o canvia el valor d'un *select* és molt costós, especialment en grafs amb molts elements.

Segon, *filtersFromDom* necessita treballar amb un objecte DOM, passat com a paràmetre, per analitzar els filtres actuals que s'estan mostrant a la pantalla. Això es deu al fet que s'ha separat l'estat del que es mostra. Malauradament, sovint obté

l'objecte incorrecte. Després d'intentar moure'm per l'arbre *DOM* per trobar l'objecte a passar, fet que trenca amb la programació en *React*, vaig descobrir el *hook* *useRef()*. Un *hook* és una eina de *React* que permet "anclar" noves característiques. En aquest cas, *useRef()* permet tenir guardada la referència a un objecte *DOM* en el moment d'inicialitzar el component.

Tercer, He escollit fer servir *visualizationCurrent* per evitar aplicar els filtres a una visualització ja filtrada. Si no fos així, si s'executessin els filtres per a un graf ja filtrat, l'algorisme només faria servir els que hi fossin en aquell moment. Per tant, cada cop hi hauria menys nodes després de cada filtració.

Quart, he tingut problemes per tenir *inputs* amb valors *stateful* (amb estat). El fet de llegir el *DOM* i a la vegada canviar una variable d'estat crea un conflicte. Com que *filtersFromDom* actualitza els filtres, el fet d'haver fet un canvi d'estat prèviament abans de cridar la funció no fa que canviï el valor de l'*input*. Al final he decidit fer servir valors per defecte (propietat *defaultValue*) als atributs en comptes d'agafar els valors directament (propietat *value*).

Un problema inherent a *JavaScript* passa quan es fa un *deep clone* d'un objecte, que es fa amb *{...obj}*. Si la propietat de l'objecte és un objecte o una llista es copia la referència a aquesta. Això m'ha causat problemes amb l'objecte *visualization*, que conté la llista de nodes i la llista d'arestes. Per solucionar-ho, he hagut de fer servir *JSON.parse(JSON.stringify(obj)*, que primer converteix l'objecte a una *string*, que el representa en format *json*, i després genera un nou objecte amb aquesta.

Un altre problema del *JavaScript* és com està implementat *parseFloat()*, que la utilitzo per convertir el valor dels *inputs* a nombres. Aquesta funció converteix una *string* en un nombre decimal. El problema rau en com tracta nombres molt grans o l'infinit ("Infinity"). En aquest segon cas, retorna *Infinity* (de tipus *number*) si la *string* conté la paraula "Infinity" precedida per cap o més espais al principi. Aquest comportament no està explicat a la documentació MDN[5] i he fet una *pull request*[6] al seu repositori per explicar això.

material-ui no té icones que representin portes lògiques per als botons que seleccionen l'operador d'agregació de filtres. Per arreglar-ho he hagut de definir noves icones *svg* (figura 5.6). L'avantatge de fer servir aquest format és que no requereix el gran espai d'emmagatzematge que poden ocupar les imatges convencionals. Ara bé, en aquest cas no importa gaire perquè es tracta d'icones, però és important seguir una bona praxi.

Un altre dilema que he tingut és com representar el ressalt per múltiples atributs. En concret, què passa quan s'ha de ressaltar un node que compleix múltiples condi-

[5] developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat
[6] github.com/mdn/content/pull/16596

Figura 5.6: Operadors d'agregació de filtres.

cions. En un principi tenia pensat representar amb un color diferent cada condició. M'he plantejat representar la combinació de condicions sumant colors, fent un degradat dels dos colors, partir el node en dos, partir l'anella del node, etc. Però al final he decidit no combinar res i mostrar-ho tot amb un sol color. En altres paraules, només representar el ressaltat si una o més condicions es compleixen. Finalment, he fet que el ressalt impliqui accentuar el color, fer-lo més clar (figura 5.7).

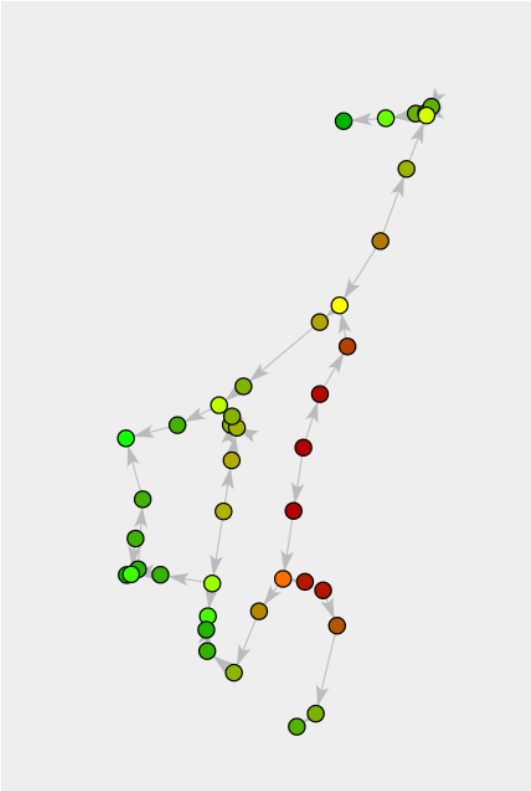

Figura 5.7: Girona: Gradació per z i ressalt per *degree* 3.

M'he trobat que treballar amb *selects* que canvien una variable d'estat en el seu *onChange* fa que s'actualitzi amb el valor antic. Per solucionar-ho he hagut de fer servir *setTimeout* per actualitzar la variable d'estat just després de canviar el valor del *select*.

Al mode 2D hi ha un *handler* (*onZoom*) que s'executa quan es fa zoom en el visualitzador, també quan es mou arrossegant amb el ratolí (zoom+pan), per actualitzar

la mida en pantalla dels nodes. Això implica que es fa una nova renderització i, per tant, es perden els valors dels filtres si aquests no s'han guardat amb les variables d'estat. Aquestes no es guarden fins que es processa un nou *visualization* quan es clica el botó de recarregar. Encara que això no afecta els *selects* amb la solució anterior, els *inputs* de tipus text es reiniciaven a cada zoom. És clar, podria haver utilitzat *setTimeout*, però es perdia el *focus*. Finalment, he decidit executar l'actualització de les variables d'estat en el mateix *handler*. Això si el menú de filtres està obert, cas contrari *filtersFromDom* rep un arbre buit.

Un altre problema relacionat amb el *onZoom* és el rendiment. Si aquest s'executa múltiples vegades quan es desplaça el visualitzador, s'ha de cridar *updateFilters* contínuament. Com a conseqüència, el moviment no és fluid. Per això he decidit utilitzar *onZoomEnd*, que s'executa quan s'ha acabat d'arrossegar. Però aquest canvi implica un comportament no esperat, els nodes es fan grans o petits durant el zoom fins que s'ajusten en acabar el desplaçament. He optat ignorar aquest comportament per garantir el rendiment.

Per acabar, m'he trobat que en el bloqueig del menú d'informació dels elements també ha d'haver-hi l'opció de desbloquejar. Primer vaig pensar fer servir una opció del *react-force-graph* que executa una funció quan es clica el fons, però aviat vaig veure que això implica canviar l'estil del cursor. Després d'intentar modificarlo manualment amb *CSS* vaig decidir fer el desbloqueig clicant al mateix element que estava blocat.

5.1.4 Abans i Després

A l'inici de l'elaboració d'aquest apartat hi havia un visualitzador poc eficient. Aquest havia de mostrar per pantalla tots els nodes i arestes. Això perjudicava el rendiment i l'experiència de l'usuari. A més, tampoc permetia visualitzar els atributs dels nodes a menys que es miressin un a un.

Ara hi ha un visualitzador que permet reduir el nombre d'elements per pantalla. Això s'aconsegueix amb un sistema de filtres que permeten definir el tipus d'interval i l'operador d'agregació entre ells. A més a més, ara es pot representar visualment, sigui via colors o altres mètodes, atributs propis dels nodes i arestes sense haver de fer *hover* per sobre d'ells.

5.1.5 Treball Futur

El nou sistema de filtres i coloració queda limitat pels atributs que té disponible el visualitzador. Cal recordar que en aquest Projecte em limito a millorar el *frontend*, on tots els valors es calculen en directe i no són permanents. Com a treball futur és

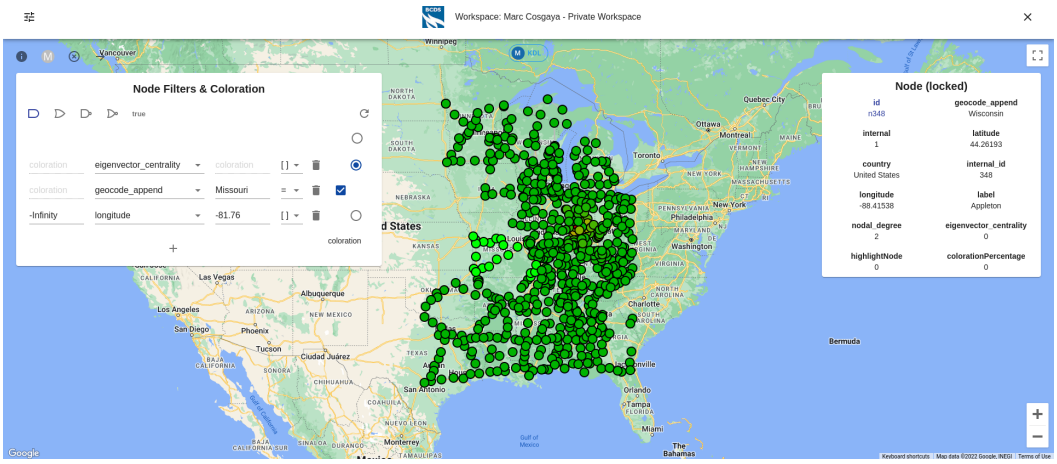

Figura 5.8: Centralitat pintada, Missouri ressaltat i long<=-81.76.

interessant el fet de definir nous atributs i configurar el servidor principal perquè els afegeixi als nodes.

És per això que he volgut elaborar una prova de concepte de com es pot demanar al servidor principal que calculi nous atributs amb la funció *getVisualizationFromApi* del *frontend*, i modificant una crida *HTTP* del servidor. Seria una bona idea definir quins atributs dels nodes i arestes es poden calcular a la configuració ("recepta") del servidor en comptes d'en un objecte del visualitzador.

5.2 3D Fix

5.2.1 Situació Actual

En els modes de visualització 2D i 3D s'utilitza l'objecte *visualization* per generar un objecte *data*. Aquest últim es passa com a paràmetre al component de *react-force-graph*. En la transformació d'un objecte a l'altre s'afegeixen dades rellevants per a l'*API* del component com l'identificador i coordenades. Aquestes últimes no són generades pel mode 3D.

5.2.2 Desenvolupament

La idea principal és aconseguir que el mode 3D guardi a *data* les coordenades x, y i z. Com que el visualitzador 2D ja ho feia amb les primeres dues, s'ha copiat i adaptat el seu codi afegint-hi la tercera coordenada. En el cas del 3D no interessa poder establir noves coordenades per defecte, com es fa en el 2D amb els botons d'avall a la dreta, perquè arrossegar nodes en un entorn 3D és poc viable. O sigui, és difícil interpretar el fet de moure per pantalla en dues dimensions un objecte de tres dimensions.

Per començar he afegit un nou panell a *VisualizerNetworkInfo* anomenat *FixedCoordinateSettings* (apèndix A.3) que permet seleccionar dos paràmetres: l'atribut i l'accentuació. El primer determina quin valor numèric se selecciona del node per representar a la tercera coordenada. I el segon indica el grau d'accentuació de la nova coordenada. Això es pot interpretar com una transformació del sistema de coordenades on es fa més gran la component z. La fòrmula del càlcul de la z per a cada node és:

$$z = \gamma\alpha$$

- α: El valor de l'atribut seleccionat.
- γ: El factor d'accentuació. $0 <= \gamma <= 500$ i $\gamma \in \mathbb{Z}$

Segon, he fet que s'habiliti la funció si els grafs tenen latitud i longitud o posicións personalitzades. No té sentit mostrar l'elevació si no es tenen coordenades horitzontals.

Tercer, per poder fer servir aquesta variant del mode 3D sense perdre l'orientació, he decidit afegir un pla/marc de referència de forma rectangular. Per implementar-ho he hagut de modificar el codi que genera el *data*, afegint-hi quatre arestes i quatre nodes. Les dimensions del rectangle depenen de les latituds i longituds màximes i mínimes.

Finalment, el visualitzador també ha de fer servir coordenades *custom*, les que

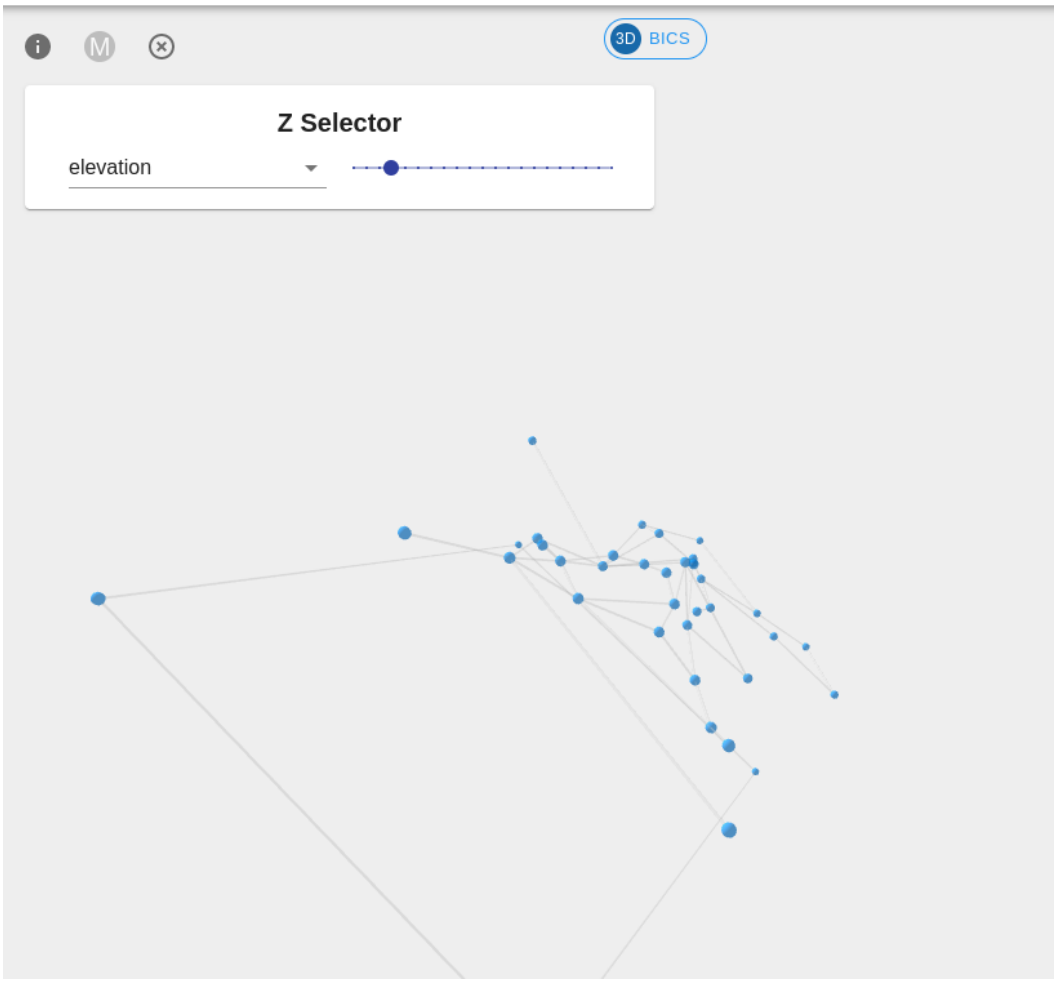

Figura 5.9: Primera versió funcional del 3D fix.

s'assignen als nodes quan es defineix una nova posició per defecte del graf des del visualitzador. Això pot ser útil per a altres projectes del grup. Per tant, ara primer es mira si existeixen aquestes coordenades i si no es dona el cas, s'intenta utilitzar les latituds i longituds.

5.2.3 Problemes i Decisions

Primer, he tornat a tenir el problema amb el *focus* dels *inputs* de tipus text. He decidit utilitzar un *slider*[7], que no queda afectat pels canvis de variables d'estat.

A continuació, en el cas del marc de referència, primer volia tenir una imatge per així poder situar els punts en un mapa. Veient que la llibreria *react-force-graph* no

[7] mui.com/material-ui/react-slider

té la possibilitat d'imatges amb perspectiva[8] i que no volia treballar amb *ThreeJS*, vaig descartar aquesta opció.

Tercer, no em quedava clar si afegir els nous elements del marc a *visualization* o directament a *data*. Com que el mode 3D no té la visualització *stateful*, modificar *data* implica menys canvis de codi. Però fer-ho així llança errors *undefined* quan es fa un *hover*. Per mitigar-ho he desactivat que es mostri la informació dels elements del marc, aquesta de fet no és rellevant perquè és completament artificial.

Quart, m'he trobat que quan les latituds i longituds estan molt properes, els nodes són massa grans per apreciar el graf (Figura 5.10). La primera idea que he tingut ha sigut utilitzar la propietat *onzoom* que ja es fa servir al component 2D de *react-force-graph*. Malauradament, aquesta propietat només està disponible pel 2D. Una altra opció que he considerat és modificar *customization* per fer més petits els nodes. Aquesta característica ja està implementada en el menú *VisualizerCustomizationDrawer*, però hi ha una mida mínima de les arestes i aquestes acaben sent massa grans. Finalment, he decidit afegir un nou *slider* per l'*spacing*, que multiplica les latituds i longituds abans de ser guardades a *data*.

Tot seguit, he vist que l'*slider* d'accentuació pot tenir una escala exponencial, per així evitar tenir massa valors possibles. El càlcul passa a ser:

$$z = 2^{\gamma}\alpha$$

- α: El valor de l'atribut seleccionat.
- γ: El factor d'accentuació. $0 <= \gamma <= 10$ i $\gamma \in \mathbb{Z}$

Al mode 3D he hagut de forçar el restabliment del zoom de la *customization* perquè aquest es mantenia entre modes. Això provocava que els nodes del 3D fossin molt grans o molt petits si abans s'havia fet zoom al 2D.

A continuació, he hagut d'arreglar un problema amb les coordenades *custom*. S'agafava aquestes coordenades quan s'esborrava la posició per defecte. Això era causat per no canviar el mode (*custom/coords/force*) abans d'actualitzar *data*.

A més a més, tenia pensat que només es mostrés el botó per habilitar aquest mode quan hi fossin disponibles els atributs dels nodes corresponents. Això es fa mitjançant la propietat *node_keys* de l'objecte *visualization*. Tanmateix, les coordenades *custom* no apareixen en aquesta propietat. No sabia si modificar *visualization* des del *frontend* o si retornar-lo directament des del servidor principal. M'he decantat per la segona opció, ja que així no s'ha de controlar contínuament si tots els nodes tenen *custom* i només s'ha de fer una consulta al servidor.

[8] vasturiano.github.io/react-force-graph/example/img-nodes

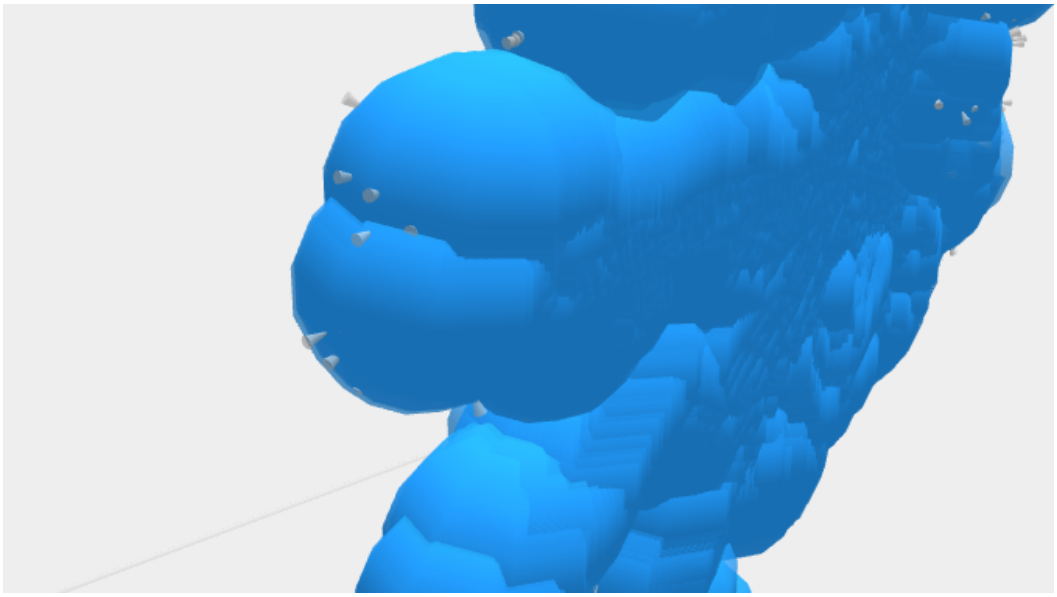

Figura 5.10: Els nodes són massa grans.

Finalment, he vist que seria adient normalitzar els valors de les z de cada node. Això pot ser útil per poder apreciar les diferències de z independentment de l'escala. El càlcul ara és:

$$z = 2^{\gamma}\alpha$$

- α: El valor de l'atribut seleccionat normalitzat per *min-max*. 0 <= α <= 1
- γ: El factor d'accentuació. 0 <= γ <= 10 i $\gamma \in \mathbb{Z}$

Per trobar l'alçada d'una capa hem de calcular la diferència entre la z d'un node que està al màxim i un node que està al mínim dins la mateixa capa:

$$2^{\gamma}\,\max(A) - 2^{\gamma}\,\min(A)$$

$$2^{\gamma}(\max(A) - \min(A))$$

$$2^{\gamma}(1 - 0)$$

$$2^{\gamma}$$

On A és el conjunt de valors α d'una capa.

L'alçada de la capa, per tant, sempre és proporcional a l'accentuació i mai depèn dels valors de l'atribut.

Pel que fa a normalitzar les x i y, he decidit no fer-ho perquè hi ha conflicte amb les coordenades *custom*. Els elements que fa servir *react-force-graph* treballen en un sistema de coordenades intern. El fet de guardar coordenades *custom* implica agafar

la coordenada interna de la llibreria. Es podria fer la traducció de la coordenada interna a una coordenada normalitzada en el moment de guardar la *custom*. Però això implicaria tenir guardat un valor de referència no normalitzat equivalent al 100% i un altre equivalent al 0%. Si el nou valor se sortís d'aquests dos valors, s'hauria de tornar a calcular la normalització amb els valors reals de tots els nodes. Poc eficient per a grafs grans.

Respecte a normalitzar les x i y per latituds i longituds, no hi ha coordenades *custom* que impliquin un conflicte. Però he decidit no implementar-ho perquè no seria coherent que el comportament canviés un cop s'establís una posició per defecte.

5.2.4 Abans i Després

Al principi d'aquest apartat hi havia un visualitzador 3D molt limitat. Aquest només permetia visualitzar els grafs amb un sistema de forces. Això feia que fos poc intuïtiu interpretar l'estructura de la xarxa.

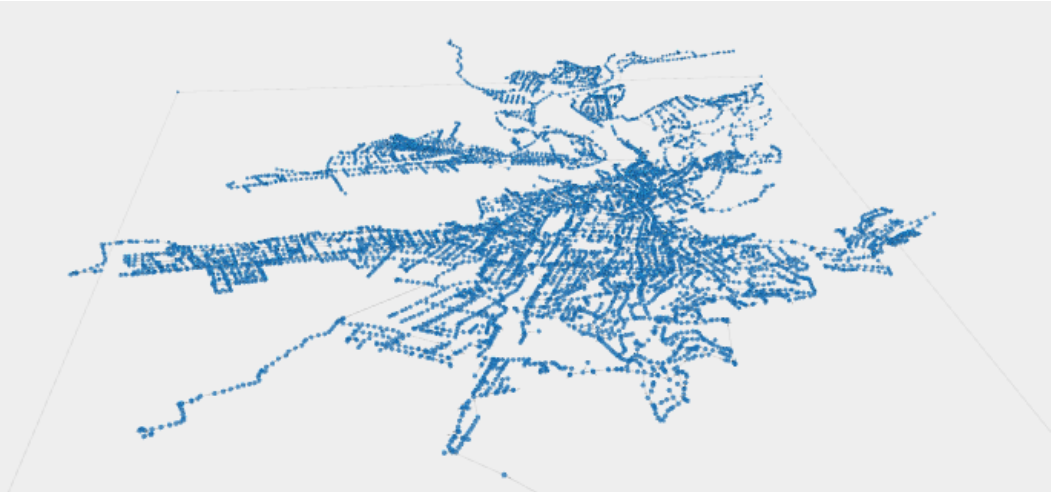

Figura 5.11: Xarxa d'aigües de Girona en 3D.

Ara hi ha un visualitzador 3D amb posicions fixes amb un marc de referència. Aquestes poden ser determinades per latitud i longitud o x i y personalitzades. A més a més, la nova coordenada z pot ser seleccionada d'entre els atributs dels nodes del graf. També hi ha l'opció d'accentuar la nova coordenada o modificar l'espaiat entre les coordenades x i y.

5.2.5 Treball Futur

S'ha mencionat a l'apartat 5.2.3 que es volia afegir una imatge fixa per tenir un marc de referència i no perdre l'orientació. Això requeria treballar amb *ThreeJS*[9]. Com a treball futur és interessant aprendre com funciona aquesta llibreria per així modificar *react-force-graph* i afegir aquesta imatge.

Respecte a normalitzar les x i y, és interessant trobar un mètode eficient per recalcular els valors normalitzats. Això o plantejar el problema d'una altra manera. Si fos possible, normalitzar també per latituds i longituds seria coherent.

[9] threejs.org

5.3 Multicapa

5.3.1 Situació Actual

Per part del servidor APP (principal), hi ha una ruta *HTTP POST* on es passa la llista de xarxes. Aquest després va generant, cridant per a cada xarxa la funció *createNewNetwork*, els fitxers *graphml* per a cadascuna de les xarxes que s'han enviat. Això es fa mitjançant *saveNewGraphmlFile*, que guarda els nous fitxers *graphml* i després de cridar *processGraphmlFile*, que acaba processant els elements de la xarxa un a un. La pila de crides és:

- *router.post('/:c_id/networks')*
  - *createNewNetwork*
    - *saveNewGraphmlFile*
      - *processGraphmlFile*

Per part del *frontend*, el component *Manager* conté tota la gestió de les xarxes. Dins d'aquest hi ha *NetworksUploader* (figura 5.12), la pestanya per pujar noves xarxes. I dins d'aquest hi ha *NetworksDragAndDrop*, que gestiona la pujada dels fitxers en si. Aquest últim té un *handler* per quan es fa clic al botó per pujar, que acaba fent la crida *HTTP POST* que s'ha mencionat anteriorment.

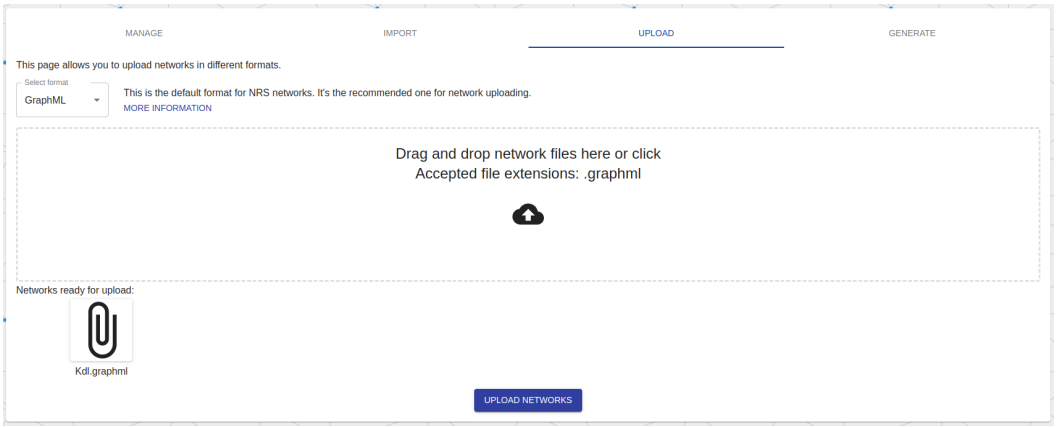

Figura 5.12: Menú per pujar fitxers.

5.3.2 Desenvolupament

Aquest Projecte se centra en el *frontend*, amb la puntual modificació del *backend*. Malgrat això, en aquest apartat hi ha hagut un esforç més gran en la segona part. Això es deu al fet que les millores d'aquest apartat es desenvolupen per sobre de les de l'apartat 5.2. Per tant, es reutilitzen funcionalitats.

La idea principal és modificar la crida del servidor perquè combini les xarxes en comptes de guardar-les per separat. Després, en el fitxer *graphml* resultant, s'afegeix un nou atribut a cada element per identificar la xarxa d'on prové. Aquest atribut podrà ser seleccionat com a z en la representació fixa, desenvolupat en l'apartat anterior. Finalment, s'assignen els identificadors interns del nou *graphml* de tal manera que no hi hagi col·lisió.

5.3.2.1 Frontend

Primer, al *frontend* he afegit un *input* de tipus *checkbox* al cantó del botó per pujar múltiples xarxes a *NetworksDragAndDrop*. Això de tal manera que només aparegui el nou *input* si s'han seleccionat dos o més fitxers.

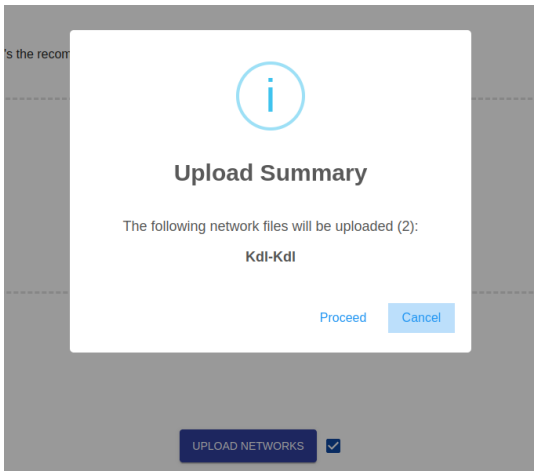

Figura 5.13: Es combinen els fitxers si el *checkbox* està seleccionat.

Després, he creat una nova crida a la *API* de nom *uploadContainerNetworksCombined*. M'he basat en la crida *uploadContainerNetworks*, que és la que penja els fitxers al servidor principal. La diferència amb la crida normal és l'addició d'un nou paràmetre de la *URL* que indica que s'han de combinar els fitxers del cos. En altres paraules, no s'afegeix una nova ruta.

5.3.2.2 Backend

Per començar, al *backend* he modificat la ruta *HTTP POST* existent perquè es cridi la nova funció *createNewCombinedNetwork*, que gestiona la creació de la xarxa a la base de dades, si el paràmetre de la *URL* ho indica. Aquesta és una modificació de *createNewNetwork*. La nova funció crida *combineNewGraphmlFile*, que guarda el fitxer *graphml* combinat en local i aquest, a la vegada, crida *processGraphmlFile*, que unifica els fitxers penjats. Això es fa element a element si un paràmetre ho indica. La pila de crides és:

- *router.post('/:c_id/networks')*
  - *createNewCombinedNetwork*
    - *combineNewGraphmlFile*
      - *processGraphmlFile* (si es demana, s'unifiquen els fitxers)

La pila de crides és quasi la mateixa. En aquest cas *processGraphmlFile* és la funció que fa la feina principal. Aquesta funció, si ho indica un paràmetre, combina els *tags* afegint un prefix a l'identificador de cada un indicant de quin fitxer prové. A més a més, afegeix un nou atribut a cada element que indica de quina xarxa es tracta. Aquest atribut és de tipus numèric.

A continuació, he hagut d'afegir l'opció de combinar grafs que no estiguin en format *graphml*. La conversió de grafs que estan en formats diferents a *graphml* la vaig desenvolupar amb un altre company del grup quan m'hi vaig unir. He adaptat el nou codi per reutilitzar-la. Això es fa amb un processament previ dels fitxers per primer convertir-los a *graphml*. D'aquesta manera sempre es fa la unificació en format *graphml*.

5.3.3 Problemes i Decisions

Per començar, el dilema principal en què m'he trobat és si crear el fitxer *graphml* combinat directament o fer-ho després de guardar els fitxers per separat. Com que les funcions existents ja processen els elements del fitxer, modificar-les per utilitzar múltiples fitxers no ha implicat molt de canvi. Tanmateix, si hagués combinat els fitxers després de processar-los un a un, hauria hagut d'afegir noves funcions.

Segon, he decidit modificar *processGraphmlFile* perquè faci la unificació al principi segons un paràmetre per no haver de repetir codi. Inicialment havia copiat la funció per crear una de nova. Però veient que el comportament era molt semblant a la funció incial, m'he decantat per l'opció del paràmetre.

Tercer, des d'un principi tenia pensat afegir el prefix a l'identificador de cada *tag* del *graphml* per evitar col·lisions. El que no em quedava clar era si el visualitzador i les rutes per obtenir la visualització al servidor principal quedarien afectats per aquest canvi. Més tard m'he adonat que a *processGraphmlFile* ja es fa una normalització d'identificadors. Per tant, el visualitzador no pot diferenciar un graf normal d'un graf combinat. Això és útil si vull estalviar codi en el *frontend*.

Per acabar, en comptes de tenir al *select* de la z l'atribut identificador, he acabat tractant-lo per separat. Això es deu a la necessitat de representar simultàniament les

capes i la z seleccionada de la visualització fixa. Per tant, càlcul de z final és:

$$z = 2^{\gamma}(\alpha + \beta)$$

- α: El valor de l'atribut seleccionat normalitzat per *min-max*. 0 <= α <= 1
- β: L'identificador de la capa. β >= 0 i $\beta \in \mathbb{Z}$
- γ: El factor d'accentuació. 0 <= γ <= 10 i $\gamma \in \mathbb{Z}$

La distància mínima possible entre capes és:

$$2^{\gamma}(\min(A_i) + \beta_i) - 2^{\gamma}(\max(A_{i-1}) + \beta_{i-1})$$

$$2^{\gamma}((\min(A_i) + \beta_i) - (\max(A_{i-1}) + \beta_{i-1}))$$

$$2^{\gamma}(\min(A_i) - \max(A_{i-1}) + \beta_i - \beta_{i-1})$$

$$2^{\gamma}(\min(A_i) - \max(A_{i-1}) + 1)$$

$$2^{\gamma}(0 - 1 + 1)$$

$$2^{\gamma}0$$

$$0$$

On A_i és el conjunt de valors α de la capa i.

En conclusió, no es pot donar el cas que un node d'una capa superior estigui per sota d'un node de la capa inferior. Com a molt podran estar a la mateixa alçada.

L'alçada d'una capa se segueix comportant igual:

$$2^{\gamma}(\max(A) + \beta) - 2^{\gamma}(\min(A) + \beta)$$

$$2^{\gamma}(\max(A) - \min(A) + \beta - \beta)$$

$$2^{\gamma}(\max(A) - \min(A))$$

$$2^{\gamma}(1 - 0)$$

$$2^{\gamma}$$

On A és el conjunt de valors α d'una capa.

5.3.4 Abans i Després

Al principi d'aquest apartat no existia l'opció de combinar múltiples xarxes. No hi havia manera de comparar les posicions dels nodes d'una xarxa a menys que es tingués el visualitzador carregat en dues pestanyes o finestres del navegador.

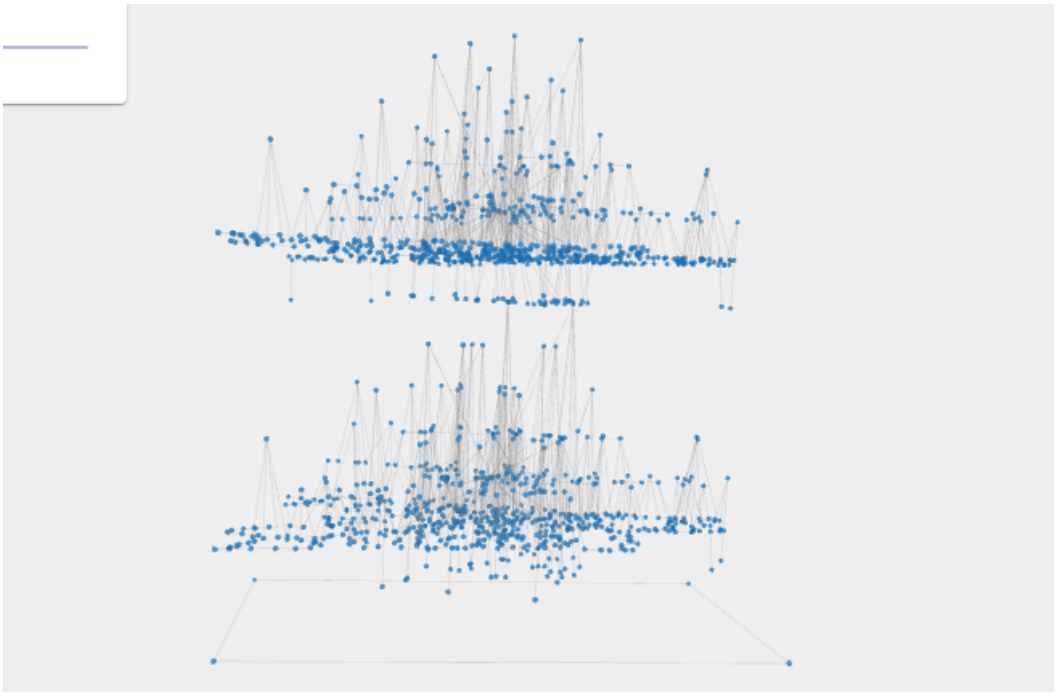

Figura 5.14: Dues xarxes *KDL* (*Kentucky Datalink*). No hi ha superposició.

Ara existeix aquesta opció de combinar. Però les xarxes no es combinen només al *frontend*, sinó que s'unifiquen els fitxers en el *backend*. Aquesta manera de treballar evita fer l'operació de combinar cada vegada que s'obre el visualitzador. A més a més, el fet que el fitxer resultant sigui únic permet calcular-ne les mètriques. És a dir, el servidor de computació calcula les mètriques al marge que el fitxer sigui resultat d'unificar una o més xarxes.

5.3.5 Treball Futur

És interessant com a treball futur afegir l'opció de generar arestes d'interdependència entre xarxes. Aquestes arestes es podrien generar durant la unificació quan dos nodes tenen coordenades molt semblants. Això pot ser útil per veure com interactuen les múltiples capes.

5.4 API d'Elevacions

5.4.1 Situació Actual

En el *NRS2* no existeix la funcionalitat d'obtenir la coordenada *z*, corresponent a l'elevació, a partir de les dades de geolocalització disponibles dels nodes. Aquesta característica pot ser útil a l'hora de visualitzar el pendent en xarxes d'aigua.

5.4.2 Desenvolupament

M'he decantat a fer servir l'*API Open-Elevation*[10]. Està implementada en *Python* i és de codi obert[11]. S'ofereix com a alternativa al *Google Elevation API*.

Obté les dades del *Consortium for Spatial Information*[12] en tres parts i genera *tiles* d'una mida determinada. Quan es fa una crida *HTTP GET* es pot passar a la *url* una llista d'ubicacions amb les seves coordenades. Després l'*API* retorna una llista amb les elevacions que s'han demanat[13].

5.4.3 Problemes i Decisions

L'*API* oficial poques vegades està disponible. Això es pot deure al fet que està servint moltes peticions a la vegada o per qüestions de manteniment. És per això que he decidit allotjar-la en un servidor virtual del grup amb *Docker*.

A la documentació d'allotjament[14] es recomana tenir prou espai d'emmagatzematge suficient perquè les dades poden ocupar més de 20GB. I com que no hi havia suficient espai en el servidor on el volia allotjar, on també hi ha allotjats altres serveis, vaig haver d'augmentar el seu emmagatzematge fins a 80GB.

Per acabar, les crides *GET* tenen un límit de mida de 1024B. Per sort, l'*API* permet fer crides *POST* sense aquest límit. A més a més, per no carregar molt el servidor, he optat per fragmentar les crides en blocs.

[10] open-elevation.com
[11] github.com/Jorl17/open-elevation
[12] srtm.csi.cgiar.org
[13] github.com/Jorl17/open-elevation/blob/master/docs/api.md
[14] github.com/Jorl17/open-elevation/blob/master/docs/host-your-own.md

5.4.4 Abans i Després

Al principi d'aquest apartat no hi havia manera d'obtenir l'elevació dels nodes per després mostrar-les amb la representació en 3D fixa desenvolupada en l'apartat 5.2.

Ara existeix un mètode per obtenir aquesta elevació. I en comptes d'estar allotjat externament[15], ara està allotjat en un servidor virtual del grup[16].

5.4.5 Treball Futur

Tal com s'ha explicat anteriorment, el servidor principal necessita tenir la possibilitat de calcular nous atributs de nodes i arestes. Per això, l'atribut de l'elevació pot ser calculat amb el que s'ha desenvolupat en aquest apartat. Com que aquest apartat es tracta d'una prova de concepte, he decidit no ajuntar-ho al codi principal al *GitLab*.

També és interessant poder generar mapes d'elevacions a partir de fonts més locals, com l'Institut Cartogràfic i Geològic de Catalunya[17], per així tenir més precisió en les consultes. Això pot ser útil en el cas de la xarxa d'aigües de Girona.

[15] api.open-elevation.com/api/v1/lookup?locations=0,0
[16] gintonic.udg.edu/api/v1/lookup?locations=0,0 Només d'accés intern.
[17] www.icgc.cat/Descarregues/Elevacions

Capítol 6

Pressupost

Ja s'ha dit al capítol 4 (Planificació) que l'estimació màxima és de 486 hores. Se suposa que el sou mitjà d'un enginyer informàtic recent iniciat és de 20400 euros bruts a l'any[1], que es tradueix en 9,81 euros l'hora. Per tant, el desenvolupament d'aquest treball implicaria un cost de 4767,66 euros bruts màxims si s'hagués dut a terme en altres circumstàncies.

S'ha de dir que part de l'elaboració d'aquest Projecte ha sigut durant una Estada a l'Entorn Laboral amb el grup *BCDS*. De fet, aquesta estada ha estat remunerada amb un sou de 5 euros l'hora i representa el 60%, aproximadament, del temps total de desenvolupament. Això sí, durant l'estada no només s'ha treballat en aquest Projecte.

[1] www.jobted.es/salario/ingeniero-informático

Capítol 7

Conclusions

M'ha agradat molt fer aquest Projecte perquè sempre m'ha fascinat el desenvolupament web. També perquè ha sigut una oportunitat per aprendre nous conceptes, encara que la base ja la coneixia. Per exemple els *hooks* de React, que han sigut molt útils per simplificar parts del codi, o el comportament del *parseFloat()* (apartat 5.1.3).

No obstant això, el desenvolupament d'aplicacions web és més difícil del que sembla. Posem per cas l'estil, que sovint es deixa de banda en favor de la funcionalitat durant el desenvolupament. En aquest cas, hi he hagut de dedicar una part considerable del temps per ajustar l'estil a les necessitats de l'usuari.

Encara que des d'un principi pot semblar que el Projecte només implica modificar el *frontend*, en alguns dels objectius he necessitat fer petites modificacions al *backend*, sobretot a l'apartat 5.3. Per tant, es pot dir que he fet de *Full Stack Developer*[1].

He afegit el codi dels nous components a l'apèndix A. No hi ha la resta del que s'ha desenvolupat per estalviar espai en aquesta memòria i per evitar problemes de llicència. Això últim és conseqüència de tenir el codi del *frontend* i del servidor principal (APP) en un repositori privat del grup.

Ara bé, encara hi ha molta feina a fer. Tots els objectius tenen possibles millores. Aquestes són oportunitats per desenvolupar noves característiques que enriqueixin encara més el que s'ha fet en aquest Projecte. Espero que el que s'ha desenvolupat pugui ser d'utilitat pel grup Comunicacions i Sistemes Distribuïts. Sobretot a què aporti valor a la nova recerca.

[1] www.w3schools.com/whatis/whatis_fullstack.asp

Apèndix A

Nous Components

A.1 NetworkFilter.js

```
1  // nrs-web: Network Research Simulator Front End Interface
2  // Author: ©BCDS Research Group - Girona University.
3  //  Contributors: Marc Cosgaya
4
5  // visualizer/NetworkFilter.js file: Contains a custom component
       for a filter.
6
7
8  // 1. DEPENDENCIES
9
10 // NPM community 'react' package: React is a JavaScript library
       for creating user interfaces.
11 import React, { useState } from 'react';
12
13 // NPM community '@material-ui' package: React component that
       implement Google's Material Design.
14 import { ListItem, Input, IconButton, Tooltip, Select, MenuItem,
       Radio, Checkbox } from '@material-ui/core';
15 import { Delete } from '@material-ui/icons';
16
17 // 2. FUNCTIONS
18
19 // NRS Private Web React Visualizer Network Info.
20 const NetworkFilter = (props) => {
21
22     // Props:
23     //  node_keys:                  List of node key attribute
           names (mandatory).
24     //  handleSelect:               Function to handle radio
           select (mandatory).
25     //  selected_indexes:           List of indexes of currently
           selected filters (mandatory).
26     //  index:                      Index of filter in the array
           of filters (mandatory).
27     //  deleteFilterParent:         Parent function to delete
           filter with an index (mandatory).
28     //  attribute:                  Attribute name (optional).
```

```
29      //  maxValue:                   Max value of attribute (
            optional).
30      //  minValue:                   Min value of attribute (
            optional).
31      //  type_prop:                  Interval type of filter (
            optional).
32
33      const { node_keys, handleSelect, selected_indexes, index,
            deleteFilterParent, attribute = 'id', maxValue = Infinity,
             minValue = -Infinity, type_prop = '[ ]' } = props;
34
35      // Store in state the type of the filter.
36      const [type, setType] = useState(type_prop);
37
38      // Get boolean whether the filter is selected.
39      const selected = type_prop !== '=' ? selected_indexes.radio
            === index : selected_indexes.checkboxes.includes(index);
40
41      // Return the render of Filter.
42      return (
43          <ListItem style={{marginTop: '-10px'}}>
44              <Input
45                  name={`minValue`}
46                  placeholder={selected ? 'coloration' : type === '
                        =' ? '' : 'min'}
47                  defaultValue={selected ? '' : type === '=' ? '' :
                         minValue}
48                  type="text"
49                  style={{ width: '100px', marginRight: '10px' }}
50                  disabled={type === '=' || selected}>
51              </Input>
52              <Select defaultValue={attribute} style={{ width: '200
                    px', marginRight: '10px', marginLeft: '10px' }}
53                      onChange={() => setTimeout(() => handleSelect
                            (undefined,undefined,false),0) }>
54                  {node_keys.map((node_key, index) => {
55                      if (!node_key.calculatable)
56                          return <MenuItem key={index} value={
                                node_key.key}>{node_key.key}</MenuItem
                                >
57                      else {
58                          return <MenuItem key={index} value={
                                node_key.key} style={{ fontStyle: '
                                italic' }}>{node_key.key}</MenuItem>
59                      }
60                  })}
61              </Select>
62              <Input
63                  name={`maxValue`}
```

```
64                  placeholder={type === '=' ? 'value' : selected ?
                        'coloration' : 'max'}
65                  defaultValue={selected && type !== '=' ? '' :
                        maxValue}
66                  type="text"
67                  style={{ width: '100px', marginLeft: '10px',
                        marginRight: '10px' }}
68                  disabled={selected && type !== '='}>
69              </Input>
70              <Select
71                  value={type}
72                  style={{ marginLeft: '10px' }}
73                  onChange={e => {
74                          setType(e.target.value);
75                          const after_type = e.target.value;
76                          setTimeout(() => {
77                              if ((after_type !== '=' && type === '
                                    =') || (type !== '=' && after_type
                                     === '='))
78                                  handleSelect(index,type === '=',
                                      true);
79                          },0);
80                      }}
81              >
82                  <MenuItem value={'[ ]'}>[ ]</MenuItem>
83                  <MenuItem value={'( )'}>( )</MenuItem>
84                  <MenuItem value={'[ )'}>[ )</MenuItem>
85                  <MenuItem value={'( ]'}>( ]</MenuItem>
86                  <MenuItem value={'='}>=</MenuItem>
87              </Select>
88              <Tooltip title="Delete Filter">
89                  <IconButton onClick={() => deleteFilterParent(
                        index)}>
90                      <Delete />
91                  </IconButton>
92              </Tooltip>
93              {type_prop !== '=' ?
94                  <Radio style={{ marginLeft: '28px' }} onClick={()
                        => handleSelect(index,true)} checked={
                        selected}></Radio> :
95                  <Checkbox style={{ marginRight: '30px' }} onClick
                        ={() => handleSelect(index,false)} checked={
                        selected}></Checkbox>
96              }
97          </ListItem>
98      )
99  }
100
101 // Export the component.
```

```
102 export default NetworkFilter;
```

A.2 EdgeColoration.js

```
1  // nrs-web: Network Research Simulator Front End Interface
2  // Author: ©BCDS Research Group - Girona University.
3  //  Contributors: Marc Cosgaya
4
5  // visualizer/EdgeColoration.js file: Contains a custom component
       for an edge coloration.
6
7
8  // 1. DEPENDENCIES
9
10 // NPM community 'react' package: React is a JavaScript library
       for creating user interfaces.
11 import React from 'react';
12
13 // NPM community '@material-ui' package: React component that
       implement Google's Material Design.
14 import { ListItem, Select, MenuItem } from '@material-ui/core';
15
16 // 2. FUNCTIONS
17
18 // NRS Private Web React Visualizer Network Info.
19 const NetworkFilter = (props) => {
20
21     // Props:
22     //  edge_coloration:        Edge coloration settings useState
           () hook (mandatory).
23     //  edge_coloration.type:   d=dots;c=color;w=width;dr=dots
           reverse;cr=color reverse;wr=width reverse;dc=direction
           check;dcr=direction check reverse
24     //  keys:                   List of edge key attribute names
           (mandatory).
25     //  reload:                 Parent function to reload the
           edge coloration (mandatory).
26     //  directed:               Boolean showing whether the graph
            is directed or not (optional).
27     //  visualizationMode:      String that stores the mode of
           visualization (mandatory).
28     const { edge_coloration, keys, reload, directed = false,
           visualizationMode } = props;
29
30     // Add empty key at the beginning of the array.
31     if (keys.find(k => k === '') !== '') keys.unshift('');
32
33     // Reset type if Map is open and dots are selected.
```

```
34      if ((edge_coloration.type === 'd' || edge_coloration.type ===
            'dr') && visualizationMode === 'M') edge_coloration.type
           = 'c';
35
36      // Return the render of Filter.
37      return (
38          <ListItem style={{marginTop: '-10px'}}>
39              <Select
40                  value={edge_coloration.type}
41                  style={{ width: '200px', marginRight: '10px',
                        marginLeft: '10px' }}
42                  onChange={e => {
43                      const new_coloration = {...edge_coloration};
44                      new_coloration.type = e.target.value;
45                      const change = !((['d','c','w','dr','cr','wr'
                            ].find(el => el === edge_coloration.type)
                            && ['d','c','w','dr','cr','wr'].find(el =>
                             el === e.target.value)) || (['dc','dcr'].
                            find(el => el === edge_coloration.type) &&
                             ['dc','dcr'].find(el => el === e.target.
                            value)));
46                      reload(new_coloration,change);
47                  }}
48              >
49                  {visualizationMode !== 'M' ? <MenuItem value={'d'
                        }>dots</MenuItem> : null }
50                  {visualizationMode !== 'M' ? <MenuItem value={'dr
                        '}>dots (reverse)</MenuItem> : null }
51                  <MenuItem value={'c'}>color</MenuItem>
52                  <MenuItem value={'cr'}>color (reverse)</MenuItem>
53                  <MenuItem value={'w'}>width</MenuItem>
54                  <MenuItem value={'wr'}>width (reverse)</MenuItem>
55                  {directed ? <MenuItem value={'dc'}>direction
                        check</MenuItem> : null}
56                  {directed ? <MenuItem value={'dcr'}>direction
                        check (reverse)</MenuItem> : null}
57              </Select>
58              <Select
59                  value={edge_coloration.attribute}
60                  style={{ width: '200px', marginRight: '10px',
                        marginLeft: '10px' }}
61                  onChange={e => {
62                      const new_coloration = {...edge_coloration};
63                      new_coloration.attribute = e.target.value;
64                      reload(new_coloration,false);
65                  }}
66              >
67                  {keys.map((edge_key, index) => <MenuItem key={
                        index} value={edge_key}>{edge_key}</MenuItem>)
```

```
                    }
68              </Select>
69          </ListItem>
70      )
71  }
72
73  // Export the component.
74  export default NetworkFilter;
```

A.3 FixedCoordinateSettings.js

```
1   // nrs-web: Network Research Simulator Front End Interface
2   // Author: ©BCDS Research Group - Girona University.
3   //  Contributors: Marc Cosgaya
4
5   // visualizer/FixedCoordinateSettings.js file: Contains a custom
        component for the fixed coordinates settings.
6
7
8   // 1. DEPENDENCIES
9
10  // NPM community 'react' package: React is a JavaScript library
        for creating user interfaces.
11  import React from 'react';
12
13  // NPM community '@material-ui' package: React component that
        implement Google's Material Design.
14  import { ListItem, Select, MenuItem, Slider } from '@material-ui/
        core';
15
16  // 2. FUNCTIONS
17
18  // NRS Private Web React Visualizer Network Info.
19  const FixedCoordinateSettings = (props) => {
20
21      // Props:
22      //  keys:                   List of node key names (mandatory
            ).
23      //  manage_accent:          Two-element array of parent
            useState of accent modifier (mandatory).
24      //  manage_spacing:         Two-element array of parent
            useState of XY spacing modifier (mandatory).
25      //  manage_selected_key:    Two-element array of parent
            useState of selected node key name (mandatory).
26      const { keys, manage_accent, manage_spacing,
            manage_selected_key } = props;
27
28      // Return the render of Filter.
```

```
29      return (
30          <ListItem style={{ marginTop: '-10px', textAlign: 'center
                ' }}>
31              <div style={{ width: '200px', marginRight: '10px',
                    marginLeft: '10px' }}>
32                  <b>Z Attribute</b>
33                  <Select
34                      defaultValue={manage_selected_key[0]}
35                      placeholder={'Select an attribute'}
36                      style={{ width: '200px', marginRight: '10px',
                            marginLeft: '10px' }}
37                      onChange={e => manage_selected_key[1](e.
                            target.value)}
38                  >
39                      {keys.map((edge_key, index) => <MenuItem key
                            ={index} value={edge_key}>{edge_key}</
                            MenuItem>)}
40                  </Select>
41              </div>
42              <div style={{ width: '200px', marginRight: '10px',
                    marginLeft: '10px' }}>
43                  <b>Z Accentuation</b>
44                  <Slider
45                      aria-label="Accent"
46                      step={1}
47                      min={-1}
48                      max={10}
49                      scale={x => x > -1 ? '2^' + x : '0'}
50                      track={false}
51                      valueLabelDisplay={'auto'}
52                      defaultValue={manage_accent[0]}
53                      style={{ width: '200px', marginRight: '10px',
                            marginLeft: '10px' }}
54                      onChangeCommitted={(_,value) => manage_accent
                            [1](value)}
55                  />
56              </div>
57              <div style={{ width: '200px', marginRight: '10px',
                    marginLeft: '10px' }}>
58                  <b>XY Spacing</b>
59                  <Slider
60                      aria-label="Spacing"
61                      step={1}
62                      min={-10}
63                      max={10}
64                      scale={x => '2^' + x}
65                      track={false}
66                      valueLabelDisplay={'auto'}
67                      defaultValue={manage_spacing[0]}
```

```
68                      style={{ width: '200px', marginRight: '10px',
                            marginLeft: '10px' }}
69                      onChangeCommitted={(_,value) =>
                           manage_spacing[1](value)}
70                  />
71              </div>
72          </ListItem>
73      )
74  }
75
76  // Export the component.
77  export default FixedCoordinateSettings;
```