Universitat de Girona
Escola Politècnica Superior

Treball final de màster

Estudi: Màster en Enginyeria Informàtica

Títol: Implementació de serveis d'optimització en Smart Grids.

Document: Memòria

Alumne: Àlex Bech Rodríguez

Tutor: Ferrant Torrent Fontbona
Departament: Enginyeria elèctrica, Electrònica i Automàtica
Àrea: Enginyeria de Sistemes i Automàtica

Convocatòria (mes/any): Gener 2019

Taula de continguts

1. Introducció 1
1.1 Context i motivació 1
1.2 Objectius 3
2 Conceptes previs 4
2.1 Conceptes elèctrics 4
2.1.1 Potència elèctrica 4
2.1.2 Hosting Capacity 4
2.1.3 Estudis dels fluxos de potència 4
2.2 RESOLVD: motivació i objectius 5
2.3 Algoritmes metaheurístics 7
3. Metodologia de treball 8
4. Requeriments 10
4.1 Requeriments del servidor 10
4.2 Requeriments del client 11
5. Disseny 13
5.1 Estructura del projecte i REST 13
5.1.1 Disseny de l'API REST 14
5.2 Disseny dels components del servidor 17
5.2.1 Diagrama de classes 17
5.2.2 Descripció dels components 17
5.3 Estructura de les dades 22
5.3.1 Estructura d'una xarxa 22
5.3.2 Estructura de la base de dades 23
5.4 Disseny del client 29
6. Tecnologies utilitzades 30
6.1 Servidor 30

6.1.1 Python....30
6.1.2 Flask....30
6.1.3 mongoDB....31
6.1.4 GridCal....32
6.2 Client....33
6.2.1 Angular....33
6.2.2 D3.js....34
7. Implementació i resultats....35
7.1 Implementació i resultats del servidor....35
7.1.1 Estructura de fitxers....35
7.1.2 Inicialització del servei web i gestió de les rutes....37
7.1.3 Restricció de les rutes per usuaris autentificats i per rol....39
7.1.4 Format de la resposta....45
7.1.5 Configuració del CORS....49
7.1.6 Implementació dels components....50
7.2 Implementació del client....54
7.2.1 Arquitectura d'una aplicació Angular....54
7.2.2 Serveis destacats....55
7.2.3 Mòduls i components destacats....56
7.2.4 Pantalles de l'aplicació....59
7.3 Algoritmes d'optimització....70
7.3.1 Hill Climbing....70
7.3.2 Particle Swarm Optimization....73
8. Conclusions....77
9. Bibliografia....79
ANNEXOS....81
ANNEX 1. Guia d'instal·lació....81
ANNEX 2. Rutes API REST....87

1. Introducció

Aquest projecte final de màster consisteix en el desenvolupament d'un servei web de planificació de xarxes elèctriques. El treball es pot dividir en dos grans blocs. El primer inclou la planificació, els requeriments, el disseny i les eines que s'han fet servir per dur a terme el desenvolupament. El segon bloc està format pels resultats obtinguts i les conclusions que se'n poden extreure a la finalització d'aquest.

A continuació s'explica el context, la motivació i els objectius a partir dels quals s'ha desenvolupat aquest treball.

1.1 Context i motivació

La generació distribuïda d'energia elèctrica, també coneguda com generació descentralitzada, consisteix en la generació i l'emmagatzematge d'energia elèctrica a partir de petites fonts d'energia connectades a la xarxa elèctrica de distribució (mitja i baixa tensió) (Wikipedia. Generación distribuida). Els elements que formen part d'aquest tipus de xarxa es coneixen com a **fonts d'energia distribuïda** (DER en anglès).

Les fonts d'energia convencionals, com les centrals tèrmiques, nuclears o també les grans granges solars, acostumen a ser lluny dels centres de consum i l'energia que generen ha de ser transportada a llocs molt distants de la seva localització. En canvi, les DER tenen una estructura descentralitzada, són modulars i pròximes al lloc on s'ha de consumir l'energia que generen. Normalment, les DER consisteixen en diferents tipus de fonts d'energies renovables com l'energia eòlica, solar, biomassa o geotèrmica, i cada cop signifiquen una porció més gran del total de l'energia aportada a la xarxa. La Figura 1 mostra una comparativa entre una xarxa de distribució d'energia elèctrica tradicional i una xarxa distribució amb generació distribuïda:

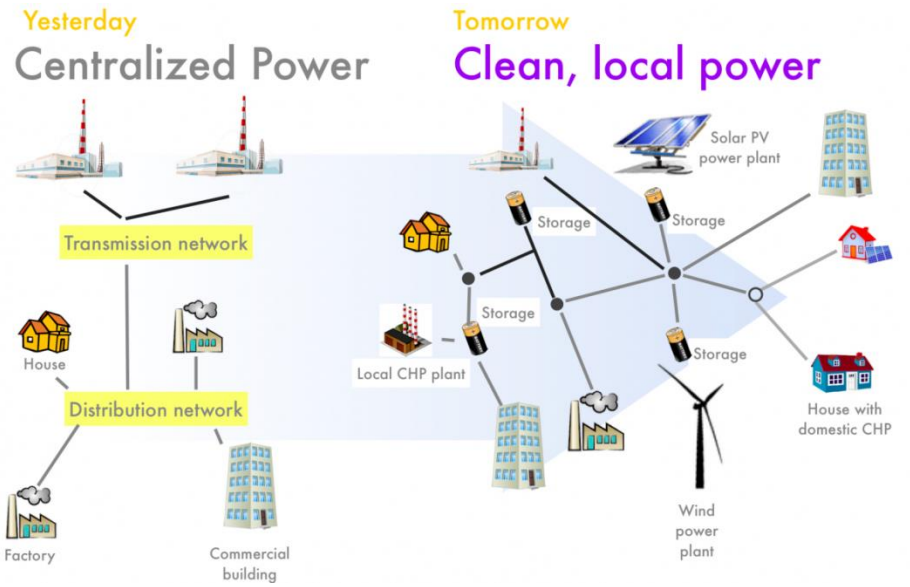

Figura 1. Representació d'una xarxa de distribució d'energia elèctrica convencional i una xarxa de distribució amb generació distribuïda. Font: Grist

Els principals avantatges de les DER respecte les fonts d'energia convencionals són:

- Al ser més pròximes al lloc de consum, redueixen les pèrdues en la transmissió.
- Poden descongestionar els sistemes de transport d'energia.
- Poden ajudar al subministrament d'energia en períodes d'alta demanda.
- Poden Millorar la fiabilitat del sistema.
- Poden Millorar la qualitat del servei elèctric.
- Necessiten inversions més petites i per tant redueixen el risc de les inversions.

La generació distribuïda és impulsada des de la UE com a possible solució que permeti incrementar la quantitat de generació d'energia elèctrica des de fonts renovables. No obstant, les xarxes elèctriques vigents no estan dissenyades per assumir una generació distribuïda significativa i seus avantatges potencials poden esdevenir desavantatges si no se'n fa una gestió activa i acurada. Per aquest motiu, és necessari dotar les xarxes elèctriques i els seus gestors de serveis per augmentar-ne el control i l'automatització de les xarxes per fer-les més flexibles i allotjar més recursos distribuïts.

RESOLVD és un projecte europeu que té com a propòsit millorar l'eficiència, la capacitat d'allotjament de generació distribuïda de les xarxes i incrementar l'ús de fonts d'energies renovables, amb eines i dispositius que dotin a les xarxes elèctriques vigents de capacitat intel·ligents per transformar-les en les anomenades *smart grids* .

Un dels objectius del projecte RESOLVD és el disseny d'eines de monitorització i planificació a partir de l'explotació de dades d'aquestes xarxes elèctriques intel·ligents. Dins d'aquest projecte, el grup de recerca eXiT de la Universitat de Girona és el responsable de desenvolupar tècniques i algoritmes per calcular la planificació òptima d'elements de la xarxa com bateries, generadors i càrregues en funció de la generació i demanda.

Aquest projecte se centra en el disseny i desenvolupament del servei web que ha de proporcionar les eines per implementar i utilitzar aquestes tècniques de planificació i poder obtenir planificacions òptimes de la xarxa.

A més a més, s'implementaran nous algoritmes de planificació i una aplicació client per interactuar amb el servei web mitjançant una interfície gràfica.

1.2 Objectius

El treball consistirà en implementar un *framework* per desenvolupar i testejar mètodes de planificació de xarxes elèctriques intel·ligents, desenvolupats pel grup de recerca de la universitat de Girona eXiT, que calcularan la planificació òptima de bateries, generadors i càrregues en xarxes elèctriques intel·ligents.

El treball té com a objectius principals els següents punts:

- Definir estàndard de fitxers Excel que defineixin la xarxa i la generació/demanda a cada punt.
- Muntar xarxa elèctrica amb algun *power flow solver* a partir de fitxers Excel estandarditzats.
- Implementar mètodes per modificar automàticament a demanda de l'algoritme de planificació, la xarxa i la demanda/generació on sigui possible.
- Implementar mètodes d'optimització i planificació de la xarxa.
- Implementar base de dades on guardar localment les dades i solucions de totes les planificacions demanades.
- Implementar gestió d'usuaris i d'accés al servei de planificació.
- Implementar el servei de planificació com a servei web.
- Desenvolupar una aplicació per interactuar amb el servei web a través d'una interfície.
- Desplegament dels serveis a un servidor de eXiT.

2 Conceptes previs

En aquest apartat s'introdueixen conceptes que poden ser d'ajuda per entendre millor la resta de les seccions d'aquest projecte.

2.1 Conceptes elèctrics

A continuació es defineixen els conceptes elèctrics clau d'aquest projecte.

2.1.1 Potència elèctrica

Per entendre què és la potència elèctrica cal primer recordar conceptes bàsics d'**energia** i **potència**.

Formalment, l'energia és la capacitat que posseeix un cos per produir un treball o una transformació. L'energia es pot transmetre en diverses formes com llum, calor, electricitat, etc. La unitat de mesura de l'energia és el joule (J) (Definición de energia. Concepto Definición)

La potència és la magnitud que mesura l'energia consumida o generada per unitat de temps. La unitat internacional de mesura és el watt (W), que equival a joules per segon (J/s). El concepte de **potència elèctrica**, doncs, fa referència a la quantitat d'energia elèctrica que consumeix o genera un cos a cada instant de temps, com per exemple un generador, una bateria o elements naturals com un llamp (Electric power. Sparkfun).

Aplicat al concepte de potència elèctrica, un **sistema de potència** és el conjunt d'elements encarregats de generar, transmetre i distribuir l'energia elèctrica als consumidors (Alcantar). La forma de generar l'energia elèctrica dependrà del tipus de font: nuclear, hidràulica, solar, eòlica, etc.

2.1.2 Hosting Capacity

En termes de generació distribuïda d'energia, el terme *hosting capacity* es defineix com la quantitat d'energia que la xarxa de distribució pot tolerar provinent de les fonts d'energia distribuïda (Bollen & Rönnberg, 2017). Si una xarxa de distribució excedeix aquest límit podria provocar problemes en la fiabilitat de la xarxa en forma de sobrecàrregues, pèrdues o baixades de voltatge, afectant a la resta d'usuaris.

2.1.3 Estudis dels fluxos de potència

L'estudi del flux de potència consisteix en analitzar numèricament el flux de potència elèctrica que circula per una xarxa (Power-flow study. Wikipedia). L'anàlisi numèric se centra en obtenir els voltatges i les potències dels diferents elements de la xarxa com bateries,

generadors, branques, etc., en unes certes condicions de generació i demanda. El càlcul dels fluxos de potència és important per planificar futures expansions de la xarxa i optimitzar-la.

Una planificació òptima de la xarxa és la configuració de la xarxa amb la potència elèctrica necessària per satisfer la demanda de tots els elements que formen part d'aquesta i que optimitza un conjunt d'objectius d'operació de la xarxa. Existeixen *softwares*, anomenats *power flow solvers,* que permeten modelar xarxes i calcular els fluxos de potència amb diferents mètodes. Els algoritmes d'optimització que s'implementaran en el servei web de planificació d'aquest projecte utilitzaran aquests *softwares* per calcular els fluxos de potència de les xarxes. Aquests resultats seran avaluats en base a determinats paràmetres i serviran per buscar la planificació òptima de les xarxes.

2.2 RESOLVD: motivació i objectius

La Unió Europea ha fixat una sèrie d'objectius per les següents dècades en matèria d'eficiència energètica i generació renovable que tenen un impacte directe en el sector de la distribució elèctrica:

- Incrementar l'ús de les fonts d'energies renovables.
- Augmentar l'eficiència de les xarxes de distribució.
- Reduir les pèrdues de les actuals infraestructures elèctriques.
- Invertir en noves infraestructures i dispositius per incrementar la generació d'energia elèctrica.

El projecte RESOLVD neix per ajudar a complir aquests objectius mitjançant la generació distribuïda d'energia amb un gran ús de fonts d'energies renovables (About the project. The Resolvd Project). El projecte té la intenció de proveir mecanismes per construir una xarxa elèctrica més eficient, flexible i intel·ligent pel benefici dels consumidors.

Partint de l'actual xarxa elèctrica, l'objectiu global del projecte és millorar l'eficiència i les capacitats de les xarxes de generació distribuïda de baix voltatge. RESOLVD es centra en les xarxes de baixa tensió ja que és on resideix més potencial per incrementar l'ús d'energies renovables. A més, són xarxes més eficients degut a que els punts de generació són més propers als llocs de consum.

El projecte engloba objectius científics, tecnològics i de negoci:

- Dissenyar, desenvolupar i provar nous dispositius electrònics per operar amb les xarxes de baix voltatge i dotar-les de capacitats intel·ligents per transformar-les en *smart grids*.
- Dotar la xarxa de mecanismes de gestió perquè sigui capaç de reconfigurar-se davant de fallades, canvis de generació o demanda amb l'objectiu de maximitzar la seva eficiència i fiabilitat.
- Implementar eines de monitorització i explotació de les dades per millorar la gestió de la xarxa i dissenyar serveis per calcular la planificació òptima dels components d'aquestes xarxes elèctriques intel·ligents.
- Integrar els diferents components del projecte en una plataforma i estandarditzar la gestió de les *smart grids*.
- Potenciar nous models de negoci i regulacions relatius a la gestió de l'energia.

Com s'ha explicat en l'apartat anterior, el grup eXiT de la Universitat de Girona s'encarregarà del desenvolupament de les eines de planificació òptima de les xarxes, que s'engloba en la secció SCO 3 de la Figura 2.

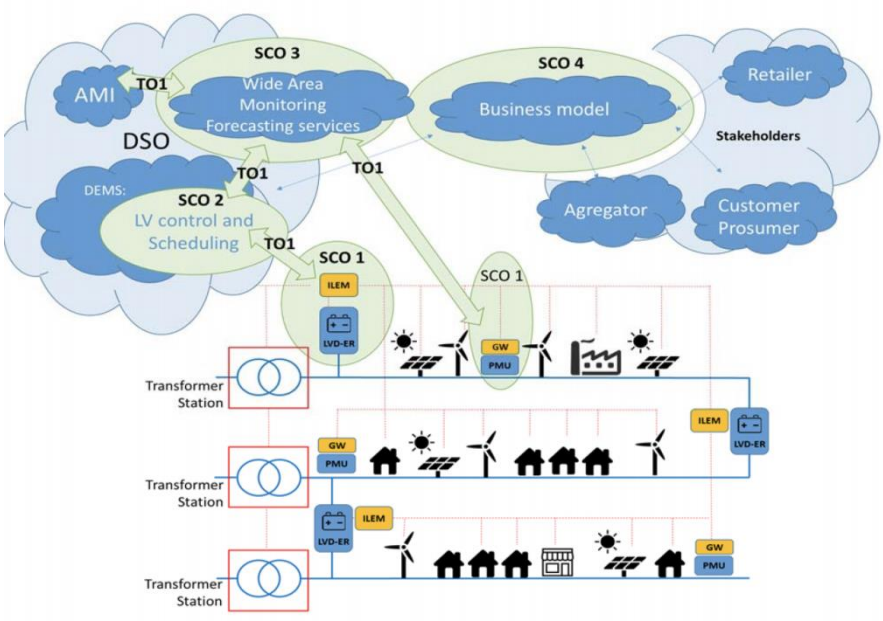

Figura 2. Mapa amb els diferents objectius del projecte RESOLVD (About the project. The Resolvd Project)

2.3 Algoritmes metaheurístics

Els algoritmes metaheurístics són aquells algoritmes que apliquen un cert grau d'aleatorietat per trobar la solució òptima, o la més òptima possible, d'un problema complex. Aquest tipus d'algoritmes es solen aplicar en problemes on inicialment no es coneix la solució òptima, però es poden generar noves solucions i avaluar-les en funció d'una heurística determinada.

Existeixen moltes classes d'algoritmes metaheurístics: algoritmes voraços, cerca tabú o optimització aleatòria, entre molts altres. Depenent de la classe, els algoritmes mantenen una sola solució candidata que van substituint a mesura que es van descobrint millors o una població de candidates que es va modificant. Els algoritmes basats en població són un subconjunt d'algoritmes metaheurístics caracteritzats per mantenir una població de solucions candidates. Les solucions que formen la població no són independents, sinó que s'afecten entre elles influint en la manera en que descobreixen l'espai de cerca. Depenent de l'algoritme, aquesta població és substituïda a cada iteració per una nova generació de solucions candidates o s'altera per produir-ne de noves, el que es coneix com a mutació. Alguns exemples d'algoritmes basats en població són els algoritmes genètics, les estratègies evolutives o l'optimització per eixam de partícules (*Particle Swarm Optimization*) (Torrent-Fontbona F. , 2015).

Donat que el nombre de solucions que l'espai de cerca pot contenir, és habitual limitar l'execució d'aquests algoritmes a un nombre màxim d'iteracions o a un temps límit (Torrent-Fontbona F. , 2015).

3. Metodologia de treball

El projecte s'ha desenvolupat seguint una metodologia de treball Agile (Tena, 2018). Aquest tipus de metodologies aporten flexibilitat i rapidesa al desenvolupament del software amb l'objectiu de satisfer al màxim els requeriments plantejats, els quals evolucionen constantment.

Altres metodologies, com la del model en cascada, estructuren el desenvolupament de software en varies fases lineals i sense retorn entre elles, prioritzant llargs processos i documentació. Això pot comportar que si els requeriments varien durant el transcurs del projecte, aquests canvis no quedin reflectits en el software final i, per tant, el producte final no satisfaci el que necessita el client.

Les metodologies Agile, en canvi, es basen en un desenvolupament incremental i iteratiu, on l'anàlisi dels requeriments, el disseny, el desenvolupament, les proves i la interacció amb el client s'inclouen a cada iteració. D'aquesta manera, s'aconsegueix adaptar el projecte al que necessita el client en tot moment. La Figura 3 mostra gràficament aquest procés iteratiu:

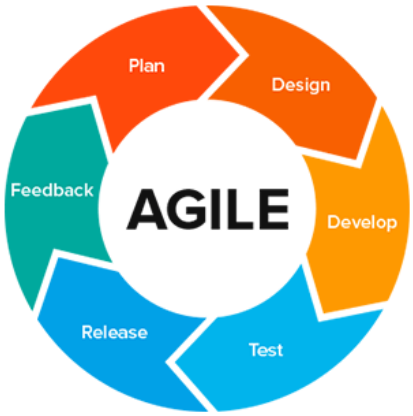

Figura 3. Representació esquemàtica d'una iteració Agile. Font: Just Digital Agency

Característiques de les metodologies Agile

- Fragmentació de les tasques.
- Entregues ràpides i contínues.
- Prioritització del software funcional per davant de la documentació.
- Emfatitzen la comunicació entre el client i els desenvolupadors de manera que qualsevol canvi en els requeriments inicials es té en compte.

- Alt grau d'implicació personal dels integrants de l'equip de desenvolupament.

Avantatges de les metodologies Agile

- Molt efectives en projectes on els requeriments inicialment no estan completament definits o poden variar.
- S'obtenen resultats abans.
- El software resultant satisfarà al màxim els requeriments del client, ja que aquest ha intervingut durant tot el procés de desenvolupament.
- Al realitzar proves de funcionament al final de cada entrega, es minimitzen els errors que pugui tenir el software final.

Desavantatges de les metodologies Agile

- La falta de documentació dota a aquestes metodologies de gran flexibilitat, però com a conseqüència provoca falta d'informació i es ressent la planificació a llarg termini.
- Degut a les nombroses entregues continues, pot augmentar l'estrès dels desenvolupadors i conseqüentment els resultats de les entregues poden empitjorar.
- La participació del client és molt important, pel que si el client no manté una participació activa durant el procés de desenvolupament pot perjudicar el resultat final.
- La necessitat de comunicació permanent entre totes les parts del projecte pot consumir molt de temps en forma de reunions.

Aplicació de la metodologia

Des de l'inici del projecte s'ha mantingut una reunió amb el tutor on s'han revisat les tasques fetes, s'han resolt els dubtes que hagin pogut sorgir durant el desenvolupament i s'han plantejat les tasques a fer per la següent reunió. D'aquesta manera el tutor ha conegut l'estat del projecte en tot moment i el resultat final s'ha ajustat al màxim al que es volia assolir inicialment.

4. Requeriments

El projecte consisteix en el disseny i desenvolupament de dues parts ben diferenciades:

- El servei web amb les funcionalitats necessàries per llançar algoritmes d'optimització sobre xarxes elèctriques. A partir d'ara, el servei web també pot ser referenciat com a servidor a la resta del document.
- Una aplicació client amb interfícies d'usuari per interactuar amb el servei web.

En els següents apartats es llisten els requeriments funcionals i no funcionals de cada part.

4.1 Requeriments del servidor

A continuació es llisten els requeriments del servei web.

Requeriments funcionals

- El servei web ha de tenir un sistema d'usuaris. S'han de poder restringir les funcionalitats accessibles des del servei web als usuaris no autenticats.
- S'han de poder classificar els usuaris en funció d'un rol. Es distingiran dos rols: administrador i usuari comú.
- S'han de poder registrar nous usuaris a través del servei web.
- Els usuaris han de poder gestionar la seva informació. Els usuaris administradors, a banda de poder gestionar la seva pròpia informació, podran gestionar la dels usuaris comuns i eliminar-los del sistema.
- Quan es registri un nou usuari s'haurà d'enviar un email d'activació al correu entrat en el registre. Un usuari no activat no podrà fer servir les funcionalitats restringides del servei web.
- El servei web ha de permetre la pujada de xarxes elèctriques i guardar-les a la base de dades. El servei ha de permetre pujar les xarxes en fitxers Excel o bé amb el format de la base de dades.
- El servei web ha de poder funcionar amb xarxes modelades amb diferents llibreries de càlcul de fluxos de potència (*power flow solvers)*.
- La informació de les xarxes guardades ha de ser accessible a través del servei web i s'ha de poder descarregar el fitxer original pujat.
- S'han de poder executar diferents algoritmes d'optimització sobre les xarxes guardades.
- Els algoritmes han de poder utilitzar diferents mètodes de càlcul de fluxos de potència.

- Les solucions que generin els algoritmes han de poder ser avaluades per diferents mètodes d'avaluació.
- Els algoritmes podran definir paràmetres addicionals necessaris per la seva execució.
- El servei web ha de permetre consultar els resultats de les planificacions. Els resultats han d'incloure tots els paràmetres de l'execució: xarxa, algoritme, mètode de càlcul utilitzat, etc.

Requeriments no funcionals

- El llenguatge de programació per desenvolupar el servei web ha de ser Python.
- Les comunicacions entre el client i el servei web hauran de ser encriptades.
- Les funcionalitats pròpies del servei web han d'estar separades de les funcionalitats de planificació i optimització.

4.2 Requeriments del client

L'aplicació client ha de servir principalment per provar les diferents funcionalitats del servei web i visualitzar les planificacions resultants.

Requeriments funcionals

- Pantalla d'autenticació.
- Pantalla de registre.
- Pantalla d'inici. En aquesta pantalla apareixeran resums de les últimes xarxes pujades per l'usuari i de les últimes execucions.
- Pantalla per pujar noves xarxes. L'usuari ha de poder especificar el tipus de xarxa que vol pujar i el format: fitxer estandarditzat o format de la base de dades.
- Llistat de xarxes on apareixen totes les xarxes pujades per l'usuari.
- Pantalla de visualització d'una xarxa, on es pugui consultar tota la informació de la xarxa, juntament amb una representació visual d'aquesta.
- Pantalla per poder llançar noves planificacions. L'usuari ha de poder triar la xarxa, l'algoritme d'optimització, el solucionador i la funció d'avaluació.
- Llistat de les planificacions llançades per l'usuari i el seu estat.
- Pantalla de visualització de la planificació, on es pugui consultar tota la informació i una comparativa gràfica entre la xarxa original i la xarxa resultant de l'optimització.
- Pantalla de perfil, on l'usuari pugui editar les seves dades.
- Pantalles exclusives pels administradors per la gestió d'usuaris:
    - Llistat dels usuaris del servei.

- Pantalla d'edició.
- Pantalla de creació de nous usuaris.

Requeriments no funcionals

- El client ha de ser una aplicació web.

5. Disseny

En aquest apartat es descriuen els principals punts de disseny que s'han plantejat en aquest projecte tant a la part del servidor com a la del client.

5.1 Estructura del projecte i REST

Com ja s'ha vist anteriorment, l'estructura del projecte es basa en dues parts: un servei web i una aplicació client. Aquesta estructura permet una separació de responsabilitats: la gestió dels recursos i el pes lògic de l'aplicació es centralitza en el servidor o servei web, mentre que el client només s'encarrega de les interfícies d'usuari. Això proporciona una sèrie d'avantatges:

- Independència de les plataformes i llenguatges del client i del servidor, permetent diferents implementacions.
- Afavoreix l'escalabilitat de l'aplicació.
- Permet que els diferents components evolucionin de forma independent.

El servidor s'ha dissenyat com una API REST (API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos, 2016). REST (*representational state transfer*) és, de manera resumida, la definició d'una interfície de comunicació entre sistemes, normalment via el protocol HTTP, que permeti accedir a recursos i executar diferents operacions, els quals poden ser retornats en qualsevol format: XML, JSON, entre d'altres. Sorgeix com a alternativa a altres protocols d'intercanvi de dades molt utilitzats, com SOAP, els quals tenen molta capacitat però alhora són molt complexos. REST en canvi és una solució més simple per manipular i intercanviar dades, fet que l'ha convertit en un dels estàndards més utilitzats a dia d'avui per construir serveis web.

Característiques bàsiques dels serveis REST

Protocol client - servidor sense estat: Cada petició HTTP conté tota la informació necessària per realitzar l'operació sol·licitada, de manera que no hi ha estats previs que el servidor o el client han de conèixer per executar la petició.

Identificadors únics per cada recurs: L'accés a cada recurs del servei està definit per un identificador de recurs uniforme (URI) únic.

Interfície uniforme de manipulació de dades: En un servei REST, l'acció a realitzar sobre els recursos s'especifica a partir de l'operació HTTP de la petició. Les operacions més comunes

que es poden realitzar en qualsevol sistema REST són POST (creació), GET (consulta), PUT (editar), DELETE (eliminar).

Utilització de recursos hipermedia: Part de la informació de les respostes han de ser enllaços a altres recursos del servei associats amb la petició. D'aquesta manera es facilita la navegació i l'accés als recursos del servei REST, el que es coneix com *Hypermedia As The Engine Of Application State (*HATEOSA).

5.1.1 Disseny de l'API REST

En aquesta secció es descriuen els dos punts més importants que s'han plantejat durant el disseny del servei web com a API REST: el sistema d'autenticació i les rutes per accedir a les funcionalitats que ofereix.

Sistema d'autenticació

Un dels requisits del projecte és la restricció d'alguns recursos del servei en cas que la petició no provingui d'un usuari autoritzat. En aplicacions on el servidor també s'encarrega de la renderització de les interfícies d'usuari, és habitual guardar en variables de sessió i cookies la informació de l'usuari per a posteriors peticions autenticades. REST es basa en un protocol sense estat, per tant el servidor no pot guardar cap mena d'informació sobre si un usuari està autenticat o no. La sol·licitud a un recurs REST ha de contenir tota la informació necessària en la petició HTTP, pel que caldrà enviar les credencials de l'usuari per cada petició. El principal avantatge de l'autenticació en protocols sense estat és l'augment de seguretat en front atacs que manipulen les variables de sessió que s'envien al servidor com els atacs *Cross-Site Request Forgery*.

Enviar les credencials de l'usuari a cada petició pot comportar altres problemes de seguretat, ja que el client s'haurà de guardar les credencials per enviar-les en cada petició i pot existir el risc de que siguin extraviades. Com a solució aquest problema és habitual implementar un sistema d'autenticació basat en *tokens*.

El funcionament és senzill. El client envia una primera petició amb les credencials de l'usuari per autenticar-se. Si les credencials són vàlides, es genera un *token* associat a aquell usuari. Un *token* es podria considerar una firma xifrada que serveix al servidor per identificar un usuari. Al realitzar noves peticions, el client inclou el *token* en la capçalera de la petició HTTP. El servidor s'encarrega de la seva validació i determina si es poden utilitzar els recursos sol·licitats en funció de l'usuari associat al *token*. Com que els *tokens* són emmagatzemats en

el client, el servidor no té informació de l'estat de l'usuari i compleix els requeriments de disseny d'una API REST.

En termes de seguretat és molt recomanat, gairebé imprescindible, que les peticions al servei web es facin mitjançant el protocol HTTPS per evitar robatoris de credencials o del *token.* En el cas dels *tokens*, és també molt habitual fixar un temps de validesa per minimitzar els riscos en el cas que s'aconseguís extraviar un *token*.

Rutes del servei web

A continuació es llisten tots els recursos i les operacions disponibles a través de l'API REST del servei web[1]. Respecte la sintaxi de les URI, les cadenes de text que comencen per : representen paràmetres d'entrada de la petició.

MÈTODE	URI	DESCRIPCIÓ
POST	/user	Registra un nou usuari a la base dades.
GET	/user/:id	Recupera la informació d'un usuari.
PUT	/user/:id	Actualitza un usuari.
DELETE	/user/:id	Elimina un usuari de la base de dades.
GET	/user/:id/send_activation_link	Envia un enllaç d'activació a l'usuari.
GET	/user/:id/activate	Activa el compte d'un usuari.
GET	/token	Genera un nou token per a un usuari.
POST	/grid	Registra una nova xarxa la base de dades.

[1] Consultar ANNEX 2 per informació més detallada de cada ruta.

GET	/grid/:id	Consulta la informació d'una xarxa.
GET	/grid/<id>/download	Descarrega el fitxer original d'una xarxa.
DELETE	/grid/:id	Elimina una xarxa de la base de dades.
GET	/grids	Retorna un llistat amb totes les xarxes d'un usuari.
GET	/grid/types	Retorna un llistat amb tots els tipus de xarxes disponibles.
POST	/scheduler	Crea i executa una nova planificació.
GET	/scheduler/:id	Consulta una planificació.
POST	/scheduler/:id/execute	Executa una planificació.
DELETE	/scheduler/:id	Elimina una planificació.
GET	/schedulers	Retorna una llista amb totes les planificacions d'un usuari.
GET	/scheduler/:id/log	Recuperar el registre d'execució d'una planificació.

5.2 Disseny dels components del servidor

A continuació es mostra l'estructura del servei web i es descriuen els components que en formen part.

5.2.1 Diagrama de classes

La Figura 4 mostra el diagrama de classes amb els diferents components del servei web. Aquest és una versió simplificada del diagrama complet, sense classes d'utilitats ni implementacions concretes. L'objectiu és mostrar els principals components del servei web per donar una idea de la seva estructura:

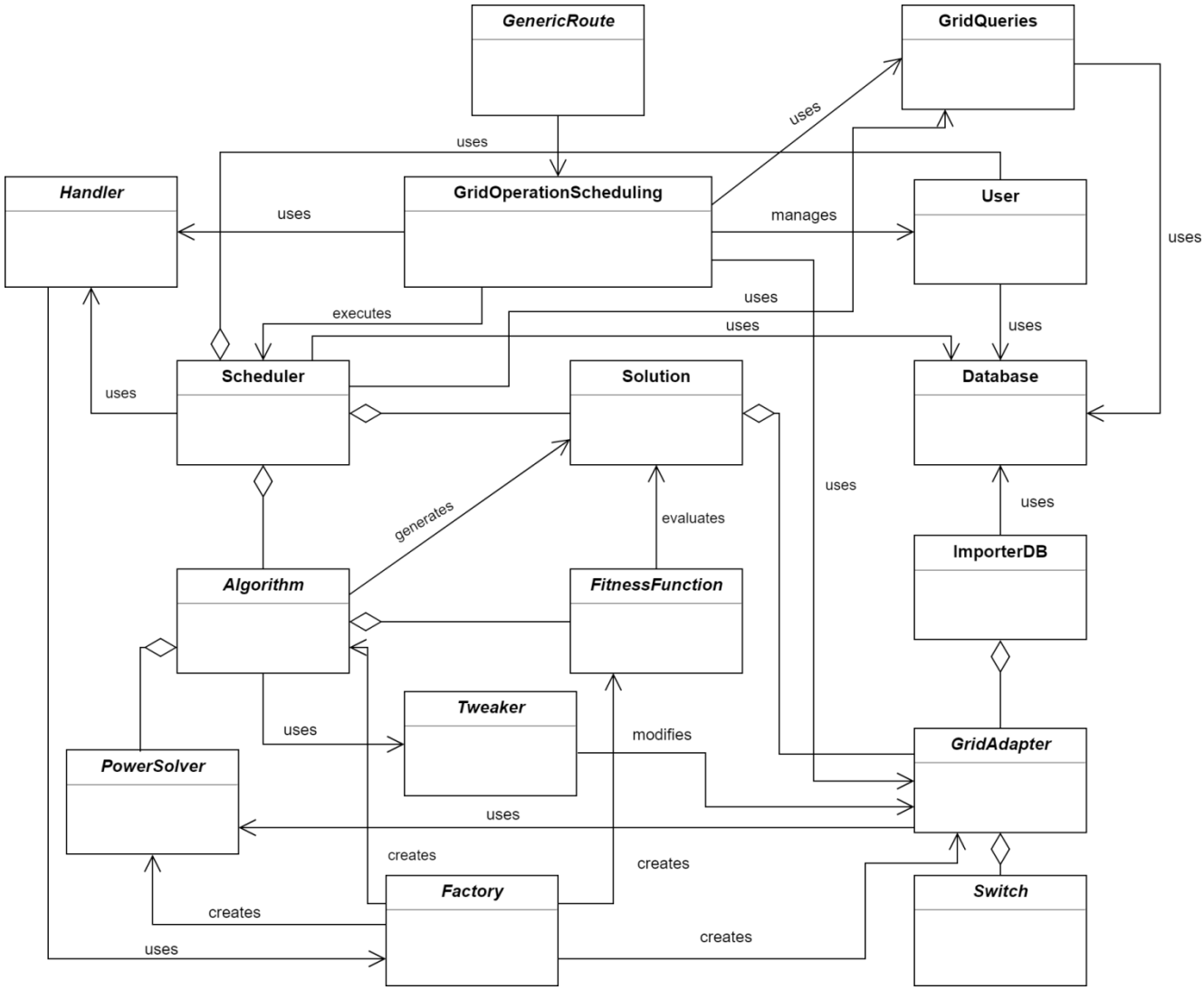

Figura 4. Diagrama de classes dels components del servei web

5.2.2 Descripció dels components

En aquesta secció es descriuen els components mostrats en la figura anterior i la seva funcionalitat dins el servei web.

Rutes (GenericRoute)

Aquesta classe és l'encarregada de processar i validar el paràmetres d'entrada de les peticions que arriben al servidor. Un cop validats els paràmetres d'entrada, la classe crida al mètode de la interfície del servei web necessari per executar la petició sol·licitada. Finalment envia la resposta generada pel servei web al client amb el format adequat.

Interfície del servei web (GridOperationScheduling)

La interfície del servei web. S'encarrega de gestionar les peticions del servei web cridant als components necessaris per realitzar cada operació.

Factoria (Factory)

Instancia els diferents components del servei web. L'ús de les factories permet separar la creació dels objectes dels components que els fan servir, amb l'objectiu de que les subclasses de les factories puguin refinar l'objecte instanciat.

Gestor (Handler)

S'encarrega d'instanciar la factoria adequada per generar cada component del servei web i retornar la instància creada per la factoria.

Base de dades (Database)

Classe encarregada de la comunicació amb la base de dades. Tots els components del servei web accedeixen i realitzen operacions a la base dades a través d'aquesta classe. El tractament d'errors produïts al realitzar operacions a la base de dades queda encapsulat en aquesta classe.

Importador (ImporterDB)

S'encarrega de transformar les xarxes al format de la base de dades i de guardar-les.

DbQueries

En aquesta classe s'implementen mètodes generals d'accés a la base de dades pels components que els necessitin, com recuperar la informació d'una xarxa o obtenir el llistat de xarxes d'un usuari en concret.

Usuari (User)

Classe encarregada de les operacions bàsiques de gestió dels usuaris del sistema: creació, consulta, edició i eliminació.

Planificador (Scheduler)

Aquesta classe s'encarrega de llançar i gestionar les planificacions. Les accions que pot fer aquesta classe són:

- Registrar una nova planificació a executar.
- Realitzar una planificació: consisteix en executar l'algoritme d'optimització seleccionat i recuperar la millorar solució que genera.
- Guardar els resultats d'una planificació a la base de dades.
- Recuperar informació d'una planificació de la base de dades.

Algoritme (Algorithm)

Les subclassses d'aquesta classe implementen els algoritmes d'optimització. Els seus atributs principals són la xarxa a la que aplicar l'algoritme, el solucionador de potències que es farà servir i la funció d'optimització per avaluar les solucions. En general l'execució d'un algoritme consta dels següents passos:

- Preparació dels paràmetres per l'execució.
- Executar l'algoritme amb *n* iteracions.
    - A cada iteració:
        - Realitzar una sèrie de modificacions sobre la xarxa a determinar segons l'algoritme.
        - Generar noves possibles solucions a partir de les modificacions sobre la xarxa.
        - Puntuar la nova solució mitjançant la funció d'avaluació
- Retornar la millor solució trobada.

Adaptador (GridAdapter)

Les xarxes són implementades per les llibreries de càlcul de fluxos de potència (*power flow solvers)*. Inicialment, el servei web només utilitza una llibreria. La resta de components del servei web com els algoritmes, funcions d'avaluació, modificadors, etc. podrien interactuar amb ella directament.

Per exemple, si una funció d'avaluació i un modificador necessitin obtenir els voltatges d'una bateria cridarien al propi mètode la llibreria com es mostra a la Figura 5:

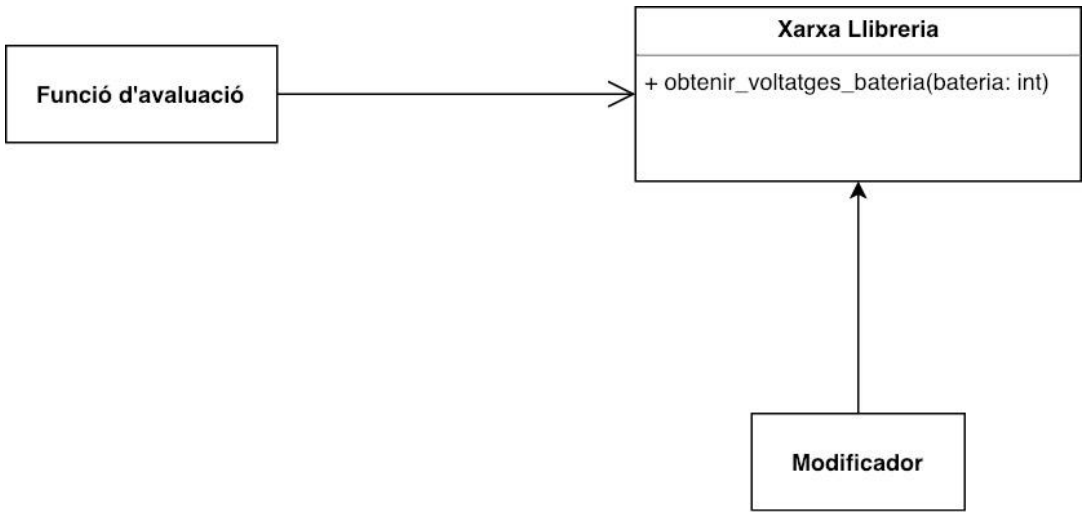

Figura 5. Exemple d'accés al mètode *obtenir_voltatges_bateria* per part d'un modificador i una funció d'avaluació

Aquest ús genera un alt acoblament dels components amb la llibreria. La llibreria és mantinguda per un tercer i és possible que rebi actualitzacions i modificacions en el futur. Es podria donar la situació que en una nova versió de la llibreria el mètode *obtenir_voltatges_bateria* necessités més paràmetres d'entrada o es substituís per un nou mètode, anomenat *obtenir_voltatges_bateria_nou.* Per tant, caldria substituir totes les crides al mètode antic pel nou mètode. També es podria donar el cas que en un futur es volgués que el servei web també fos compatible amb xarxes implementades per una altra llibreria de càlcul, amb una implementació totalment diferent de la primera llibreria. És necessari doncs que els components del servei web no coneguin les diferents implementacions de les xarxes: cal definir una interfície amb la qual els components puguin interactuar amb elles.

Els adaptadors implementen els mètodes definits per la interfície en funció de la llibreria de càlcul associada. D'aquesta manera s'aconsegueix desacoblar les llibreries de càlcul del servei web, permetent la coexistència de vàries llibreries, encapsulant els canvis i delimitant els errors associats al seu ús dins els adaptadors.

La Figura 6 exemplifica com la resta d'elements del servei web interactuen amb les xarxes a través dels adaptadors:

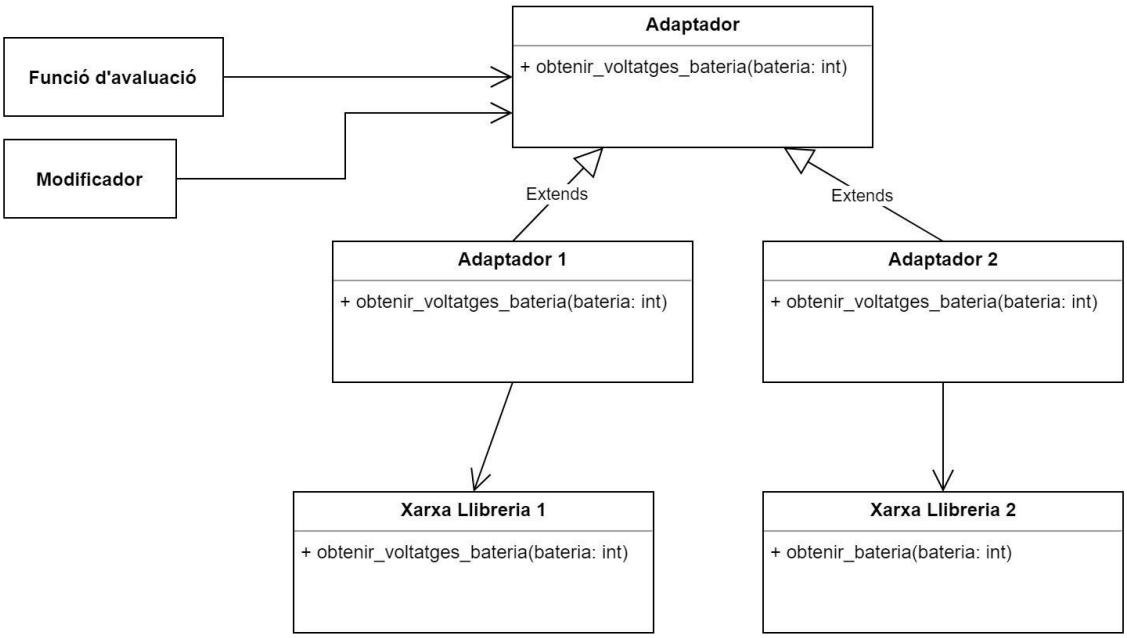

Figura 6. Exemple de l'ús d'adaptadors per utilitzar les xarxes

A part dels adaptadors, hi ha dos components més del servei web que interactuen directament amb les llibreries de càlcul: els solucionadors de potències i els interruptors. A les descripcions respectives dels components s'explica el motiu pel qual no segueixen aquest patró.

Solucionador de potències (Power solver)

S'encarrega de calcular els fluxos de potència d'una xarxa. Cada implementació concreta d'aquesta classe aplica un mètode concret d'una llibreria de càlcul de fluxos de potència, pel que és necessari que la xarxa estigui implementada amb la mateixa llibreria que fa servir el solucionador de potències. Un cop calculats els fluxos de potència, assigna els resultats a la xarxa.

Interruptor (Switch)

Controla una branca de la xarxa. Caldrà implementar una subclasse específica per cada llibreria, ja que l'interruptor manté una referència a l'objecte directa a la branca, implementada de manera diferent en funció de la llibreria.

Solució (Solution)

Representa el resultat de l'execució d'un algoritme d'optimització. Una solució està formada per una xarxa solucionada i una puntuació. La puntuació s'obtindrà a partir de la funció amb la que s'avaluï la solució.

Funció d'avaluació (Fitness solution)

Aquesta classe s'encarrega de l'avaluació de les solucions trobades. L'avaluació d'una solució consisteix en valorar diferents paràmetres dels resultats retornats pels solucionadors de problemes sobre una xarxa. Cada funció d'avaluació determina quins paràmetres valorar i assigna una puntuació a la solució. El format de la puntuació també està definit per la funció: enter, matriu, vector, etc.

A més a més, aquesta classe també permet comparar solucions i determinar si una solució és òptima. Una solució es considera òptima quan assoleix la màxima puntuació segons la funció.

Modificador (Tweaker)

Aquesta classe implementa accions per modificar les xarxes, com per exemple activar/desactivar branques de la xarxa o modificar les potències dels generadors.

5.3 Estructura de les dades

En aquesta secció es descriuen les estructures que es guarden a la base de dades.

5.3.1 Estructura d'una xarxa

Les xarxes elèctriques estan formades per illes. Una illa és un circuit tancat dins la xarxa, de manera que pot contenir una o vàries illes. Una illa es pot representar com un graf format per un conjunt de nodes (busos) i arestes (branques):

- Busos: Són els contenidors de tots els possibles dispositius que pot tenir la xarxa: càrregues, generadors, bateries, etc.
- Branques: Connecten els diferents busos d'una illa i permeten el pas del corrent.

Tanmateix, la xarxa conté interruptors. Els interruptors permeten habilitar o inhabilitar les branques per alterar el curs del corrent. Cada interruptor té una branca associada que controla.

L'estructura representada a la Figura 7 és amb la qual el servei web espera treballar. No obstant, com s'ha explicat en l'apartat anterior, la implementació i l'estructura de les xarxes

depenen del *power flow solver* que la modeli. Per tant, cal que els adaptadors facin les operacions necessàries per tal de simular aquesta estructura.

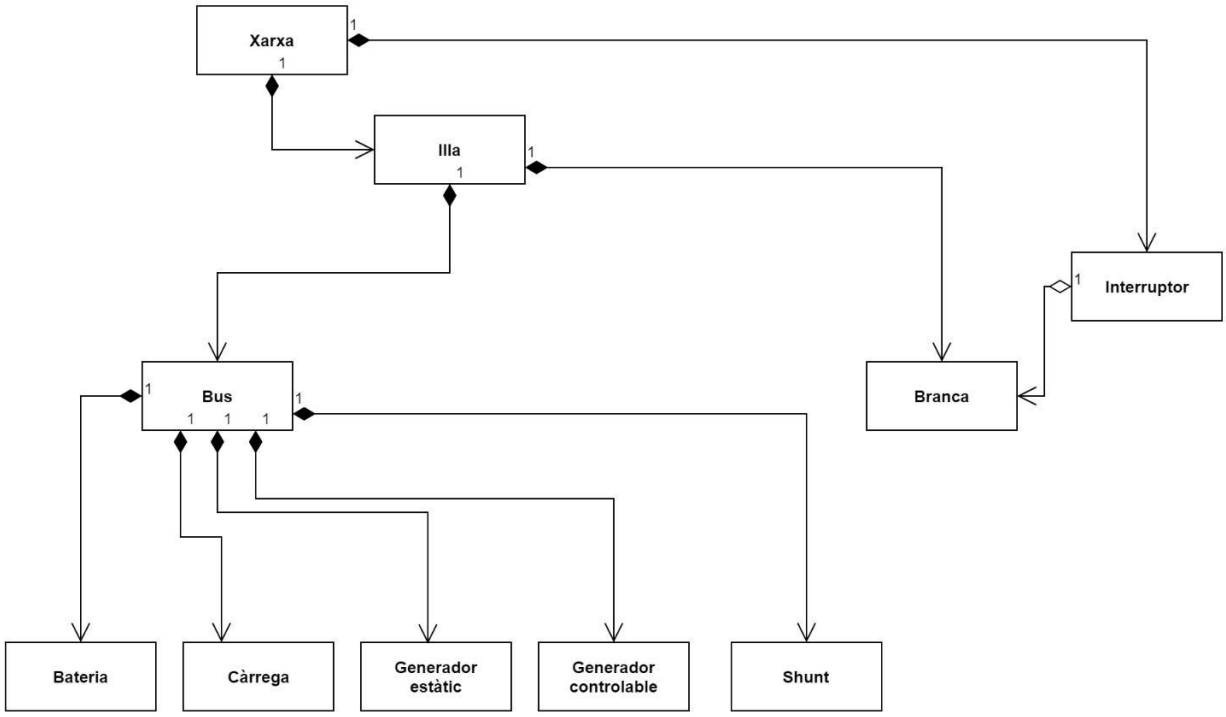

Figura 7. Estructura d'una xarxa del servei web

Valors per instants de temps

Els dispositius de la xarxa poden contenir diferents valors de les seves variables per diferents instants de temps. Per exemple, una bateria pot tenir el valor de la seva potència registrat a les 00:00 i el valor registrat a les 00:05, de manera que es pot veure l'evolució de la bateria al llarg del temps. Els interruptors també tenir diferents estats per cada instants de temps.

5.3.2 Estructura de la base de dades

Degut a la composició de les xarxes i al fet de tenir valors diferents per instant de temps en els dispositius, s'ha optat per fer servir una base de dades no relacional (noSQL) enfront a una base de dades relacional tradicional.

La base de dades s'estructura en tres grans entitats: els usuaris (User), les xarxes (Grid) i les planificacions (Scheduling) com es mostra en la Figura 8:

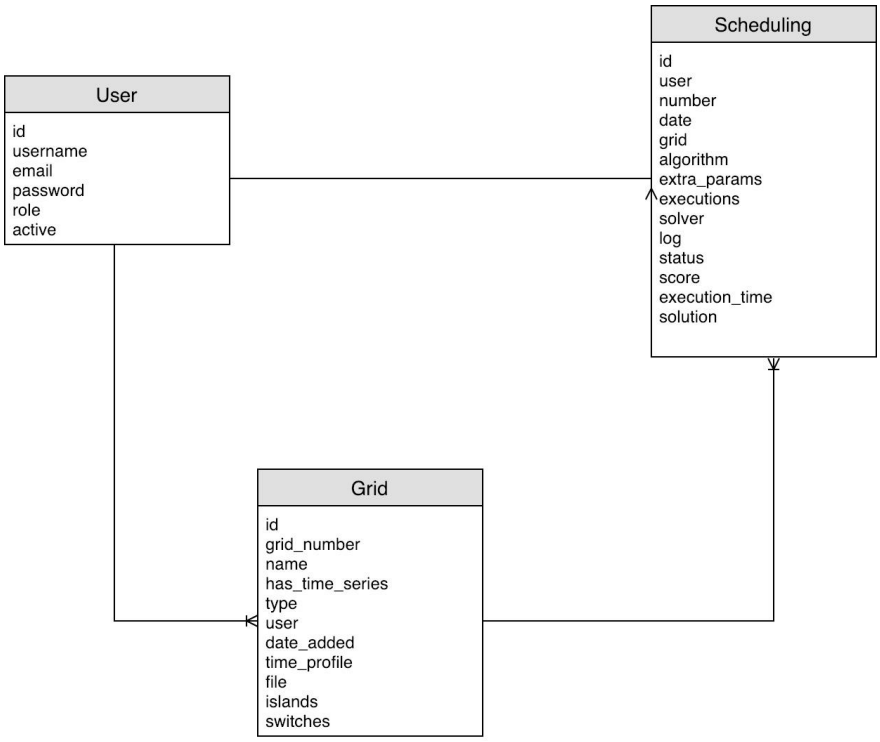

Figura 8. Estructura de la base de dades

Descripció dels camps d'un usuari

- id: Identificador de la base de dades.
- username: Nom de l'usuari. Camp únic per usuari.
- email: Igual que el camp username és únic per cada usuari.
- password
- role: Rol de l'usuari dins el servei web.
- active: Indica si és un usuari actiu del servei web.

Descripció dels camps d'una xarxa

- id: Identificador de la base de dades.
- grid_number: Indica el número de xarxa relativa a l'usuari propietari.
- name: Nom de la xarxa.
- has_time_series: Indica si la xarxa consta de valors per diferents instant de temps.
- type: Llibreria de càlcul de fluxos amb la que la xarxa està implementada.
- user: Referència a l'usuari propietari de la xarxa.

- date_added: Data de creació de la xarxa.
- time_profile: Llista dels instants de temps disponibles a la xarxa.
- file: Fitxer original de la xarxa en cas que fos pujada a partir d'un fitxer Excel.
- islands: Llistat de les illes de la xarxa. Cada illa conté els següents camps:
    - id: Identificador de l'illa dins la xarxa.
    - Busos: Llistat dels busos de l'illa.
    - Branques: Llistat de branques.
- switches: Llistat d'interruptors.

Camps d'un bus

- id: Identificador del bus dins la xarxa.
- type: Indica el tipus del bus.
- name: Nom del bus.
- active: Indica si el bus està activat o deshabilitat.
- is_slack: Indica si el bus és de tipus *slack.*
- Vnom: Voltatge nominal del bus en kV.
- vmin: Voltage mínim expressat en tant per u.
- vmax: Voltage màxim expressat en tant per u.
- qmax: Màxima energia reactiva.
- qmin: Mínima energia reactiva.
- Zf: Impedència del circuit, expressada en forma complexa i en Ohms.
- x: Coordenada x del bus.
- y: Coordenada y del bus.
- dispatch_storage:
- loads: Llistat de càrregues.
- batteries: Llistat de bateries.
- controlled_generators: Llistat de generadors controlables.
- static_generators: Llistat de generadors estàtics.
- shunts: Llistat de shunts.

Camps d'una càrrega

- id: Identificador de la càrrega.
- type: Tipus de càrrega.
- name: Nom de la càrrega.
- active: Indica si la càrrega està activa.

- I: Intensitat en forma complexa expressada en kA (kiloamperes).
- Iprof: Llistat de valors de I per cada instant de temps de la xarxa.
- S: Potència en forma complexa expressada en MVA (megavoltiamperes).
- Sprof: Llistat de valors de S per cada instant de temps de la xarxa.
- Z: Impedància en forma complexa expressada en Ohms.
- Zprof: Llistat de valors de Z per cada instant de temps de la xarxa.

Camps d'un generadors estàtic

- id: Identificador del generador.
- name: Nom del generador.
- type: Tipus de generador.
- active: Indica l'estat del generador.
- S: Potència en forma complexa expressada en MVA (megavoltiamperes).
- Sprof: Llistat de valors de S per cada instant de temps de la xarxa.

Camps d'un generador controlable

- id: Identificador del generador.
- type: Tipus de generador.
- name: Nom del generador.
- active: Indica l'estat del generador.
- P: Potència activa expressada en MW.
- Pprof: Llistat de valors de P per cada instant de temps de la xarxa.
- qmax: Màxima energia reactiva.
- qmin: Mínima energia reactiva.
- Snom: Potència nominal en MVA.
- Vset: Valor d'ajust de la tensió expressat en tant per u.
- Vset_prof: Valors de Vset per cada instant de tems de la xarxa.

Camps d'una bateria

- id: Identificador de la bateria.
- type: Tipus de bateria.
- name: Nom de la bateria.
- active: Indica l'estat de la bateria.
- Enom: Energia nominal en MWh.
- max_soc: Màxim estat de càrrega de la bateria expressat en tant per u.

- min_soc: Mínim estat de càrrega de la bateria expressat en tant per u.
- soc: Estat de càrrega de la bateria.
- charge_effiiency: Eficiència de la bateria al carregar.
- charge_per_cycle: Capacitat de càrrega per cicle expressada en tant per u.
- discharge_efficiency: Eficiència de la bateria al descarregar.
- discharge_per_cycle: Capacitat de descàrrega per cicle expressada en tant per u.
- P: Potència activa expressada en MW.
- Pprof: Llistat de valors de P per cada instant de temps de la xarxa.
- qmax: Màxima energia reactiva.
- qmin: Mínima energia reactiva.
- Snom: Potència nominal en MVA.
- Vset: Valor d'ajust de la tensió expressat en tant per u.
- Vset_prof: Valors de Vset per cada instant de tems de la xarxa.

Camps d'una shunt

- id: Identificador de la shunt.
- type: Tipus de shunt.
- name: Nom de la shunt.
- Y: Admitància expressada en tant per u.
- Yprof: valors de Y per cada instant de temps de la xarxa.

Camps d'una branca

- id: Identificador de la branca.
- type: Tipus de branca.
- name: Nom de la branca.
- active: Estat de la branca.
- bus_from: Identifiador del bus origen.
- bus_to: Identificador del bus destí.
- angle: Angle de la branca expressat en radiants.
- rate: Potència expressada en MVA.
- tap_module: Bus amb tensió controlada.
- y_shunt: Admitància de la branca.
- z_series: Impedància de la branca.
- length: Longitud de la branca.

- mttf: Temps mitjà de fallada expressat en hores.
- mttr: Temps mitjà de reparació expressat en hores.
- vset: Tensió de control si la branca té un bus amb tensió controlada.

Camps d'un interruptor

- id: Identificador de l'interruptor.
- branch: Identificador de la branca que controla.
- active: Estat de l'interruptor.
- profiles: Llistat dels estats de l'interruptor a cada instant de temps.

Descripció dels camps d'una planificació

- id: Identificador de la planificació.
- user: Identificador de l'usuari de la planificació.
- number: Número de planificació relatiu a l'usuari.
- date: Data de creació.
- log: Nom del fitxer amb el registre de la planificació.
- grid: Xarxa amb la que s'ha fet la planificació.
- solver: Nom del solucionador de potències aplicat.
- algorithm: Nom de l'algoritme utilitzat.
- extra_params: Llistat d'objectes clau-valor amb el nom dels paràmetres addicionals requerits per l'algoritme.
- status: Estat de la planificació.
- executions: Nombre d'iteracions que ha realitzat l'algoritme.
- execution_time: Temps que ha tardat la planificació.
- solution: Solució associada a la planificació. Conté els següents camps:
    - score: Puntuació que ha obtingut la solució.
    - changes: Objecte amb les modificacions que ha patit la xarxa durant l'optimització.
    - result_grid: Xarxa resultat de la planificació. Consta de la mateixa estructura descrita a l'apartat anterior.

5.4 Disseny del client

L'aplicació client s'implementarà com una *Single Page Application* (SPA) (Single-page application. Wikipedia). Les SPA són aplicacions web d'un sola pàgina, la qual es descarrega un únic cop al navegador. A mesura que s'interactua amb la pàgina, es modifiquen parts del contingut d'aquesta sense necessitat de refrescar tot el document ni realitzar noves peticions al servidor (en aquest cas el servidor és l'aplicació client), a diferència de les aplicacions webs tradicionals. Això proporciona una millor experiència d'usuari i millora l'eficiència de l'aplicació al realitzar només les peticions necessàries al servidor.

En el cas d'aquest projecte, a mesura que els usuaris naveguin per l'aplicació s'aniran efectuant peticions via AJAX al servei web demanant les dades que l'aplicació client necessiti mostrar: llistat de xarxes de l'usuari, consulta de les planificacions, visualització de les dades d'una xarxa, etc.

6. Tecnologies utilitzades

En aquest apartat es descriuran les eines principals utilitzades en el desenvolupament del projecte. Com en apartats anteriors, la secció es divideix en la part del servidor i en la part del client.

6.1 Servidor

De les tecnologies utilitzades en la implementació del servidor es destaca el llenguatge de programació, el *framework* per llançar el servei web, la base de dades i la llibreria de càlcul de fluxos de potència.

6.1.1 Python

Python és un llenguatge de programació interpretat, d'alt nivell i de propòsit general. Llançat l'any 1991, Python emfatitza una sintaxi molt clara per una bona llegibilitat del codi i reduir els costos de d'aprenentatge i manteniment del codi, incrementant la velocitat de producció de noves aplicacions. Es tracta d'un llenguatge multi-paradigma, ja que permet l'ús de diferents tipus de programació com l'orientada a objectes o la funcional, utilitza un tipatge dinàmic i és multi-plataforma.

Figura 9. Logo de Python. Font: Python

De base, Python compta un gran nombre de funcions i classes de diferents àmbits: processament de cadenes text, protocols d'internet, eines d'enginyeria del software, operacions amb interfícies del sistema, etc. Python és fàcilment extensible gràcies al seu sistema de mòduls que permet afegir noves funcionalitats des del mateix llenguatge o d'altres com C i C++, afavorint la programació modular, l'escalabilitat i la reutilització del codi.

El principal motiu pel que s'ha fet servir Python ha estat que la llibreria de càlcul de fluxos de potència amb la que treballa el servei web està implementada en aquest llenguatge.

6.1.2 Flask

Flask és un *micro-framework* per Python pel desenvolupament web. Basat en la llibreria d'utilitats web de Python anomenada Werkzeug, Flask proporciona les eines necessàries per implementar i llançar aplicacions web en poc temps.

Figura 10. Logo de Flask. Font: Flask

Segons els seus creadors, Flask és considerat un *micro-framework* ja que la seva intenció és mantenir el seu nucli el més simple possible i ampliar les seves funcionalitats mitjançant diferents mòduls a mode d'extensió. D'aquesta manera el servei web s'ajusta a les necessitats del projecte i permet als desenvolupadors tenir més llibertat per construir l'aplicació.

Pel desenvolupament d'aquest projecte la principal extensió que s'ha fet servir ha estat Flask-Restful, una extensió que amplia les funcionalitats del *framework* amb utilitats orientades al desenvolupament d'aquest tipus de servei web.

Els principals motius pels quals s'ha triat Flask per construir el servei web han estat la seva amplia documentació, l'elevat nombre d'extensions i la facilitat per construir i llançar un servei web en poc temps.

6.1.3 mongoDB

mongoDB és una base de dades noSQL que ofereix una gran escalabilitat i flexibilitat. De codi lliure i multiplataforma, mongoDB emmagatzema les dades en documents amb un format similar a JSON, anomenada BSON. Cada document pot contenir diferents camps i estructures de dades. La Figura 12 mostra un exemple de document de la base de dades:

Figura 11. Logo de mongoDB. Font: mongoDB

Figura 12. Estructura d'un document BSON. Font: mongoDB

Característiques principals

- Suport per a cerques per camp, consultes de rangs, agregacions i expressions regulars.
- Sistema d'indexació de camps similar als existents en les bases de dades relacionals.
- Alta disponibilitat de les dades gràcies al seu sistema de replicació.
- Aposta per l'escalabilitat horitzontal gràcies al ser una base de dades distribuïda i al seu sistema de *sharding* . *Sharding* consisteix en dividir una base de dades en varis subconjunts i distribuir-los en diferents màquines o servidors, amb l'objectiu de balancejar la càrrega que un servidor en contextos amb grans volums dades i un nombre elevat de trànsit i operacions. A l'hora d'accedir a les dades, mongoDB

internament s'encarrega d'ajuntar tots els subconjunts i presenta la informació com si es tractés d'un sol conjunt de dades.

Existeixen varies eines per treballar amb mongoDB amb diferents llengüatges de programació. En el cas de Python, la llibreria PyMongo proporciona una API per poder interactuar amb aquest tipus de base de dades.

En un inici es va plantejar que la informació dels diferents elements de l'aplicació es guardessin en una base de dades SQL. A mesura que es va anar avançant en el projecte, es va veure que estructurar la informació de les xarxes en taules SQL era molt costós i poc pràctic a l'hora de consultar la informació guardada.

En canvi, era més senzill transformar les xarxes a diccionaris de Python, els quals a la vegada són fàcilment transformables a format JSON. Al buscar una alternativa noSQL, mongoDB va ser la millor opció degut a l'emmagatzematge de les dades en format BSON, pel que permetia guardar i recuperar la informació de forma més ràpida i eficient. La seva escalabilitat horitzontal també va ser un factor de pes, ja que cada xarxa i les diferents xarxes resultants de llançar les planificacions suposen un gran volum de dades, pel que el seu sistema distribuït potser molt útil a l'hora de balancejar la càrrega dels servidors en un futur.

6.1.4 GridCal

GridCal és un software de càlcul de fluxos de potència en sistemes elèctrics (*power flow solver)*. Permet modelar xarxes i els seus elements i calcular els seus fluxos de potència amb diferents mètodes. El software es pot fer servir a través de la seva interfície gràfica (Figura 14) o com a mòdul de Python.

Figura 13. Logo de GridCal. Font: Github

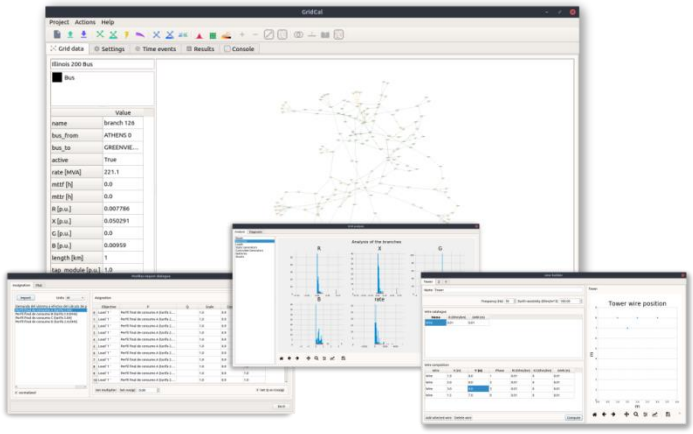

Figura 14. Interfície gràfica de GridCal

Es va decidir utilitzar GridCal perquè és un software lliure i proporciona les eines necessàries per modelar xarxes i calcular els fluxos de potència. L'estructura de les xarxes amb les que treballa el servei web està basada en l'estructura de les xarxes de GridCal. A la Figura 15 es mostra el diagrama de classes d'una xarxa GridCal:

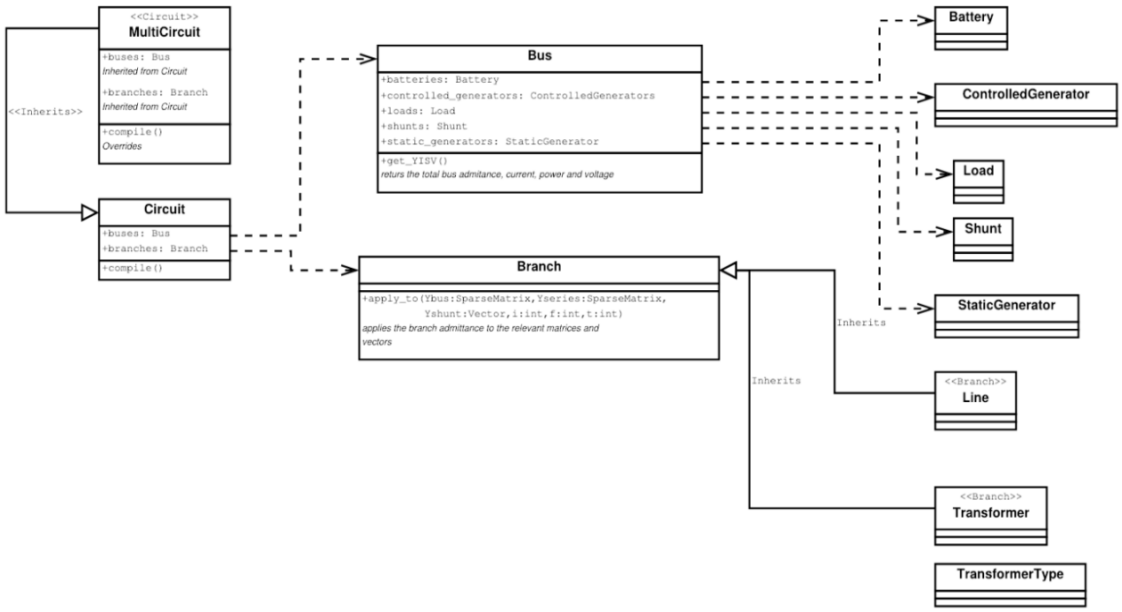

Figura 15. Estructura de les xarxes GridCal

6.2 Client

El client és una aplicació web, pel que s'ha construït amb tecnologies web com HTML, CSS i Javascript per aconseguir interfícies dinàmiques i interactives. En aquest apartat s'expliquen el *framework* utilitzat per la seva implementació i la llibreria de representació utilitzada per visualitzar les xarxes.

6.2.1 Angular

Angular 2, també conegut com a Angular, és un *framework* desenvolupat amb TypeScript que s'utilitza per crear i mantenir *single page applications.* Angular és l'evolució d'Angular JS.

Figura 16. Logo d'Angular. Font: Angular

El *framework* es basa en l'arquitectura model - vista - controlador (MVC), amb l'objectiu de separar la lògica de l'aplicació de les dades i la seva representació. A partir de directives, natives o personalitzades, Angular amplia l'HTML proporcionant un mecanisme de sincronització bidireccional entre els

models de dades i les vistes, anomenat *data binding*. D'aquesta manera s'aconsegueix deslligar la lògica d'aplicació del disseny de les interfícies d'usuari, millorant la capacitat de testeig i el desenvolupament.

6.2.2 D3.js

D3 és una llibreria de Javascript de visualització i manipulació de dades a través dels navegadors webs. Per mitjà de tecnologies com HTML, CSS i SVG permet representar les dades en diferents models interactius: gràfics, taules, mapes etc. D3 és flexible, ràpid i compta amb moltes opcions de personalització.

Figura 17. Logo de D3. Font: D3

En aquest projecte es fa servir la llibreria D3 per la representació de les xarxes en l'aplicació client ja que és una de les llibreries més completes de visualització de dades en Javascript.

7. Implementació i resultats

Tant el servei web com l'aplicació client s'han desplegat en un servidor d'eXiT mitjançant **Docker**. Docker és una eina de virtualització que permet desplegar aplicacions dins de contenidors de software. Un contenidor conté l'aplicació i totes les seves dependències proporcionant un entorn d'execució aïllat de la resta de contenidors i dels sistema on està allotjat. D'aquesta manera s'aconsegueix que una aplicació pugui executar-se en qualsevol entorn, augmentant la portabilitat i la flexibilitat de les aplicacions.

En aquest apartat es descriuen els punts més destacats del desenvolupament del projecte i es mostren els resultats obtinguts en les dues parts que el formen.

7.1 Implementació i resultats del servidor

A continuació es descriuen els punts més importants de la implementació del servidor com l'estructura de fitxers, els passos per configurar el servei web, el sistema d'autenticació d'usuaris o els components implementats.

7.1.1 Estructura de fitxers

La figura 18 mostra l'estructura de fitxers del servidor. En aquest primer nivell es troben els fitxer **server.py** i **config.py**, els qual són necessaris per llançar el servei web. El servei web es troba dins la carpeta **app**. Les carpetes **grids** i **logs** contenen les xarxes pujades al servidor i els registres que es van generant durant l'execució del servei web.

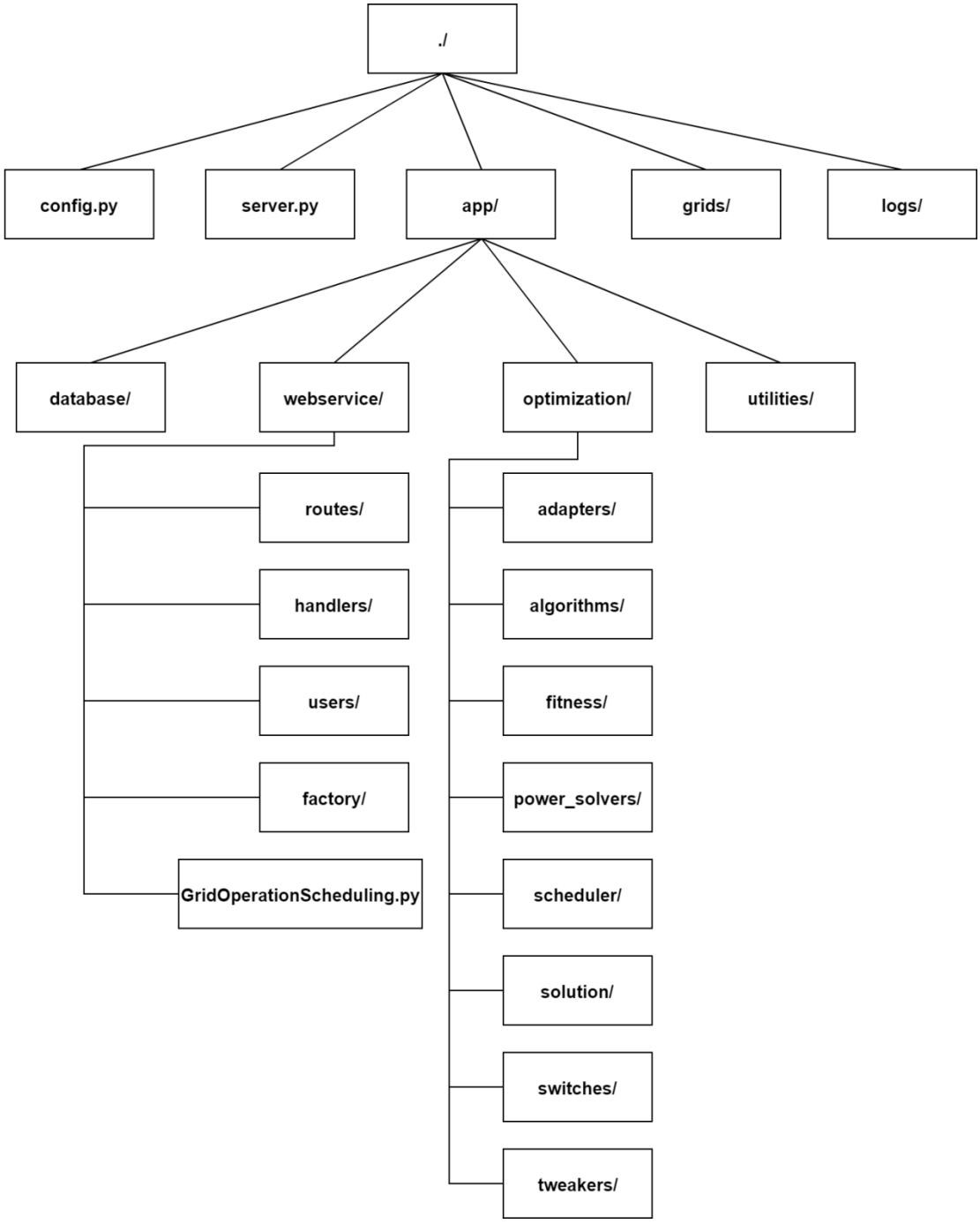

Figura 18. Estructura de fitxers del servidor

Dins la carpeta app es troben quatre directoris principals. Dins el directori **webservice** es troben els directoris i fitxers relatius a processos propis de les funcionalitats web del servidor, com la definició de les rutes o la gestió d'usuaris. Per altra banda, el directori **optimization** conté els fitxers de la part de planificació de xarxes del servei web: algoritmes, adaptadors, modificadors, funcions d'avaluació, etc. Finalment, les carpetes **database** i **utilities** content els fitxers relatius a la connexió amb la base de dades i classes auxiliars.

7.1.2 Inicialització del servei web i gestió de les rutes

Per inicialitzar un servei web amb Flask cal importar la classe Flask del mòdul del *framework* i instanciar-la:

```
# Fragment del fitxer app/__init__.py
from flask import Flask, request

app = Flask(__name__)
```

A partir de la variable *app* es poden configurar les diferents extensions que s'afegiran al servei web. També es pot passar un objecte de configuració amb variables globals que es podran importar on es necessitin:

```
# Fragment del fitxer app/__init__.py
from flask import Flask
from config import Config
from flask_pymongo import PyMongo

app = Flask(__name__)
# Importació de variables globals definides a la classe Config
app.config.from_object(Config)
# Configuració de l'extensió de la base de dades
mongo = PyMongo(app)
```

Com s'ha descrit a l'apartat 6.1.2, per construir el servei web com a API REST es farà servir l'extensió Flask-RESTful. Com en l'anterior fragment de codi l'extensió s'instancia a partir de la variable *app*:

```
# Fitxer app/webservice/routes/__init__.py
# Importació de la variable app definida en el fitxer app/__init__.py
from app import app
from flask_restful import Api

# Inicialització de l'extensió Flask Restful
api = Api(app)
```

L'extensió Flask-RESTful ofereix una solució orientada a objectes per tractar les peticions que arriben al servei web, associant la gestió de cada recurs a una classe. Les classes han d'heretar de la classe **Resource** de Flask per ser vàlides com a gestores d'una ruta. Seguint el

diagrama descrit a la secció 5.2, la classe **GenericRoute** hereta de la classe **Resource** i implementa utilitats que les subclasses que tracten les rutes poden fer servir. Aquesta part del diagrama queda ampliada a la Figura 19:

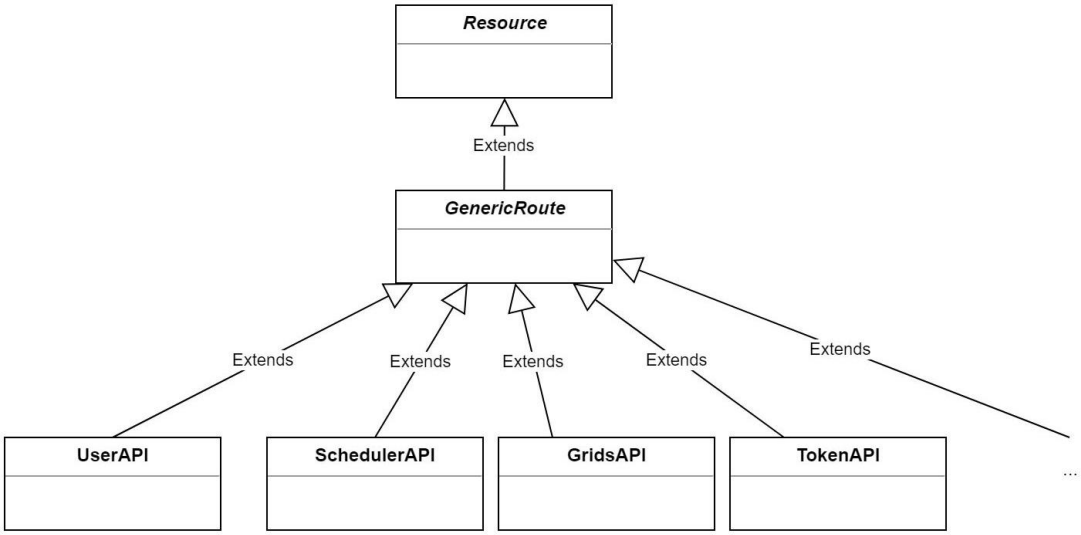

Figura 19. Exemple d'implementacions de la classe GenericRoute

A cada classe cal implementar els mètodes per tractar cada operació HTTP associada a la ruta. El següent exemple mostra la classe UserAPI, que gestiona les peticions de les rutes /user i /user/:id i conta amb les operacions HTTP GET, POST i PUT:

```
# Fragment del fitxer app/webservice/routes/Routes.py
class UserAPI(GenericRoute):
    def get(self, id_user):
     # Retorna la informació d'un usuari
      …
    def post(self):
      # Crea un nou usuari
      …
    def put(self, id_user):
      # Actualitza un usuari
      …
```

Perquè Flask associï la classe amb la ruta corresponent cal cridar el mètode *add_resource* que proporciona l'extensió:

```
# Fragment del fitxer app/webservice/routes/__init__.py
# Importació de la variable app definida en el fitxer app/__init__.py
from app import app
# Importació de les classes gestores de les rutes
from app.webservice.routes import Routes
from flask_restful import Api

# Inicialització de l'extensió Flask Restful
api = Api(app)

# Assignar la classe UserAPI a les rutes corresponents
api.add_resource(Routes.UserAPI, '/user', '/user/<id_user>')
…
```

Finalment, podem llançar el servei web des de línia de comanda executant la comanda **flask run**. Aquesta comanda executa l'aplicació Flask a partir del fitxer server.py i llança un servidor de desenvolupament en local per poder testejar el servei web.

```
$ flask run
 * Environment: production
   WARNING: Do not use the development server in a production
environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

7.1.3 Restricció de les rutes per usuaris autentificats i per rol

Per controlar l'accés als recursos restringits a usuaris autenticats es pot implementar un sistema d'autenticació mitjançant l'extensió de Flask **flask_httpauth**. Aquesta extensió proporciona autenticació HTTP bàsica i Digest per les rutes de Flask. Com amb la resta d'extensions primer caldrà instanciar-la:

```python
# Fragment del fitxer app/webservice/routes/Routes.py
# Importació de la variable app definida en el fitxer app/__init__.py
from app import app
from flask_httpauth import HTTPBasicAuth

# Instanciació de l'extensió
auth = HTTPBasicAuth()
```

Un cop instanciada és necessari definir una funció que ens permeti determinar si un usuari està autenticat o no:

```python
# Fragment del fitxer app/webservice/routes/Routes.py
# Importació de la variable app definida en el fitxer app/__init__.py
from app import app
from flask_httpauth import HTTPBasicAuth

# Instanciació de l'extensió
auth = HTTPBasicAuth()

@auth.verify_password
def verify_password(username, password):
    # Instanciació la interfície del servei web
    webservice = GridOperationScheduling()
    user = webservice.authenticateUser(username, password)
    if user:
        # Es guarda l'usuari a la variable global de Flask g
        g.user = user
        return True
    else:
        return False
```

Punts clau del fragment de codi anterior:

- Afegint el decorador **@auth.verify_password** a sobre la declaració de la funció *verify_password* s'indica que es vol fer servir aquesta funció per validar l'autenticació dels usuaris.
- La funció de validació rep el nom d'usuari i el password inclosos a les capçaleres d'autenticació de la petició HTTP.

- Per validar l'usuari cridem la funció *authenticateUser* de la interfície del servei web. Aquesta funció retorna un objecte de la classe User si s'ha trobat l'usuari amb aquelles credencials o fals si no s'ha trobat.
- En cas de que s'hagi trobat l'usuari, es guarda la referència a l'objecte *user* dins la variable global *g* de Flask per posteriors usos durant l'execució de la petició.

Un cop definida la funció només caldrà afegir el decorador **@auth.login_required** abans de la declaració de les rutes que s'han de restringir:

```
# Fragment del fitxer app/webservice/routes/Routes.py
# Importació de la variable app definida en el fitxer app/__init__.py
from app import app
from flask_httpauth import HTTPBasicAuth
from flask import g

# Instanciació de l'extensió
auth = HTTPBasicAuth()

@auth.verify_password
def verify_password(username, password):
    # Instància de la interfície del servei web
    webservice = GridOperationScheduling()
    user = webservice.authenticateUser(username, password)
    if user:
        # L'usuari es guarda a la variable global de Flask g
        g.user = user
        return True
    else:
        return False

# Classe encarregada de les rutes de generació de token
class TokenAPI(GenericRoute):
    # El decorador login_required indica que la ruta està
      restringida
    @auth.login_required
    def get(self):
      token = self.webservice.generate_token(g.user)
      response = {'token': token}
      return response
```

Seguint l'exemple anterior, quan s'intenti accedir a la ruta per generar un nou *token,* GET /token, s'executarà la següent cadena d'accions:

- Flask detectarà que aquesta ruta necessita autenticació i cridarà a la funció de validació definida amb el decorador **@auth.verify_password** el nom d'usuari i password de la petició HTTP.
- Si la funció de validació retorna cert, s'executarà el mètode get de la classe *TokenAPI*. Si pel contrari la funció de validació retorna fals, Flask aturarà l'execució de la petició i retornarà un error HTTP 401.
- Dins el mètode *get,* es crida la funció de la interfície del servei web per generar un nou *token* passant com a paràmetre l'usuari prèviament guardar a la variable *g* de Flask durant la funció de validació.
- Finalment, es retorna el nou *token* com a resposta.

Autenticació via token: generació i validació

Segons el disseny proposat l'autenticació també ha de poder ser via *token*. Podem generar els *tokens* mitjançant les utilitats que es poden trobar dins els paquets de Python **itsdangerous**. La classe *User* és l'encarregada de la generació:

```
# Fragment del fitxer app/webservice/users/User.py
from app import app
from app.database.Database import *
from itsdangerous import (TimedJSONWebSignatureSerializer as
Serializer, BadSignature, SignatureExpired)
…
class User():
  …
  def generate_auth_token(self, expiration=3600):
    # Es prepara el nou token amb la clau secreta
    s = Serializer(app.config['SECRET_KEY'], expires_in=expiration)
    dumped = s.dumps({'user': self.username})

    # S'actualitza la base de dades amb l'últim token generat
   Database.updateOne('users', {'username': self.username}, {'$set':
      {'last_token': dumped.decode('ascii')}})
```

En aquesta funció s'instancia un objecte de la classe *Serializer*, el qual s'encarrega de la generació del *token.* A l'instanciar-lo es passa com a paràmetre una clau secreta definida en el fitxer config.py i el temps de validesa amb un temps d'una hora com a valor per defecte. Un cop instanciat, es crida al mètode *dumps* per generar un nou *token* xifrat amb la clau secreta passada com a paràmetre. Dins el *token* s'afegeix el nom de l'usuari per poder validar-lo quan es desxifri. Finalment, per augmentar la seguretat es guarda l'últim *token* generat a la base de dades.

Per utilitzar el *token* per l'autenticació cal afegir-lo a la capçalera d'autenticació HTTP com si fos el nom d'usuari. Llavors a la funció *authenticateUser* de la classe *GridOperationScheduling* s'intentarà primer tractar el paràmetre com a *token* i si falla es provarà a fer l'autenticació amb credencials:

```
# Fragment del fitxer app/webservice/GridOperationScheduling.py
class GridOperationScheduling:
      …
      def authenticateUser(self, username, password):
            # Autenticació via token (considerant username com el
                  token)
            user = User.verify_auth_token(username)
            if not user:
                  # Si l'autenticació amb token falla es prova a
                  fer l'autenticació amb usuari i password
                  user = User.get_user(username)
                  if not user or not user.verify_password(password):
                        return False
            # Si l'usuari no està actiu també es retorna Fals
            if user.active != 1:
                  return False
            return user
```

A la funció *verify_auth_token* de la classe User es comprova que el token sigui vàlid, es desxifra i es retorna l'usuari associat al nom d'usuari que es va incloure en la generació del *token*:

```
# Fragment del fitxer app/webservice/users/User.py
from itsdangerous import (TimedJSONWebSignatureSerializer as
Serializer, BadSignature, SignatureExpired)
class User():
  …
  @staticmethod
  def verify_auth_token(token):
    s = Serializer(app.config['SECRET_KEY'])
    # S'intenta desxifrar el token
    try:
        data = s.loads(token)
    except SignatureExpired:
        return None  # token caducat
    except BadSignature:
        return None  # token invàlid
    # Recupera l'usuari associat a l'usuari
    user = User.get_user(data['user'])
    # Comprova que sigui l'ultim token generat
    if User.is_last_token(user.username, token):
        return user
    else:
        return None
```

Restricció per rol

Per acabar, s'afegeix una última capa de validació per rol en certes rutes. Com a requisit d'aquest projecte només cal fer una distinció entre usuaris comuns i usuaris administradors. El procediment per afegir aquesta nova capa és similar al seguit per crear la capa d'autenticació. Es defineix una funció que es fa servir com a decorador dels mètodes de les rutes en les que es vol aplicar la restricció per rol:

```
# Fragment del fitxer app/webservice/routes/Routes.py

def admin_required(f):
      # El paràmetre f és el mètode de la ruta que es vol executar
      @wraps(f)
      def wrapper(*args, **kwargs):
            # Si l'usuari guardat de l'autenticacio és administrador
            s'executarà la funció
            if g.user is not None and g.user.is_admin():
                  return f(*args, **kwargs)
            return 'No access'

      return wrapper
```

Un cop definida la funció només s'ha d'afegir a sobre les declaracions dels mètodes a restringir:

```
# Fragment del fitxer app/webservice/routes/Routes.py
class UserAPI(GenericRoute):

    @auth.login_required
    @admin_required
    def delete(self, id_user):
      # Esborra un usuari. Només poden executar aquesta funció els
      usuaris administradors
      …
```

7.1.4 Format de la resposta

Flask-RESTful serialitza totes les respostes en format JSON mitjançant el mòdul base de Python **json**. En general aquest comportament és molt pràctic ja que no cal formatar explícitament els valors que retornen les classes que gestionen les rutes. No obstant, pot ser un inconvenient en determinades situacions:

- El mòdul **json** només pot serialitzar tipus bàsics de dades. Per exemple, no és capaç de serialitzar un número complex o una data a JSON. Si el contingut d'una resposta té un tipus de dada que el mòdul no pot serialitzar provocarà un error del servidor.

- Si el contingut d'una resposta ja ha estat serialitzat en JSON prèviament, com al recuperar elements de la base de dades, Flask no l'hauria de tornar a serialitzar.
- El contingut de la resposta d'algunes rutes no pot ser JSON, com per exemple la ruta GET /grid/<id>/download, que retorna el fitxer Excel d'una xarxa.

Per solucionar aquests problemes s'ha d'ampliar el funcionament de la generació de la resposta. Primer caldrà ampliar el mòdul de **json** de manera que sigui capaç de serialitzar nous formats de dades:

```
# Fragment del fitxer app/utilities/CustomEncoder.py
import json
import datetime
from bson.objectid import ObjectID
…
# Per ampliar el mòdul de json cal crear una classe que hereti de la classe JSONEncoder que proporciona el mòdul. En aquest exemple el nou codificador permetrà serialitzar dades en format data, nombres complexos i identificadors de la base de dades
class ExtendedEncoder(json.JSONEncoder):
      # El mètode default s'executa al serialitzar cada dada
      def default(self, o):
            # El mètode s'extén amb els nous tipus dades a controlar
            if isinstance(o,datetime.datetime):
                  return o.isoformat()
            elif isinstance(o, complex):
                  return EncodingUtilities.complex_number_to_string(o)
            elif isinstance(o, ObjectId):
                  return str(o)
            # Si no es cap format que controlem es serialitza la dada amb la funció original
            return json.JSONEncoder.default(self, o)
```

Un cop definit, s'ha de modificar la funció de serialització que fa servir Flask per una nova on es faci servir el nou codificador. Flask permet definir funcions personalitzades i associar-les a un tipus de format específic de resposta, per tant cal definir una nova funció amb el nou codificador i associar-la al tipus de resposta JSON:

```python
# Fragment del fitxer app/webservice/routes/__init__.py
from app import app
from app.utilities.CustomEncoder import ExtendedEncoder
import flask
from flask_restful import Api

# Inicialització de l'extensió Flask Restful
api = Api(app)
# Amb el decorador s'associa el tipus de resposta JSON a aquesta
funció
# La funció rep com paràmetres el contingut de la resposta, el codi
HTTP que es retornarà i les capçaleres extra si n'hi ha
@api.representation('application/json')
def output_json(data, code, headers=None):
    # Primer es comprova si el contingut de la resposta ja està
      serialitzat en JSON
    try:
        # Per comprovar si el contingut ja està serialitzat es pot
        provar a deserialitzar amb la funció json.loads.
        json_test = json.loads(data)
    except TypeError:
        # Si la deserialització falla vol dir que el contingut no
        està codificat en JSON, per tant es codifica amb la funció
        json.dumps indicant que es vol fer servir el nou
        codificador amb el paràmetre cls
        data = json.dumps(data, cls=ExtendedEncoder)
    # Finalment es construeix la resposta mitjançant el mètode
      make_response de flask
    resp = flask.make_response(data)
    resp.headers.extend(headers or {})
    return resp
```

Retornar fitxers Excel

Per les rutes que no han de retornar respostes en format JSON es pot aplicar el procediment anterior de definir una funció i associar-la a un tipus de format amb el decorador de la variable *api*. També és possible trobar alguna extensió de Flask que gestioni un format concret. En el cas dels fitxers Excel existeix una extensió anomenada **flask_excel** que s'encarrega de configurar les capçaleres de la resposta HTTP i el contingut per descarregar fitxers en format Excel. Un cop descarregada l'extensió cal inicialitzar-la al crear la variable *app*:

```
# Fragment del fitxer app/__init__.py
from flask import Flask
from config import Config
from flask_pymongo import PyMongo
import flask_excel as excel

app = Flask(__name__)
# Importació de variables globals definides a la classe Config
app.config.from_object(Config)
# Inicialització de l'extensió de la base de dades
mongo = PyMongo(app)
# Inicialització de l'extensió flask_excel
excel.init_excel(app)
```

Un cop inicialitzada es pot fer servir per retornar fitxers Excel a través de les rutes del servei web:

```
# Fragment del fitxer app/webservice/routes/Routes.py
…
import flask_excel as excel
…

class DownloadGridAPI(GenericRoute):

    @auth.login_required
    def get(self, id):
      # El servei web recupera el fitxer Excel
      file_info = self.webservice.download_grid_file(id)
      # L'extensió flask_excel s'encarrega de generar la resposta amb el
      contigut i les capçaleres adequades
      return excel.make_response(file_info['file'], 'xlsx', 200,
             file_info['name'])
```

7.1.5 Configuració del CORS

L'intercanvi de recursos d'origen creuat (CORS) és un mecanisme que utilitza capçaleres HTTP addicionals per permetre que les peticions HTTP provinents de dominis o ports diferents del servidor tinguin accés als recursos sol·licitats (Control de acceso HTTP (CORS). MozillaDevelopers). La figura 20 mostra gràficament aquest concepte:

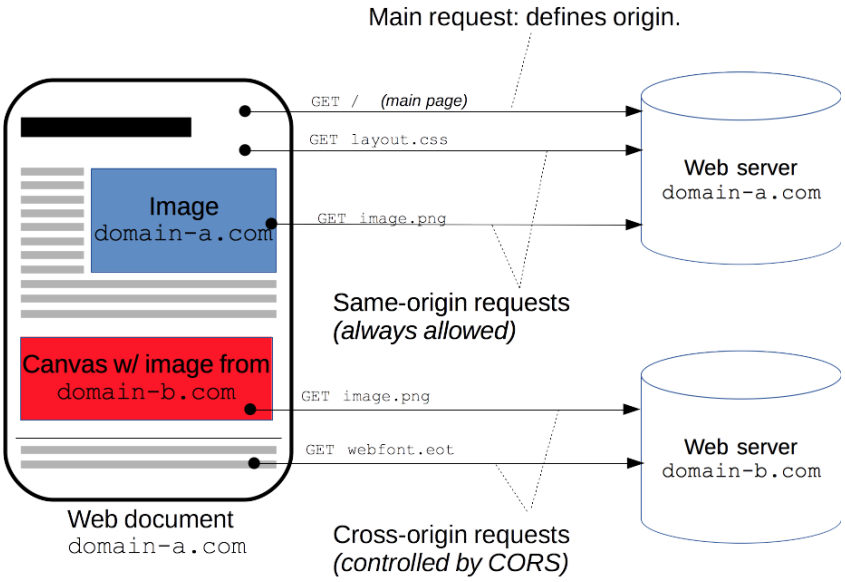

Figura 20. Exemple de CORS (Control de acceso HTTP (CORS))

Per defecte Flask no incorpora les capçaleres necessàries per habilitar el CORS. Per incloure aquestes capçaleres cal utilitzar l'extensió **flask_cors.** Amb aquesta extensió podem controlar els recursos del servei web que es volen incloure en el CORS, els orígens permesos, pas de credencials, etc.

```
# Fragment del fitxer app/__init__.py
…
from flask_cors import CORS

app = Flask(__name__)
…
# S'habilita CORS per tots els recursos i orígens de les peticions
cors = CORS(app, resources={r"/*": {"origins": "*"}})
```

7.1.6 Implementació dels components

A banda de llançar i configurar el servei web, també s'han realitzat implementacions dels principals components del servei web per poder començar a utilitzar el servei de planificació. En aquest apartat es descriuran tots breument a excepció dels algoritmes que es tractaran amb més detall a l'apartat 7.3.

Adaptador GridCal (GridCalAdapter)

Aquest adaptador s'encarrega de gestionar les xarxes implementades amb la llibreria GridCal. De moment GridCal és l'únic *power flow solver* disponible en el servei web, pel que totes les xarxes seran gestionades per aquest adaptador. Les seves principals funcions són:

- Inicialitzar la xarxa amb GridCal des d'un fitxer Excel.
- Transformar la xarxa al format de la base de dades guardar-la mitjançant els importadors.
- Fer el procés invers: transformar la xarxa en format de la base de dades a format GridCal.
- Retornar informació sobre els diferents components de la xarxa: branques, bateries, generadors, etc.
- Aplicar modificacions sobre la xarxa definides pels modificadors.
- Aplicar solucionadors de potències a la xarxa i gestionar els resultats.
- Gestionar els interruptors associats de la xarxa.

Interruptor GridCal (SwitchGridCal)

Aquesta classe és la implementació especifica de la classe *Switch* per la llibreria GridCal. Permet habilitar i deshabilitar branques de xarxes implementades amb aquesta llibreria.

Modificador aleatori (RandomTweaker)

Aquest modificador altera un instant de temps aleatori de la xarxa realitzant dues accions:

- Canvia l'estat d'un interruptor, també escollit aleatòriament.
- Modifica les potències dels generadors controlables i bateries amb valors aleatoris.

Cal tenir present que els modificadors no interactuen directament amb la xarxa, sinó que criden els mètodes corresponents de l'adaptador en concret per realitzar les modificacions.

Control de les potències

Tot i que les potències dels generadors controlables i les bateries són modificats amb valors aleatoris, cal mantenir un control i ajustar els valors resultants perquè estiguin dins el rang de

valors permesos per cada element. La nova potència ha d'estar dins els límits de potència mínima i màxima:

$$P_{max} \geq P_{new} \leq P_{min}$$

En cas de les bateries també s'ha de comprovar que l'energia de la bateria estigui dins els seus límits. L'energia que produeix o consumeix una bateria en un instant determinat per determinat pel seu estat de càrrega (soc) i la seva energia nominal (E_{nom}):

$$E_{max} = Soc_{max} * E_{nom}$$

$$E_{min} = Soc_{min} * E_{nom}$$

$$E_{current} = Soc_{current} * E_{nom}$$

La següent expressió calcula l'energia que tindrà la bateria amb el canvi de potència:

$$E_{new} = E_{current} - P_{new} * \Delta t * eff$$

on:

$E_{current}$ és l'energia que produeix la bateria en l'instant actual.

P_{new} és la nova potència proposada.

Δt és la diferència de temps entre l'instant anterior i l'actual expressada en hores.

eff és l'energia produïda o l'energia consumida en un cicle de la bateria depenent de si la potència proposada és negativa (bateria consumint) o positiva (bateria carregant):

$$eff = E_{nom} * ChargeEfficiency$$

$$eff = E_{nom} * DischargeEfficiency$$

La nova energia haurà d'estar dins els límits d'E_{max} i E_{min} :

$$E_{max} \geq E_{new} \leq E_{min}$$

En cas que la nova potència no satisfaci alguna de les regles la potència serà ajustada al límit màxim o mínim pel dispositiu.

Solucionador de potències Newton-Raphson (GridCalNR)

El solucionador de potències GridCalNR calcula els fluxos de potència de les xarxes de tipus GridCal aplicant el mètode Newton-Raphson.

El solucionador retorna un objecte amb els resultats de l'execució amb vàries mètriques que fan servir les funcions d'avaluació. Algunes d'aquestes mètriques són:

- Voltatges unitaris dels busos.
- Potències dels busos.
- Intensitat de les branques.
- Potència de les branques.
- Càrrega de les branques.
- Pèrdues de les branques.
- Errors dels fluxos de potència de la xarxa.

Funció d'avaluació (VVLFitness)

La funció d'avaluació implementada puntua les solucions a partir de tres factors: l'error de voltatge total, l'excés de càrrega total i les pèrdues de la xarxa. Es suposa que la solució a avaluar consta d'una xarxa amb resultats de l'aplicació d'un solucionador de potències per cada instant de temps.

Error de voltatge total

L'error de voltatge total consisteix en la mitjana de l'error de voltatge mitjà en cada instant de temps de la xarxa. L'error de voltatge d'un bus s'obté del valor absolut de la diferència entre el seu voltatge expressat en forma unitària respecte 1:

$$errorVol = \frac{1}{T \cdot N} \sum_{i=1}^{N} \sum_{t=1}^{T} |1 - V_{i,t}|$$

on:

T és el nombre d'instants de temps de la xarxa.

N és el nombre de busos de la xarxa

$V_{i,t}$ és el voltatge unitari del bus i a l'instant t retornat pel solucionador de potències.

Excés de càrrega total

L'excés de càrrega és la diferència entre la càrrega d'una branca i un límit respecte a la càrrega total que pot assolir. L'excés de càrrega en una branca es pot calcular mitjançant la següent expressió:

$$loadExcess = \frac{\sqrt{CLoad_{real}^2 + CLoad_{imag}^2}}{TLoad} - threshold$$

on:

$CLoad_{real}^2$ és la part real del valor de la càrrega retornat pel solucionador de potències, expressat en forma complexa.

$CLoad_{imag}^2$ és la part imaginària del valor de la càrrega retornat pel solucionador de potències, expressat en forma complexa.

$TLoad$ és el valor màxim de càrrega de la branca.

$threshold$ és el límit de càrrega expressat en tant per u fixat per la funció d'avaluació.

L'excés de càrrega total de la xarxa consisteix en la mitjana de l'excés de càrrega total mig a cada instant de temps:

$$errorLoading = \frac{1}{T \cdot M} \sum_{j=1}^{M} \sum_{t=1}^{T} LoadExcess_{t,j}$$

on:

T és el nombre d'instants de temps de la xarxa.

M és el nombre de branques de la xarxa .

$LoadExcess_{t,j}$ és l'excés de càrrega de la branca j a l'instant t.

Pèrdues de la xarxa

L'últim factor d'aquesta funció d'avaluació són les pèrdues de les branques de la xarxa. Com en la resta de factors es calcularà a partir de les pèrdues mitjanes totals de la xarxa en cada instant de temps:

$$errorLosses = \frac{1}{T \cdot M} \sum_{j=1}^{M} \sum_{t=1}^{T} losses_{t,j}$$

on:

T és el nombre d'instants de temps de la xarxa.

M és el nombre de branques de la xarxa .

$losses_{t,j}$ són les pèrdues de la branca j a l'instant i retornades pel solucionador de potències.

La puntuació final vindrà determinada per la suma d'aquests tres factors:

$$score = errorVol + errorLoading + totalLosses$$

Com més petita sigui la puntuació millor serà la solució ja que es tracta d'una xarxa amb menys error acumulat. La xarxa òptima, doncs, serà la tingui puntuació 0.

7.2 Implementació del client

En aquest apartat s'expliquen els diferents elements que s'han implementat durant el desenvolupament client i es mostren les pantalles que la formen.

7.2.1 Arquitectura d'una aplicació Angular

Una aplicació desenvolupada amb Angular està estructurada en una sèrie de blocs anomenats **mòduls**. Els mòduls són unitats funcionals reutilitzables formades per **components**. Els components defineixen i controlen les **vistes** de l'aplicació. Una vista és un fitxer HTML amb marcatge específic d'Angular que permet modificar l'HTML en funció de la lògica del component i lligar de manera bidireccional les dades de l'aplicació, de manera que a la vista es representin les dades dels models de l'aplicació i a la vegada es puguin actualitzar a partir de les accions dels usuaris.

Els components poden utilitzar **serveis**, que implementen funcionalitats que no estan lligades a cap component. Els serveis són injectats com a dependències dins els components i

altres serveis, afavorint la modularitat, la reutilització i l'eficiència del codi. Alguns exemples de serveis poden ser la validació de dades entrades pels usuaris a través de les vistes o les crides a un servei web per obtenir o enviar dades.

En els següents apartats es descriuen els elements més importants d'aquesta arquitectura implementats durant el desenvolupament de l'aplicació client del projecte.

7.2.2 Serveis destacats

ApiGridService

Aquest servei s'encarrega de la comunicació amb el servei web. Implementa totes les crides necessàries pel funcionament del client: autenticació de l'usuari, recuperar llistats de xarxes, executar una nova planificació, etc. Qualsevol element de l'aplicació que necessiti interactuar amb el servei web ho farà a través d'aquest servei com es representa a la Figura 21.

GridCreatorService

Aquest servei transforma les dades de les xarxes en format JSON a instàncies de classes que representen els models dels diferents elements d'una xarxa: busos, branques, bateries, interruptors, etc. Aquests models representen les dades amb la mateixa estructura que es representen les xarxes en el servidor i implementa mètodes que els components i serveis de l'aplicació poden utilitzar per accedir a informació de les xarxes o manipular-les.

D3Service

Aquest servei actua com una interfície de la llibreria D3 amb la finalitat que la resta d'elements de l'aplicació puguin interactuar amb la llibreria per construir la representació de les xarxes.

AuthenticationService

Aquest servei s'encarrega d'autenticar els usuaris a l'aplicació, via el servei *ApiGridService*, i de la gestió de les seves dades com el *token* o el rol.

MessageService

Aquest servei permet als elements de l'aplicació encuar missatges per mostrar-los en pantalla en forma de missatges emergents.

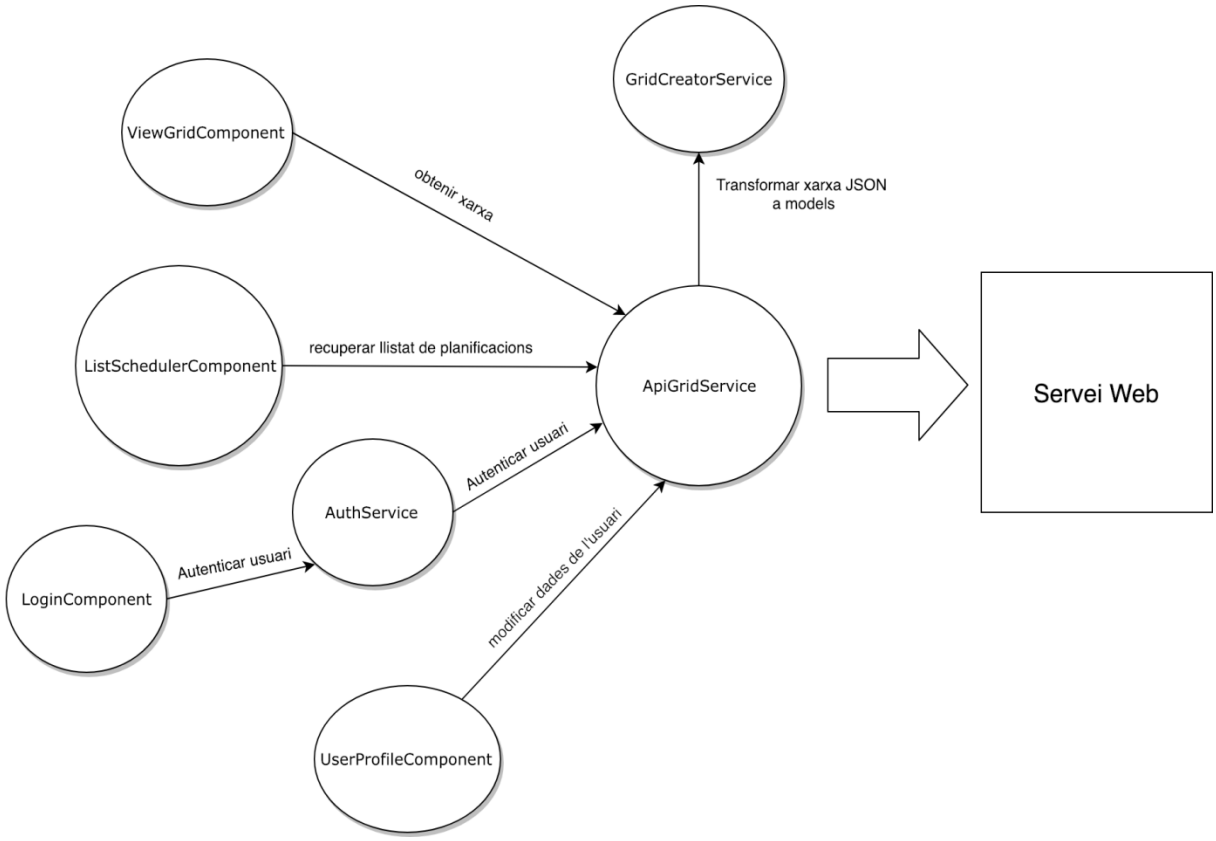

Figura 21. Els diferents components del client utilitzen el servei ApiGridService per comunicar-se amb el servidor

7.2.3 Mòduls i components destacats

SignupModule

Aquest mòdul està format per un sol component, *SignupComponent,* que controla la vista de la pantalla de registre. S'encarrega de recuperar les dades introduïdes en el formulari de registre i enviar la petició de nou usuari al servidor.

LoginModule

Mòdul d'autenticació. El seu únic component, *LoginComponent*, recull les dades del formulari d'autenticació i interactua amb el servei d'autenticació *AuthenticationService*.

LayoutModule

Gestiona els elements comuns que contenen les pantalles corresponents a l'àrea interna de l'aplicació un cop l'usuari ha entrat correctament. Consta de quatre components:

- *LayoutComponent*: Defineix l'estructura d'una pantalla interna de l'aplicació, la qual està formada pel cap, la barra lateral i el contingut de la pantalla. El cap i la barra lateral vindran definits pels mòduls *HeaderComponent* i *SidebarComponent* respectivament, mentre que el contingut de la pantallla estarà definit pels mòduls que es descriuran a continuació.
- *HeaderComponent*: Controla la barra superior d'una pantalla interna.
- *SidebarComponent*: Controla la barra lateral d'una pantalla internta.
- *MessageComponent*: Controla els missatges emergents que es mostren en pantalla provinents del servei *MessageService*.

DashboardModule

Està format pel component *DashboardComponent*, encarregat de la gestió de la pantalla de benviguda de l'aplicació un cop autentificat.

UploadGridModule

Mòdul format pel component *UploadGridComponent*, que gestiona la pantalla de nova xarxa. Recull les dades introduïes i envia la petició de creació de nova xarxa al servidor.

RenderGridModule

Mòdul amb els components per mostrar la informació d'una xarxa. Està format per quatre components:

- *RenderGridComponent:* Mostra la informació general d'una xarxa com el nom, data de creació, tipus de xarxa...
- *RenderFeaturesComponent*: Mostra els diferents elements dels que està formada la xarxa: bateries, generadors, branques, etc.
- *RenderFieldComponent:* Representa visualment els camps dels elements en funció del seu tipus.
- *RenderProfilesComponent:* Mostra la llista de valors per instant de temps dels elements.

RenderGraphModule

Mòdul per representar les xarxes en forma de graf. El formen tres components:

- *RenderGraphComponent*: S'encarrega de transformar la xarxa com a conjunt de nodes i arestes i mostrar-lo.

- *NodeVisualComponent*: Representació dels nodes del graf. Cada node té associat un bus de la xarxa.
- *LinkVisualComponent*: Representació de les arestes del graf. Cada aresta té associada una branca de la xarxa.

GridsModule

Agrupa els components encarregats de les pantalles de visualització de les xarxes. Consta dels següents components:

- *ListGridsComponent*: Recupera el llistat de xarxes de l'usuari del servidor i el mostra.
- *ViewGridComponent*: Fitxa d'una xarxa. Fa servir les funcionalitats dels mòduls *RenderGraphModule* i *RenderGrid*.

SchedulerModule

Consta dels components per llançar noves planificacions i visualitzar-les. Està format pels següents components:

- *CreateSchedulerComponent*: Gestiona la pantalla per llançar una nova planificació.
- *ListSchedulersComponent*: Recupera el llistat de planificacions de l'usuari i les mostra per pantalla. Igual que amb el llistat de xarxes, aquest component permet accedir a la fitxa d'una planificació i també eliminar-la.
- *ViewSchedulerComponent*: Fitxa d'una planificació. Mostra la informació de la planificació i sobre la xarxa resultant, juntament amb la representació d'aquesta xarxa i la xarxa original.

AdminModule

Aquest mòdul està format pels components que defineixen les pantalles dels administradors. Consta d'aquests components:

- *AdminUsersListsComponent*: Controla la pantalla de llistat d'usuaris des d'on es pot accedir a la fitxa d'edició o eliminar-lo.
- *AdminNewUserComponent*: Controla la pantalla de creació d'un usuari des de l'administrador.
- *AdminUserComponent*: Controla la pantalla d'edició dels usuaris com a administrador.

UsersModule

Consta del component *UserProfileComponent*, que gestiona la pantalla de perfil dels usuaris.

7.2.4 Pantalles de l'aplicació

A continuació es mostraran les pantalles que formen l'aplicació, esmenant els mòduls i components per les que estan formades. El disseny parteix d'una plantilla gratuïta que proporciona les eines necessàries per construir pantalles a partir d'estils predefinits. Per evitar la repetició dels mateixos components, es donarà per entès que totes les pantalles a excepció de la d'autenticació i registre estan formades pels components del mòdul *LayoutModule*.

Pantalla d'autenticació

Components: *LoginComponent* (LoginModule)**.**

Aquesta és la primera pantalla (Figura 22) que es visualitza quan s'accedeix a l'aplicació. Permet iniciar sessió omplint les credencials en el formulari o accedir al formulari de registre.

Figura 22. Pantalla d'autenticació

Pantalla de registre

Components: *SignupComponent* (*SignupModule*)**.**

Formulari de registre per registrar-se com a usuari de l'aplicació com mostra la Figura 23. Si el registre s'efectua amb èxit es mostrarà un missatge de notificació indicant a l'usuari que ha d'activar el seu compte a través del seu correu electrònic.

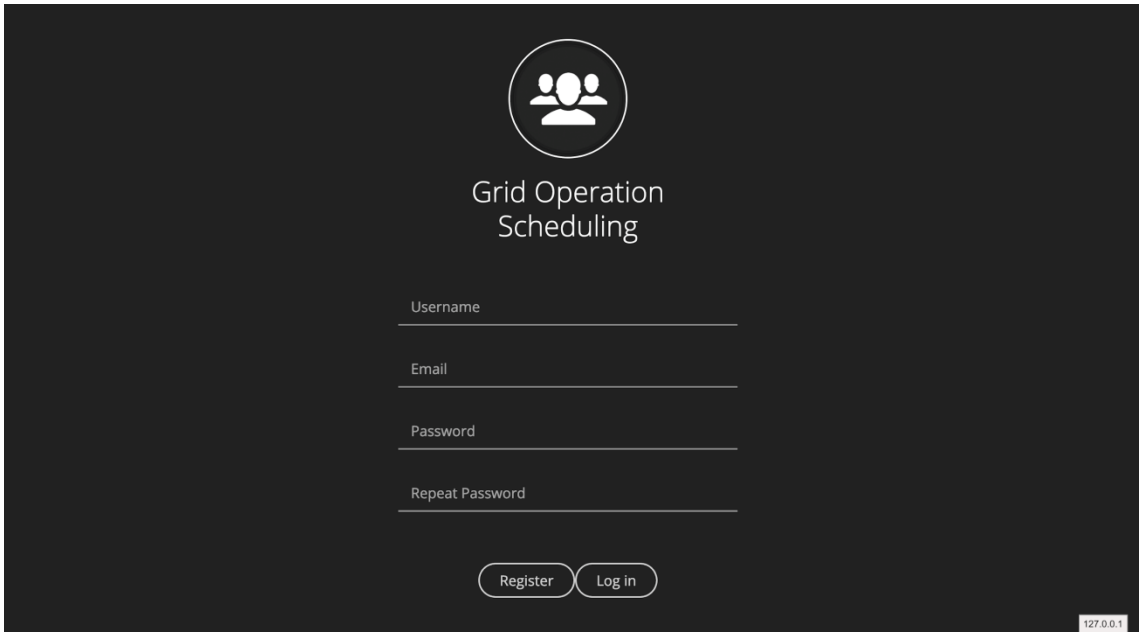

Figura 23. Pantalla de registre

Pantalla de benviguda

Components: *DashboardComponent* (*DashboardModule*).

Pantalla inicial un cop l'usuari s'ha autenticat. Com es mostra a la Figura 24, en aquesta pantalla es visualitzen les últimes xarxes pujades i les últimes planificacions llançades per l'usuari.

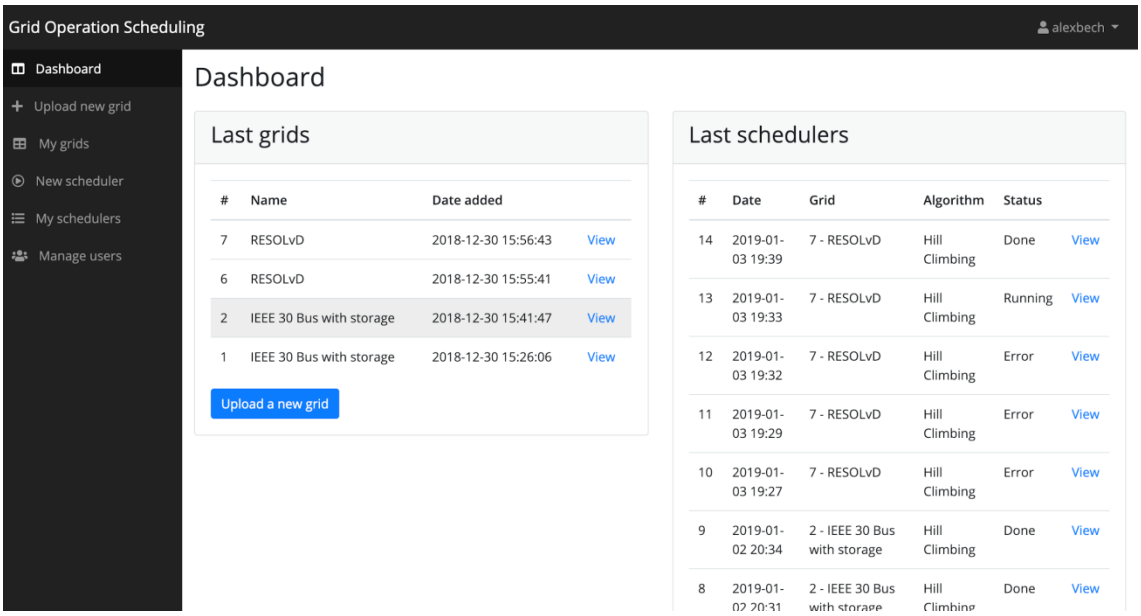

Figura 24. Pantalla de benvinguda

Pantalla de nova xarxa

Components: *UploadGridComponent* (*UploadGridModule*).

La Figura 25 mostra la pantalla amb el formulari per pujar una nova xarxa. Permet pujar la xarxa des d'un fitxer Excel o amb una cadena JSON amb el format de la base de dades.

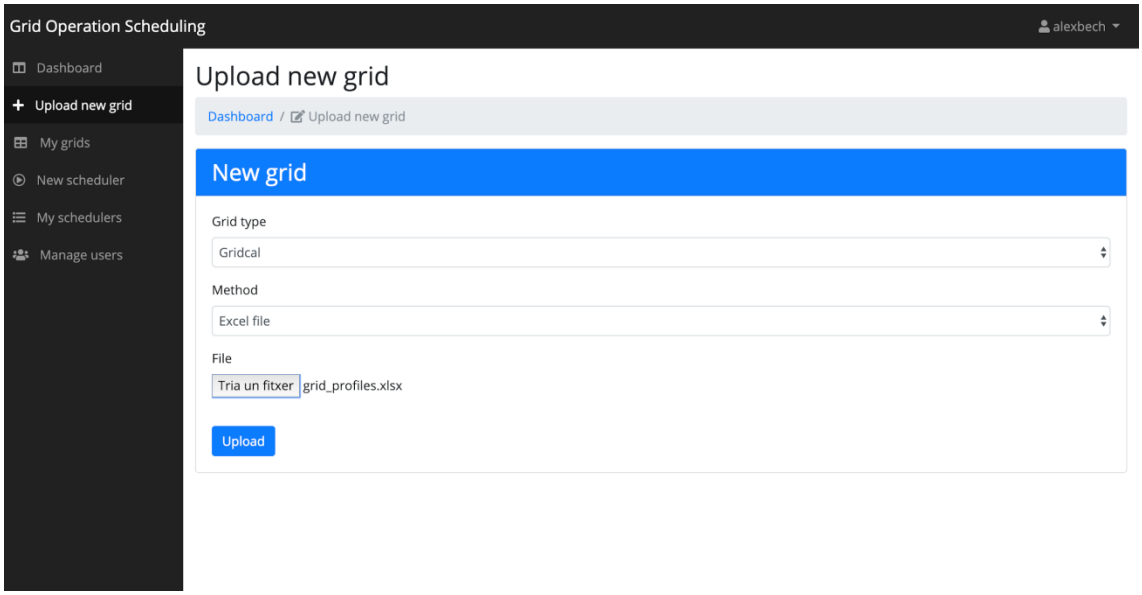

Figura 25. Pantalla de nova xarxa

Quan es crea una xarxa es mostra un missatge notificant el resultat de la creació amb i un enllaç a la fitxa de la xarxa si s'ha creat correctament, juntament amb la possibilitat de pujar una nova xarxa tal com es mostra en la Figura 26.

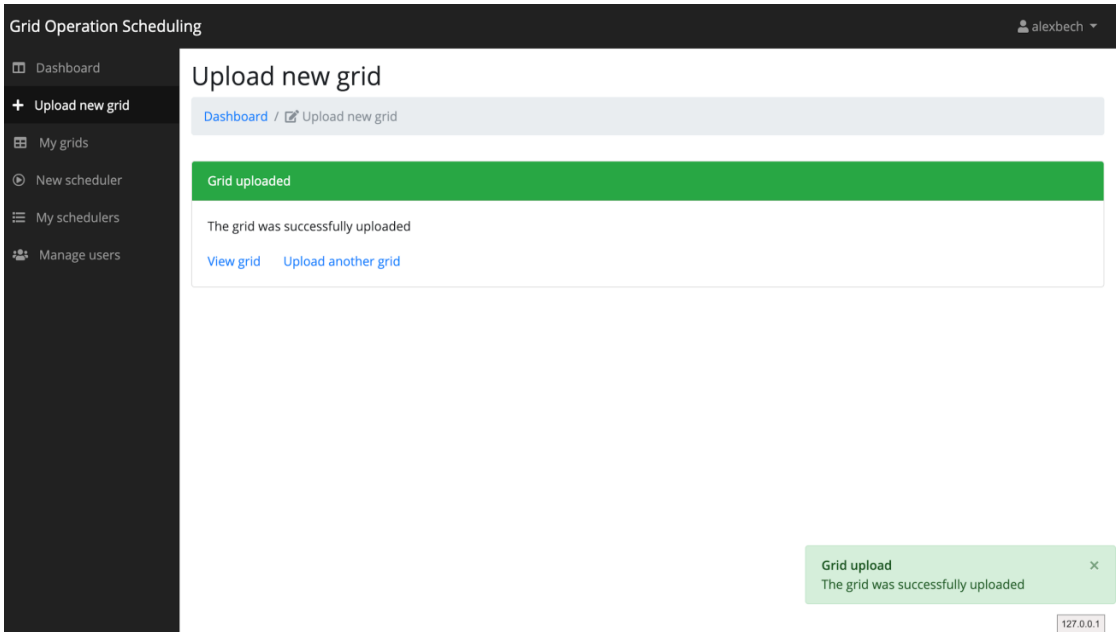

Figura 26. Missatge amb la resposta de la creació de la xarxa

Llistat de xarxes

Components: *ListGridsComponent* (*GridsModule*).

Pantalla amb el llistat de les xarxes pujades per l'usuari amb informació rellevant, juntament amb un enllaç per visualitzar la fitxa o eliminar la planificació com mostra la Figura 27.

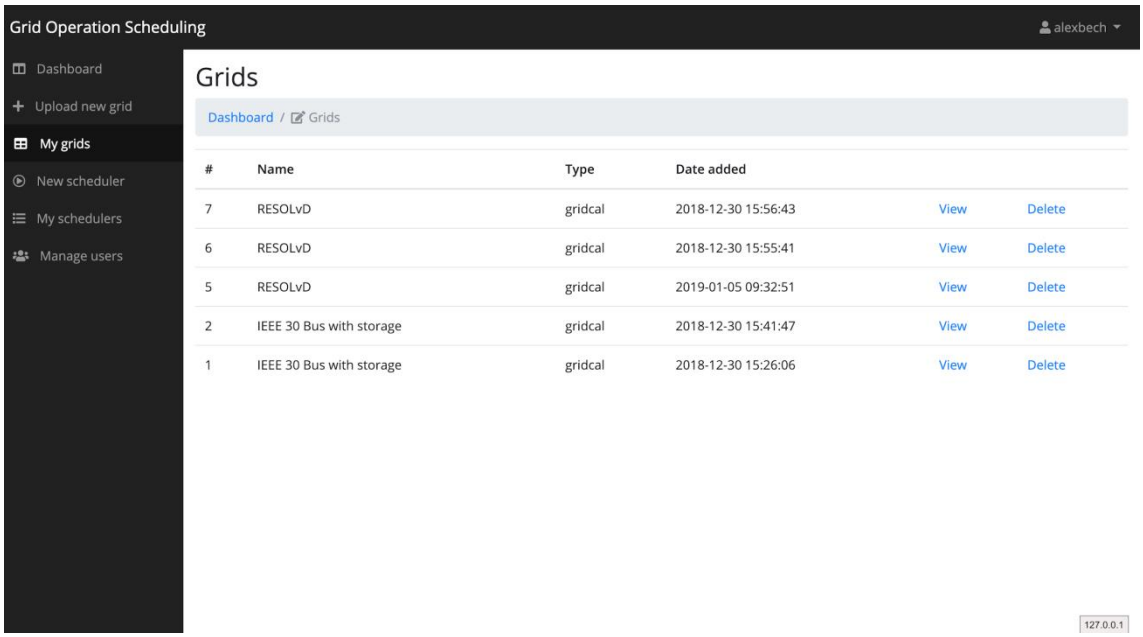

Figura 27. Llistat de xarxes

Pantalla de visualització d'una xarxa

Components: *ViewGridComponent (GridsModule)*, *RenderGridComponent (RenderGridModule), RenderFeaturesComponent (RenderGridModule), RenderFieldComponent (RenderGridModule)*, *RenderProfilesComponent (RenderGridModule)*, *RenderGraphComponent (RenderGraphModule), NodeVisualComponent (RenderGraphModule)* i *LinkVisualComponent (RenderGraphModule)*.

Les Figures 28, 29, 20 mostren la pantalla de visualització d'una xarxa. Permet veure les dades de tots els seus elements i una representació gràfica. Des d'aquesta pantalla també és possible descarregar-se el fitxer original de la xarxa o començar una nova planificació.

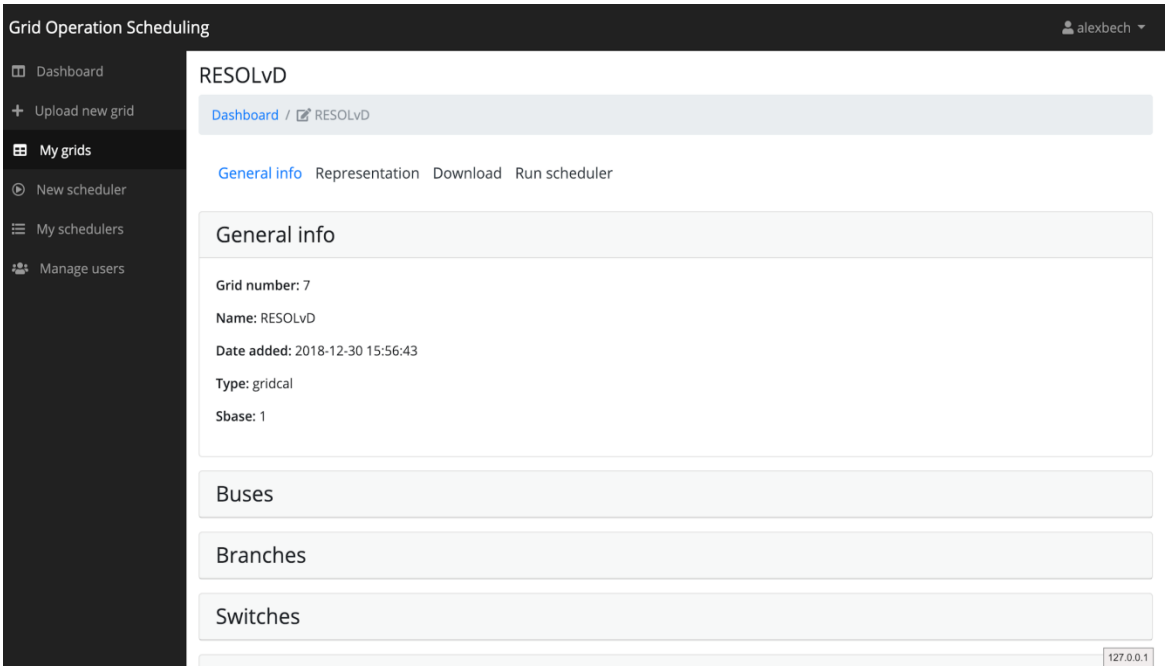

Figura 28. Informació general de la xarxa

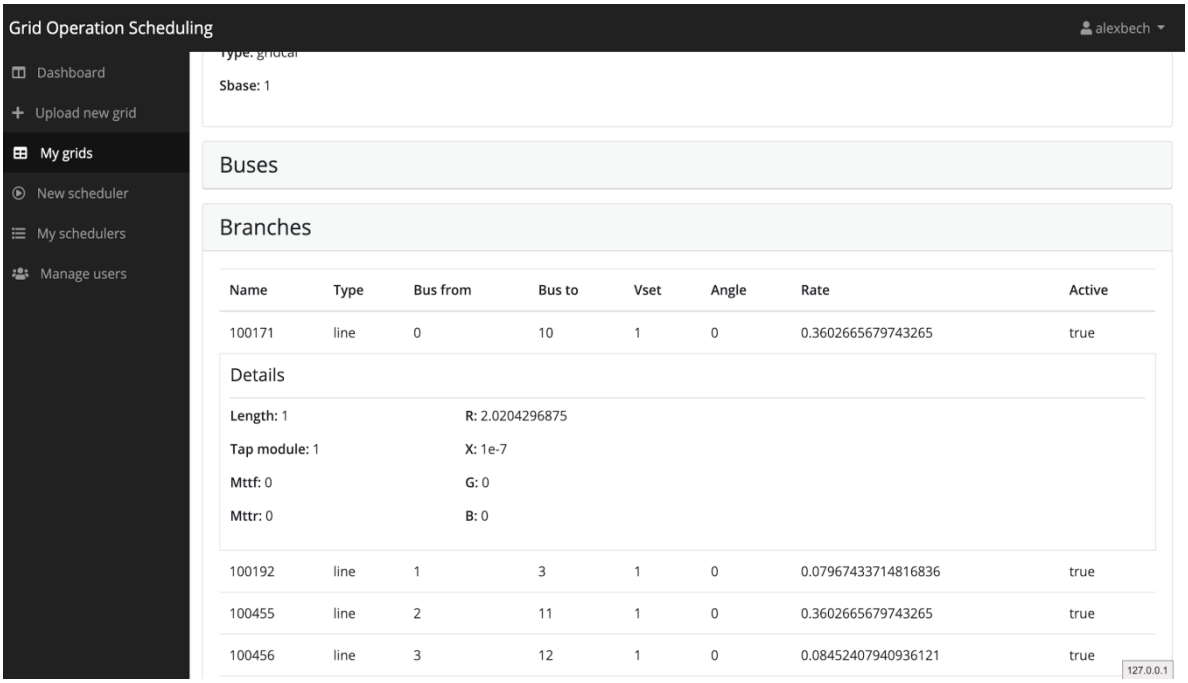

Figura 29. Descripció detallada dels components de la xarxa

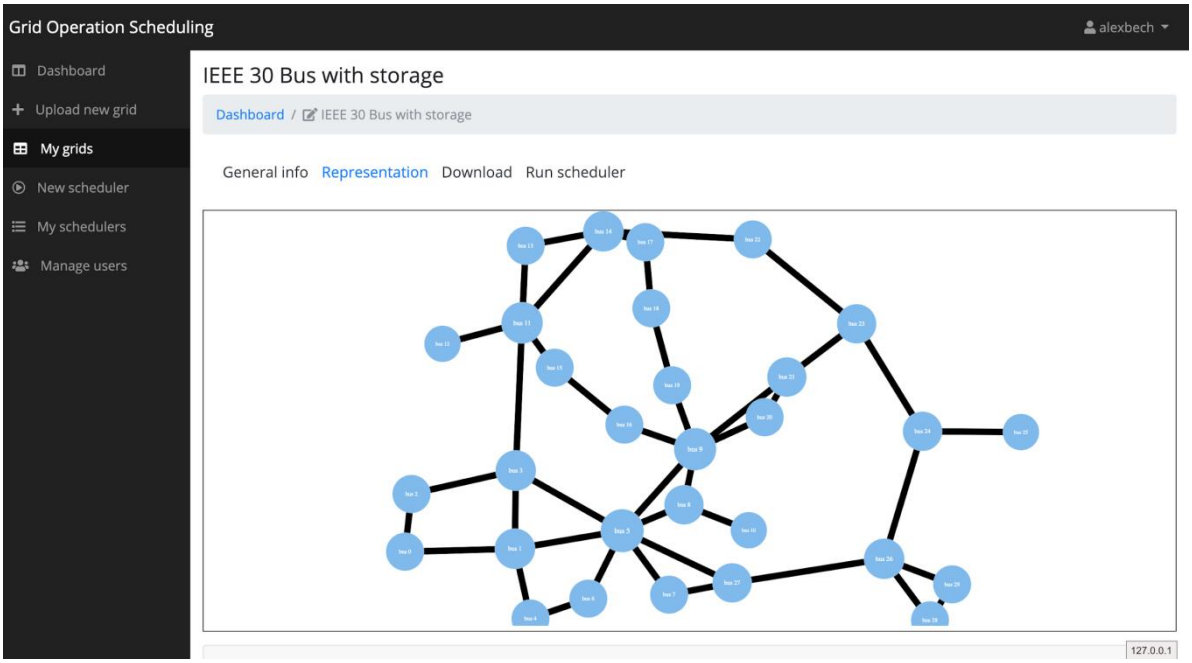

Figura 30. Descripció detallada dels components de la xarxa

Pantalla de nova planificació

Components: *CreateSchedulerComponent* (*SchedulerModule*).

La Figura 31 mostra la pantalla amb el formulari amb tots els paràmetres necessaris per llançar una nova planificació de la xarxa. En funció de l'algoritme d'optimització s'afegiran nous paràmetres dinàmicament al formulari.

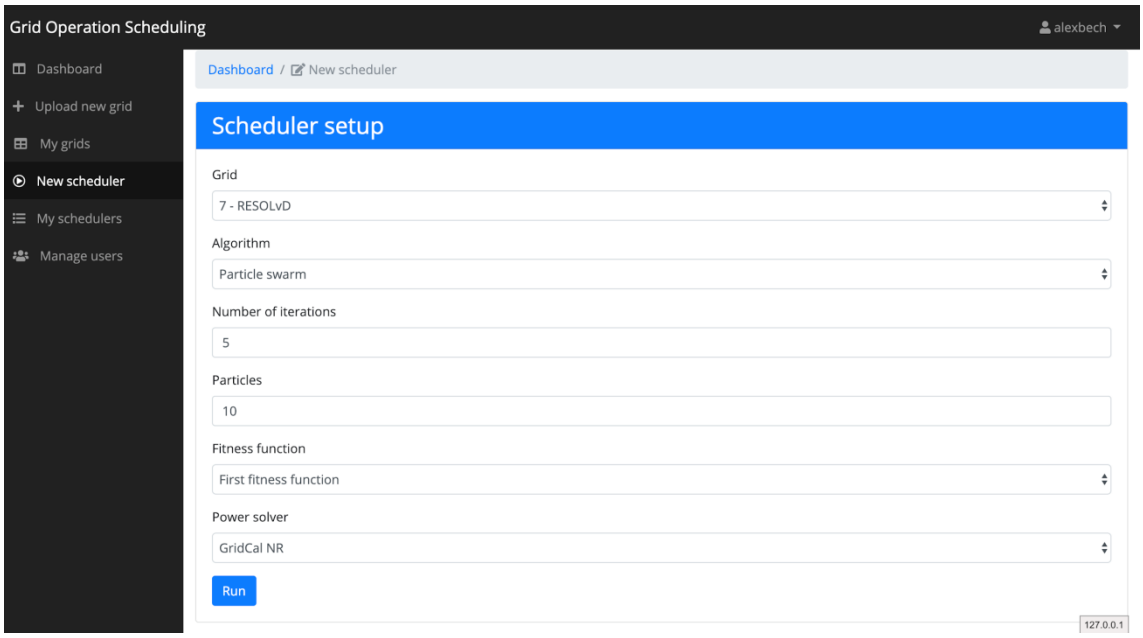

Figura 31. Formulari per llançar una nova planificació

Un cop llançada la planificació podrem accedir a la seva respectiva fitxa i visualitzar els resultats quan hagi finalitzat o llançar una nova planificació com mostra la Figura 32.

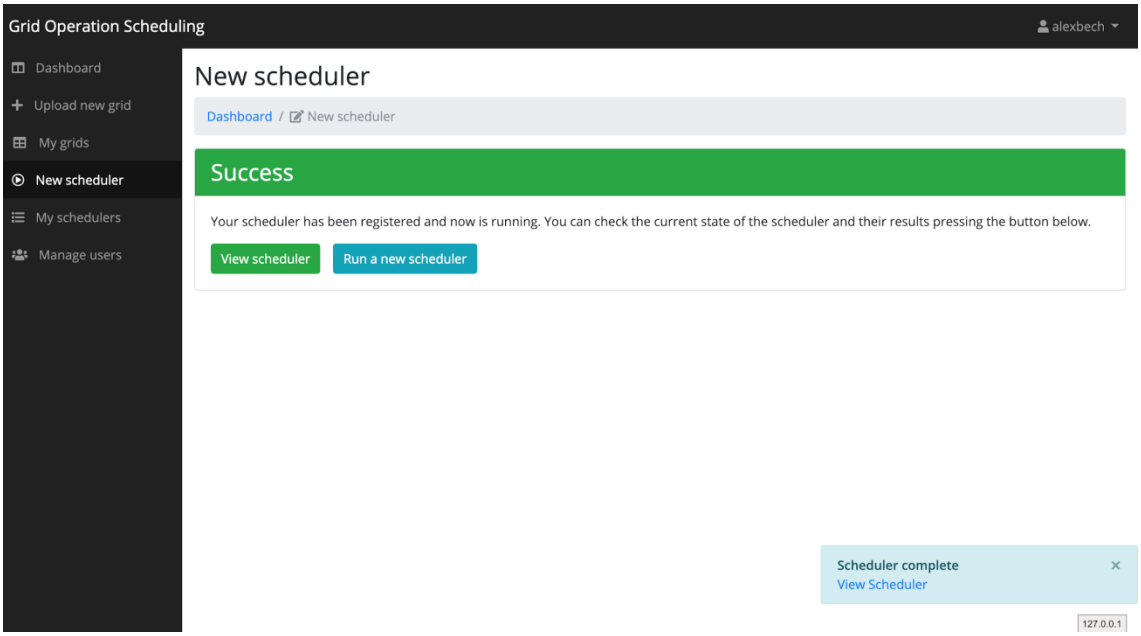

Figura 32. Notificació del llançament d'una nova planificació

Llistat de planificacions

Components: *ListSchedulersComponent* (*SchedulerModule*).

Pantalla amb la llista de planificacions llançades. Com mostra la Figura 33, permet veure detalls rellevants, consultar-les o eliminar-les.

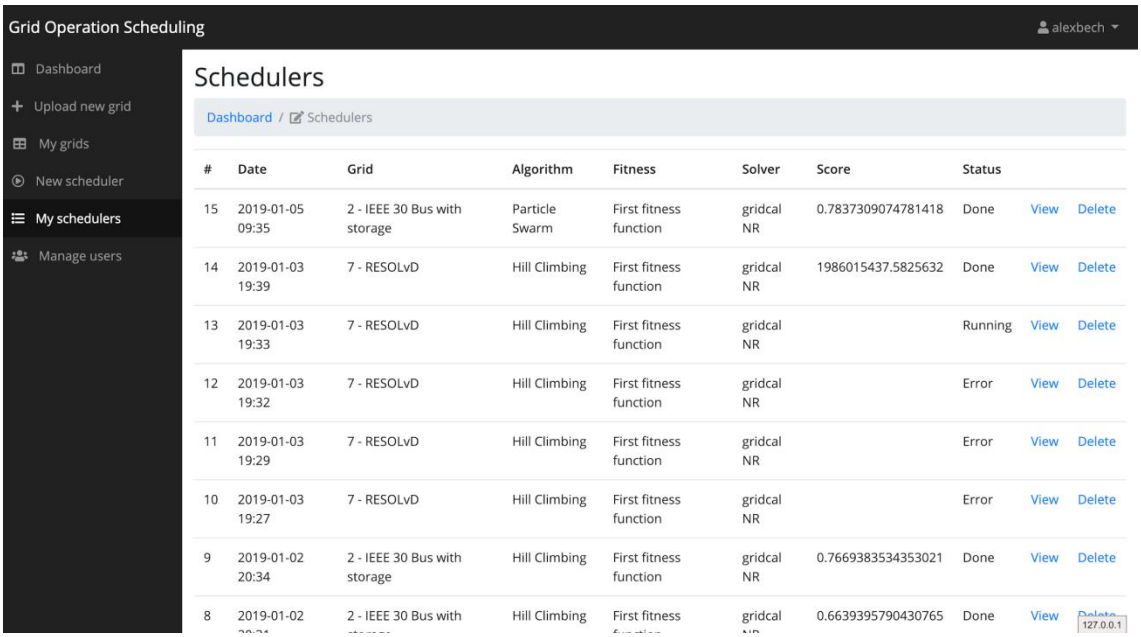

Figura 33. Llistat de planificacions

Pantalla de visualització de la planificació

Components: *ViewSchedulerComponent (SchedulerModule)*, *RenderGridComponent (RenderGridModule)*, *RenderFeaturesComponent (RenderGridModule)*, *RenderFieldComponent (RenderGridModule)*, *RenderProfilesComponent (RenderGridModule)*, *RenderGraphComponent (RenderGraphModule)*, *NodeVisualComponent (RenderGraphModule)* i *LinkVisualComponent (RenderGraphModule)*.

Les Figures 34, 35 i 36 mostren la pantalla de visualització d'una planificació. Consta de la pestanya dels resultats de l'execució de l'algoritme, les dades de la xarxa resultant i la representació gràfica de la xarxa optimitzada i l'original.

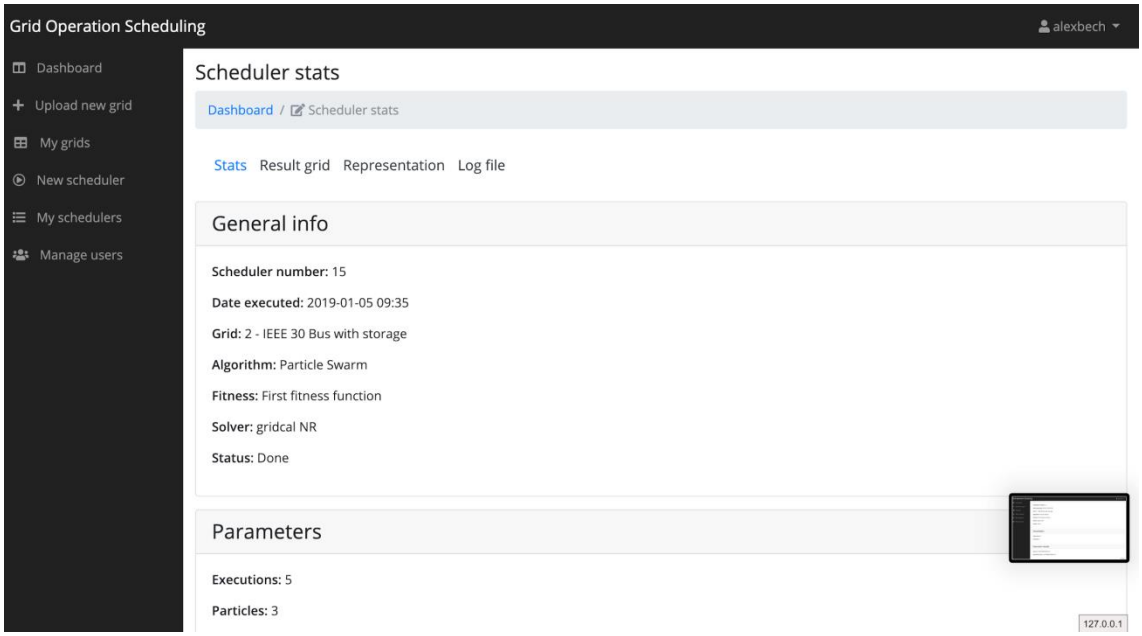

Figura 34. Informació de l'execució de la planificació.

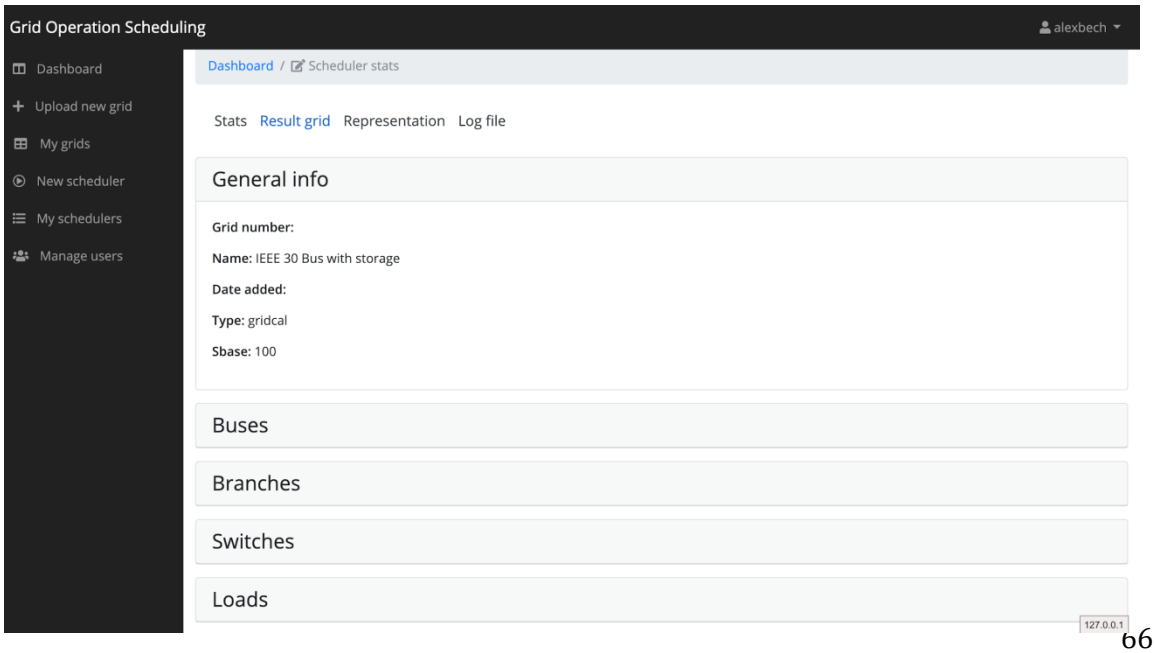

Figura 35. Informació de la xarxa resultant de l'optimització

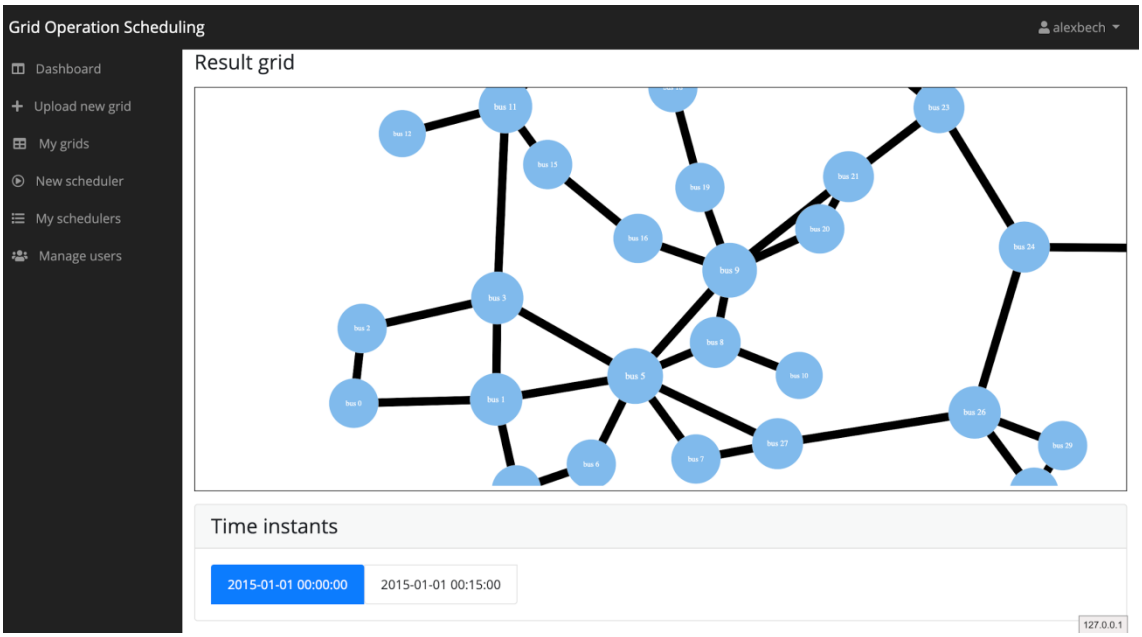

Figura 36. Representació de la xarxa resultant.

Llistat d'usuaris de l'administrador

Components: *AdminUsersListsComponent* (*AdminModule*)

La Figura 37 mostra la pantalla que permet als administradors visualitzar un llistat dels usuaris del servei web i gestionar-los.

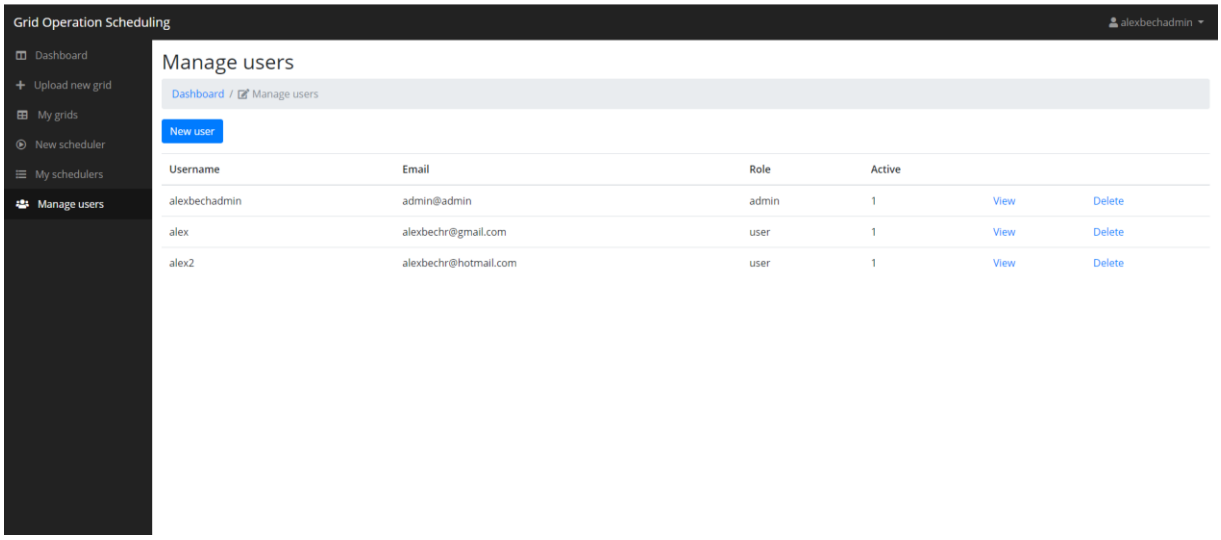

Figura 37 Llistat dels usuaris des de l'administració

Pantalla de creació d'un nou usuari

Components: *AdminNewUserComponent* (*AdminModule*)

Pantalla amb el formulari per crear un nou usuari del servei web com es mostra en la Figura 38.

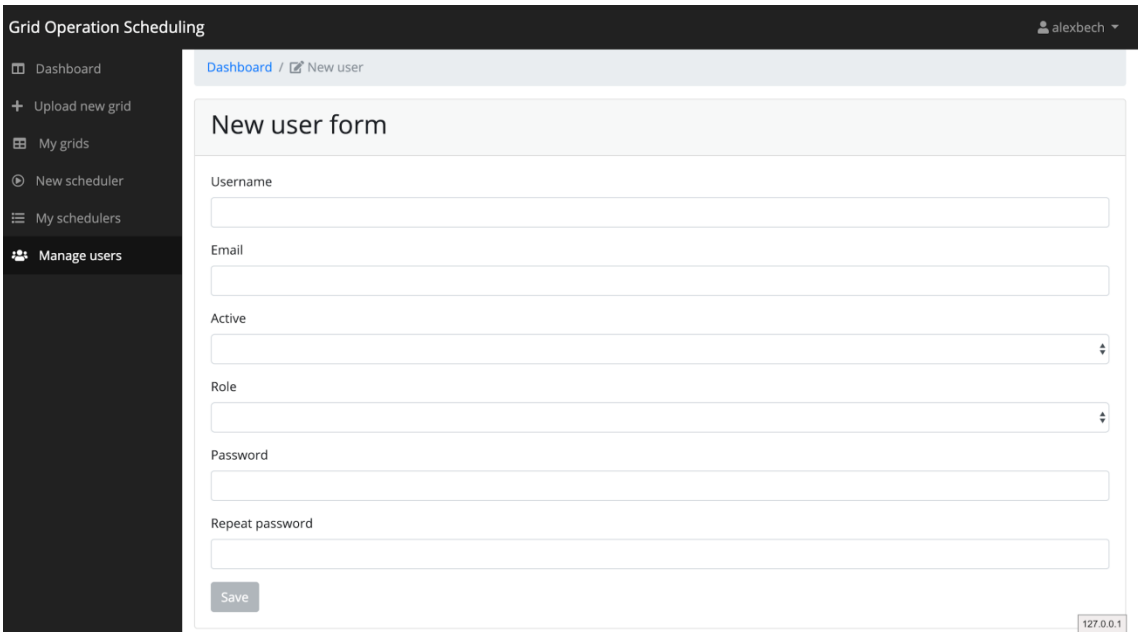

Figura 38. Formulari de creació d'un nou usuari

Pantalla d'edició d'un usuari

Components: *AdminUserComponent* (*AdminModule*)

La pantalla d'edició d'un usuari, mostrada a la Figura 39, està formada pel mateix formulari que la pantalla anterior però per editar les dades d'un usuari ja registrat.

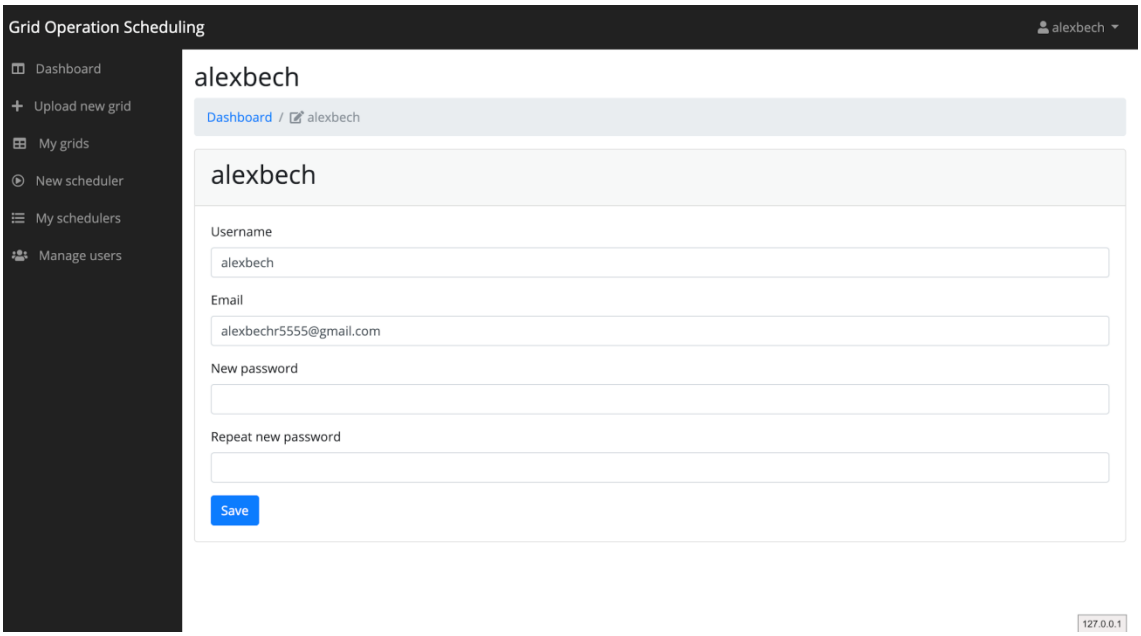

Figura 39. Formulari d'edició d'un usuari

Pantalla de perfil de l'usuari

Components: *UserProfileComponent* (*UsersModule*)

Similar a les pantalla d'edició d'un usuari des de l'administrador, la Figura 40 mostra la pantalla de perfil de l'usuari des d'on pot editar les seves dades.

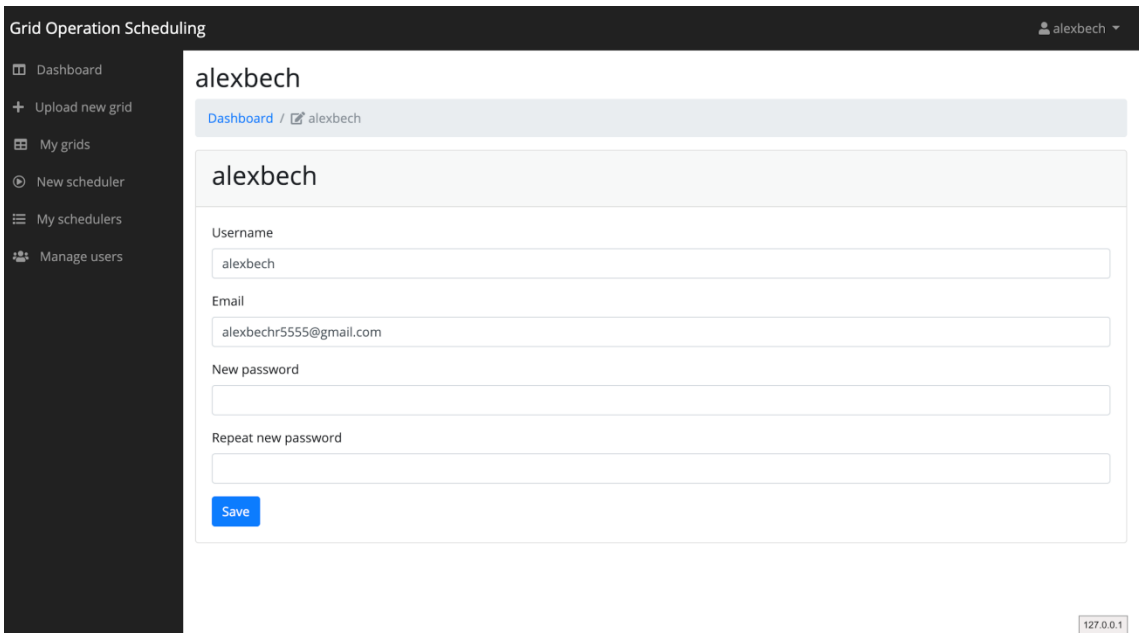

Figura 40. Pantalla de perfil de l'usuari

7.3 Algoritmes d'optimització

En aquest apartat es descriuran els dos algoritmes metaheurístics implementats en el desenvolupament d'aquest projecte

7.3.1 Hill Climbing

L'algoritme *Hill Climbing* és un algoritme d'optimització basat en el mètode del gradient. L'optimització del gradient és una tècnica que consisteix en buscar el punt òptim d'una funció a partir de seguir el seu pendent, ja sigui el màxim (gradient ascendent) o el mínim (gradient descendent) (Luke, 2013).

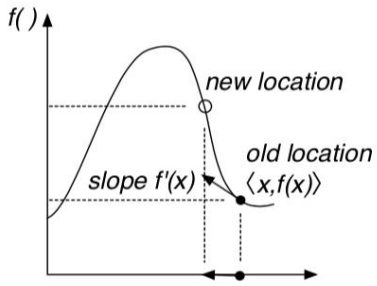

Figura 41. Representació de la tècnica del gradient (Luke, 2013)

Com es pot observar a partir de la Figura 41, el principal requeriment dels mètodes d'optimització del gradient és poder calcular el pendent de la funció a optimitzar. Amb l'algoritme *Hill Climbing* no es necessita saber el pendent de la funció ni quina forma té. Aquest algoritme inicia un procés iteratiu a partir d'un primer punt conegut o solució inicial, a partir de la qual busca noves solucions properes a l'actual. Si troba una millor solució, es buscaran noves solucions des d'aquest nou punt. D'aquesta manera es va "escalant" fins a trobar la millor solució possible (Luke, 2013). La figura 42 mostra el pseudocodi d'aquest algoritme:

Algorithm 4 *Hill-Climbing*

```
1: S ← some initial candidate solution                    ▷ The Initialization Procedure
2: repeat
3:     R ← Tweak(Copy(S))                                 ▷ The Modification Procedure
4:     if Quality(R) > Quality(S) then                    ▷ The Assessment and Selection Procedures
5:         S ← R
6: until S is the ideal solution or we have run out of time
7: return S
```

Figura 42. Algoritme *Hill Climbing*. (Luke, 2013)

El procés d'optimització d'una xarxa amb aquest algoritme consistirà en:

- Executar el solucionador de potències sobre la xarxa inicial, generant una primera solució.
- Avaluar la primera solució amb la funció d'avaluació.
- Fins que no es trobi la solució òptima (segons la funció d'avaluació) o el nombre d'iteracions actuals sigui menor que el nombre màxim d'iteracions:
- Generar una còpia de la xarxa associada a la millor solució trobada.
- Aplicar un modificador sobre la xarxa per generar-ne una de nova. Per modificar la xarxa es farà servir el modificador **RandomTweaker** explicat anteriorment, que modificarà els estats dels interruptors i les potències dels generadors i bateries de manera aleatòria.
- Aplicar el solucionador de potències a la xarxa modificada per obtenir els paràmetres necessaris per avaluar la nova solució.
- Avaluar la nova solució amb la funció d'avaluació.
- Si la nova solució té millor puntuació que la millor solució actual, substituir-la.
- Retornar la millor solució amb la xarxa optimitzada.

Degut al funcionament de l'algoritme és possible que la millor solució trobada no sigui la millor de totes, sinó un màxim (o un mínim) local com mostra la Figura 43:

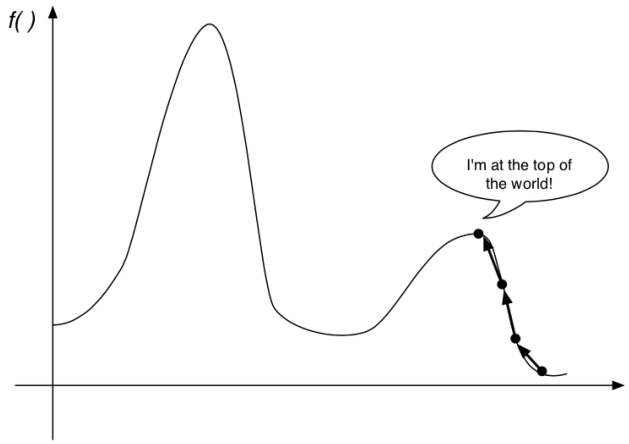

Figura 43. L'algoritme *Hill Climbing* es queda "atrapat" en un màxim local (Luke, 2013)

L'única manera de sortir d'aquests punts locals és esperar que el component aleatori generi noves solucions més properes al punt més òptim.

Aplicació de l'algoritme

A continuació es mostra el resultat d'una execució d'aquest algoritme sobre una xarxa per comprovar si l'algoritme és capaç de trobar millors planificacions. La xarxa inicial que s'ha fet servir en aquesta execució consta de les següents característiques:

- 30 busos.
- 40 branques.
- 19 càrregues.
- 1 generador estàtic.
- 2 bateries.
- 5 generadors controlables.
- 2 shunts.

Aplicant la funció d'avaluació **VLL** descrita anteriorment a la xarxa inicial s'obtenen els mostrats a la Taula 1:

Error de voltatge (p.u)	Excés de càrrega (p.u)	Pèrdues de la xarxa (MWh)	Puntuació
0.013962937374083582	0	1.4675326054858233	1.4814955428599068

Taula 1. Solució inicial de l'algoritme *Hill Climbing*

A partir d'aquests valors es pot observar que la xarxa inicial de la que parteix l'algoritme no presenta excés de càrrega i l'error de voltatge i les pèrdues de la xarxa són baixes. La Taula 2 mostra les puntuacions que s'han obtingut durant l'execució de l'algoritme amb 10 iteracions:

Nº iteració	Puntuació	Millor puntuació
1	1.3657067882942249	1.3657067882942249
2	1.1993043973211988	1.1993043973211988
3	1.1449641492331775	1.1449641492331775
4	1.1294129961870458	1.1294129961870458
5	1.2036492485365333	1.1294129961870458
6	1.132025557547843	1.1294129961870458
7	1.1849991516581748	1.1294129961870458
8	1.2470900333875403	1.1294129961870458
9	1.3201351041178813	1.1294129961870458
10	1.2838399953175461	1.1294129961870458

Taula 2. Puntuacions de les solucions trobades durant l'execució de l'algoritme *Hill Climbing*

Un cop executat l'algoritme, s'observa que a les quatre primeres iteracions s'ha trobat una planificació millor de la xarxa. A partir de la cinquena iteració l'algoritme no ha pogut trobar cap solució millor, possiblement perquè s'ha arribat a un màxim local o el factor aleatori de l'algoritme ha realitzat modificacions sobre la xarxa que no han ajudat a disminuir l'error total. Aplicant la funció d'avaluació sobre la xarxa resultant s'obtenen els valors de la Taula 3, confirmant la millora de la xarxa:

Error de voltatge (p.u)	Excés de càrrega (p.u)	Pèrdues de la xarxa (MWh)	Puntuació
0.013330002719018252	0	1.1160829934680276	1.1294129961870458

Taula 3. Millor solució trobada per l'algoritme *Hill Climbing*

7.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) és una tècnica d'optimització basada en població. Desenvolupada per James Kennedy i Russell Eberhart durant els anys 90, aquesta tècnica es basa en el comportament observat en les bandades d'ocells o els bancs de peixos. Similar als algorismes genètics (GA), PSO manté una població on els seus membres, anomenats partícules, exploren l'espai n-dimensional del problema. Cada punt de l'espai representa una solució. Els moviments que fan aquests partícules són aleatoris, però alhora estan influïts pel millor punt trobat localment i el millor punt trobat entre totes les partícules. Les partícules consten de tres atributs n-dimensionals (Torrent-Fontbona & López, 2016):

- La posició actual dins l'espai: $\overrightarrow{x_i}$
- La millor posició trobada per la partícula: $\overrightarrow{p_i}$
- La velocitat de moviment: $\overrightarrow{v_i}$

La Figura 44 mostra el pseudocodi de l'algoritme:

Algorithm 4 PSO

1: Initialise population with random positions
2: Compute the fitness of each particle
3: **for** $i \leftarrow 1$ **to** $N_{iterations}$ **do**
4: **for each** particle **do**
5: **if** $fitness(\vec{x}_i) > fitness(\vec{p}_i)$ **then**
6: $\vec{p}_i \leftarrow \vec{x}_i$
7: **if** $fitness(\vec{x}_i) > fitness(\vec{p}_g)$ **then**
8: $\vec{p}_g \leftarrow \vec{x}_i$
9: **end if**
10: **end if**
11: **end for**
12: **for each** particle **do**
13: $\vec{v}_i \leftarrow \vec{v}_i \cdot \omega + \Phi_1(\vec{p}_i - \vec{x}_i) + \Phi_2(\vec{p}_g - \vec{x}_i)$
14: $\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$
15: **end for**
16: Compute the fitness of the position of each particle
17: **end for**
18: Select the best position as solution

Figura 44. Algoritme *Particle Swarm Optimization*. Font: (Torrent-Fontbona & López, 2016)

L'execució de l'algoritme comença inicialitzant un nombre determinat de partícules en posicions aleatòries dins l'espai de la solució i s'avaluen les solucions associades a cada punt. A continuació s'executa un procés de N iteracions on a cada iteració s'actualitzen les millors posicions a nivell de partícula i a nivell global. Seguidament s'actualitza la velocitat de les partícules amb la següent expressió:

$$\overrightarrow{v_i} = \overrightarrow{v_i} * \omega + \Phi_1(\overrightarrow{p_i} - \overrightarrow{x_i}) + \Phi_2(\overrightarrow{p_g} - \overrightarrow{x_i})$$

on:

$\overrightarrow{v_i} * \omega$ representa la inèrcia inherent de la partícula.

$\Phi_1(\overrightarrow{p_i} - \overrightarrow{x_i})$ representa la força d'atracció que exerceix el millor punt trobat per la partícula sobre la velocitat de la partícula.

$\Phi_2(\overrightarrow{p_g} - \overrightarrow{x_i})$ representa la força d'atracció que exerceix el millor punt trobat globalment sobre la velocitat de la partícula.

Les variables ω, Φ_1 i Φ_2 són paràmetres per assegurar la convergència de l'algoritme. El concepte de convergència (Figura 45) en un PSO és el fet que les diferents partícules es concentrin en un punt en particular de l'espai de cerca, el qual pot ser òptim o no. Segons un estudi sobre la convergència del PSO fet per Maurice Clerc i James Kennedy (Clerc & Kennedy, 2002), uns possibles valors que asseguren la convergència de l'algoritme són: $\omega = 0.7298$ i $\Phi_1 = \Phi_2 = 1.49618$.

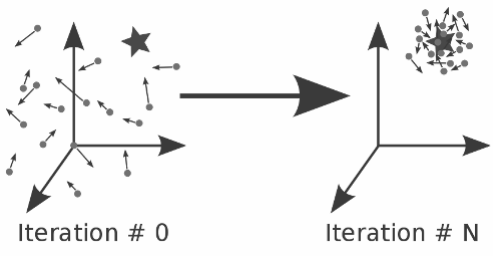

Figura 45. Convergència de les partícules (Clerc & Kennedy, 2002)

Finalment s'actualitza la posició de la partícula amb la nova velocitat i s'avalua la nova posició. Un cop finalitzat el procés iteratiu es retorna el millor punt trobat en l'exploració de l'espai.

A continuació es descriuen les particularitats de l'algoritme PSO implementat per realitzar optimitzacions sobre les xarxes:

- Cada posició de l'espai representa una variació de la xarxa amb diferents valors de potència en generadors controlables i bateries i d'estats de les seves branques en cada instant de temps.
- Per inicialitzar les n partícules es creen n solucions aplicant el modificador RandomTweaker sobre la xarxa inicial.
- La velocitat de les partícules estarà composta de dues matrius de dimensions GxT i BxT, on G és el número de generadors controlables, B el nombre de bateries i T és el nombre d'instants de temps de la xarxa.
- La posició de les partícules s'actualitza modificant les potències dels generadors i de les bateries de la xarxa de la posició actual amb els valors de les matrius de velocitat. L'estat de les branques es canviarà aleatòriament amb el modificador RandomTweaker.

Aplicació de l'algoritme

Finalment es mostra una execució d'aquest algoritme sobre la mateixa xarxa utilitzada en l'execució de l'algoritme *Hill Climbing*. La solució inicial de la que parteix l'algoritme és mostra a la taula 4:

Error de voltatge (p.u)	Excés de càrrega (p.u)	Pèrdues de la xarxa (MWh)	Puntuació
0.013962937374083582	0	1.4675326054858233	1.4814955428599068

Taula 4. Solució inicial de l'algoritme *Particle Swarm Optimization*

L'algoritme s'ha llançat amb 10 partícules i ha constat de 20 iteracions. La Figura 46 mostra el moviment de les partícules durant cada iteració de l'algoritme:

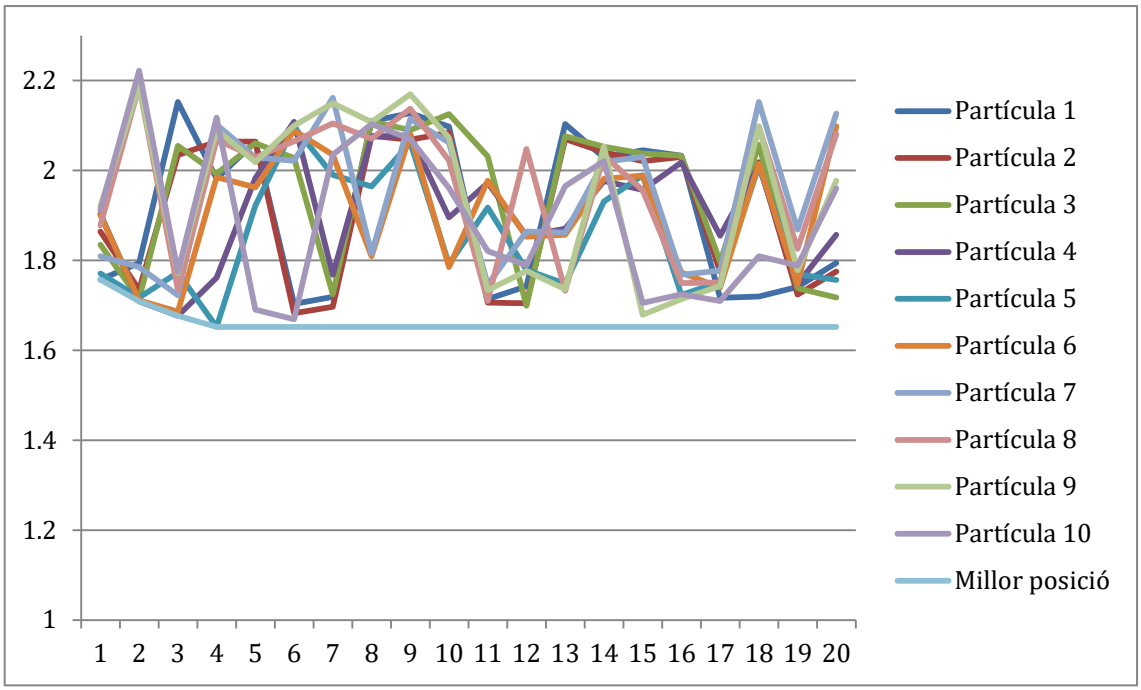

Figura 46. Moviment de les partícules durant l'execució de l'algoritme

A partir del gràfic anterior es pot observar el fenomen de convergència durant algunes iteracions de l'algoritme, com per exemple a les iteracions 9 o 17. També s'observa que la millor posició trobada ha estat a la iteració 4 i que a partir de llavors les partícules no n'han trobat una de millor. La solució associada a la millor posició, i per tant la millor planificació trobada, consta dels següents valors:

Error de voltatge (p.u)	Excés de càrrega (p.u)	Pèrdues de la xarxa (MWh)	Puntuació
0.017632781438503827	0	1.634222730030788	1.6518555114692919

Taula 5. Millor solució trobada per l'algoritme *Particle Swarm Optimization*

8. Conclusions

L'objectiu d'aquest projecte ha estat dissenyar i implementar un servei de planificació de xarxes elèctriques. Aquest servei consisteix en un servei web amb funcionalitats i algoritmes necessaris per trobar una planificació òptima per a la xarxa elèctrica. A més a més, s'ha desenvolupat un client que serveix d'interfície d'usuari.

El servei s'ha dissenyat de tal manera que pugui funcionar amb xarxes construïdes amb diferents llibreries de càlculs de potències o *power flow solvers*. Les xarxes, igual que amb la informació dels usuaris i les planificacions llançades, es guarden en una base de dades per poder ser recuperades posteriorment. El sistema d'usuaris permet portar un control de l'ús del servei web i restringir les funcionalitats en funció del rol de l'usuari.

De la part de planificació del servei web destacar els dos algoritmes metaheurístics d'optimització implementats: l'algoritme *Hill Climbing* i l'algoritme *Particle Swarm Optimization*. També s'ha implementat una funció d'avaluació per mesurar la qualitat de les solucions trobades en funció de tres paràmetres: l'error de voltatge, excés de càrrega i pèrdues totals de la xarxa. Els algoritmes també poden modificar la generació/demanda i l'estat de les branques de la xarxa gràcies als modificadors. Tots aquests elements permeten començar a utilitzar el servei de planificació.

Tot i ser un servei operatiu, el software desenvolupat s'ha dissenyat com a *framework* on implementar i testejar altres algorismes de planificació i altres funcions d'avaluació tal i com demanaven els requeriments del projecte.

L'aplicació client també està operativa i permet interactuar amb el servei web mitjançant una interfície gràfica. A través de la interfície es poden registrar nous usuaris, pujar xarxes al servidor, consultar la seva informació i executar planificacions amb els elements de planificació disponibles. També compta amb una àrea administrativa per gestionar els usuaris.

Finalment, les dues aplicacions s'han desplegat en el servidor d'eXiT en contenidors de Docker. Per tant, es pot afirmar que els objectius definits en aquest projecte s'han complert.

Com a treball futur, el software implementat és susceptible d'afegir nous components, així com paral·lelitzar l'execució d'alguns dels seus elements per millorar el rendiment. D'altra banda, seria convenient implementar un sistema d'integració contínua per millorar la testabilitat i traçabilitat dels algoritmes, detectar i solucionar errors més ràpidament.

Pel que fa al client, també s'hi poden afegir seccions o funcionalitats noves com podrien ser estadístiques dels resultats obtinguts en l'execució dels algoritmes de planificació.

9. Bibliografia

About the project. (sense data). Consultat el 4 / 12 / 2018, a The Resolvd Project: https://resolvd.eu/about-the-project/

Alcantar, L. (sense data). *Apuntes de Sistemas Eléctrics de Potencia.* Consultat el 03 / 12 / 2018, a Research Gate: https://www.researchgate.net/publication/275770730_Apuntes_de_Sistemas_Electricos_de_Potencia

Angular. (sense data). Consultat el 03 / 01 / 2019, a https://angular.io/

API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos. (3 / 2016). Consultat el 10 / 12 / 2018, a BBVAOPEN4U: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

Bollen, M., & Rönnberg, S. (2017). Hosting Capacity of the Power Grid for Renewable Electricity Production and New Large Consumption Equipment. *MDPI: Energies 10*, 1-28.

Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans Evol Comput 6*, 58-73.

Control de acceso HTTP (CORS). (sense data). Consultat el 27 / 12 / 2018, a MozillaDevelopers: https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

Data-Drive Documents. (sense data). Consultat el 03 / 01 / 2019, a D3JS: https://d3js.org/

Electric power. (sense data). Consultat el 23 / 12 / 2018, a Sparkfun: https://learn.sparkfun.com/tutorials/electric-power/all

Flask. (sense data). Consultat el 30 / 12 / 2018, a http://flask.pocoo.org/

Flask-RESTful: User's Guide. (sense data). Consultat el 30 / 12 / 2018, a Flask-RESTful: https://flask-restful.readthedocs.io/en/latest/

Generación distribuida. (sense data). Consultat el 03 / 12 / 2018, a Wikipedia: https://es.wikipedia.org/wiki/Generaci%C3%B3n_distribuida

Grinberg, M. (12 / 2017). *The Flask Mega-Tutorial.* Consultat el 16 / 06 / 2018, a https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Luke, S. (2013). *Essentials of Metaheuristics.* Fairfax: Lulu.

Peñate, S. (10 / 2018). *Manual of GridCal.* Consultat el 23 / 10 / 2018, a GitHub: https://github.com/SanPen/GridCal/blob/master/Documentation/GridCal/Manual_of_GridCal.pdf

Principales ventajas y limitaciones de las metodologías ágiles. (sense data). Consultat el 10 / 12 / 2018, a OBS Business School: https://www.obs-edu.com/es/blog-project-management/metodologia-agile/principales-ventajas-y-limitaciones-de-las-metodologias-agiles

Python. (sense data). Consultat el 23 / 12 / 2018, a https://www.python.org/

Quiroga, B. (Maig / 2017). *Generación distribuida: Una puerta hacia la autogeneración.* Consultat el 23 / 12 / 2018, a Twenergy: https://twenergy.com/a/generacion-distribuida-una-puerta-hacia-la-autogeneracion-2647

Sharir, L. (03 / 2017). *Visualizing Data with Angular and D3.* Consultat el 12 / 10 / 2018, a Medium: https://medium.com/netscape/visualizing-data-with-angular-and-d3-209dde784aeb

Single-page application. (sense data). Consultat el 18 / 12 / 2018, a Wikipedia: https://en.wikipedia.org/wiki/Single-page_application

Tena, M. (2018). *¿Qué es la metodología 'agile'?* Consultat el 10 / 12 / 2018, a BBVA: https://www.bbva.com/es/metodologia-agile-la-revolucion-las-formas-trabajo/

The MongoDB 4.0 Manual. (sense data). Consultat el 22 / 12 / 2018, a MongoDB: https://docs.mongodb.com/manual/

Torrent-Fontbona, F. (2015). *Optimisation methods meet the smart grid: new methods for solving location and allocation problems under the smart grid paradigma.* Universitat de Girona.

Torrent-Fontbona, F., & López, B. (2016). Decision support for grid-connected renewable energy generators planning. *Energy, 115*, 577-590.

ANNEXOS

ANNEX 1. Guia d'instal·lació

En aquest annex es mostraran els passos necessaris per executar el servei web i l'aplicació client localment i en contenidors Docker. Durant aquest annex es considera que s'ha obtingut el codi del projecte del repositori.

Instal·lació del servei web en entorn local

Per poder utilitzar el servei web de planificació caldrà tenir instal·lada una versió de Python superior o igual a 3.6 i una versió de MongoDB superior o igual a 4.0.0:

- Python: https://www.python.org/downloads/
- MongoDB: https://www.mongodb.com/es

Un cop satisfets aquests requeriments es pot procedir a la instal·lació de les llibreries necessàries per executar el servei. Abans és recomanable crear un entorn virtual a l'arrel de la carpeta *server*. Un entorn virtual permet instal·lar les dependències d'un projecte de manera aïllada a la resta del sistema per evitar conflictes entre diferents projectes que necessitin la mateixa llibreria però en versions diferents. La següent comanda crea un entorn virtual en el directori actual:

```
python3 –m venv venv
```

On el primer *venv* és el mòdul Python que permet la creació d'entorns virtuals i el segon *venv* és el nom de l'entorn que es crea. Per instal·lar les llibreries requerides s'utilitza la següent comanda:

```
pip install –r requirements.txt
```

Aquesta comanda executa l'instal·lador de paquets de Python, PyPi, que instal·larà totes les llibreries en la versió corresponent llistades al fitxer *requirements.txt*. Un cop finalitzi l'execució de la comanda ja estaran instal·lades totes les llibreries necessàries pel funcionament del servei web dins l'entorn virtual creat prèviament,

Seguint la instal·lació, cal crear dos fitxers de configuració addicional: *dev.env* i *prod.env*. En aquests fitxers es defineixen variables d'entorn per l'execució del servei web com la URL de la base de dades, la configuració del servei de correu o les rutes de les carpetes de les xarxes i

les planificacions, per exemple. Els camps que es defineixen en aquests fitxers estan definits en el fitxer *environment_schema.txt*:

```
MONGO_URI =
SECRET_KEY =
HTTPS =
BASE_URL =
UPLOAD_GRID_FOLDER =
LOG_FOLDER =
SMTP_HOST =
SMTP_PORT =
SMTP_AUTH =
USERNAME =
PASSWORD =
SMTP_SECURE =
ADMIN_USER =
ADMIN_EMAIL =
ADMIN_PASSWORD =
```

Els valors d'aquests fitxers seran importats durant el llançament del servei web. Depenent de l'entorn definit s'importaran els valors del fitxer *dev.env* o *prod.env*. Un cop creats els dos fitxers només falta llançar el servei web de la base de dades amb la següent comanda:

```
mongod
```

Amb aquesta comanda s'inicialitza el servidor de MongoDB en el seu port per defecte, el 27017. És possible passar més paràmetres en aquesta comanda per personalitzar la inicialització del servei com el port o la direcció IP. Aquestes personalitzacions caldrà tenir-les present a l'hora de definir la URL de la base de dades en el camp MONGO_URI dels fitxers de les variables d'entorn.

Finalment, per executar el servei web de manera local es pot utilitzar la comanda *flask run*, que desplega un servidor local amb el servei desplegat:

```
$ flask run
 * Environment: production
   WARNING: Do not use the development server in a production
environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Per defecte el servidor es desplega en el port 5000. Per comprovar si el desplegament s'ha efectuat correctament a l'accedir a la url http://localhost:5000 s'hauria de visualitzar un missatge de confirmació del desplegament.

Desplegament del servei web amb Docker

Com prerequisit principal per desplegar el servei web amb Docker primer caldrà descarregar-se el software del web oficial: https://docs.docker.com/

El desplegament d'aplicacions amb Docker consta de dos elements principals: les imatges i els contenidors. Les imatges són fitxers d'instruccions que defineixen l'entorn en el que s'executaran les aplicacions. És habitual construir noves imatges a partir d'altres, el que es coneixen com imatges derivades. Els contenidors, per la seva part, construeixen els entorns descrits a les imatges i permeten el desplegament d'aplicacions de manera aïllada d'altres contenidors i del sistema que els conté. L'avantatge principal d'aquesta sistema és que un cop creada la imatge, aquesta pot ser desplegada en qualsevol sistema amb Docker. A la carpeta *server* es troba el fitxer *Dockerfile* que representa la imatge del servei web:

```
FROM python:3.6-stretch

RUN apt update
RUN apt install -y libgl1-mesa-glx

RUN useradd gridoperationscheduling

WORKDIR /home/gridoperationscheduling

COPY requirements.txt requirements.txt
RUN python -m venv venv
RUN venv/bin/pip install -r requirements.txt
RUN venv/bin/pip install gunicorn

COPY app app
COPY prod.env server.py config.py boot.sh install_user.py ./

RUN chmod +x boot.sh

ENV FLASK_APP server.py

RUN chown -R gridoperationscheduling:gridoperationscheduling ./
USER gridoperationscheduling

RUN mkdir grids
RUN mkdir logs

CMD ["python", "install_user.py"]

EXPOSE 5000
ENTRYPOINT ["./boot.sh"]
```

En aquest fitxer s'indica que la imatge es construeix a partir de la imatge *python:3.6-stretch*, que conté una versió del sistema operatiu Debian 9 Stretch amb la versió de Python 3.6 ja està instal·lada. Les següents línies consisteixen en la instal·lació de les llibreries necessàries per l'execució del servei web, similar al procediment fet durant la instal·lació local, la definició de les variables d'entorn i els fitxers que el contenidor ha de copiar per desplegar el servei web. Finalment, les dues últimes línies indiquen el port pel qual es lligarà el contenidor amb el *host* i el fitxer que s'executarà quan es construeixi el contenidor. El fitxer *boot.sh* conté les instruccions per llançar el servei web un cop s'executi el contenidor:

```
#!/bin/bash
source venv/bin/activate
exec gunicorn –b :5000 ––access–logfile – ––error–logfile – server:app
```

Abans de llançar un contenidor amb l'anterior imatge cal configurar un altre contenidor amb el servidor la base de dades i configurar-lo perquè sigui accessible des del contenidor del servei web. Per realitzar aquesta configuració es pot fer servir l'eina **Docker Compose** que permet gestionar de manera senzilla les relacions entre contenidors a partir d'un fitxer de configuració. El fitxer *docker-compose.yml* conté la configuració entre els dos contenidors:

```
version: "3"
services:
  web:
    build: .
    ports:
      – "8000:5000"
    volumes:
      – .:/app
    environment:
      – PROD_MODE=1
    links:
      – db
    depends_on:
      – db
  db:
    image: mongo:latest
    environment:
      – MONGO_DATA_DIR=/usr/data/db
      – MONGO_LOG_DIR=/dev/null
    volumes:
      – ./data/db:/usr/data/db
    ports:
      – 27017:27017
```

En aquest fitxer es defineixen dos contenidors com a serveis: el servei web (web) i la base de dades (db). En el cas del servei web, s'especifica que es construeix a partir del fitxer Dockerfile de la mateixa carpeta, la relació entre els ports del host i del contenidor, les variables d'entorn que es passen al contenidor i les relacions i dependències que té, en aquest cas les de la base de dades. Pel contenidor de la base de dades s'especifica que es construeix a partir de la imatge *mongo:latest*, el mapeig dels ports, les variables d'entorn i el mapeig dels volums de dades. És important especificar el mapeig dels volums de dades per no perdre les dades un cop els contenidors es destrueixin.

Finalment, per llançar l'aplicació en contenidors de Docker es fa servir la següent comanda:

```
docker-compose up --build
```

Aquesta comanda llegeix el fitxer *docker-compose.yml* anterior i construeix els contenidors amb el servei web i la base de dades. Per comprovar que el desplegament ha anat bé a la url http://localhost:8000 hauria d'aparèixer el mateix missatge que apareixia al desplegar el servei web de manera local.

Desplegament de l'aplicació client en entorn local

Per desplegar l'aplicació client cal tenir instal·lat NodeJS en una versió superior a la 9.6: https://nodejs.org/es/.

Un cop instal·lat NodeJS, cal instal·lar les dependències de l'aplicació amb el gestor de paquets NPM, el qual ja ve inclòs amb NodeJS. Per instal·lar les dependències cal executar la següent comanda a l'arrel del directori de l'aplicació:

```
npm install
```

Aquesta comanda instal·la les dependències definides al fitxer *package.json*. Un cop acabi la instal·lació, només cal executar la següent comanda per llançar un servidor amb l'aplicació client:

```
ng serve --open
```

Si l'aplicació es desplega correctament a la url http://localhost:4200 es podrà veure la pantalla d'autenticació.

Desplegament de l'aplicació client amb Docker

Similar al desplegament del servei web, es defineix la imatge de l'aplicació client amb un fitxer *Dockerfile* a l'arrel de l'aplicació:

```
FROM node:9.6.1

RUN mkdir /usr/src/app
WORKDIR /usr/src/app

ENV PATH /usr/src/app/node_modules/.bin:$PATH

COPY package.json /usr/src/app/package.json
RUN npm install
RUN npm install –g @angular/cli@1.7.1

COPY . /usr/src/app

CMD ng serve ––host 0.0.0.0
```

En aquest cas, la imatge es construeix a partir d'una imatge amb NodeJS instal·lat i es configura l'entorn. Finalment, la comanda a fer servir quan es construeixi el contenidor permet llançar l'aplicació. El fitxer *docker-compose.*yml per aquesta aplicació, similar a l'anterior, és el següent:

```
version: '3'
services:
  client:
    container_name: client
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      – '.:/usr/src/app'
      – '/usr/src/app/node_modules'
    ports:
      – '4200:4200'
```

Per construir el contenidor es fa servir la comanda *docker-compose up --build* utilitzada prèviament. Com abans, si el desplegament ha anat bé a la url http://localhost:4200 es podrà veure l'aplicació.

ANNEX 2. Rutes API REST

Per cada ruta es llista el seu URI, l'operació HTTP, els paràmetres que s'esperen en la petició i una breu descripció del que contindrà la resposta. Si no s'especifica un format en concret, es considera que el format de les respostes és JSON.

Les rutes es poden classificar en tres gran grups:

- Rutes d'usuari.
- Rutes de xarxes.
- Rutes de planificacions.

Per cada ruta també s'especifica si es necessita autenticació per accedir al recursos i el rol de l'usuari requerit. En cas que no una petició no satisfaci els requeriments per accedir a un recurs el servei web retornarà un error HTTP 401.

POST /user

Descripció

Registra un nou usuari a la base dades.

Autenticació

Necessita autenticació?	No
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom	Tipus	Obligatori?	Descripció
username	string	Sí	Nom de l'usuari
email	string	Sí	Email de l'usuari
password	string	Sí	Contrasenya de l'usuari

Resposta

Nom del paràmetre	Tipus	Descripció
added	boolean	Resultat del registre. Valor true si el registre s'ha efectuat amb èxit, false si hi hagut algun problema.
msg	string	Missatge auxiliar en cas d'error.

Notes

En el cas que el registri s'efectuï correctament, s'enviarà un correu electrònic al nou usuari amb un enllaç per activar el seu compte. Els usuaris que no hagin activat el seu compte no podran fer servir les seccions restringides del servei web.

POST /admin/user

Descripció

Ruta pels administradors per registrar nous usuaris a la base de dades. La ruta permet paràmetres addicionals que no es poden incloure en un registre normal.

Autenticació

Necessita autenticació?	Sí
Necessita rol d'administrador?	Sí

Paràmetres d'entrada

Nom	Tipus	Obligatori?	Descripció
username	string	Sí	Nom de l'usuari
email	string	Sí	Email de l'usuari
password	string	Sí	Contrasenya de l'usuari
active	enter	Sí	Estat del nou usuari
role	string	Sí	Rol de l'usuari

Resposta

Nom del paràmetre	Tipus	Descripció
added	boolean	Resultat del registre. Valor true si el registre s'ha efectuat amb èxit, false si hi hagut algun problema.
msg	string	Missatge auxiliar en cas d'error.

GET /user/:id

Descripció

Recupera la informació d'un usuari.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de l'usuari a la base de dades.

Resposta

JSON amb la informació de l'usuari buscat.

Notes

Un usuari no administrador només pot fer servir aquesta ruta per accedir a la seva pròpia informació, mentre que un usuari administrador pot consultar la informació de tots els usuaris.

Si un usuari no administrador intenta consultar la informació d'un altre usuari retornarà un error HTTP 401.

Si l'usuari a buscar no existeix es retornarà un error 404.

PUT /user/:id

Descripció

Actualitza un usuari.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
email	string	Sí	Nou compte de l'usuari.
password	string	No	Nova contrasenya.

Resposta

Nom del paràmetre	Tipus	Descripció
updated	boolean	Cert si l'usuari s'ha actualitzat correctament, false si l'actualització ha fallat.
msg	string	Missatge amb informació auxiliar sobre el resultat de l'operació.

Notes

Els usuaris només editar la seva pròpia informació, mentre que els usuaris administrador poden modificar la de tots els usuaris.

DELETE /user/:id

Descripció

Elimina un usuari de la base de dades.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	Sí

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades de l'usuari

Resposta

Nom del paràmetre	Tipus	Descripció
deleted	boolean	Resultat de l'operació

Notes

Es retornarà un error 404 si l'usuari buscat no existeix.

GET /user/:id/send_activation_link

Descripció

Envia un correu amb un enllaç d'activació a l'usuari.

Informació

Necessita autenticació?	No
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades de l'usuari

Resposta

Nom del paràmetre	Type	Descripció
result	boolean	Resultat de l'enviament

GET /user/:id/activate

Descripció

Activa el compte d'un usuari.

Informació

Necessita autenticació?	No
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de l'usuari

Resposta

Nom del paràmetre	Tipus	Descripció
result	string	Missatge amb el resultat de l'activació

GET /token

Descripció

Genera un nou *token* per a un usuari.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
user	enter	No	Indica si es vol recuperar el rol i l'identificador de l'usuari

No necessita paràmetres d'entrada. S'identificarà l'usuari pel que es generarà el *token* a partir de les credencials adjuntes a la capçalera de la petició HTTP.

Resposta

Nom del paràmetre	Tipus	Descripció
token	string	Nou token assignat a l'usuari
role	string	Rol de l'usuari (si user = 1)
id	string	Identificador de l'usuari (si user=1)

POST /grid

Descripció

Registra una nova xarxa la base de dades.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
input_type	string	Sí	Tipus de pujada (fitxer o JSON)
type	string	Sí	Tipus de xarxa pujada (més informació a la secció de paràmetres)
file	Fitxer	No	Si el paràmetre *input_type* és *file*, aquest camp contindrà un fitxer Excel amb la xarxa
json_input	string	No	Si el paràmetre *input_type* és *json*, aquest camp contindrà una cadena de text en format JSON amb la informació de la xarxa

Resposta

Nom del paràmetre	Type	Descripció
sucess	boolean	Resultat del registre de la xarxa.
id_grid	string	Si el registre s'ha efectuat amb èxit, aquest camp contindrà l'identificador de la base de dades de la xarxa

Notes

Encara que ni el camp *file* o el camp *json_input* estan especificats com a obligatoris, és necessari que almenys un dels dos estigui present en la petició.

GET /grid/:id

Descripció

Consulta la informació d'una xarxa.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades

Resposta

JSON amb la informació de la base de dades.

Notes

Si la xarxa sol·licitada no existeix es retornarà un error 404.

GET /grid/:id/download

Descripció

Descarrega el fitxer d'una xarxa.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades

Resposta

Fitxer de la xarxa en format Excel o JSON depenent de la forma en que es va pujar al servidor.

Notes

Si la xarxa sol·licitada no existeix es retornarà un error 404.

DELETE /grid/:id

Descripció

Elimina una xarxa de la base de dades.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades

Resposta

Nom del paràmetre	Type	Descripció
result	string	Missatge amb el resultat de l'operació

Notes

Si la xarxa sol·licitada no existeix es retornarà un error 404.

GET /grids

Descripció

Retorna un llistat amb totes les xarxes d'un usuari.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

No necessita paràmetres d'entrada.

Resposta

Retorna una llista amb totes les xarxes de l'usuari.

GET /grid/types

Descripció

Retorna un llistat amb tots els tipus de xarxes disponibles

Informació

Necessita autenticació?	No
Necessita rol d'administrador?	No

Paràmetres d'entrada

No necessita paràmetres d'entrada.

Resposta

Llista amb tots els tipus de xarxa disponibles.

GET /scheduler/params

Descripció

Retorna un llistat amb tots els elements disponibles per llançar noves planificacions: algoritmes, funcions d'avaluació i solucionadors de potències.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

No necessita paràmetres d'entrada.

Resposta

Llistat amb tots els elements disponibles per llançar noves planificacions.

POST /scheduler

Descripció

Executa una nova planificació.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
grid	string	Sí	Identificador de la base de dades de la xarxa
algorithm	string	Sí	Nom de l'algoritme a fer servir
fitness	string	Sí	Nom de la funció d'avaluació
solver	string	Sí	Nom del solucionador
executions	enter	No	Nombre d'iteracions de l'algoritme
register_only	enter	No	Indica si es vol executar la planificació immediatament després de registrar-la

Resposta

JSON amb les dades de la planificació. En cas que el paràmetre register_only sigui 0, el JSON també contindrà els resultats i la xarxa optimitzada.

Notes

En funció de l'algoritme és possible que la petició inclogui paràmetres addicionals, a la secció de paràmetres es descriuen totes les possibilitats.

POST /scheduler/:id/execute

Descripció

Executa una planificació de la base de dades.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades de la planificació

Resposta

JSON amb la mateixa informació que en la ruta POST scheduler amb register_only = 0.

Notes

Si la planificació sol·licitada no existeix es retornarà un error HTTP 404. Només s'executarà la planificació si no s'ha executat prèviament.

GET /scheduler/:id

Descripció

Consulta els resultats d'una planificació.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades de la planificació

Resposta

Informació de la planificació consultada.

Notes

Si la planificació sol·licitada no existeix es retornarà un error HTTP 404.

GET /scheduler/:id/log

Descripció

Descarrega el registre de l'execució d'una planificació.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades de la planificació

Resposta

Fitxer .log amb el registre de l'execució de la planifiació

Notes

Si la planificació sol·licitada no existeix es retornarà un error HTTP 404. Només es retornarà el fitxer si la planificació ha estat executada prèviament.

DELETE /scheduler/:id

Descripció

Elimina una planificació.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

Nom del paràmetre	Tipus	Obligatori?	Descripció
id	string	Sí	Identificador de la base de dades de la planificació

Resposta

Nom del paràmetre	Tipus	Descripció
result	boolean	Resultat de l'operació

Notes

Si la planificació sol·licitada no existeix es retornarà un error 404.

GET /schedulers

Descripció

Retorna un llistat de les planificacions d'un usuari.

Informació

Necessita autenticació?	Sí
Necessita rol d'administrador?	No

Paràmetres d'entrada

No necessita paràmetres d'entrada.

Resposta

Llistat amb totes les planificacions de l'usuari.

Informació sobre els paràmetres

Tipus de xarxes disponibles

Tipus	Nom del paràmetre
GridCal	gridcal

Algoritmes de planificació disponibles

Algoritme	Nom del paràmetre
Hill climbing	hill-climbing
Particle swarm	Particle-swarm

Paràmetres addicionals dels algoritmes

Algoritme	Nom del paràmetre	Tipus	Descripció
Particle swarm	particles	enter	Nombre de partícules a generar en l'execució de l'algoritme

Funcions d'avaluació disponibles

Nom	Descripció	Nom del paràmetre
VLL	Avalua les solucions en funció del sobrevoltatge, la sobrecàrrega I les pèrdues de la xarxa	vll

Solucionadors de potència disponibles

Nom	Descripció	Nom del paràmetre
NR-GridCal	Aplica el mètode Newton Raphson linear per calcular els fluxos de potència	gridcal-nr