FESA WEB

Jordi Ustrell Garrigos
Màster en Enginyeria Informàtica
Desenvolupament d'aplicacions web

César Pablo Cócoles Briongos
Ignasi Lorente Puchades

Gener 2019

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Copyright ©2019 Jordi Ustrell Garrigos.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

© Jordi Ustrell Garrigos
Reservats tots els drets. Està prohibit la reproducció total o parcial d'aquesta obra per qualsevol mitjà o procediment, compresos la impressió, la reprografia, el microfilm, el tractament informàtic o qualsevol altre sistema, així com la distribució d'exemplars mitjançant lloguer i préstec, sense l'autorització escrita de l'autor o dels límits que autoritzi la Llei de Propietat Intel•lectual.

FITXA DEL TREBALL FINAL

Títol del treball:	*FESA WEB*
Nom de l'autor:	Jordi Ustrell Garrigos
Nom del consultor/a:	*Ignasi Lorente Puchades*
Nom del PRA:	*César Pablo Cócoles Briongos*
Data de lliurament (mm/aaaa):	*01/2019*
Titulació o programa:	*Màster en Enginyeria Informàtica*
Àrea del Treball Final:	*Desenvolupament d'aplicacions web*
Idioma del treball:	*Català*
Paraules clau	*Pàgina web, dades en temps real, web sockets.*

Resum del Treball (màxim 250 paraules):

L'objectiu d'aquest treball és crear una interfície lleugera per sol·licitar dades en temps real de diferents dispositius que monitoritzen i controlen els feixos de protons dels 4 experiments que avui en dia treballen en el CERN (ATLAS, ALICE, CMS i LHCb). Actualment s'utilitzen una o varies aplicacions fetes amb JAVA, C++ o Python per a cada dispositiu.

Per fer això s'ha desenvolupat un front-end i back-end d'una pàgina web per realitzar subscripcions sobre els diferents dispositius i rebre les dades d'aquests. Els usuaris poden visualitzar les dades en diferents formats, com un únic valor, com una taula, com una gràfica o com una imatge.

Pel desenvolupar el front-end s'ha utilitzat HTML5, CSS3, Bootstrap 4, JavaScript i Jquery. En back-end consisteix en un servidor Apache Tomcat juntament amb Java, Maven i fitxers amb format JSON per emmagatzemar les subscripcions creades pels usuaris.

Abstract (in English, 250 words or less):

The goal of this work is to create a light interface for requesting real-time data from different devices, which monitor and control the proton beams of the 4 experiments that are presently operating at CERN (ATLAS, ALICE, CMS and LHCb). Up to now, one or more applications made with JAVA, C ++ or Python have been used for each device.

To do this, a front-end and back-end of a webpage has been developed to make subscriptions to the different devices and receive their data. Users can view the data in different formats, as a single value, such as a table, such as a graphic or as an image.

To develop the front-end, HTML5, CSS3, Bootstrap 4, JavaScript and Jquery have been used. The back-end consists of an Apache Tomcat server using Java, Maven and JSON files to store the subscriptions created by users.

Índex

1. Introducció 1
1.1 Context i justificació del Treball 1
1.2 Objectius del Treball 2
1.3 Enfocament i mètode seguit 2
1.4 Planificació del Treball 3
1.5 Breu sumari de productes obtinguts 4
1.6 Breu descripció dels altres capítols de la memòria 4
2. Diagrames UML 5
2.1 Front-end 5
2.2 Back-end 7
3. Usabilitat/UX (DCU) 11
3.1 Perfils d'usuari 11
3.2 Diagrama dels casos d'ús 11
3.3 Especificació dels casos d'ús 12
3.4 Prototips 20
4. Arquitectura 33
4.1 Client 33
4.2 Servidor 33
4.3 Bases de dades 34
4.4 APIs/Llibreries de tercers 36
4.5 Arquitectura de la capa Front-end 38
4.6 Arquitectura de la capa Back-end 45
5. Desenvolupament 61
5.1 Extractes de codi 61
5.2 Test 73
5.3 Versions 74
5.4 Bugs 75
6. Conclusions 79
7. Bibliografia 80

Llista de figures

Figura 1 Diagrama UML del front-end 5
Figura 2 Diagrama UML del Back-end 7
Figura 3 Diagrama dels casos d'ús 11
Figura 4 Prototip Lo-Fi de la pàgina web 21
Figura 5 Prototip en Lo-Fi de la pàgina web amb una subscripció activa 22
Figura 6 Prototip Lo-Fi de la opció de menú "selecció d'usuari" desplegada ... 22
Figura 7 Prototip Lo-Fi de la opció de menú "subscripció sota demanda" desplegada 23
Figura 8 Prototip Lo-Fi de la opció de menú "configuració" desplegada 24
Figura 9 Prototip Lo-Fi de la representació d'un bloc d'un camp de tipus escalar 24
Figura 10 Prototip Lo-Fi de la representació d'un bloc amb la resta d'opcions del menú desplegades 26
Figura 11 Prototip Lo-Fi de l'aplicació per la creació de Settings 27
Figura 12 Prototip en Hi-Fi de la pàgina web buida 27
Figura 13 Prototip Hi-Fi amb una subscripció activa 28
Figura 14 Prototip Hi-Fi opció de *menú "Load Settings"* 28
Figura 15 Prototip Hi-Fi opció de menú "On-demand Subscription" i exemple recerca de dispositius 29
Figura 16 Prototip Hi-Fi opció de menú "Settings" 29
Figura 17 Prototip Hi-Fi visualització de dades com valor escalar 30
Figura 18 Prototip Hi-Fi visualització de dades com valor escalar amb menú desplegat 30
Figura 19 Prototip Hi-Fi visualització de dades com una gràfica 31
Figura 20 Prototip Hi-Fi visualització de dades com una imatge 31
Figura 21 Prototip Hi-Fi visualització de dades com una taula 32
Figura 22 Prototip Hi-Fi de l'aplicació per la creació de Settings 32
Figura 23 Model de l'arquitectura de FESA Web 33
Figura 24 Diagrama de flux de la creació d'un objecte per emmagatzemar dades d'un camp 40
Figura 25 Diagrama de flux per la creació d'una instància de les classes que hereten de FieldBox 41
Figura 26 Diagrama UML del front-end versió 1.0 74

Llista de taules

Taula 1 Fites de la planificació 3
Taula 2 Esdeveniment: “Realitzar un subscripció sota demanda”..... 14
Taula 3 Esdeveniment: “Carregar llistat de Settings d'un usuari específic”..... 15
Taula 4 Esdeveniment: “Realitzar un subscripció per Setting” 17
Taula 5 Esdeveniment "Crear un Setting" 18
Taula 6 Esdeveniment: "Editar un Setting"..... 19
Taula 7 Esdeveniment: "Eliminar un Setting" 20
Taula 8 Configuració servidor..... 33
Taula 9 Llista de les etiquetes necessàries per la creació d'un Settings..... 34
Taula 10 Llista de les etiquetes necessàries per la creació de Sub-settings.... 36

1. Introducció

1.1 Context i justificació del Treball

Al CERN (Centre Europeu per la Recerca Nuclear), físics i enginyers de tot el món treballen junts per estudiar l'estructura fonamental de l'univers. Per fer-ho, fan servir un complex d'acceleradors de partícules, on els protons van guanyant energia fins arribar al més gran de tots, el *Large Hadron Collider* (LHC). Al LHC els protons xoquen entre sí a una velocitat gairebé igual a la velocitat de la llum, i les noves partícules "creades" durant les col·lisions són detectades i posteriorment analitzades pels 4 experiments que avui en dia treballen al CERN: ATLAS, ALICE, CMS i LHCb. Dins del CERN, el grup de "Instrumentació del Feix" (*Beam Instrumentation*), és responsable del disseny, construcció i manteniment dels instruments que permeten l'observació dels feixos de partícules i la mesura de paràmetres relacionats amb ells. Dins d'aquest grup, la secció de Software desenvolupa, construeix i manté software per obtenir informació a temps real per aquests instruments, que fan servir els operadors dels acceleradors i els especialistes de hardware.

Per tal de controlar i adquirir dades dels instruments que monitoritzen i controlen els feixos de protons, al CERN s'ha desenvolupat una arquitectura de programari anomenada FESA ("Arquitectura de Programari de Front-End"). FESA és un entorn complet per especialistes d'instruments, perquè puguin dissenyar, desenvolupar, provar e implementar programari de control en temps real, escrit amb C++. Per tant, és possible connectar-se a un instrument o dispositiu seleccionat a través del middleware de comunicació CERN i escoltar qualsevol notificació de dades desencadenada per FESA.

L'objectiu del projecte és desenvolupar un lloc web adaptatiu, accessible des de la *General Purpose Network* (GPN) i des de xarxa protegida o *Technical Network* (TN) del CERN per obtenir dades en temps real de diferents servidors FESA, permetent als usuaris accedir a tota aquesta informació ràpidament i des de qualsevol lloc del CERN, ja sigui des de un dispositiu mòbil o un ordinador personal, sense necessitat d'instal·lar cap aplicació. Fins ara, l'accés a aquestes dades només era possible a través d'aplicacions realitzades amb Java, C++ o Python. L'accés a la pàgina web és obert a tots els usuari que estiguin connectats a alguna de les xarxes esmentades anteriorment, sense qui hi hagi cap tipus d'identificació per part de l'usuari.

La pàgina disposarà d'una aplicació que permetrà als usuaris crear subscripcions complexes amb gran quantitat d'opcions de configuració. Aquestes subscripcions creades pels usuaris es coneixen amb el nom de Settings. L'accés a aquesta aplicació es realitzarà des de el menú de la pàgina web, però per motius de seguretat per poder obrir l'aplicació caldrà que l'usuari estigui connectat a la TN, on prèviament s'ha

identificat amb el seu usuari del CERN i serà aquest usuari el que tindrà en compte l'aplicació a l'hora d'administrar els Settings de l'usuari.

1.2 Objectius del Treball

L'objectiu principal del treball és crear un lloc web adaptatiu per obtenir dades en temps reals dels servidors FESA del CERN. El lloc web creat haurà de complir les especificacions següents:

- El servidor ha de ser capaç de crear un nombre elevat de subscripcions a servidors FESA i emmagatzemar els últims valors rebuts de cada un d'aquests.
- Cal establir una comunicació en temps real entre totes les sessions obertes pels clients i el servidor perquè aquestes puguin sol·licitar les subscripcions amb els servidors FESA, rebre informació sobre la configuració de les dades que es rebran i finalment que cada sessió rebi les dades que li corresponen quan aquestes siguin actualitzades.
- El navegador ha de ser capaç de processar un nombre elevat de dades i actualitzar el contingut de la plana amb la representació oportuna d'aquestes dades amb una freqüència màxima de 10Hz.
- La plana web ha de funcionar i visualitzar-se correctament des de qualsevol smartphone, tablet o ordinador personal mitjanament modern.
- Els usuaris han de disposar d'un formulari per poder realitzar de manera ràpida subscripcions senzilles als diferents servidors FESA. També han de disposar d'algun sistema per poder crear i modificar subscripcions més complexes, de manera que, un cop creades aquestes subscripcions puguin ser executades i accessibles per la resta d'usuaris sense grans complicacions.

1.3 Enfocament i mètode seguit

S'ha escollit realitzar una pàgina web per permetre que qualsevol usuari pugui accedir a la informació de l'instrument que necessiti d'una manera molt senzilla i usable, sense necessitat d'instal·lar cap software i pel fet que es pugui utilitzar en pràcticament qualsevol dispositiu mòbil que disposi de wifi. La intenció principal d'aquesta pàgina és facilitar la feina als operadors del LHC de manera que puguin veure des dels seus dispositius mòbils, en temps real, els efectes que tenen les modificacions que estiguin realitzant sobre el dispositiu en el què estiguin treballant o configurant. Això facilitarà molt la feina de molts usuaris ja que fins ara era necessari realitzar aquestes consultes des d'un ordinador de sobre taula o desplaçar-se amb un ordinador portàtil fins a on es troba el dispositiu per poder visualitzar els efectes, necessitant la prèvia instal·lació d'un software específic creat per a tal fi i que de vegades resultava massa complex d'utilitzar.

Per la part del servidor s'ha decidit utilitzar un servidor Java per poder utilitzar les llibreries existents per d'accedir a les bases de dades on es troba tota la informació dels diferents servidors FESA existents i que s'encarreguen de realitzar la comunicació amb aquests.

1.4 Planificació del Treball

Per a la creació del projecte s'utilitzarà Java 8 juntament amb Maven per la part del servidor. La part del client s'implementarà amb HTML5, CSS3 i Bootstrap 4. La programació del costat client es farà amb JavaScript en la versió ECMAScript 6, Jquery 3.3.1 i múltiples llibreries de Jquery i JavaScript amb diverses funcionalitats com la creació de gràfiques, taules dinàmiques, etc. Per a la comunicació en temps real entre el client i el servidor s'utilitzaran WebSockets.

Com a software per la creació del codi de la plana web s'utilitzarà el IDE Eclipse for Java EE Developers i el servidor on s'executarà aquesta plana serà un servidor Apache Tomcat en la seva versió 9.0.12.

La planificació seria la següent:

Fites	Inici	Fi	Dies
PAC1			**12**
Definició del projecte i objectius	9/07/18	17/07/18	7
Redacció primer document	18/07/18	27/07/18	8
PAC2			**59**
Aprenentatge funcionament FESA	31/07/18	17/08/18	14
Estructura de continguts i disseny	20/08/18	21/08/18	2
Disseny Wireframes	22/08/18	22/08/18	1
Test comunicació Websocket + Tomcat + FESA	23/08/18	28/08/18	4
Prototip funcional bàsic sense format	29/08/18	25/09/18	20
Adaptació HTML+CSS	26/09/18	5/10/18	8
Creació vistes per tipus de dades	8/10/18	19/10/18	10
PAC3			**35**
Subscripció per Setting	22/10/18	8/11/18	14
Agrupació múltiples dispositius	9/11/18	19/11/18	7
Proves, correccions i millores	20/11/18	7/12/18	14
Lliurament final			**28**
Finalització del projecte	10/12/18	31/12/18	16
Elaboració del vídeo	1/01/19	16/01/19	12

Taula 1 Fites de la planificació

Per accelerar el desenvolupament i garantir la màxima compatibilitat amb la majoria de dispositius mòbils i navegadors s'ha buscat una plantilla que s'ajustés a les necessitats del projecte i fos fàcil d'adaptar i modificar. Aquesta plantilla utilitzada es diu Metronic i s'ha comprat a la pàgina https://themeforest.net/item/metronic-responsive-admin-dashboard-template/4021469

1.5 Breu sumari de productes obtinguts

Finalment s'ha obtingut una pàgina web moderna, senzilla d'utilitzar i amb un disseny adaptatiu per garantir la correcta visualització des de qualsevol dispositiu amb el que s'accedeixi a la pàgina. Aquesta pàgina web permet realitzar de manera ràpida subscripcions per adquirir dades dels instruments que monitoritzen i controlen els feixos de protons dels acceleradors del CERN.

També s'ha configurat un servidor web Tomcat que s'encarrega de respondre les sol·licituds d'accés a la pagina web i d'administrar totes les subscripcions sol·licitades via WebSocket, ja siguin per la pròpia pàgina web o per altres aplicacions.

1.6 Breu descripció dels altres capítols de la memòria

En el capítol 2 es mostren els diagrames UML on es representen els elements més importants de l'aplicació amb una breu descripció d'aquests.

El capítol 3 parla de tots els elements relacionats amb la usabilitat, com poden ser els perfils dels usuaris, els casos d'us, i els diagrames de baixa i d'alta fidelitat dels diferents elements que intervenen a la pàgina web i de l'administrador de Settings.

El capítol 4 explica con funciona l'arquitectura del front-end i del back-end, així com els elements que s'utilitzen per fer funcionar Fesa Web, una descripció de les bases de dades i de les Apis/Llibreries utilitzades.

En el capítol 5 hi ha algunes parts del codi i una petita descripció d'aquests, hi ha codis que formen part tant del back-end i d'altres del front-end que s'ha considerat que són més rellevants, ja sigui per la seva complexitat o per la seva importància dins de l'aplicació. Dins d'aquest capítol també es parla d'alguns dels tests que s'han realitzat sobre la pàgina web i en el servidor. També hi ha una breu descripció de les versions en les que s'han realitzat modificacions importants en l'estructura del codi.

El capítol 6 consisteix en una conclusió del projecte, on es parla dels beneficis que m'ha aportat personalment aquest projecte, si s'han complert els objectius definits inicialment i sobre futures ampliacions que en el moment de l'entrega han quedat pendents de desenvolupar.

El capítol 7 és bàsicament la bibliografia de totes les pàgines web que s'han anat consultant al llarg del desenvolupament de Fesa Web.

2. Diagrames UML

2.1 Front-end

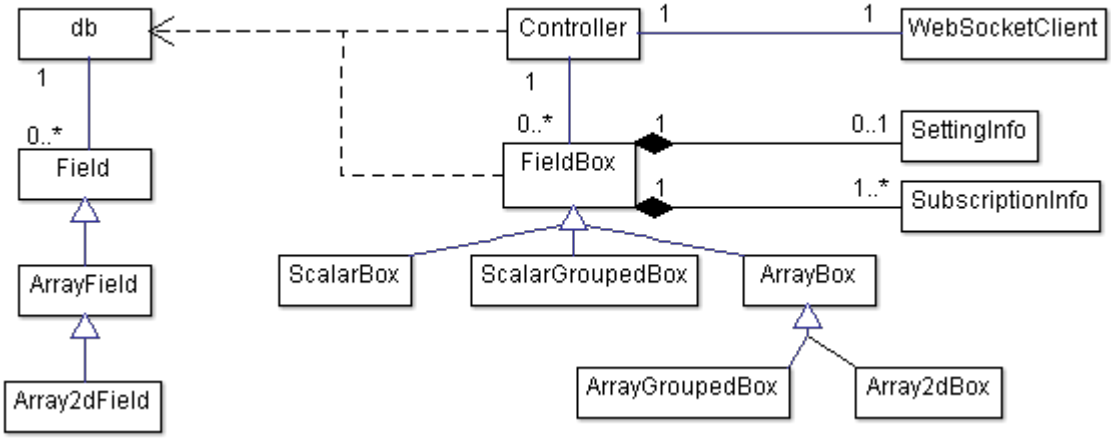

Figura 1 Diagrama UML del front-end

WebSocketClient: S'encarrega d'establir i gestionar la connexió permanent amb el servidor mitjançant WebSockets. També s'encarrega de gestionar la comunicació bidireccional amb el servidor, de manera que cada vegada que rep un missatge del servidor s'encarrega d'informar a *Controller* per que pugui processar la informació del missatge depenent del tipus de missatge.

Controller: És l'element principal de la pàgina web i gestiona pràcticament tot el que passa en aquesta. Quan un usuari realitza una sol·licitud de subscripció o realitza una acció que modifica l'estat actual d'alguna subscripció, ho comunica al servidor enviant-li un missatge mitjançant la classe *WebSocketClient*. També gestiona el contingut de *db* i dels objectes de tipus *FieldBox* que s'han creat al realitzar les subscripcions, així com, els esdeveniment que succeeixen sobre aquests.

Db: Actua com una base de dades local gestionant la informació que es va rebent de cada camp que s'ha generat amb les subscripcions i eliminant-la quan un camp és eliminat.

Field: És la definició del tipus de dada més bàsica que pot emmagatzemar la base de dades, equivaldria a un sol valor numèric o una cadena de text. Aquesta classe defineix un espai per guardar el nom del camp, l'últim valor rebut, el seu tipus, el nom del dispositiu, propietat, cicle al que fa referencia i d'un històric per emmagatzemar els últims valors que ha rebut, en aquest cas són els 120 últims valors.

ArrayField: És una especialització de la classe *Field,* pensada per emmagatzemar valors de tipus vector. En aquest cas es modifica l'extensió de l'històric a 60, s'implementa un mètode per reiniciar l'històric quan és rep un vector amb una longitud diferent a la que s'havia rebut fins el moment i implementa un mètode per filtrar les dades que

conté, permetent obtenir l'últim vector rebut o consultar el contingut d'una posició especifica d'aquest (columna) o fer el mateix amb l'històric, retornant un vector amb tot l'històric sencer o simplement la posició especifica de tots els vectors que conté.

Array2dField: És una especialització de la classe *ArrayField* i serveix per emmagatzemar vectors de dues dimensions. El seu funcionament és exactament igual que el de la seva classe para, tret del mètode per filtrar les dades, que addicionalment permet filtrar indicant una columna o una fila, retornant un vector d'una dimensió, o especificant una columna i una fila, retornant un sol valor. Aquest filtre també es pot aplicar al històric, retornant un vector que conté en cada posició un vector d'una dimensió en el cas que s'especifiqui una columna o una fila, o que en cada posició només conté un sol valor si s'especifica una fila i columna. En el cas que no s'especifiqui cap columna ni fila, el que és retorna són els vectors de dues dimensions originals.

FieldBox: Definició genèrica, dels blocs que es generen per mostrar la informació dels camps. Defineix l'estructura HTML dels blocs, les opcions de menú, de visualització, mètodes per controlar l'estat del bloc, per si es pot actualitzar el seu contingut o no, en el cas que estigui pausat o ocupat executant algun procés, mètodes per donar format a l'estructura de les dades depenent de com aquests s'han de mostrar per pantalla i mètodes per imprimir aquestes dades en els blocs com un sol valor, com una gràfica, com una taula, com una gràfica, o per exportar-les a un fitxer CSV.

SettingInfo: Quan un bloc o Box fa referencia a un camp que s'ha creat a partir d'un Setting, conté una instància d'aquesta classe amb la informació referent al Setting.

SubscriptionInfo: Tots els blocs o Box, tenen una instància d'aquesta classe amb la informació de la subscripció, com pot ser: identificador, nom del dispositiu, nom de la propietat, nom del camp, tipus de camp i el nom del cicle.

ScalarBox: Especialització de la classe *FieldBox* per generar blocs referents a valors escalars o de cadena de text. Aquesta implementació evita la vista de gràfica en cas de que el tipus del valor sigui una cadena de text.

ScalarGroupedBox: Especialització de la classe *FieldBox.* Serveix per mostrar el valor d'un camp específic de múltiples dispositius a l'hora.

ArrayBox: Especialització de la classe *FieldBox* per generar blocs referents a valors vectorials d'una sola dimensió. Activa les vistes del menú necessàries per aquest tipus de dades, exceptuant que provingui d'una subscripció per Setting i s'hagi especificat que només es vol consultar una columna concreta del vector, en aquest cas les opcions de visualització i de funcionament seran les per defecte de l'objecte pare.

També personalitza els mètodes per exportar a les dades a un fitxer CSV, per donar-lis format i n'inclou un de nou, dona format per imatge, juntament amb el seu mètode per imprimir la imatge.

ArrayGroupedBox: Especialització de la classe *ArrayBox*. La intenció d'aquesta classe és la mateixa que la de *ScalarGroupedBox* però per camps de tipus vector. També personalitza els mètodes per dona format a les dades, d'impressió i exportació de les dades en CSV.

Array2dBox: Especialització de la classe *ArrayBox* per generar blocs referents a valors vectorials de dues dimensions. Activa les vistes del menú necessàries per aquest tipus de dades, exceptuant que provingui d'una subscripció per Setting i s'hagi especificat que només es vol consultar una columna o fila concreta del vector. En aquest cas les opcions de visualització i de funcionament seran les mateixes que les d'un objecte de tipus ArrayBox. En el cas que s'hagi especificat una fila i una columna les opcions de visualització seran les de la classe FieldBox. També personalitza els mètodes per dona format a les dades i exportació de les dades en CSV.

2.2 Back-end

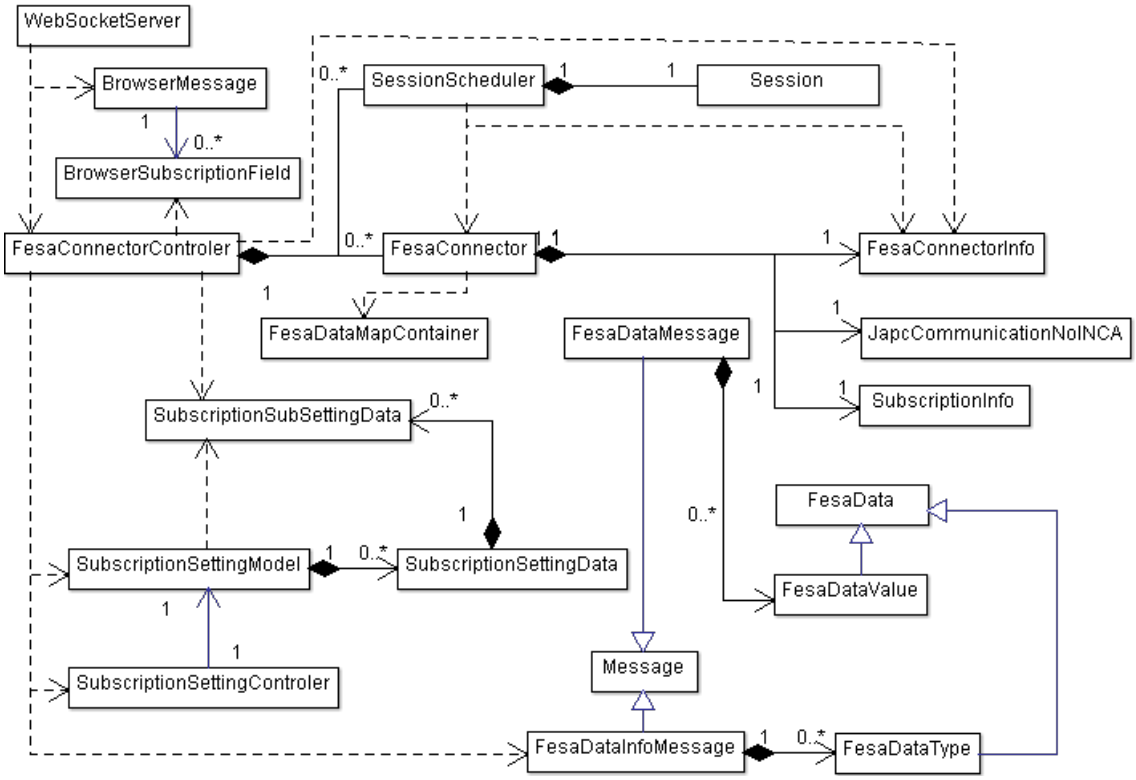

Figura 2 Diagrama UML del Back-end

WebSocketServer: Actua com a Web Socket Server i és la classe mitjançant la qual s'estableix una connexió permanent entre el client i el servidor. Aquesta classe és la que rep les notificacions de quan s'obra i és tanca una connexió amb un client i quan aquest envia un missatge. Segons el tipus de missatge crida al mètode corresponent al tipus de

missatge de la classe *FesaConnectorControler.* Els mètodes d'aquesta classe reben com a paràmetre una instància de la classe Session que fa referencia al client connectat.

Session: Aquesta classe forma part del paquet *javax.websocket* i representa una conversació entre dos web socket endpoints. Les instancies d'aquesta classe són les que ens permeten principalment saber l'estat d'una comunicació i enviar missatges al client, entre altres coses.

BrowserMessage: Quan *WebSocketServer* rep un missatge d'algun dels clients, el que rep és una cadena de text, amb format JSON i el que fa és transformar aquesta cadena de text en un objecte vàlid de Java i aquests objectes són de tipus *BrowserMessage* i contenen les propietats necessàries per emmagatzemar la informació dels diferents tipus de missatge que podem rebre.

BrowserSubscriptionField: *BrowserMessage* conté una llista d'instàncies d'aquesta classe quan el missatge que rep del navegador es per realitzar una reconnexió després d'haver perdut la comunicació via WebSocket amb el servidor i enviar, un cop s'ha restablert la comunicació, tota la informació necessària sobre les subscripcions que tenia actives, abans de perdre la comunicació, perquè el servidor pugui tornar a generar aquestes subscripcions i el client tornar a rebre les dades.

Gson: Forma part del paquet *com.google.gson.Gson* i és utilitzada per deserializar un JSON a un objecte Java o per serializar un objecte Java a JSON.

FesaConnectorControler: Classe estàtica que gestiona les sessions i les subscripcions existents.

FesaConnector: Cada instància d'aquesta classe s'encarrega de realitzar la subscripció amb el dispositiu indicat mitjançant *JapcCommunicationNoInca*. FesaConnector disposa principalment de mètodes per crear, eliminar i rebre les dades de la subscripció. Cada vegada que rep noves dades aquestes són enviades a *FesaDataMapContainer.*

JapcCommunicationNoINCA: Llibreria creada pel CERN per poder realitzar la comunicació amb un dispositiu específic i que existeixi en un servidor FESA3 (versió actual de FESA).

SubscriptionInfo: Classe creada pel CERN. Un cop tenim la comunicació establerta amb el dispositiu mitjançant *JapcCommunicationNoINCA,* l'utilitzem per crear la subscripció amb el dispositiu, indicant-li el cicle i propietat que volem consultar, per poder començar a rebre informació d'aquest. Al crear la subscripció ens

retorna un objecte de tipus *SubscriptionInfo* que serà necessari per eliminar la subscripció i deixar de rebre dades.

FesaConnectorInfo: Conté la informació general de la subscripció realitzada, així com el nom de tots els camps, el tipus i el tamany i dimensions dels camps de tipus vector.

SessionScheduler: La primera vegada que un client sol·licita una subscripció es crea una instància d'aquesta classe que contindrà la instància de *Session,* que fa referencia a aquest client, per poder realitzar una comunicació directa, i el llistat de tots els camps de les subscripcions que té actives per enviar-li les dades d'aquests cada vegada que disposin de nous valors.

FesaDataMapContainer: Classe estàtica que actua com una base de dades guardant les dades rebudes per totes les instàncies de tipus *FesaConnector* existents, juntament amb un timestamp de la última vegada que ha estat actualitzat cada camp. També és utilitzada per les instancies de tipus *SessionScheduler* a mode de consulta per saber si algun dels camps que necessiten ha estat actualitzat, en cas afirmatiu, sol·liciten el valor de tots els camps han estat actualitzats i els envien al client corresponent.

Message: Defineix l'estructura bàsica dels missatges que s'enviaran via WebSocket entre el servidor i el client.

FesaDataInfoMessage: Especialització de la classe *Message,* s'utilitza per enviar informació relativa a la subscripció sol·licitada i dels camps pertinents mitjançant una llista d'objectes del tipus *FesaDataType*.

FesaDataMessage: Especialització de la classe *Message,* s'utilitza per enviar les dades de les subscripcions actives dels clients mitjançant una llista de tipus *FesaDataValue*.

FesaData: Defineix l'estructura d'informació bàsica sobre un camp d'una subscripció.

FesaDataType: Especialització de la classe *FesaData* utilitzada per contenir informació referent a un camp en concret d'una subscripció.

FesaDataValue: Especialització de la classe *FesaData* utilitzada per emmagatzemar el valor d'un camp en concret d'una subscripció.

SubscriptionSettingController: Gestiona els Settings existents en un fitxer json. Actualment només s'utilitza per accedir al contingut d'aquest fitxer, no per modificar-lo.

SubscriptionSettingModel: *SubscriptionSettingController* crea una instància d'aquesta classe a partir del fitxer json que conté tots els Settings creats pels usuaris i actua com una base de dades de Settings,

permetent-nos per exemple accedir a Settings concrets mitjançant el seu identificador, obtenir els Settings que formen part d'una secció o obtenir totes les seccions existents. Dins d'aquest objecte els Settings es transformen en instancies de *SubscriptionSettingData* i s'agrupen totes les que formen part d'una mateixa secció dins d'una llista en objectes del tipus *SubscriptionSubSettingData*.

SubscriptionSubSettingData: Cada instància d'aquesta classe es correspon a un Setting pròpiament dit i esta format per tota la informació de configuració d'aquest Setting.

SubscriptionSettingData: Cada instància d'aquesta classe equival a una *secció* creada per l'usuari des de l'aplicació per gestionar Settings i esta formada pel nom de la secció i el llistat d'instàncies de tipus *SubscriptionSubSettingData* que formen part d'aquesta secció.

3. Usabilitat/UX (DCU)

3.1 Perfils d'usuari

Actualment qualsevol usuari que accedeixi a Fesa Web, té accés a la mateixa informació i funcionalitats que la resta, ja que no existeix cap procés d'identificació.

En el únic punt on sí que es realitza una diferenciació entre usuaris és a l'aplicació per crear subscripcions personalitzades. Per poder accedir a aquesta aplicació és obligatori que l'usuari estigui accedint a Fesa Web des de la Technical Network (TN), xarxa amb accés limitat a internet i que requereix que l'usuari s'hagi identificat amb les seves credencials del CERN, de manera que, al accedir a aquesta aplicació, el primer que farà aquesta serà mirar si el nom de l'usuari identificat està reconegut com administrador de l'aplicació, en cas afirmatiu, tindrà autorització per accedir, modificar o eliminar qualsevol subscripció existent o crear noves subscripcions en nom de qualsevol usuari. En el cas de que el nom de l'usuari no estigui reconegut com administrador, aquest usuari podrà realitzar totes les accions comentades anteriorment però només a les subscripcions creades prèviament per ell i totes les subscripcions que creï estaran vinculades al seu usuari.

3.2 Diagrama dels casos d'ús

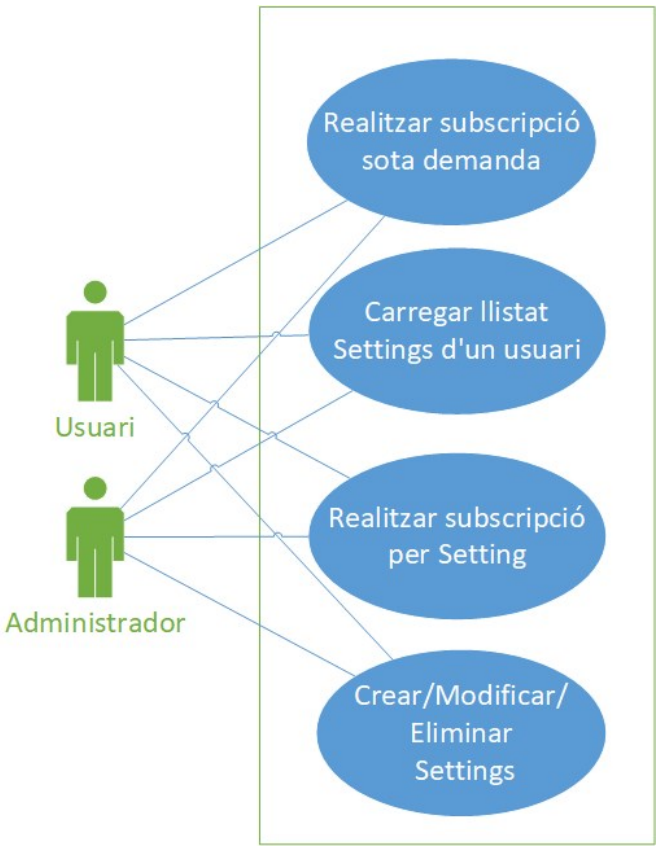

Figura 3 Diagrama dels casos d'ús

3.3 Especificació dels casos d'ús

Realitzar subscripció sota demanda

Actors: Qualsevol usuari que accedeixi al sistema
Descripció: L'usuari accedeix a la pàgina web i mitjançant un formulari realitza una subscripció sota demanda (subscripció senzilla) a un dispositiu específic.
Condició prèvia: El dispositiu des del que accedeix l'usuari ha d'estar connectat a la TN o a la GPN del CERN.
Curs de l'esdeveniment:

Actor	Sistema
1. L'usuari accedeix a Fesa Web	
	2. Es genera una connexió permanent entre la pestanya del navegador i el servidor via WebSocket i es crea un objecte que contindrà tota la informació d'aquesta sessió creada pel navegador.
	3. El sistema analitza la URL per veure si hi ha una referencia a una subscripció per Setting. En cas afirmatiu realitza de manera automàtica el pas 24 i posteriors, enviant-li, via WebSocket, tota la informació relativa a la subscripció que s'ha trobat a la URL.
4. L'usuari clica sobre la icona amb una fulla i un llapis per visualitzar el formulari "On-demand Subscription".	
	5. El sistema mostra el formulari per introduir les dades de la subscripció sota demanda.
6. L'usuari comença a introduir una part del nom del dispositiu en el camp del formulari "Device".	
	7. El servidor rep via AJAX la cadena de text introduïda i envia el llistat de dispositius que el seu nom conté la cadena de text rebuda.
	8. El sistema mostra sota el camp "Device" el llistat de dispositius enviat pel servidor.
9. L'usuari selecciona un dels dispositius del llistat.	
	10. El servidor rep via AJAX el nom del dispositiu seleccionat i retorna el llistat de propietats d'aquets dispositiu
	11. El sistema rep el llistat de propietats i les carrega en el camp "*Property*".
12. L'usuari clica sobre el camp "*Property*" i	

se li mostra el llistat de propietats del dispositiu.	
	13. El sistema mostra el llistat de propietats relacionades amb el dispositiu seleccionat anteriorment,
14. L'usuari pot introduir part del nom de la propietat. Si no introdueix part del nom passem al punt 16.	
	15. El sistema filtrarà el llistat mostrant només les propietats que contenen la cadena de text introduïda.
16. L'usuari selecciona la propietat.	
	17. El servidor rep via AJAX el nom del dispositiu i de la propietat seleccionades i retorna el llistat de camps que formen part d'aquesta propietat del dispositiu.
18. Si l'usuari ho desitja pot clicar sobre el camp "*Fields*".	
	19. En el cas que l'usuari cliqui sobre el camp "*Fields*", els sistema mostrarà el llistat de camps que pertanyen a la propietat seleccionada.
20. L'usuari selecciona un, cap o múltiples camps del llistat.	
21. Si l'usuari ho desitja introdueix el nom d'un cicle en el camps "*Cycle*".	
22. L'usuari clica sobre el botó "Send subscription" per sol·licitar la subscripció pertinent amb les dades introduïdes.	
	23. El sistema modifica la URL del navegador per indicar que s'ha realitzat una subscripció sota demanda indicant el nom del dispositiu, la propietats, els camps seleccionats i en el cas que s'hagi introduït, el nom del cicle.
	24. El servidor rep, via WebSocket, les dades per realitzar la subscripció i comprova si alguna altre sessió ja ha creat una subscripció per aquest dispositiu i propietat, en cas contrari, crea aquesta subscripció i guarda dins del objecte de la sessió la informació de la subscripció que ha sol·licitat.
	25. Un cop el servidor disposa de la informació sobre els tipus de camps d'aquesta propietat, s'envia via WebSocket tota la informació relativa del dispositiu, de la propietat i dels camps seleccionats per l'usuari. En el cas que l'usuari no hagi especificat cap camp, s'enviarà la informació de tots els camps de la propietat.

	26. El navegador rep aquesta informació i mira si l'usuari ha seleccionat la opció de conservar subscripcions anteriors. En cas afirmatiu realitza al punt 27, en cas contrari elimina els blocs dels camps existents i totes les referencies de les subscripcions que s'havien realitzat prèviament. Acte seguit envia la notificació al servidor per cancel·lar la recepció de dades d'aquestes subscripcions.
	27. El navegador crea tots els objectes necessaris per gestionar aquesta subscripció, així com, un contenidor per a cada un dels camps que s'hagin rebut dins del cos de la pàgina web i envia, via WebSocket, una confirmació al servidor per comunicar-li que ja esta preparat per rebre dades de la subscripció.
	28. Un cop el servidor rep aquesta confirmació, crea un temporitzador a l'objecte de la sessió de l'usuari, amb una freqüència de 10Hz. Aquest temporitzador consisteix en un Thread que s'encarregarà de mirar si existeixen dades noves per les diferents subscripcions sol·licitades per aquesta sessió i en cas afirmatiu envia, via WebSocket, totes les dades noves.
	29. El sistema, processa les dades rebudes i actualitza el contingut de tots els contenidors relacionats amb la nova informació

Taula 2 Esdeveniment: "Realitzar un subscripció sota demanda"

Cas alternatiu:

2a. No es pot crear la connexió via WebSocket amb el servidor, ja sigui perquè el servidor no esta disponible o l'usuari no disposa d'accés a internet. El sistema informa amb un missatge que no s'ha pogut establir la connexió i va intentant crear la connexió cada 5 segons o quan es recupera la connexió a internet. Un cop es recupera la connexió i existeix una o més subscripcions en el navegador, que abans de perdre la connexió estaven activa, es torna a sol·licitar al servidor que realitzi aquesta subscripció per tornar a rebre les dades d'aquestes.

6a, 14a. El sistema no troba cap coincidència amb el text introduït per l'usuari. En aquest cas s'informa a l'usuari que no s'ha trobat cap coincidència.

24a. Es produeix un error al intentar realitzar la subscripció al dispositiu. S'informa a l'usuari de que ha succeït un problema.

28a. L'usuari tanca la pestanya del navegador o perd la connexió a internet. El servidor elimina tota la informació relativa a aquesta sessió. Quan es detecta que ha passat una hora sense que hi hagi cap sessió interessada en una subscripció, aquesta és cancel·lada i eliminada.

Carregar llistat Settings d'un usuari específic

Actors: Qualsevol usuari que accedeixi al sistema
Descripció: L'usuari accedeix a la pàgina web i mitjançant un desplegable selecciona un usuari per carregar el llistat de Settings creats per aquest usuari.
Condició prèvia: El dispositiu des del que accedeix l'usuari ha d'estar connectat a la TN o a la GPN del CERN i ja ha carregat la pàgina web en el navegador.
Curs de l'esdeveniment:

Actor	Sistema
1. L'usuari clica sobre la icona d'usuari a la barra de menú que hi ha a la capçalera.	
	2. El sistema mostra el formulari per seleccionar un usuari.
3. L'usuari clica sobre el desplegable i selecciona un usuari. Acte seguit clica el botó enviar.	
	4. El servidor rep via AJAX l'usuari seleccionat i busca els Settings creats per aquest.
	5. El servidor envia, via AJAX, la informació dels Settings trobats.
	6. El navegador substitueix el contingut del desplegable, que conté el llistat de Settings, per la informació que ha rebut.
	7. S'informa a l'usuari que els Settings de l'usuari seleccionat s'han carregat correctament.

Taula 3 Esdeveniment: "Carregar llistat de Settings d'un usuari específic"

Realitzar subscripció per Setting

Actors: Qualsevol usuari que accedeixi al sistema
Descripció: L'usuari accedeix a la pàgina web i mitjançant un desplegable selecciona un Setting existent i realitza la subscripció.
Condició prèvia: El dispositiu des del que accedeix l'usuari ha d'estar connectat a la TN o a la GPN del CERN.
Curs de l'esdeveniment:

Actor	Sistema
1. L'usuari accedeix a Fesa Web	
	2. Es genera una connexió permanent entre la pestanya del navegador i el servidor via WebSocket i es crea un objecte que contindrà tota la informació d'aquesta sessió creada pel navegador.
	3. El sistema analitza la URL per veure si

	hi ha una referencia a una subscripció per Setting. En cas afirmatiu realitza de manera automàtica el pas 10 i posteriors, enviant-li, via WebSocket, l'identificador que s'ha trobat a la URL.
4. L'usuari clica sobre el desplegable que hi ha a la capçalera de la pàgina.	
	5. El sistema mostra el llistat de Settings per l'usuari seleccionat prèviament. Si és la primera vegada que s'accedeix el llistat conté els Settings de l'usuari Public.
6. L'usuari pot introduir part del nom del Setting en el que esta interessat, en cas de no fer-ho saltar al punt 8	
	7. El sistema filtra el llistat mostrant només els Settings que contenen la cadena de text introduïda.
8. L'usuari selecciona un Setting dels dispositius del llistat i clica sobre el botó per enviar la sol·licitud de subscripció.	
	9. El sistema modifica la URL del navegador per indicar que s'ha realitzat una subscripció per Setting i especificant l'identificador del Setting seleccionat.
	10. El servidor rep via WebSocket el identificador del Setting, busca si aquest Setting existeix i en carrega la informació d'aquesta.
	11. El servidor analitza aquesta informació per crear les subscripcions que siguin necessàries, si aquestes no han estat creades prèviament per alguna altre sessió. Acte seguit guarda dins del objecte de la sessió la informació de la subscripció o subscripcions a les que feia referencia el Setting.
	12. Un cop el servidor disposa de la informació sobre els tipus de camps de la subscripció o subscripcions sol·licitades, s'envia, via WebSocket, tota la informació necessària dels dispositius, de las propietats i dels camps.
	13. El navegador rep aquesta informació i mira si l'usuari ha seleccionat la opció de conservar subscripcions anteriors. En cas afirmatiu realitza al punt 12 directament. En cas contrari primer elimina els blocs dels camps existents i totes les referencies de les subscripcions que s'havien realitzat prèviament. Acte seguit envia la notificació al servidor per cancel·lar la recepció de dades d'aquestes subscripcions.
	14. El navegador crea tots els objectes

	necessaris per gestionar aquesta subscripció o subscripcions, així com, un contenidor per a cada un dels camps que s'hagin rebut dins del cos de la pàgina web i envia, via WebSocket, una confirmació al servidor per comunicar-li que ja esta preparat per rebre les dades.
	15. Un cop el servidor rep aquesta confirmació, crea o modifica un temporitzador a l'objecte de la sessió de l'usuari, amb la freqüència definida en el Setting. Aquest temporitzador consisteix en un Thread que s'encarregarà de mirar si existeixen dades noves per les diferents subscripcions sol·licitades per aquesta sessió i en cas afirmatiu envia, via WebSocket, totes les dades noves.
	16. El sistema, processa les dades rebudes i actualitza el contingut de tots els contenidors relacionats amb la nova informació.

Taula 4 Esdeveniment: "Realitzar un subscripció per Setting"

Cas alternatiu:
2a. No es pot crear la connexió via WebSocket amb el servidor, ja sigui perquè el servidor no esta disponible o l'usuari no disposa d'accés a internet. El sistema informa amb un missatge que no s'ha pogut establir la connexió i va intentant crear la connexió cada 5 segons o quan es recupera la connexió a internet. Un cop es recupera la connexió i existeix una o més subscripcions en el navegador, que abans de perdre la connexió estaven activa, es torna a sol·licitar al servidor que realitzi aquesta subscripció per tornar a rebre les dades d'aquestes.
6a. El sistema no troba cap coincidència amb el text introduït per l'usuari. En aquest cas s'informa a l'usuari que no s'ha trobat cap coincidència.
8a. El sistema no troba cap Setting amb l'identificador rebut. S'informa a l'usuari de que no existeix cap Setting amb aquest identificador.
11a. Es produeix un error al intentar realitzar la subscripció o subscripcions. S'informa a l'usuari de que ha succeït un problema.
15a. L'usuari tanca la pestanya del navegador o perd la connexió a internet. El servidor elimina tota la informació relativa a aquesta sessió. Quan es detecta que ha passat una hora sense que hi hagi cap sessió interessada en una subscripció, aquesta és cancel·lada i eliminada.

<u>Creació d'un Setting</u>

Actors: Qualsevol usuari que accedeixi al sistema
Descripció: L'usuari accedeix a l'aplicació des de la pàgina web per crear un nou Setting.
Condició prèvia: El dispositiu des del que accedeix l'usuari ha d'estar connectat a la TN del CERN.
Curs de l'esdeveniment:

Actor	Sistema
1. L'usuari accedeix a Fesa Web i clica sobre la icona d'engranatge.	
	2. El sistema mostra l'apartat "*Settings*".
3. L'usuari clica sobre l'opció "*Setting configuration tool*" per accedir a l'aplicació.	
	3. El sistema mira si l'usuari es reconegut com administrador, si és així carregarà en un desplegable el llistat d'usuaris que han creat algun Setting i per defecte carregarà les seccions i els Settings de l'usuari *Public.* Sí l'usuari no és reconegut com administrador, en el desplegable d'usuaris només apareixerà el seu usuari i es carregaran les seccions i els Settings que ell hagi creat.
4. L'usuari ha de seleccionar una secció creada prèviament per ell, de manera que el Setting que creï formarà part d'aquesta secció.	
5. Acte seguit introduirà un nom pel Setting i clicarà sobre el botó "New".	
	6. El sistema crearà el Setting.
7. L'usuari seleccionarà del llistat de Settings el Setting que acaba de crear.	
	8. El sistema carrega el en formulari de configuració de Settings la informació referent a aquest Setting.
9. L'usuari omple i configura les opcions necessàries pel Setting que vol crear.	
10. L'usuari clica sobre el boto "*Save*".	
	11. El sistema guarda les modificacions del Setting.

Taula 5 Esdeveniment "Crear un Setting"

Cas alternatiu:
3a. Si l'usuari no esta accedint des de la TN, rebrà un error al intentar obrir l'aplicació.
4a. Si no existeix cap secció o la secció que interessa a l'usuari, primer tindrà que crear.la. Sota el llistat de seccions hi ha un camp de text per introduir el nom d'una secció i al costat el botó "New", al clicar sobre aquest botó es crearà la secció i apareixerà en el llistat de seccions.

Editar d'un Setting

Actors: Qualsevol usuari que accedeixi al sistema
Descripció: L'usuari accedeix a l'aplicació des de la pàgina web per editar la configuració d'un Setting existent.
Condició prèvia: El dispositiu des del que accedeix l'usuari ha d'estar connectat a la TN del CERN.

Curs de l'esdeveniment:

Actor	Sistema
1. L'usuari accedeix a Fesa Web i clica sobre la icona d'engranatge.	
	2. El sistema mostra l'apartat "*Settings*".
3. L'usuari clica sobre l'opció "*Setting configuration tool*" per accedir a l'aplicació.	
	3. El sistema mira si l'usuari es reconegut com administrador, si és així carregarà en un desplegable el llistat d'usuaris que han creat algun Setting i per defecte carregarà les seccions i els Settings de l'usuari *Public.* Sí l'usuari no és reconegut com administrador, en el desplegable d'usuaris només apareixerà el seu usuari i es carregaran les seccions i els Settings que ell hagi creat.
4. L'usuari ha de seleccionar una de les seccions existents.	
	5. El sistema carregarà el llistat de tots els Settings que hi ha dins d'aquesta secció.
6. L'usuari seleccionarà del llistat de Settings el Setting que vol modificar.	
	7. El sistema carrega el en formulari de configuració de Settings la informació referent al Setting seleccionat.
8. L'usuari modifica les opcions necessàries del Setting.	
9. L'usuari clica sobre el boto "*Save*"	
	10. El sistema guarda les modificacions del Setting,

Taula 6 Esdeveniment: "Editar un Setting"

Cas alternatiu:
3a. Si l'usuari no esta accedint des de la TN, rebrà un error al intentar obrir l'aplicació.
4a. Si l'usuari es administrador, podrà seleccionar un usuari del llistat per carregar les seccions creades per aquest i poder triar els Settings pertinents.
4b. Si no existeix cap secció o la secció que interessa a l'usuari, primer tindrà que crear.la. Sota el llistat de seccions hi ha un camp de text per introduir el nom d'una secció i al costat el botó "New", al clicar sobre aquest botó es crearà la secció i apareixerà en el llistat de seccions. Si l'usuari és administrador la secció creada estarà vinculada a l'usuari seleccionat en el desplegable d'usuaris.

<u>Eliminar un Setting</u>

Actors: Qualsevol usuari que accedeixi al sistema

Descripció: L'usuari accedeix a l'aplicació des de la pàgina web amb la intenció d'eliminar un Setting existent.
Condició prèvia: El dispositiu des del que accedeix l'usuari ha d'estar connectat a la TN del CERN.
Curs de l'esdeveniment:

Actor	Sistema
1. L'usuari accedeix a Fesa Web i clica sobre la icona d'engranatge.	
	2. El sistema mostra l'apartat "*Settings*".
3. L'usuari clica sobre l'opció "*Setting configuration tool*" per accedir a l'aplicació.	
	3. El sistema mira si l'usuari es reconegut com administrador, si és així carregarà en un desplegable el llistat d'usuaris que han creat algun Setting i per defecte carregarà les seccions i els Settings de l'usuari *Public.* Sí l'usuari no és reconegut com administrador, en el desplegable d'usuaris només apareixerà el seu usuari i es carregaran les seccions i els Settings que ell hagi creat.
4. L'usuari ha de seleccionar una de les seccions existents.	
	5. El sistema carregarà el llistat de tots els Settings que hi ha dins d'aquesta secció.
6. L'usuari seleccionarà del llistat de Settings el Setting que vol eliminar.	
	7. El sistema carrega el en formulari de configuració de Settings la informació referent al Setting seleccionat.
8. L'usuari clica sobre el botó "Del".	
	9. El sistema esborra el Setting seleccionat i actualitza la llista.

Taula 7 Esdeveniment: "Eliminar un Setting"

3.4 Prototips

Fesa Web és una pàgina web Single Page, és a dir, només esta formada per una plana on és carregaran els continguts pertinents. Per tenir una primera idea de com serà l'estructura de la pàgina s'ha creat un prototip Lo-Fi (baixa fidelitat) d'aquesta plana, representat a la Figura 4, i de l'aplicació que tindran que utilitzar els usuaris per crear les subscripcions personalitzades, representat a la Figura 11.

A partir del prototip Lo-Fi de baixa fidelitat de la pàgina web, s'ha creat el prototip Hi-Fi o d'alta fidelitat quedant representat a la Figura 12.

3.4.1 Lo-Fi

Visió general de la pàgina

Pàgina inicial:

A la Figura 4 es pot veure la distribució dels elements a la pàgina. A continuació es detallen els elements principals.

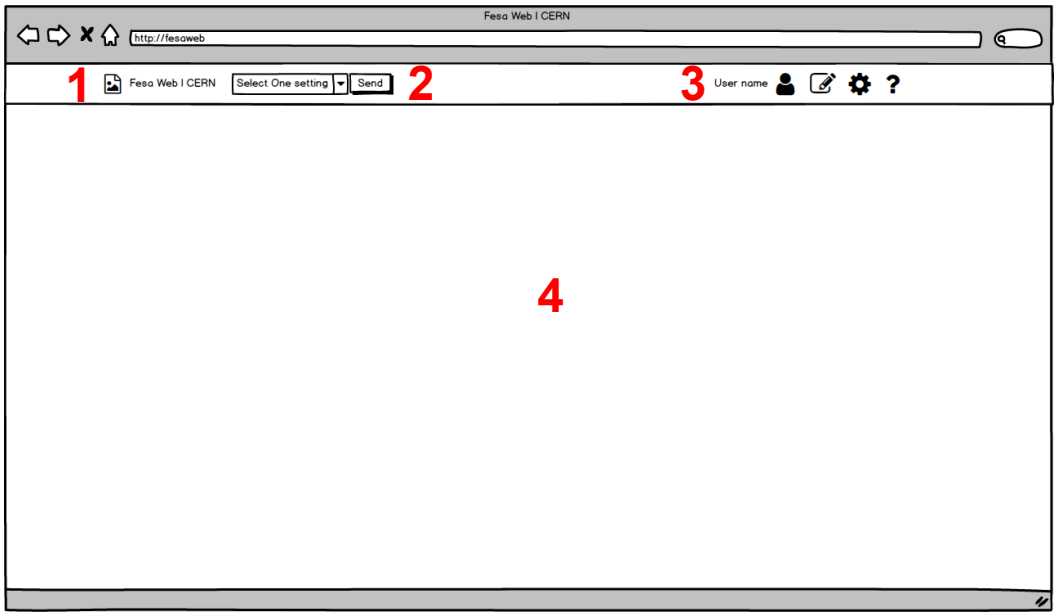

Figura 4 Prototip Lo-Fi de la pàgina web

1. Logotip del CERN i nom de la pàgina
2. Formulari per sol·licitar una subscripció per Setting. Al clicar sobre el desplegable es mostraran tots els Settings de l'usuari per defecte (Public) o els Setting d'un usuari específic si prèviament n'havia seleccionat un.
3. Menú de la pàgina.
4. Contingut de la pàgina, aquí es carregaran els blocs per mostrar els valors dels camps de la subscripció. Més endavant s'expliquen els diferents elements que formen el menú.

Pàgina amb una subscripció realitzada:

Aquí es visualitza com en el contingut de la pàgina s'han creat un seguit de blocs per representar les dades que es reben al realitzar una subscripció. Es crearà un bloc per a cada camp que es rebi informació, la visualització de les dades i les opcions de visualització dependran del tipus de dada rebuda.

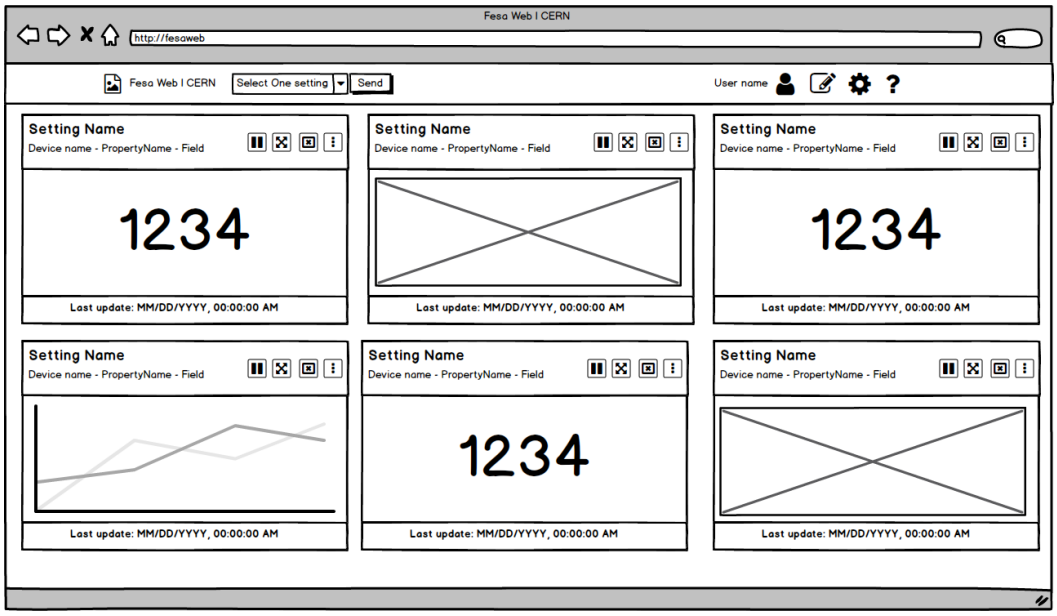

Figura 5 Prototip en Lo-Fi de la pàgina web amb una subscripció activa

Opcions desplegables del menú principal

<u>Menú selecció d'usuari:</u>

Al clicar sobre la icona es mostrarà la finestreta amb el formulari per escollir l'usuari del qual es desitja consultar els Settings. Al seleccionar el botó "Load" es substituirà el contingut del desplegable de Settings pel llistat de Settings de l'usuari escollit i es mostrarà un missatge informant a l'usuari si s'ha pogut carregar correctament o no, també es substituirà el text que a la Figura 5 apareix com "User name" pel nom de l'usuari seleccionat.

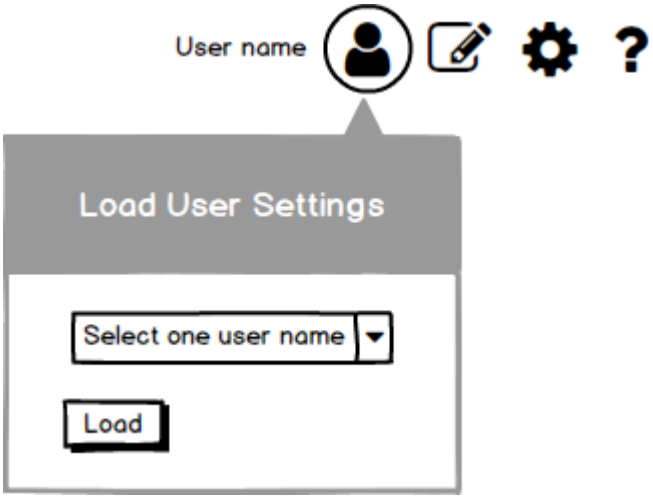

Figura 6 Prototip Lo-Fi de la opció de menú "selecció d'usuari" desplegada

<u>Menú de subscripció sota demanda:</u>

Al clicar sobre la icona es mostrarà la finestreta amb el formulari per crear una subscripció sota demanda. Al clicar sobre el camp "Device" es

demanarà que introduïm com a mínim 4 caràcters o més, ens anirà mostrant el llistat de dispositius que el seu nom conté el text introduït.

Un cop seleccionat un dispositiu s'actualitzarà el contingut del desplegable "Property" amb les propietats del dispositiu seleccionat, per defecte apareixerà seleccionada la primera. Quan es selecciona una propietat s'actualitza el camp de "Fields", de manera que quan l'usuari cliqui sobre aquest camp es mostrarà el llistat de camps disponibles, poden seleccionar si ho desitja tots els que vulgui. Aquest camps és opcional.

El camp "Cycle" és opcional i l'usuari ha de introduir el contingut manualment.

Quan es clica sobre el botó "Send subscription" es realitza la validació i es ressalten els camps que són incorrectes. Si tots els camps són correctes s'envia la sol·licitud de subscripció.

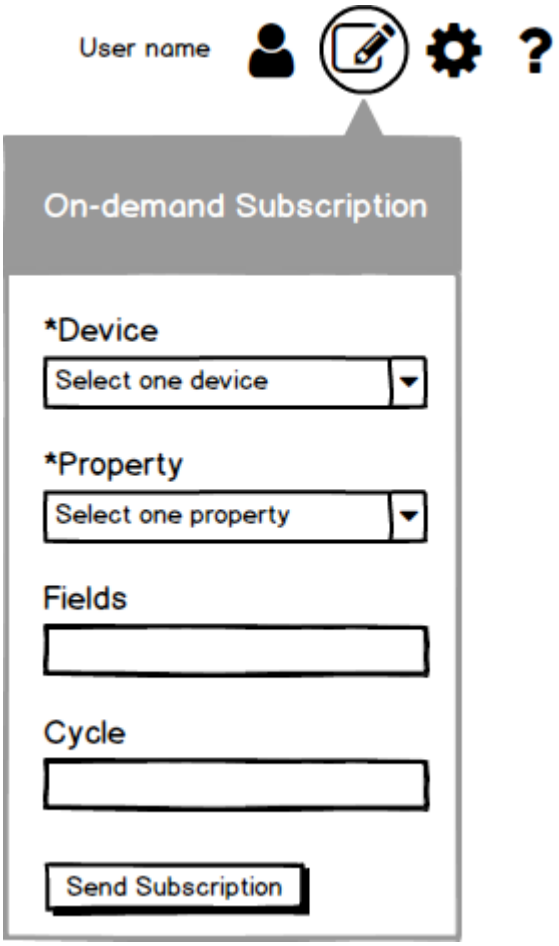

Figura 7 Prototip Lo-Fi de la opció de menú "subscripció sota demanda" desplegada

<u>Menú de configuració:</u>

Al clicar sobre la icona es mostrarà la finestreta amb diferents opcions per realitzar configuracions. La primera opció ens permetrà accedir a la aplicació per poder crear Settings de subscripció personalitzats. La

segona actuarà com un checkbox, de manera que si no esta activa al realitzar una nova subscripció s'eliminarà tota la informació i els blocs referents a les subscripcions realitzades prèviament, mentre que si està activa aquestes subscripcions anteriors es conservaran i els blocs de la nova subscripció s'afegiran a continuació dels blocs existents.

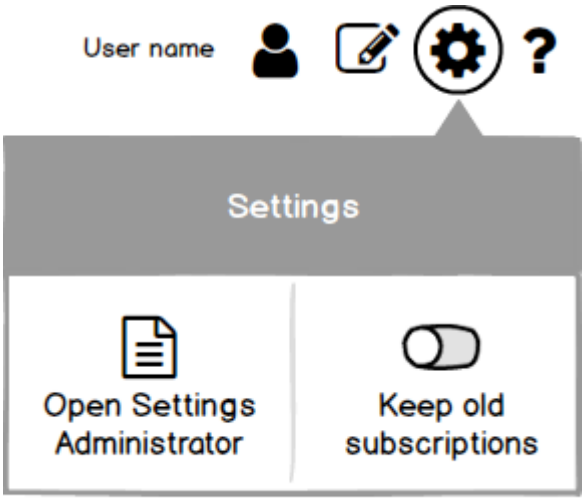

Figura 8 Prototip Lo-Fi de la opció de menú "configuració" desplegada

Representació visualització de dades

<u>Bloc per representar dades d'un camp:</u>

Per cada camp que formi part de la subscripció realitzada es crearà un bloc per mostrar la seva informació. A continuació s'expliquen els elements comuns a tots els blocs i que es troben ressaltats a la Figura 9.

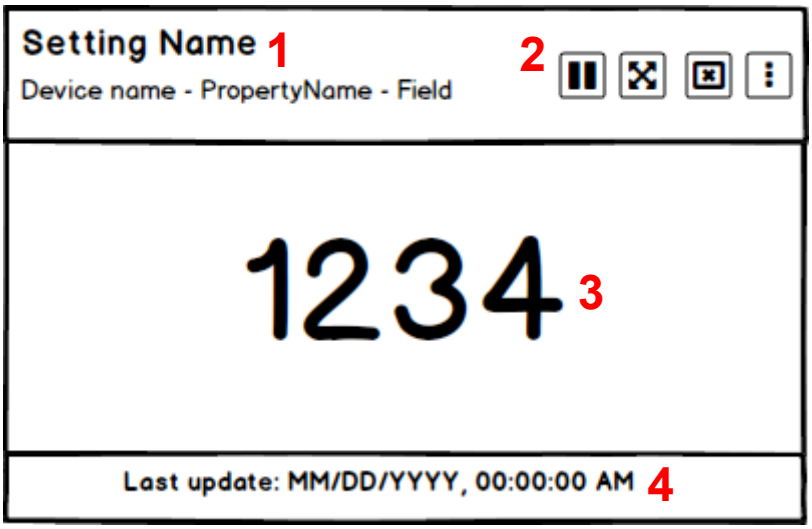

Figura 9 Prototip Lo-Fi de la representació d'un bloc d'un camp de tipus escalar

1. Informació relativa a la subscripció. En el cas que la subscripció s'hagi realitzat mitjançant un Setting es mostrarà el nom del Setting, en cas contrari, es mostrarà el nom del dispositiu i la propietat. A sota es mostrarà en lletra més petita, indicant que és

cada cosa el nom del dispositiu, la propietat i en el cas de que s'hagi especificat també s'afegirà el nom del cicle.

2. Menú amb les opcions a realitzar sobre el bloc:
    a. Aturar/Reiniciar la recepció de dades del bloc,
    b. Mostrar el bloc a pantalla completa,
    c. Eliminar el bloc.
    d. Desplegar la resta d'opcions.
3. Visualització del valor del camp.
4. Data i hora de la última vegada que s'ha rebut un nou valor.

<u>Bloc amb la resta d'opcions del menú desplegades:</u>

La resta d'opcions es divideixen en dos blocs formats per diverses opcions, tal i com es pot veure a la Figura 10. A continuació s'expliquen aquests blocs i les seves opcions.

1. Accions ràpides:
    a. Exporta dades a CSV: Aquesta opció permetrà exportar les dades rebudes en un fitxer CSV.
    b. Aturar/Reiniciar camps del Setting: Opció només disponible si el bloc s'ha creat a partir d'un Setting. La seva funció es aturar o reiniciar la recepció de dades de tots els camps que formin part la mateixa subscripció que aquest camp. Al clicar es modificarà e text de aturar a reiniciar o de reiniciar a aturar, segons l'acció que es tingui que realitzar a totes les opcions d'aquest tipus de tots els blocs afectats.
    c. Aturar/Reiniciar tots els camps del dispositiu: realitza les mateixes tasques que l'opció anterior però només en els camps que formin part del mateix dispositiu.
    d. Aturar/Reiniciar tots els camps de la propietat: Realitza les mateixes tasques que l'opció anterior però només en els camps que formen part del mateix dispositiu i propietat.
2. Vistes: Aquí es mostren les opcions per modificar la manera de visualitzar les dades. Aquestes opcions seran diferents depenent del tipus de dada:
    a. Escalar o cadena de text:
        1. Valor únic,
        2. Taula,
        3. Gràfica.
    b. Vector d'una dimensió:
        1. Taula,
        2. Gràfica
        3. Imatge
    c. Vector de dues dimensions:
        1. Gràfica
        2. Imatge

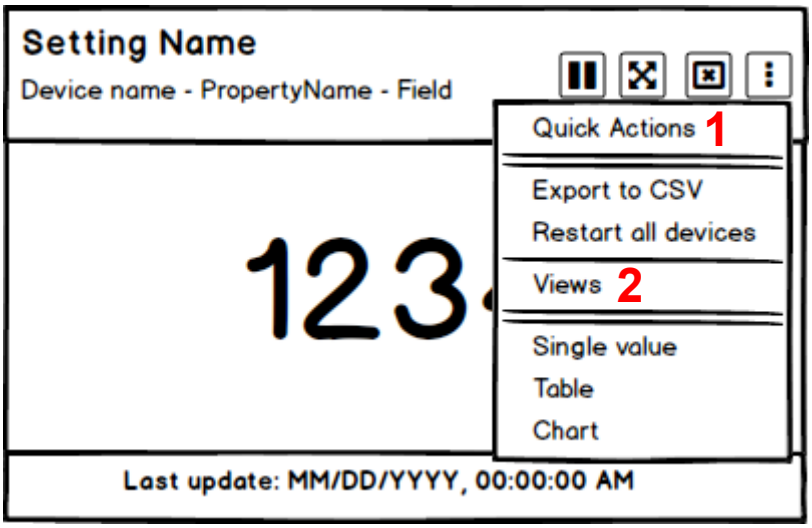

Figura 10 Prototip Lo-Fi de la representació d'un bloc amb la resta d'opcions del menú desplegades

Aplicació per crear Setting personalitzats

La distribució de l'aplicació queda representada a la Figura 11 i la seva funcionalitat és la següent:

1. Desplegable amb el llistat d'usuaris que han creat algun Setting personalitzat. Al seleccionar un usuari es carregaran totes les seccions que ha definit i el llistat de Settings de la primera secció. En el cas que l'usuari no sigui reconegut com administrador, el llistat d'usuaris només contindrà el seu nom i l'aplicació carregarà les seves seccions i Settings.
2. Apartat de seccions. Aquí hi ha el llistat de seccions, al clicar sobre una secció es carregarà el llistat de Settings de la secció seleccionada a l'apartat de Settings i s'introduirà el nom i la descripció d'aquesta secció en el formulari que hi ha just a sota del llistat. En aquest formulari el primer camp equival al nom de la secció i el segon a la descripció. Si es clica sobre el botó "*Add*" es crearà una nova secció, si els camps contenien la informació d'una secció existent i es clica sobre el botó "*Del*" s'eliminarà aquesta secció i tots els Settings que conté, si el botó clicat és "*Update*" s'actualitzarà la informació d'aquesta secció.
3. Apartat de Settings. Aquí hi ha el llistat de Settings per la secció seleccionada anteriorment. Al clicar sobre un Setting es carregarà el nom del Setting en el formulari que hi ha just a sota del llistat i la resta de informació en els camps corresponents de l'apartat de configuració del Setting. Al igual que a l'apartat anterior, si es clica sobre el botó "*Add*" es crearà un nou Setting, si l'únic camp d'aquest formulari, que equival al nom del Setting, conté el nom d'un Setting seleccionat prèviament i es clica sobre el botó "*Del*" s'eliminarà aquest Setting, si el botó clicat és "*Update*" s'actualitzarà el nom d'aquest Setting.
4. Apartat configuració de Setting. Aquest apartat conté tot un conjunt de camps que l'usuari pot omplir per configurar com

funcionarà aquest Setting i personalitzar alguns dels elements, com per exemple el nom dels eixos de coordenades, els seus valors màxim i mínim, si ha d'executar altres Settings existent, etc...

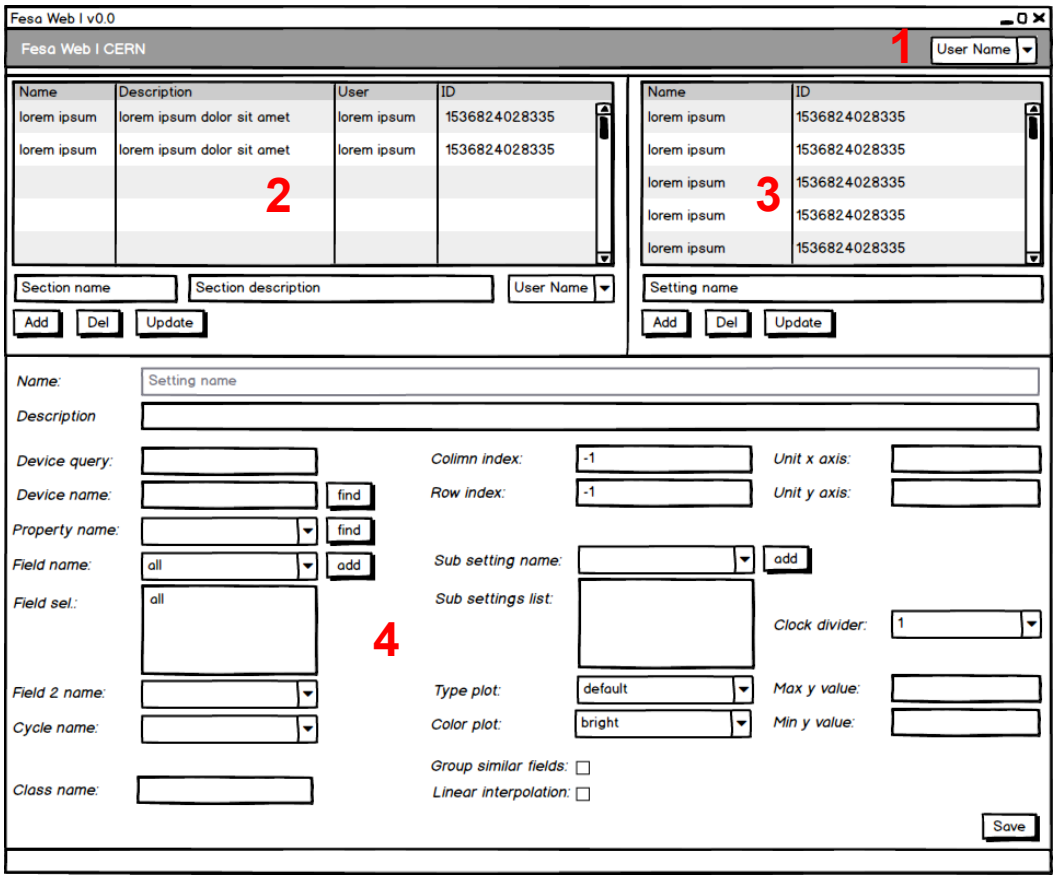

Figura 11 Prototip Lo-Fi de l'aplicació per la creació de Settings

3.4.2 Hi-Fi

Visió general de la pàgina

<u>Pàgina inicial:</u>

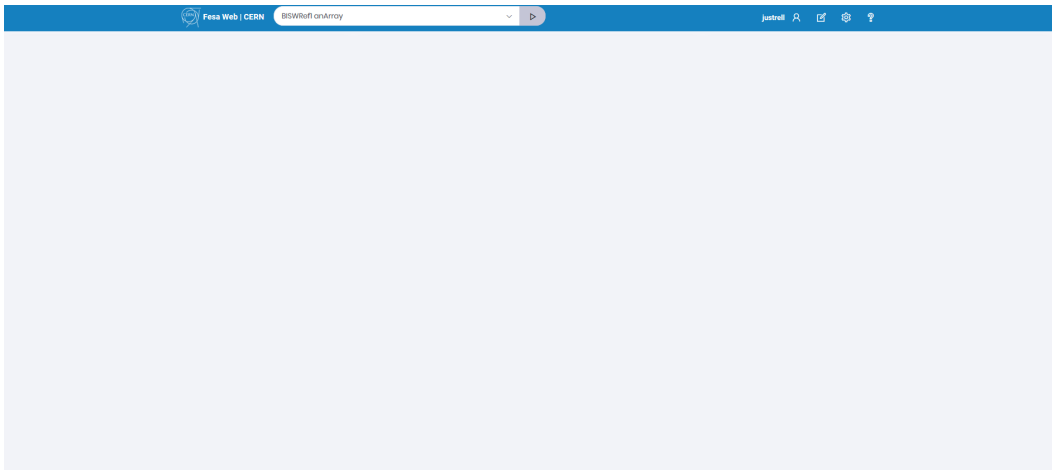

Figura 12 Prototip en Hi-Fi de la pàgina web buida

Pàgina amb una subscripció realitzada:

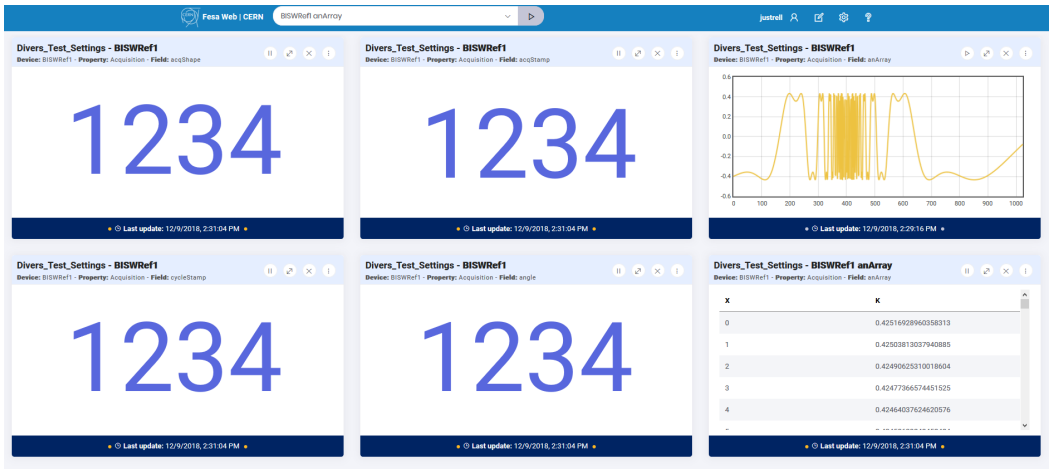

Figura 13 Prototip Hi-Fi amb una subscripció activa

Opcions desplegables del menú principal

Menú selecció d'usuari:

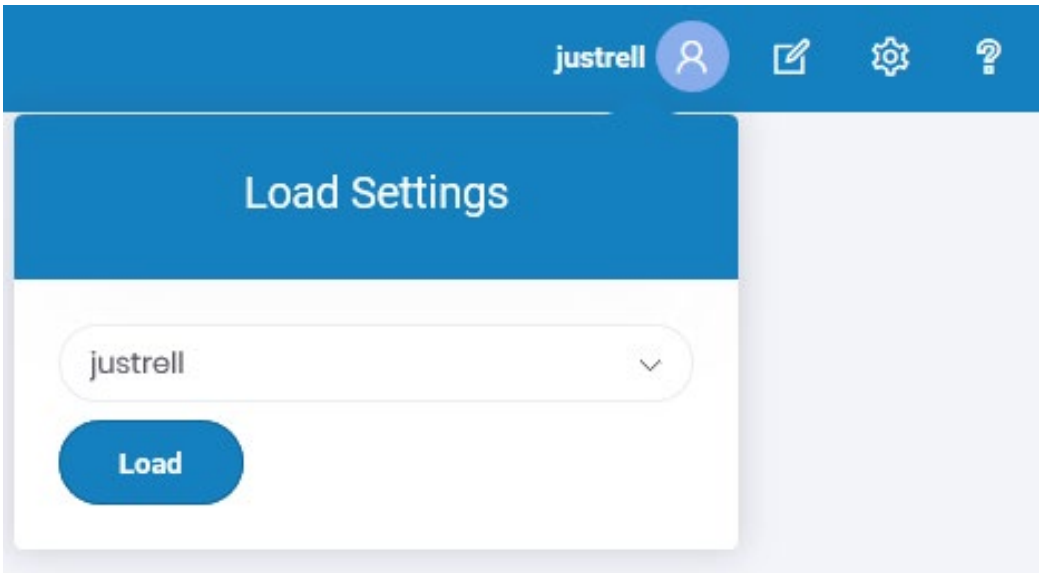

Figura 14 Prototip Hi-Fi opció de *menú "Load Settings"*

Menú de subscripció sota demanda:

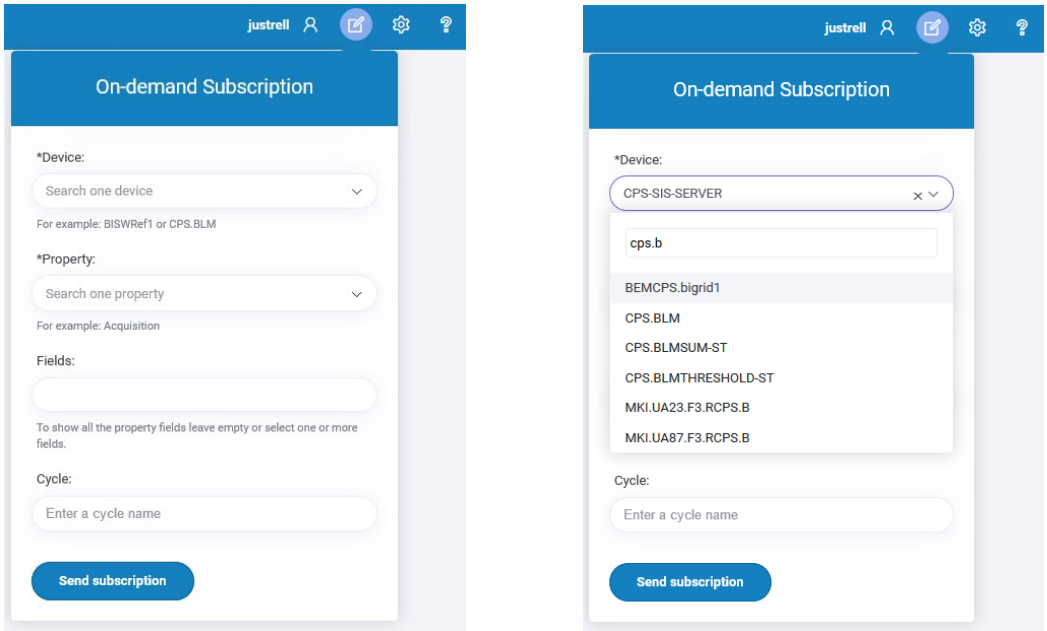

Figura 15 Prototip Hi-Fi opció de menú "On-demand Subscription" i exemple recerca de dispositius

Menú de configuració:

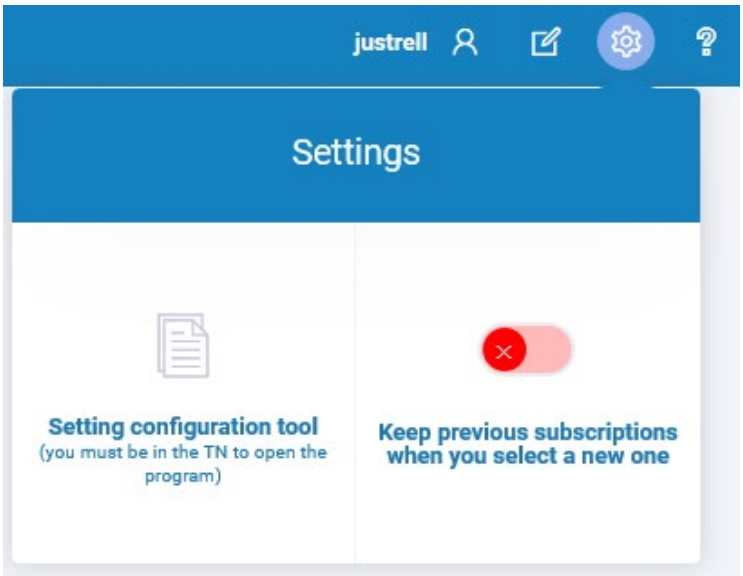

Figura 16 Prototip Hi-Fi opció de menú "Settings"

Representació tipus de visualització de dades

Representació del bloc amb la vista de valor únic:

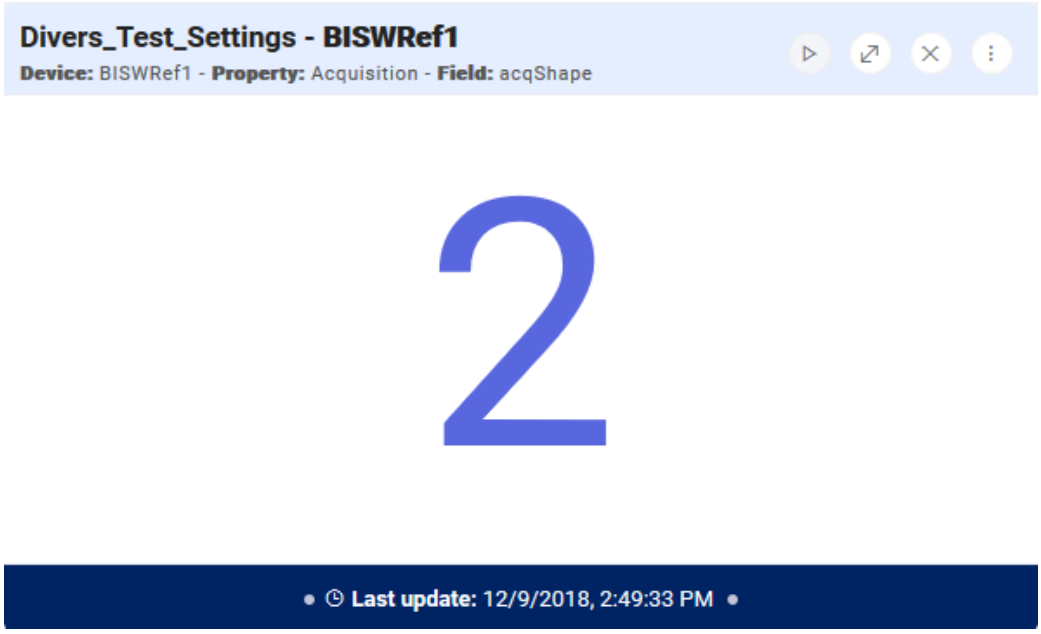

Figura 17 Prototip Hi-Fi visualització de dades com valor escalar

Representació del bloc amb la resta d'opcions del menú desplegades

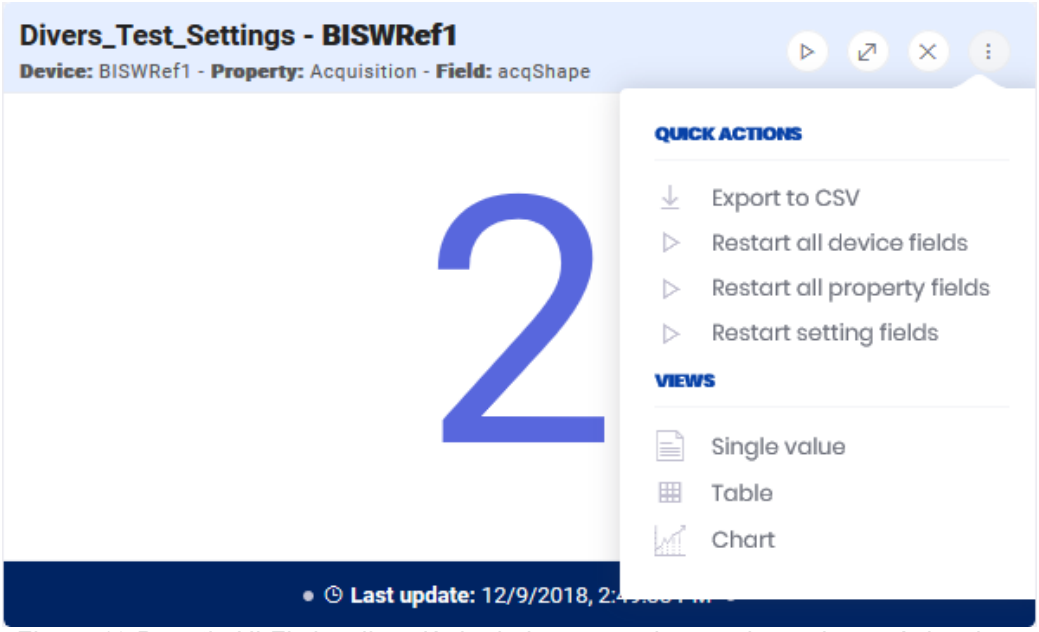

Figura 18 Prototip Hi-Fi visualització de dades com valor escalar amb menú desplegat

Representació del bloc amb la vista de gràfica seleccionada:

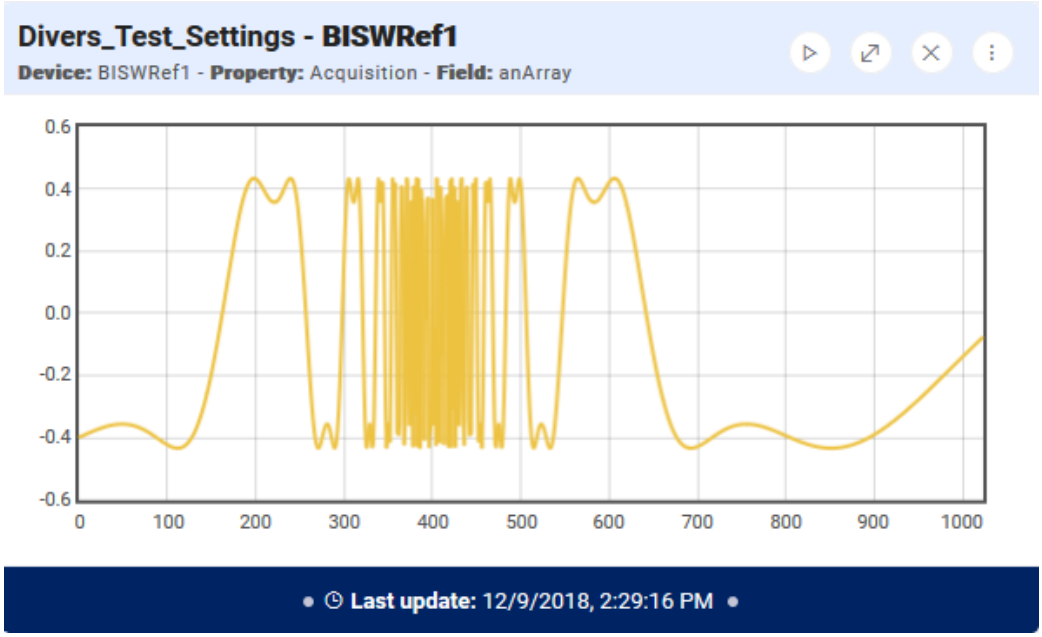

Figura 19 Prototip Hi-Fi visualització de dades com una gràfica

Representació del bloc amb la vista d'imatge:

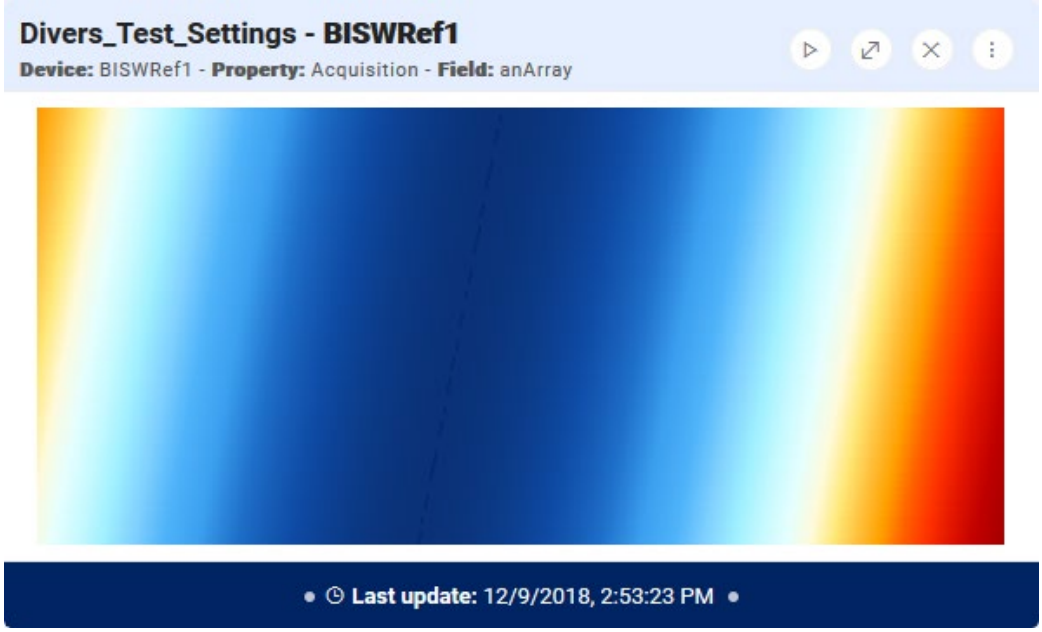

Figura 20 Prototip Hi-Fi visualització de dades com una imatge

Representació del bloc amb la vista de taula:

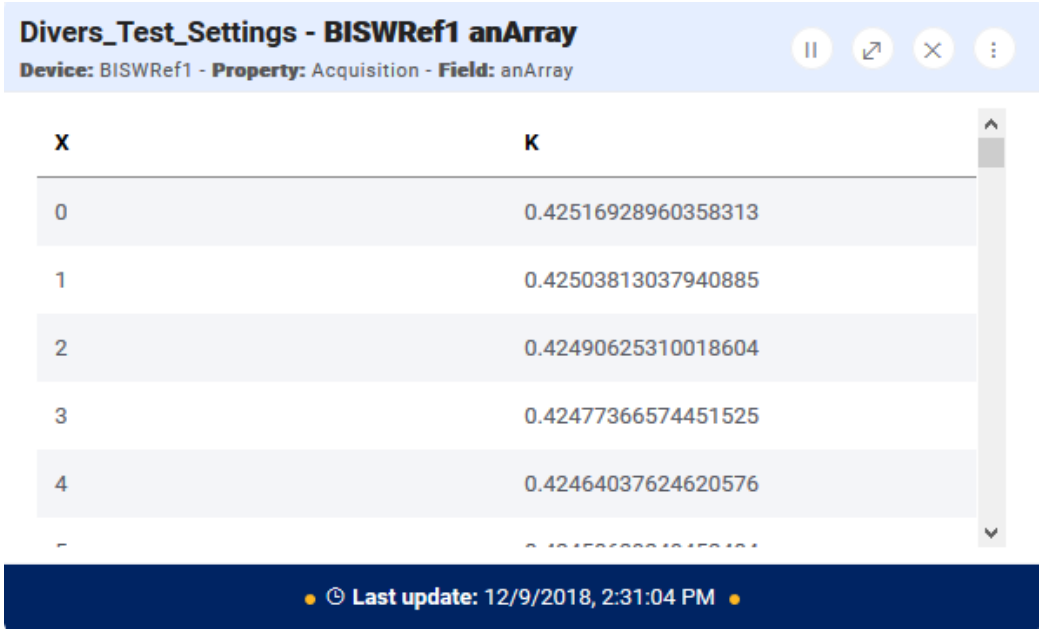

Figura 21 Prototip Hi-Fi visualització de dades com una taula

Aplicació per crear Setting personalitzats

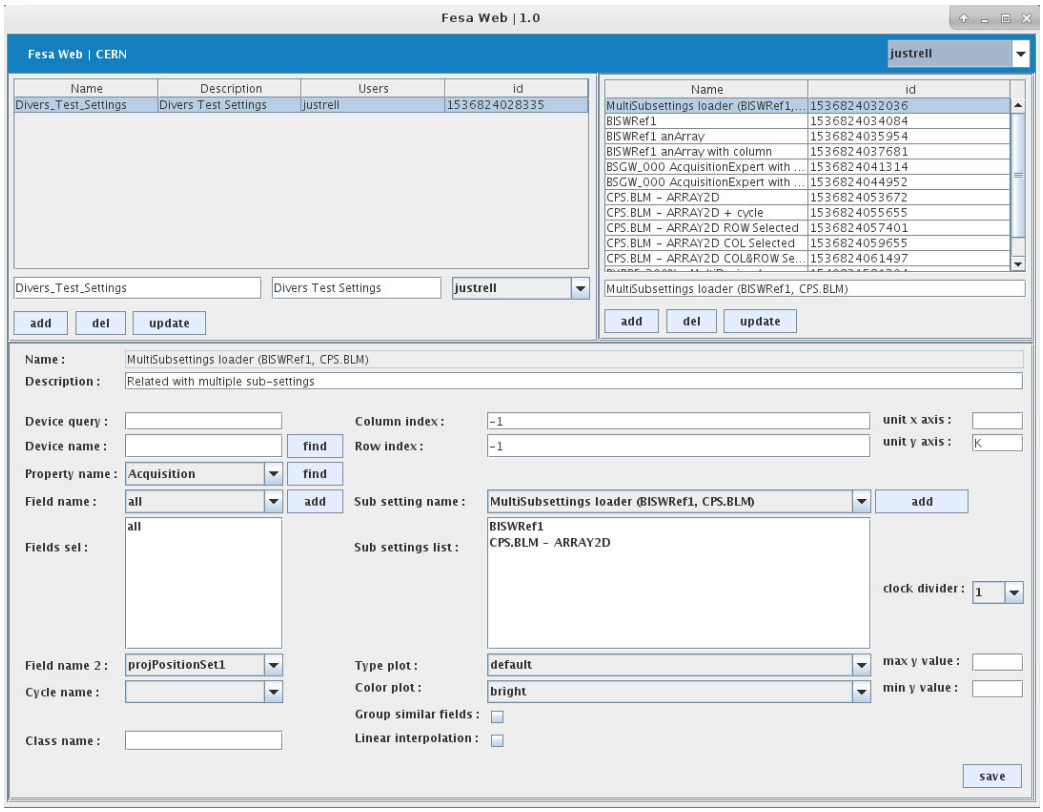

Figura 22 Prototip Hi-Fi de l'aplicació per la creació de Settings

4. Arquitectura

El projecte FESA Web utilitza una arquitectura client-servidor, aquesta arquitectura queda representada de manera esquemàtica a la Figura 23.

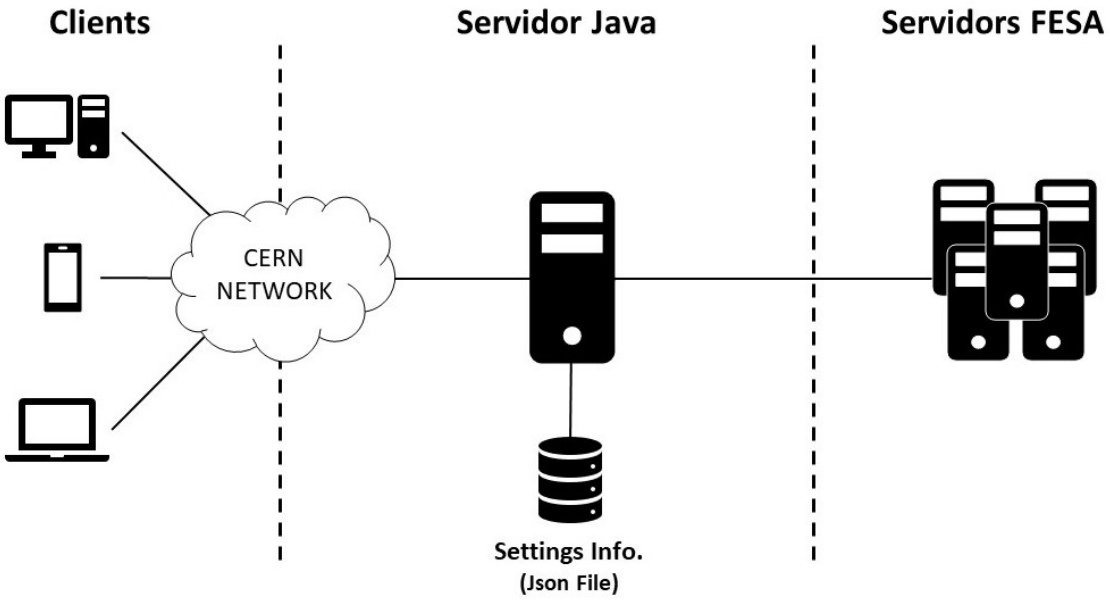

Figura 23 Model de l'arquitectura de FESA Web

4.1 Client

Es considera un client qualsevol dispositiu que estigui connectat a la xarxa del CERN i disposi d'un navegador web, de manera que, mitjançant la URL especifica del projecte pugui establir una connexió permanent amb el servidor web mitjançant WebSockets.

Fesa Web funciona en qualsevol navegador web modern, exceptuant Internet Explorer i Edge, en aquests dos navegadors per poder utilitzar connexions WebSockets cal que aquestes connexions siguin segures; al tractar-se d'una pàgina web accessible només des de la xarxa del CERN, no s'ha contemplat aquesta opció.

4.2 Servidor

El servidor web és servidor principal, que s'encarrega de rebre les sol·licituds dels clients, processar-les i d'enviar periòdicament la informació específica per a cada client. A la Taula 8 es mostra la configuració d'aquest servidor.

Sistema operatiu	CentOS 7 (64 bits)
Servidor Web	Apache Tomcat 9.0.12
Llenguatge de programació	Java 9

Taula 8 Configuració servidor

També disposem de múltiples servidors FESA amb els quals el servidor principal s´ha de comunicar. Aquests servidors són creats, executats i mantinguts per la secció de Software del grup Beam Instrumentation del CERN.

4.3 Bases de dades

Com a base de dades s'utilitza un fitxer JSON que conté un llistat de Settings i Sub-settings creats pels diferents usuaris. Un Setting serveix per agrupar múltiples Sub-settings, mentre que, els Sub-settings contenen la informació necessària per realitzar les subscripcions amb els servidors FESA desitjats, així com diferents configuracions sobre com s'haurà de processar i mostrar la informació que s'obtingui d'aquesta subscripció en el lloc web. Les taules 9 i 10, mostren respectivament les etiquetes i la descripció de les etiquetes per poder crear un Setting i els diferents Sub-settings que formaran part d'aquest Setting. Per la creació de Settings i Sub-settings s'utilitza una aplicació Java creada expressament per facilitar als usuaris aquesta tasca i només accessible des de la *Technical Network*[1] del CERN.

Etiquetes per la creació d'un Setting

Etiqueta	Tipus	Descripció
id_setting	string	Identificador únic del setting
name_setting	string	Nom del setting
description_setting	string	Descripció del setting
user	string	Usuari que ha creat el setting

Taula 9 Llista de les etiquetes necessàries per la creació d'un Settings

Etiquetes per la creació de Sub-settings

Etiqueta	Tipus	Descripció
id_sub_setting	string	Identificador únic del sub-setting
name_sub_setting	string	Nom del sub-setting
description_sub_setting	string	Descripció del sub-setting
device_name	string	Nom del dispositiu amb el que es vol fer la subscripció
fesa_property_name	string	Propietat del dispositiu de la qual volem consultar els camps
fesa_field_name	string	Camp o camps, separats per comes que es volen consultar. Si el camp esta buit es consultaran tots els camps de la propietat.
fesa_field_name_2	string	Aquest camp serveix per veure la relació que hi ha entre els valors de dos camps de tipus Array i serà utilitzat en la representació de la resta de valors de tipus Array que tinguin la mateixa quantitat de valors per a l'eix de les X i els valors de l'altre array es representaran a l'eix de les Y.

[1] Xarxa local del CERN des de la qual no es disposa d'accés a internet.

cycle	string	Nom del cicle del qual és vol obtenir els valors. Si els dispositiu es dependent de cicles i no s'especifica un cicle en concret es consultaran els valors dels diferents cicles. Si el dispositiu no depèn dels cicles no ha de contenir cap valor.
fesa_class_name	string	En aquesta etiqueta es pot introduir el nom el nom d'una classe[2] especifica i només és necessària quan es vol realitzar una consulta a múltiples dispositius a l'hora. Una classe esta formada per un o varis dispositius.
device_query	string	Aquesta etiqueta és obligatòria en el cas que s'hagi introduït el nom d'una classe. Aquest camp serveix per consultar múltiples dispositius a l'hora de la classe indicada. Aquí s'ha d'introduir part del nom del dispositiu i la consulta es realitzarà a tots els dispositius que el seu nom estigui format per aquesta paraula.
similar_fields_group	boolean	Quan s'ha introduït part del nom del dispositiu a l'etiqueta *device_query* si el valor d'aquesta etiqueta és FALSE es mostrarà la informació de manera individual de cada camp dels diferents dispositius. Sí el seu valor és TRUE el que es farà serà agrupar la informació dels camps que diguin igual en els diferents dispositius en una mateixa finestra.
clock_divider int	integer	Permet modificar la freqüència en la que s'enviaran les dades als clients independentment de la freqüència amb la que el dispositiu enviï noves dades al servidor web. Els valors que pot prendre van de 1 a 10, de manera que el valor 1 equival a 1Hz i 10 a 10Hz.
fesa_field_col_index	integer	Aquesta etiqueta té efecte sobre els camps de tipus Array o Array 2D. El que fa és especificar que només volem rebre informació d'una columna en concret. En el cas d'un Array, això suposarà que només consultarem un valor, de manera que es comportarà com si fos un valor de tipus escalar. En el cas de l'array 2D, suposarà consultar la columna especifica per a cada fila, de manera que es comportarà com si fos un array d'una dimensió. El valor per defecte és -1, això vol dir que no es filtraran els valors per cap columna, si el valor és -2, vol dir que volem consultar la última posició de l'array. Qualsevol altre valor serà equivalent a l'índex de la columna que es vol consultar, en cas que l'índex sigui superior al nombre d'elements de l'array aquest valor quedarà representat per 0.
fesa_field_row_index	integer	Aquest camp només té efecte sobre els camps de tipus Array 2D i serveix per indicar que només volem consultar els valors d'una fila en concret. El funcionament i valors que pot prendre són els mateixos que l'etiqueta *fesa_field_col_index.* En el cas que també s'hagi especificar un valor diferent a -1 en

[2] Un servidor FESA és una instància d'una classe programada en C++ i aquesta s'identifica pel seu nom.

		aquesta etiqueta vol dir que només volem consultar el valors d'una posició especifica de l'array 2D de manera que aquest es comportarà com si fos un valor escalar.
type_plot	integer	Per defecte cada tipus de dada te una visualització per defecte, aquest camp permet modificar la visualització per defecte dels camps poden triar entre: Etiqueta, Taula, Gràfica o Imatge.
color_plot	integer	Permet modificar la paleta de colors que s'utilitzarà per representar els valors.
plot_unit_x	string	Indicar un nom per l'eix de les X o en el cas de les taules, el nom de la columna/fila corresponent.
plot_unit_y	string	El mateix que l'etiqueta anterior però per l'eix de les Y.
max_value	float	Especifica un valor màxim que poden prendre els camps, de manera que el valor màxim en la representació d'una gràfica sempre serà el mateix.
min_value	float	És com l'etiqueta anterior però amb el valor mínim.
fit	boolean	Si el seu valor és TRUE, farà que en les vistes de gràfica aparegui una línia horitzontal indicant el valor intermedi de la gràfica.
id_sub_setting_list	string	Aquesta etiqueta pot contenir el identificador únic d'altres sub-settings, separats per comes, de manera que si s'escull aquest sub-Setting a la pàgina web, el que farà serà crear les subscripcions d'aquests sub-settings.

Taula 10 Llista de les etiquetes necessàries per la creació de Sub-settings

Per la consulta de l'estat, versió, configuració o simplement per poder realitzar una subscripció amb els diferents servidors FESA, cal consultar una base de dades que conté aquesta informació, però per fer-ho s'utilitza una llibreria externa creada pel CERN.

4.4 APIs/Llibreries de tercers

En el front-end:

- Bootstrap4: Framework utilitzat per facilitar la creació d'un disseny adaptatiu.
- BlockUI: La intenció d'utilitzar aquesta llibreria és per crear blocs informatius mentre la pàgina esta carregant o processant informació. Aquesta part en aquests moments en cara es troba pendent d'implementar.
- Boostrap Select: S'utilitza per dotar donar format i augmentar les funcionalitats dels desplegable de selecció de Settings i llistat d'usuaris.
- Flot: Llibreria utilitzada per la creació de les gràfiques.
- Google Web Font Loader: La seva funcionalitat es carregar fonts de text externes i diferents a les que implementen els navegadors per defecte.

- Jquery: Necessària perquè moltes de les llibreries JavaScript estan basades en Jquery.
- Select2: S'utilitza per donar format i augmentar les funcionalitats dels desplegables del formulari per realitzar una subscripció sota demanda.
- Toastr: Llibreria utilitzada per crear les notificacions de la pàgina web.

En el Back-end:
S'han utilitzat llibreries creades pel CERN per poder accedir i comunicar-se amb els servidors FESA. Aquestes llibreries són:

- JapcCommunicationNoINCA: S'utilitza en el servidor per establir la comunicació amb un dispositiu concret dels servidors FESA i després realitzar la subscripció per una propietat especifica.

  Per realitzar la comunicació es crea una instància d'aquesta classe especificant-li el dispositiu amb el que es desitja establir la comunicació i acte seguit es crida al mètode *subscribe* d'aquest objecte per realitzar la subscripció enviant com a paràmetres el nom del cycle i de la propietat a la que volem subscriure'ns.
- ControlDeviceService: S'utilitza via Ajax en el formulari de subscripció sota demanda de la pàgina web quan un usuari comença a introduir el nom d'un dispositiu per obtenir via Ajax el llistat de dispositius que contenen la cadena de text introduïda.

  Proporciona dades relacionades amb *ControlDevice*. De manera que mitjançant el mètode *getControlDevices()* es realitza una consulta per obtenir tots els dispositius que compleixin aquesta consulta retornant un llistat d'objectes de tipus *ControlDevice* i amb *getControlDevice*() s'obté un objecte de tipus *ControlDevice* que coincideix amb el nom del dispositiu enviat com a paràmetre.
- ControlDevice: Serveix per consultar la informació d'un dispositiu com per exemple el seu nom, versió o les propietats que conté. De fet per és per aquest últim motiu pel qual l'utilitzem, per poder retornar via Ajax el llistat de propietats d'un dispositiu quan aquest es seleccionat en el formulari de subscripció sota demanda.
- DevicePropertyService: Aquesta classe s'utilitza per obtenir els camps que conté la propietat seleccionada en el formulari per realitzar una subscripció sota demanda de la pàgina web. Aquest llistat és enviada via Ajax.
- ServiceLocator: Aquesta classe implementa el patró Singleton i l'utilitzem per obtenir la instància de *ControlDeviceService* i *DevicePropertyService*.

En el capítol 5.1, hi ha una mostra de com s'utilitzen aquestes llibreries en el costat del servidor.

4.5 Arquitectura de la capa Front-end

4.5.1 Accés a la pàgina web i realització d'una subscripció

Quan l'usuari accedeix a Fesa Web el servlet *IndexController* s'encarrega de generar la pàgina web i enviar-la al navegador, un cop s'ha carregat tot el contingut de la plana en el navegador es crida al mètode *init()* de *Controller* que inicialitza tots els esdeveniments que caldrà controlar, per exemple: la inclinació del dispositiu, modificació de les dimensions del navegador, si es clica sobre el botó per realitzar una subscripció sota demanda o per Setting, etc. També es crea una comunicació permanent amb el servidor utilitzant *WebSockets*, a patir d'una instància de la classe *WebSocketClient* que és l'encarregada de crear aquesta comunicació i de rebre i enviar missatges al servidor. Un cop establerta la comunicació, l'usuari pot sol·licitar la creació de subscripcions a dispositius FESA. Per fer-ho disposa de tres sistemes:

1. Formulari subscripció sota demanda: l'usuari ha d'omplir aquest formulari indicant el nom del dispositiu al que es vol realitzar la subscripció, la propietat del dispositiu sobre la que vol rebre la informació, els camps que vol consultar, en el cas que no s'indiqui cap es rebran tots els camps de la propietat i el cicle del que es vol rebre informació. El cicle serveix per especificar una fase concreta del accelerador de la que es volen obtenir dades. Si no s'especifica cap cicle, s'aniran obtenint dades dels diferents cicles o simplement perquè aquest dispositiu no te cap dependència amb els cicles i no s'ha d'especificar.
2. Subscripció via Setting: l'usuari utilitza un desplegable que conté un llistat de Settings d'un usuari prèviament seleccionat i envia la sol·licitud.
3. Subscripció per URL: En aquest cas al accedir a Fesa Web la URL conté les dades d'una subscripció sota demanda feta prèviament o l'identificador d'un Setting, de manera que, al carregar-se la pàgina i establir-se la connexió mitjançant WebSocket s'envia la sol·licitud per generar la subscripció de manera automàtica.

Quan s'envia una sol·licitud de subscripció, el sistema no fa res fins que no rep un missatge del servidor o es perd la comunicació amb aquest i ho comunica a l'usuari. Quan la instància de *WebSocketClient* rep un missatge del servidor el que fa és transformar aquest missatge en un objecte literal mitjançant *JSON.parse()* i l'envia a *Controller* cridant al mètode *processMessage()* d'aquesta classe, on és mira el tipus de missatge rebut. Els tipus de missatge que es poden rebre són:

1. DATA_INFO: En aquest cas estem reben informació referent a una subscripció feta sota demanda, o mitjançant una URL amb la informació relativa a aquesta.
2. DATA_SETTING_INFO: El contingut d'aquest missatge és la informació relativa a una subscripció per Setting o d'una URL amb el identificador d'un Setting.
3. ERROR: S'ha produït un error i rebem aquest missatge per part del servidor.

4. INFO: S'està rebent algun tipus d'informació.
5. SUCCES: Es rep una confirmació de que alguna cosa sol·licitada s'ha realitzat correctament en el costat del servidor.
6. DATA_MULTIPLE: Es reben dades dels camps de les subscripcions que han estat actualitzats.

En el moment en el que l'usuari clica sobre el botó per realitzar una subscripció, ja sigui sota demanda (es crida al mètode *subscribeOnDemand* de *Controller*) o mitjançant un Setting (es crida al mètode *subscribeToSetting* de *Controller*), l'aplicació mira si l'opció de conservar subscripcions anteriors no esta seleccionada per saber si cal eliminar tota la informació relativa a les subscripcions anteriors.

Un cop enviada una sol·licitud de subscripció, utilitzant qualsevol dels tres sistemes comentats anteriorment, es poden rebre dos tipus de missatges relacionats amb aquesta sol·licitud: ERROR, DATA_INFO o DATA_SETTING_INFO. Sí el missatge és de tipus ERROR, a les hores es mostrarà el missatge d'error amb el text relatiu a aquest error, en cas contrari, vol dir que el servidor ha pogut generar la subscripció sol·licitada i ens esta enviant total al informació relacionada amb aquesta incloent informació sobre els camps que s'han de consultar com el nom, el tipus,... *Controller realitza les següents tasques:*

1. **Crida 1** db.add(): Envia a *db* la informació del missatge perquè pugui crear els camps que necessiti per emmagatzemar les seves dades.
2. **Crida 2** *createBoxes()*: mètode de *Controller* per crear els diferents elements blocs per representar les dades i els afegeix a la llista de blocs existents, cada un d'aquest sap a quin o quins camps fa referencia.
3. Un cop s'han creat els elements per emmagatzemar les dades que rebrem i s'han creat el blocs on es mostraran aquestes dades *Controller* crida al mètode *resizeBoxes()* per ajustar el tamany dels blocs depenent de la quantitat total de blocs que s'estan visualitzant. Acte seguit *Controller* envia un missatge al servidor informant-li que ja pot començar a enviar-li dades de la subscripció.

Crida 1: db.add()

Aquí DB recorre tots els camps que conté la informació que ha rebut com a paràmetre creant en cas de que sigui necessari un objecte que emmagatzemarà les dades relatives a aquest camp. A la Figura 24 es pot veure el procés que es segueix per saber si és necessari crear o no un objecte per aquest camp i com s'escull el tipus de l'objecte.

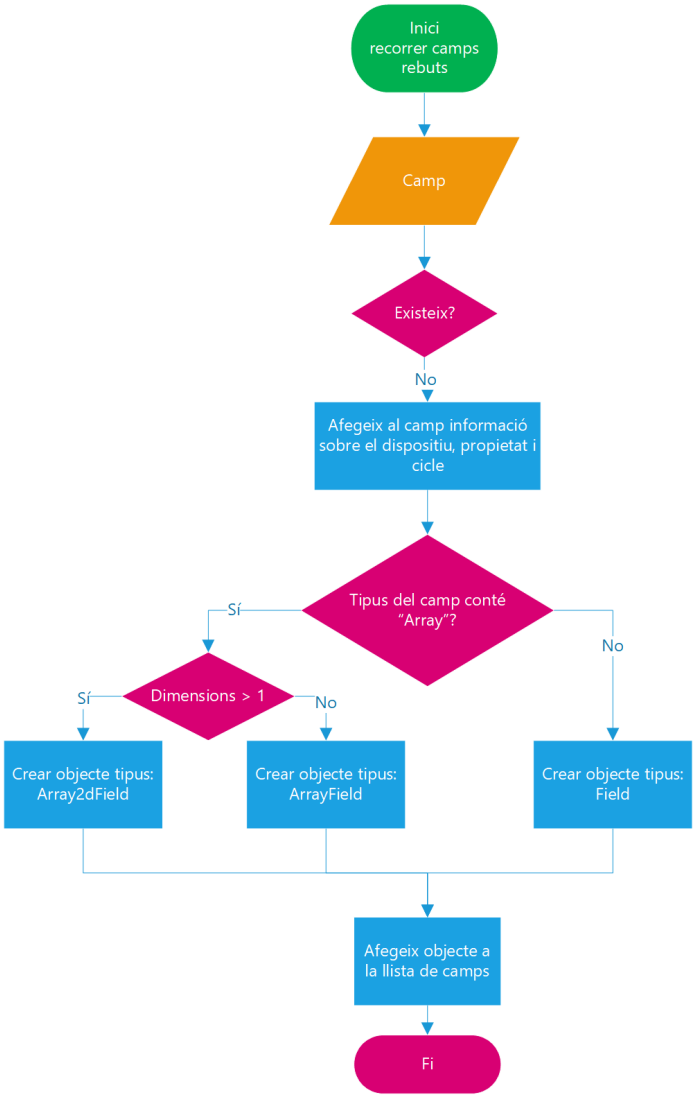

Figura 24 Diagrama de flux de la creació d'un objecte per emmagatzemar dades d'un camp

Crida 2: *createBoxes()*

Aquest mètode el que fa és crear una instància, d'alguna de les classes que hereten de *FieldBox,* per a cada un dels camps que s'han rebut. El tipus de la instància creada depenent del tipus del camp al que fa referencia i de la configuració del Setting, en el cas que la subscripció s'hagi creat a partir d'un Setting. Les possibles classes per crear aquestes instàncies són:

- *ScalarBox*: Gestiona la visualització de camps de tipus escalar o cadena de text.
- *ScalarGroupedBox*: Les instancies d'aquesta classe només es poden crear quan la subscripció s'ha realitzar mitjançant un Setting i es necessària quan obtenim dades, de tipus escalar o cadena de text, de múltiples dispositius que pertanyen a una mateixa subscripció i s'ha triat la opció d'agrupar els camps. Això el que fa es crear un bloc en el que es mostrarà el valors d'un mateix camp en múltiples dispositius.
- *ArrayBox*: Gestiona la visualització de vectors d'una sola dimensió.

- *ArrayGroupedBox*: Al igual que *ScalarGroupedBox,* gestiona la visualització dels valors d'un mateix camp de múltiples dispositius en un bloc, però en aquest cas per camps de tipus vector d'una dimensió i només es pot crear una instància si la subscripció s'ha realitzat mitjançant un Setting i amb la opció d'agrupar els camps.
- *Array2dBox*: Gestiona la visualització de vectors de dues dimensions.

A la Figura 25 es representa el procés per saber quin tipus d'instància s'ha de crear.

El següent pas és cridar al mètode *createPortlet()* d'aquesta instància perquè generi i afegeixi al contingut de la plana el codi HTML necessari del bloc en el que es mostrarà la informació del camp, configuri les opcions de menú del bloc i generi els esdeveniment relacionats amb les accions que es podran realitzar sobre aquest bloc.

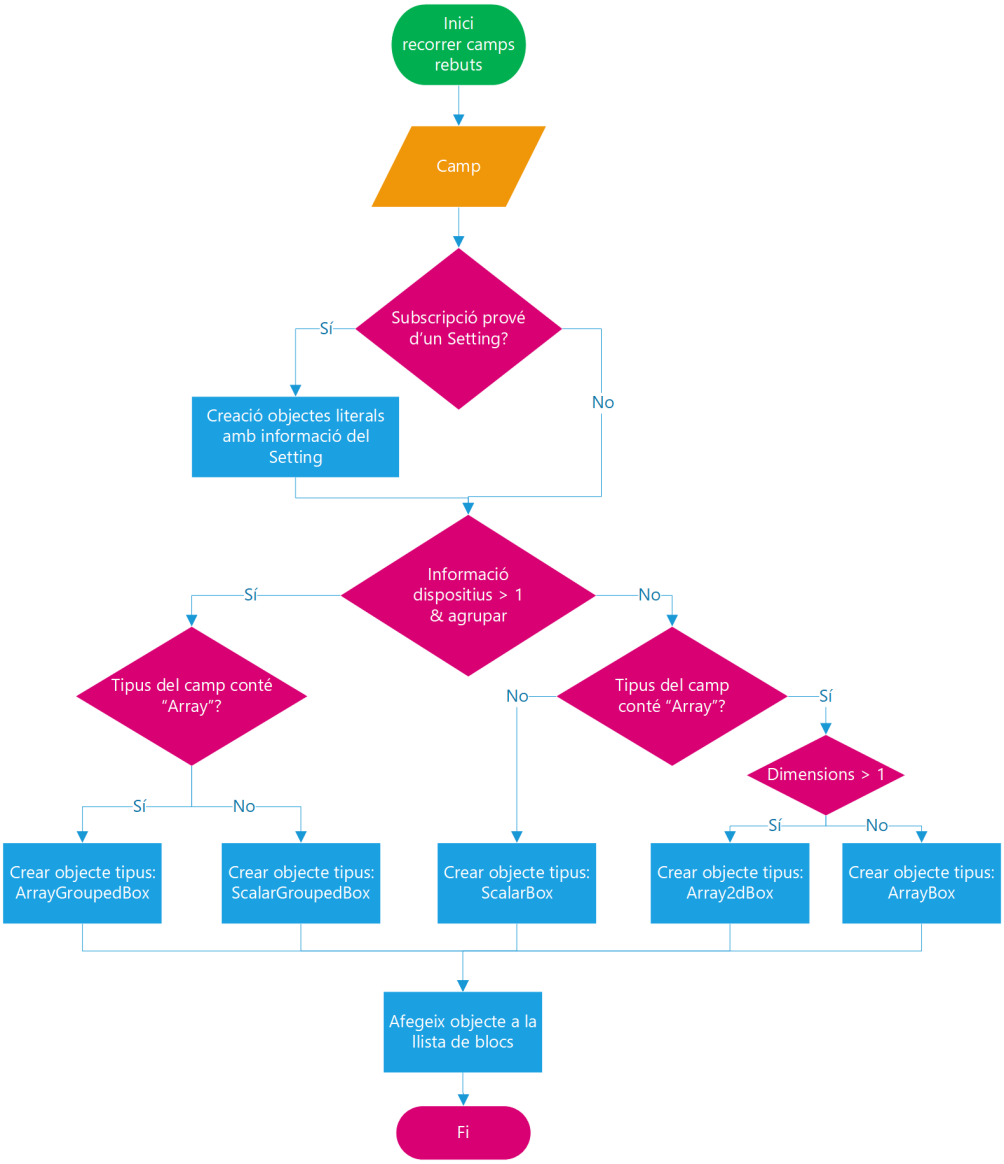

Figura 25 Diagrama de flux per la creació d'una instància de les classes que hereten de FieldBox

4.5.2 Recepció de dades de les Subscripcions actives

Un cop s'ha enviat el missatge al servidor confirmant que pot començar a rebre dades, l'aplicació pot rebre missatges amb les dades de les subscripcions en qualsevol moment. Això passa quan es rep un missatge del tipus DATA_MULTIPLE, aquest tipus de missatge té una propietat anomenada *dataList* que conté un vector amb objectes, on cada objecte té una propietat amb l'identificador de la subscripció i una propietat amb un vector que conté la informació dels camps actualitzats d'aquesta subscripció.

El procés que segueix *Controller* quan rep un missatge de tipus MULTIPLE és el següent:

1. **Crida 1:** *db.update()*, enviant com a paràmetre el contingut de la propietat *dataList* perquè *db* actualitzi els valors i els històrics dels camps que s'estan rebent.
2. **Crida 2:** *updateBoxes()*

Crida 1: *db.update()*

Recorre els camps de cada una de les subscripcions rebudes en un array com a paràmetre. En cada iteració mira si existeix informació d'aquest camp en cas afirmatiu crida al mètode *update(),* d'aquest camp, enviant-li el nou valor rebut i en el cas de que sigui un vector també la informació sobre les dimensions del vector.

Dins del mètode *update*() de cada camp, si el camps és de tipus vector, el que es fa és mirar si es rep informació sobre les dimensions i si aquesta es diferent a la que existia prèviament, si és diferent s'elimina la informació existent de l'històric i guarda les noves dimensions del vector. Acte seguit crida al mètode *update()* de la seva classe pare que és *Field.*

El mètode *update()* de *Field* el que fa és fer una conversió dels valors rebuts al tipus de dades que li correspon, ja que tots els valors els hem rebut com una cadena de text, això és necessari sobre tot en el cas dels vectors. Acte seguit guarda aquest nou valor i actualitza l'històric.

Crida 2: *updateBoxes()*

Recorre els camps de cada una de les subscripcions rebudes com a paràmetre en format de vector i crea un nou objecte on les seves propietats seran els identificadors de cada subscripció rebuda i contindran un vector amb els noms dels seus camps. Aquest objecte és necessari ja que un bloc pot estar mostrant informació de múltiples dispositius. Acte seguit es recorre el vector de *Controller* que conté tots els blocs existents i es crida al mètode *isUpdatable()* de cada un d'aquest blocs, enviant com a paràmetre l'objecte creat anteriorment, de manera que, cada bloc mira si mostra informació d'algun dels camps dels que s'han rebut noves dades. Sí el resultat d'aquest mètode és afirmatiu i el bloc no esta pausat es crida al mètode *print()* del bloc.

Dins del mètode *print()* es mira el tipus de visualització del bloc i es crida al mètode específic per imprimir per pantalla aquest tipus de visualització.

4.5.3 Càrrega dels Settings d'un usuari seleccionat

Un cop la pàgina web s'ha acabat de carregar es crida al mètode *init()* de *Controller* que crea un esdeveniment per detectar quan es clica sobre el botó enviar del formulari *"Load Settings".* Quan un usuari clica sobre aquest botó s'estableix una comunicació asíncrona amb el servidor via AJAX i se li envia un missatge especificant que l'acció que es desitja realitzar és *"getSettingsByUser"* juntament amb el nom de l'usuari seleccionat. Al rebre la resposta del servidor es realitzen els passos següents:

1. Es crida al mètode *Controller.updateSettingsSelect*() enviant-li com a paràmetre la resposta del servidor. Aquest mètode el que fa és:
    1. Buidar el contingut del desplegable.
    2. Recorrer les seccions rebudes i per a cada secció mira si existeixen Settings, en cas afirmatiu crea un grup en el desplegable amb el nom de la secció i a dins d'aquest crea una opció per a cada un dels Settings que conté la secció, utilitzant el nom del Setting com a text i el seu ID com a valor.
2. S'actualitza a la barra superior de la pàgina el nom del usuari del que s'ha carregat el llistat de Settings.
3. Es crea una cookie amb el nom d'aquest usuari per fer que en els propers accessos es carreguin per defecte els Settings d'aquest usuari.
4. Es mostra un missatge a l'usuari de que s'han carregat els Settings correctament, en el cas que es produeixi un error es mostra un missatge d'error.

4.5.4 Comunicació entre formulari Subscripció sota demanda i el servidor

En aquest formulari s'ha utilitzat la llibreria Select2 en els desplegables *Device, Property* i *Fields*. Pels dos primers desplegables, cal establir una comunicació amb el servidor per poder carregar i actualitzar els seus continguts. A continuació s'explica com es realitza aquest procés en el costat del client.

Quan l'usuari clica sobre el desplegable per seleccionar un dispositiu, el desplegable *Device,* apareix un camp de text demanant a l'usuari que introdueixi com a mínim quatre caràcters del nom del dispositiu. A partir del quart caràcter, cada vegada que s'introdueix un nou caràcter s'estableix una comunicació asíncrona amb el servidor via AJAX i se li envia un missatge indicant que l'acció que es desitja realitzar és *getDevicesByName* i el text introduït. Quan aquesta comunicació rep una resposta del servidor, es mostra a l'usuari un llistat amb els noms dels dispositius rebuts que el seu nom conté el text enviat. Si l'usuari selecciona un d'aquests dispositius, s'estableix una nova comunicació asíncrona amb el servidor via AJAX i se li envia un missatge indicant que

l'acció que es desitja realitzar és *getDeviceProperties* juntament amb el nom dels dispositiu seleccionat, per rebre el llistat de propietats d'aquest dispositiu. Tan aviat com aquesta comunicació rep un missatge satisfactori del servidor, es substitueix el contingut del desplegable *Property* pel llistat de propietats rebudes del servidor i es provoca l'esdeveniment *change* d'aquest desplegable. Si es produeix un error es mostrarà un missatge informant que s'ha produït un problema al carregar les propietats.

En el desplegable *Property* l'esdeveniment *change* és el mateix que es crida quan un usuari selecciona una propietat del desplegable. Quan això passa s'estableix una comunicació asíncrona amb el servidor via AJAX i se li envia un missatge indicant que l'acció que es desitja realitzar és *getPropertyFields* juntament amb el nom del dispositiu i propietat seleccionats, amb la intenció de rebre el llistat de camps que formen part de la propietat per al dispositiu indicat. Quan es rep un missatge satisfactori del servidor es substitueix el contingut del desplegable *Fields* amb el nom dels camps. Si es produeix un error es mostrarà un missatge informant que s'ha produït un problema al carregar els camps.

4.5.5 Reconnexió amb el servidor

En el moment en el que es tanca la comunicació via WebSocket amb el servidor, en el costat client es crida automàticament al mètode *onclose()* de la instància de *WebSocketClient* que havia creat la comunicació persistent amb el servidor. Quan es produeix aquesta crida, el que es fa és mirar sí l'esdeveniment que ha generat aquesta crida té el valor 1000, això vol dir que ha estat un tancament normal i no s'ha de fer res, en cas contrari significa que ha s'ha produït un tancament anormal de la comunicació i a les hores es crida al mètode *reconnect()* de la instància de *WebSocketClient.*

Dins del mètode ***reconnect()*** el procediment que es segueix és:

1. Es mira si el navegador sí el navegador esta connectat a la xarxa, en cas afirmatiu s'informa que hi ha hagut un problema de comunicació amb el servidor i que s'està intentant recuperar la comunicació amb aquest i en cas contrari informa a l'usuari de que torni a connectar el dispositiu a la xarxa.
2. S'assigna a la propietat *reconnected* de l'objecte el valor *TRUE* per indicar que s'està realitzat una reconnexió.
3. És genera un setTimeout que passats X segons crida al mètode "*open()"* per intentar crear la connexió amb el servidor. En aquest cas poden passar dues coses:
    a. *La creació de la connexió falla*, es torna a cridar de manera automàtica al mètode *onclose(),*
    b. *La creació de la connexió té èxit*, es crida al mètode *onOpen()* que mira si s'està realitzant una reconnexió, és a dir, que el valor de la propietat *reconnected* és TRUE. Si és així es crida al mètode *getSubscriptionsToReconnect()* de la classe *Controller.* Aquest mètode tornar la informació de totes les subscripcions i camps que estaven actius

abans de perdre la comunicació amb el servidor. Acte seguit se li envia un missatge especificant que l'acció que es desitja realitzar és *reconnect* i la informació que ha retornat el mètode *getSubscriptionsToReconnect().*

4. En aquest punt es tornen a rebre dades i s'actualitza el contingut dels blocs amb aquestes dades.

4.6 Arquitectura de la capa Back-end

4.6.1 Comunicacions via WebSocket

Pràcticament la totalitat de funcionalitats que ofereix aquesta pàgina web es realitzen mitjançant una comunicació persistent via WebSockets. La classe del servidor que s'encarrega de gestionar aquestes comunicacions persistents és *WebSocketServer.* Quan es rep un missatge via WeSocket el que es fa és cridar al mètode *onMessage(),* d'aquesta classe, que rep com a paràmetres una cadena de text amb el missatge rebut i un objecte de tipus *Session* que representa una conversa entre dos *web socket endpoints* permetent enviar missatges des d'un punt final a un altre.

El mètode *onMessage()* s'encarrega de transformar aquest missatge, que ha de tenir format JSON, en una instància de la classe *BrowserMessage* mitjançant *gson.fromJson()*. Un cop s'ha creat la instància d'aquesta classe, es mira el tipus de missatge que s'ha rebut cridant al mètode *getAction()* i segons el tipus de missatge es realitzen unes accions o unes altres. El tipus de missatges que podem rebre són:

1. **subscribe**, per realitzar una subscripció sota demanda.
2. **subscribe-setting**, per realitzar una subscripció per Setting.
3. **reconnect**, per tornar a crear les subscripcions que siguin necessàries en el cas que s'hagués perdut la connexió amb el client.
4. **unsubscribe**, per cancel·lar una subscripció.
5. **unsubscribe-setting**, per cancel·lar una subscripció realitzada per Setting.
6. **unsubscribe-all**, per cancel·lar totes les subscripcions d'una sessió.
7. **add-session**, realitza les configuracions necessàries, utilitzant la sessió rebuda, perquè un client comenci a rebre les dades del dispositiu al que fa referencia la subscripció.
8. **add-session-multi-device**, realitza les configuracions necessàries utilitzant la sessió rebuda perquè un client comenci a rebre les dades de múltiples dispositius quan s'ha realitzar una subscripció que fa referencia a més d'un dispositiu, això només és possible en subscripcions realitzades per Setting.
9. **quit**, aquest tipus de missatge simplement és per evitar l'error que produeix el servidor Tomcat quan un client tanca el navegador tenint una o més subscripcions actives.

A continuació s'explica amb més detall el passos que es realitzen al rebre un missatge d'aquests tipus.

Subscribe

Aquest tipus de missatge es rep quan és vol crear una subscripció utilitzant el formulari "*Subscription on-demand*" de la pàgina web. El que es fa és cridar al mètode *subscribe()* de la classe *FesaConnectorController* enviant-li com a paràmetre la instància de *Session*, una instància de *FesaConnectorInfo,* i en el cas de que es rebin camps, també s'envia el llistat de camps rebuts.

La classe *FesaConnectorInfo* serveix per emmagatzemar la informació principal del connector que crearà la subscripció amb un dispositiu *FESA.* El constructor d'aquesta classe espera rebre el nom del connector, que vindria a ser l'identificador de la subscripció, el nom del dispositiu, la propietat i el nom del cicle. En el cas que no es conegui el nom del connector o el cicle, es pot enviar una cadena de text buida, en aquest cas el constructor crearà un identificador únic per aquest connector (subscripció) a partir de la informació facilitada.

En el mètode ***subscribe()*** es crida al mètode *subscriptionOnDemand()* enviant-li com a paràmetre la sessió, la instància de *FesaConnectorInfo* i en el cas de rebre un llistat de camps també el llistat de camps. Si la crida d'aquest mètode retorna TRUE, vol dir que la subscripció s'ha realitzat correctament i s'envia un missatge a la sessió (el client que ha realitzat la sol·licitud) un missatge amb tota la informació de la subscripció, com el identificador, el nom del dispositiu, el nom de la propietat, la informació sobre els tipus de dada de cada un dels camps de la propietat, etc.

En el mètode ***subscriptionOnDemand()*** és realitzen els passos següents:

1. Es crida al mètode *addSession()* enviant-li com a paràmetre la instància de *Session*. Aquí és mira si ja existeix dins del vector *sessionMap,* de tipus *Map<String, SessionScheduler>*, alguna clau que coincideixi amb l'identificador de la instància *Session,* en cas afirmatiu retorna l'objecte que es troba en aquesta clau, en cas contrari, crea una instància de *SessionScheduler* enviant-li la sessió com a paràmetre i l'afegeix a *sessionMap* utilitzant com a clau l'identificador de la sessió i retorna aquesta instància creada.

   Les instàncies de *SessionScheduler* actuen com intermediàries de la sessió a la que fan referencia guardant les subscripcions que ha sol·licitat aquesta sessió, juntament amb els camps dels que vol rebre informació, els timestamp en els que cada subscripció va rebre noves dades, i utilitza aquesta informació per comprovar si les subscripcions han rebut noves dades in enviar a la sessió un missatge amb totes les dades noves.
2. Es crida al mètode *addConnector()* enviant-li com a paràmetre la instància de *FesaConnectorInfo.* Aquí es mira si ja existeix dins del vector *connMap,* de tipus *Map<String, FesaConnector>*, una

clau que coincideixi amb l'identificador d'aquesta instància, en cas afirmatiu, retorna l'objecte que es troba en aquesta clau, en cas contrari, es crea una instància de *FesaConnector,* enviant-li com a paràmetre l'objecte de tipus *FesaConnectorInfor,* s'afegeix aquesta instància al vector *connMap* utilitzant el seu identificador com a clau i acte seguit es retorna aquesta instància.

Les instancies de la classe *FesaConnector* són les que realment creen la subscripció a un dispositiu, que es troba en un servidor FESA, mitjançant la llibreria *JapcCommunicationCreatorNoINCA* creada pel CERN. Cada instància de *FesaConnector* esta relacionada amb un dispositiu i una propietat, de manera que només es crearan noves instancies si es sol·licita una subscripció a un dispositiu diferent o al mateix dispositiu però a una propietat diferent. El codi per crear la subscripció es pot consultar a l'apartat *5.1.1* .

3. S'espera a que la instància de *FesaConnector* rebi dades i a partir de la informació rebuda defineixi tota la informació necessària sobre els camps pertinent a la subscripció, com el nom del camps, el seu tipus i en el cas de que siguin vectors, la informació sobre les seves dimensions.
4. Un cop s'ha definit la informació dels camps de la subscripció es mira si a les dades rebudes en el missatge del client no s'havia especificat cap camp, en cas afirmatiu, es crida al mètode *addFields()* de la instància de *SessionScheduler* enviant-li l'identificador de la subscripció i un vector amb tots els noms dels camps que s'han definit dins de *FesaConnector.* La instància de *SessionScheduler* guardarà aquesta informació per saber quins camps de la subscripció especificada ha de tenir controlats per enviar-los a la sessió a la que fa referencia. En el cas de que sí s'haguessin especificat alguns camps, primer es mira si aquests existeixen a la instància de *FesaConnector*, si existeixen es crida al mètode *addFields()* de *SessionScheduler* enviant-li els camps especificats.
5. En el cas que tot hagi anat bé, retorna TRUE, en cas contrari s'elimina tota la informació generada, s'envia un missatge informant de l'error que ha succeït i retorna FALSE.

subscribe-setting

Aquest tipus de missatge es rep quan és vol crear una subscripció a partir d'un *Setting*. Al igual que en el cas anterior es crida al mètode *subscribe()* de la clase *FesaConnectorController* però en aquest cas només se li envia la instància de *Session* i l'identificador del Setting seleccionat.

El mètode ***subscribe()*** el que fa és crear una instància de *SubscriptionSettingModel,* aquesta classe ens permet accedir a tota la informació dels Settings que es troben guardats en un fitxer JSON. A partir d'aquesta instància es recupera la informació que pertany al

Setting amb el mateix identificador que el que s'ha rebut en una instància de *SubscriptionSubSettingData*. En el cas de que no existeixi cap Setting amb aquest identificador o es produeixi un error, s'enviarà un missatge informatiu a la sessió que ha realitzat la sol·licitud.

Si existeix un Setting amb l'identificador que s'ha rebut, a les hores es mira si aquest Setting ha de cridar altres Settings, en cas afirmatiu, crea un vector de tipus cadena amb tots els identificadors dels Settings que ha d'executar. En cas contrari, es crea un altre vector de tipus cadena que només contindrà l'identificador d'aquests Setting. Acte seguit es crida al mètode *subscriptionOnMultiSetting()* perquè s'encarregui de crear totes les subscripcions que siguin necessàries, per que ho pugui fer se li envia com a paràmetre la instància de *Session,* el vector amb els identificadors de les subscripcions i la instància de *SubscriptionSettingModel*.

El mètode ***subscriptionOnMultiSetting()*** recorre el vector que conté els identificadors dels Settings i per a cada identificador realitza les següents accions:

1. Recupera la informació del Setting que el seu identificador coincideixi amb l'indicat.
2. Mira si aquest Setting ha cridar altres Settings, si és així es crida a si mateix enviant-se el llistat de Settings que ha de cridar.
3. Si no ha de cridar altres Settings mira si en el Setting s'ha especificat el nom d'una classe i si conté una consulta, això vol dir que aquest Setting fa referencia a múltiples dispositius que compleixin la consulta especificada. En aquest cas es crida al mètode *subscriptionOnSettingMultiDevice()* enviant-li com a paràmetre la instància de *Session* i la informació del Setting per crear les subscripcions per als diferents dispositius. Més endavant, en aquest mateix punt, s'explica amb detall aquest mètode.
4. En el cas de que no s'hagi realitzat els punts 2 i 3, s'agafa el nom del dispositiu per defecte del Setting. Acte seguit es mira si s'ha definit una classe o si s'ha especificat alguna consulta, en cas afirmatiu, es busquen els dispositius que compleixin la condició o formin part d'aquesta classe i es substitueix el nom del dispositiu pel nom del primer dispositiu dels que s'han trobat.

   Es crea una instància de *FesaConnectorInfo* enviant-li el nom del dispositiu i el nom de la propietat i del cicle indicats en el Setting. Acte seguit, es crida al mètode *subscriptionOnSetting()* perquè creï la subscripció enviant-lo com a paràmetre la instància de *Session*, la instància de *FesaConnectorInfo* i la informació del Setting.

El procés que es realitza en el mètode ***subscriptionOnSetting()*** és pràcticament idèntic al del mètode *subscriptionOnDemand():*

1. Es crida al mètode *addSession()*.
2. Es crida al mètode *addConnector()*.

3. S'espera a que la instància de *FesaConnector* que ha retornat el mètode *addConnector()* tingui els camps definits.
4. S'afegeixen els camps a la instància de *SessionScheduler*. Si en el Setting no s'ha especificat cap camp, s'afegeixen tots els camps que s'han definit la instància de *FesaConnector.* En el cas de que sí s'hagin especificat alguns camps en el Setting es mira si cada un d'aquests camps estan definits dins de la instància de *FesaConnector*, si ho estan, s'afegeixen, en cas contrari, s'ignoren.
5. Si tot ha anat bé, retorna TRUE, en cas contrari s'elimina tota la informació generada, s'envia un missatge informant de l'error que ha succeït i retorna FALSE.

El mètode **subscriptionOnSettingMultiDevice*()*** es segueix el següent procediment:

1. Es crida al mètode *addSession()*.
2. Es defineixen el vector *devicesToSubscribe* de tipus *ArrayList<String>* que contindrà el llistat de tots els dispositius als que tenim que crear una subscripció.
3. Es defineixen el vector *devicesList* de tipus *ArrayList<String>* que mitjançant el mètode *getDevices()* de la classe *FesaDeviceInfo* ens retorna el llistat de tots els dispositius que pertanyen a una classe especifica i compleixen la consulta indicada en el Setting. El codi del mètode *getDevices()* es troba explicat a l'apartat *5.1.3* .
4. Es defineixen el vector *connectors* per guardar les instancies de tipus *FesaConnector* que calgui crear pels diferents dispositius. El tipus d'aquest vector és *ArrayList<FesaConnector>*.
5. Es defineixen el vector *devicesKey* de tipus *ArrayList<String>* per guardar els identificadors de les subscripcions creades.
6. Es recorren tots els nos dels dispositius que conté el vector *devicesList* i per a cada un es fa el següent:
    1. Es crea una instància de *FesaConnectorInfo* enviant-li el nom del dispositiu i el nom de la propietat i del cicle especificats en el Setting.
    2. Es crida al mètode *addConnector()* enviant-li la instància de *FesaConnectorInfo* creada i es recull la instància de *FesaConnector* que retorna.
    3. Es crida al mètode *existProperty()* de la instància *FesaConnector,* mirant si aquest ens retorna el valor TRUE, per assegurar-nos de que realment existeix la propietat indicada en el moment de crear la subscripció. Això és necessari ja que per un defecte en el codi de la llibreria del CERN utilitzada per crear les subscripcions, permet crear una subscripció a un dispositiu per una propietat especifica, encara que aquesta propietat no existeixi en el dispositiu sense provocar cap error no excepció. L'error apareix en el moment que en teoria rebem dades i volem intentar accedir a aquestes.

Si la crida a *existProperty()* retorna TRUE, afegim la instància de *FesaConnector* al vector *connectors,* afegim l'identificador d'aquesta instància al vector *devicesKey* i per últim, afegim el nom del dispositiu al vector *devicesToSubscribe.* En el cas que ens retorni FALSE cridem al mètode *removeConnector()* perquè cancel·li i elimini aquesta subscripció.

4. Es recorre el vector connectors i per a cada instància de *FesaConnector* que conté es crida al mètode *waitToFieldsDefined()*, per esperar a que la informació dels camps de la subscripció estiguin definits.
5. En aquest punt agafem la primera instància de *FesaConnector* que conté el vector *connectors* i guarden en el vector *fieldsDimensions,* de tipus *Map<String, int[]>* la informació de les dimensions de tots els camps que són de tipus vector. També guardem en el vector *fieldsName,* de tipus *String[]*, el nom de tots els camps que conté aquesta subscripció.
6. Acte seguit recorrem cada un dels camps que conté el vector *fieldsName* i es mira si aquest camp existeix en tots els dispositius i en el cas de que sigui de tipus array es mira si el vector no és més gran d'una dimensió i si en tots tenen la mateixa quantitat de dades. Sí això es compleix s'afegeix aquest camp a la instància de *SessionScheduler* per a cada una de les instàncies *FesaConnector* que conté el vector *connectors.*
7. Per últim es crea i s'envia un missatge a la Sessió amb la informació del Setting juntament amb tota la informació de les diferents subscripcions a les que fa referencia, de manera que, el navegador del client pugui utilitzar aquesta informació per crear tots els blocs que siguin necessàries per mostrar la informació dels diferents camps i subscripcions generades.

reconnect

Aquest missatge es rep quan una sessió a perdut la connexió amb el servidor, ja sigui perquè el client s'ha quedat sense connexió a la xarxa o perquè el servidor ha estat indisponible per algun motiu. L'objectiu d'aquest missatge és que la sessió/client torni a rebre la informació dels camps de les diferents subscripcions que tenia actives en el moment de perdre la connexió. Per aconseguir-ho es crida al mètode *restartSubscriptions()* de *FesaConnectorControler* enviant-li com a paràmetre la instància de *Session* rebuda, i la informació de les subscripcions que conté el missatge.

En el mètode ***restartSubscriptions()*** la informació que es rep, com a paràmetre, del missatge sobre les subscripcions és un vector de tipus *HashSet<BrowserSubscriptionField>* on cada una de les instancies de tipus *BrowserSubscriptionField* conté el nom d'un dispositiu i propietat

específics, el nom del cicle que es vol consultar, en el cas de que no s'hagi especificat, aquesta propietat estarà buida i els camps de la propietat dels quals es desitja rebre informació. Dins d'aquest mètode es recorren totes les instancies de tipus *BrowserSubscriptionField* que conté aquest vector i en cada iteració el que es fa és:

1. Es crea una instància de *FesaConnectorInfo* a partir de la informació que conté la instància de *BrowserSubscriptionField.*
2. Es crida al mètode *subscriptionOnDemand()*, explicant anteriorment, enviant-li com a paràmetres la instància de *Session,* la instància de *FesaConnectorInfo* i el llistat de camps que conté la instància de *BrowserSubscriptionField.*
3. Si *subscriptionOnDemand()* retorna TRUE, es crida al mètode *addSessionToConnector()* enviant-li com a paràmetre la instància de *Session* i la instància de *FesaConnectorInfo* amb la intenció d'informar a la subscripció, o millor dit la instància de *FesaConnector* d'aquesta subscripció, que hi ha alguna sessió interessada en rebre dades d'alguns dels seus camps . En cas contrari s'assigna el valor FALSE a la variable *success.*
4. Si el valor de la variable *success* és TRUE, s'envia un missatge a la sessió informant que la reconnexió s'ha realitzat amb èxit, en cas contrari, s'envia un missatge de que la reconnexió s'ha realitzat amb alguns problemes.

El mètode ***addSessionToConnector()*** com ja s'ha comentat, té la funció d'informar a la instància de *FesaConnector,* relacionada amb la informació de la subscripció que rep com a paràmetre, que hi ha alguna sessió interessada en rebre els seus camps. Això és necessari perquè quan no hi ha cap sessió interessada en una subscripció aquesta guarda el timestamp del moment en que això ha succeït i quan ha passat el temps indicat a la propietat *secondsToUnsubscribe* la subscripció és cancel·lada i eliminada per optimitzar recursos. Per tant, quan hi ha una sessió interessada en aquesta subscripció, el que es fa és mirar si aquesta propietat té algun valor, en vas afirmatiu s'elimina i queda buida.

L'altre funcionalitat que té aquest mètode és cridar al mètode *startDataControl()* de la instància *SessionScheduler* corresponent a la sessió rebuda, d'aquesta manera s'està dient a la instància de *SessionScheduler* que pot començar a controlar si hi ha noves dades en els camps de les subscripcions en les que esta interessada i enviar-les a la sessió. El funcionament d'aquest mètode s'explica amb més detall a l'explicació del tipus de missatge *add-session.*

A diferencia dels tipus de missatges *subscribe* i *subscribe-setting,* aquí es crea la subscripció, si és necessari, i s'avisa a la instància de *SessionScheduler* que pot començar a enviar les noves dades dels camps que es vagin reben de les subscripcions, mentre que, en els altres dos, el que es fa és enviar la informació de la subscripció i dels seus camps a la sessió. Això es deu a que en aquests dos missatges el navegador del client ha de crear els blocs HTML i els objectes per els diferents camps i per fer-ho necessita saber tota aquesta informació,

però en el cas del missatge de tipus *reconnect* aquesta feina ja esta feta i els objectes ja estan creats, de manera que, només es necessita tornar a rebre dades.

unsubscribe

Aquest missatge serveix per cancel·lar la recepció de dades de tots els camps d'una subscripció o només d'uns camps concrets, en el cas de que el missatge s'especifiquin aquests camps. Per fer-ho es crida al mètode *unSubscribe()* de la classe *FesaConnectorControler* enviant-li com a paràmetre la instància de *Session* rebuda i una instància de *FesaConnectorInfo* creada amb la informació rebuda del missatge, si en el missatge s'han rebut camps, a les hores enviem el llistat de camps.

En el mètode ***unSubscribe()*** si no rep el llistat de camps com a paràmetre el que fa és:

1. Crida al mètode *removeConnectorFieldsFromSession()* enviant-li la instància de *Session* i la instància de *FesaConnectorInfo.* Aquest mètode el que fa és recuperar la instància de *SessionScheduler* equivalent a la sessió rebuda del vector *sessionMap* i crida al mètode *removeFields()* d'aquesta instància enviant-li l'identificador de la subscripció, de manera que eliminarà tota la informació existent dels camps d'aquesta subscripció en el vector *fields* de *SessionScheduler*.
2. Acte seguit es crida al mètode *existConnectorInSessions(),* enviant-li com a paràmetre l'identificador de la subscripció. Aquest mètode el que fa és recórrer totes les instancies *SessionScheduler* que conté el vector *sessionMap* i mira si alguna sessió encara vol rebre dades d'aquesta subscripció, en el cas de que no hi hagi cap sessió interessada, es recupera la instància de *FesaConnector* equivalent a aquesta subscripció que es troba guardada en el vector *connMap* i es crida al mètode *setNoSessions()* enviant-li la data actual, de manera que quan hagin passat X segons sense que hi hagi cap sessió interessada en la subscripció aquesta serà cancel·lada i eliminada.

En el mètode ***unSubscribe()*** si rep el llistat de camps com a paràmetre el que fa és:

1. S'eliminen els camps de la instància de *SessionScheduler* corresponent a la sessió rebuda.
2. Un cop eliminats els camps que s'han rebut com a paràmetre, es mira si encara queda algun camps per rebre informació per aquesta subscripció dins de la instància de *SessionScheduler* cridant al mètode *isConnectorEmpty(),* en el cas de que ja no hi hagi cap camp, es realitzen els passos 1 i 2 indicats al mètode *unSubscribe()* anterior.

unsubscribe-all

La intenció d'aquest missatge és eliminar totes les subscripcions actives d'una sessió abans de crear-ne una nova. Per fer-ho el que fa és cridat al mètode *unSubscribeAllFromSession()* de la classe *FesaConnectorControler,* enviant-li com a paràmetre la instància de *Session* rebuda.

En el mètode ***unSubscribeAllFromSession()*** el que es fa és recuperar la instància de tipus *SessionScheduler* que fa referencia a la sessió rebuda com a paràmetre i es crida al mètode *getConnectors()* per recuperar i recórrer tots els identificadors de les subscripcions que té actives. En cada iteració es crida al mètode *unSubscribe(),* explicat anteriorment, enviant-li només la instància de *Session* i la instància *FesaConnector* equivalent al identificador de la subscripció.

add-session

Aquest missatge es rep desprès d'haver rebut un missatge *subscribe* o *subscribe-setting*, quan es rep un d'aquest dos missatges, es creen les subscripcions pertinents i s'envia a la sessió un missatge amb tota la informació de la subscripció i dels camps d'aquesta, de manera que el navegador aprofita aquesta informació per crear els elements que siguin necessaris i els diferents blocs HTML que mostraran la informació dels camps. Un cop fet això envia aquest missatge per avisar al servidor que ja esta preparat per començar a rebre dades. Per fer-ho es crida al mètode *addSessionToConnector()* de la classe *FesaConnectorControler*, explicat en el tipus de missatge *reconnect,* enviant-li com a paràmetre la instància de *Session* rebuda, la instància de *FesaConnectorInfo* creada amb la informació rebuda del missatge i el divisor del rellotge, rebut també en el missatge.

Quan es crida al mètode *addSessionToConnector()* enviant-li com a paràmetre el divisor del rellotge, al final de tot, aquest crida al mètode *startDataControl()* de la instància de *SessionScheduler* corresponent a la sessió rebuda, enviant-li aquest divisor de rellotge.

En el mètode ***startDataControl()*** s'utilitza el divisor del rellotge que es rep com a paràmetre, en el cas de que no és rebi s'utilitza el valor de la propietat de clockDivider de l'objecte, per calcular la freqüència de temps en la que la instància de *SessionScheduler* mirarà si s'han rebut noves dades. Aquest divisor tenir un valor enter entre 1, que equival a 10Hz i 10 que equival a 1Hz. Un cop s'ha calculat la freqüència es crea un thread que crida al mètode *sendRecentData(),* encarregat de consular i enviar les dades noves, amb la freqüència indicada. En el cas de que ja existeixi aquest thread es modificarà la freqüència d'execució per la nova. Aquest mètode es troba explicat amb més detall en el punt 4.6.3.

add-session-multi-device

Al igual que el missatge anterior, aquest missatge es rep un cop el navegador ha acabat de crear els elements que siguin necessaris i els

diferents blocs HTML que mostraran la informació dels camps, per avisar que ja està preparat per rebre les dades de la subscripció, però en aquest cas la subscripció s'ha realitzat via Setting i fa referencia a més d'un dispositiu, per tant, caldrà indicar a la instància de *SessionScheduler* corresponent que comenci a enviar les dades de tots aquests dispositius. Per fer-ho el que es fa és cridar al mètode *addSessionToMultipleConnector()* de la classe *FesaConnectorControler,* enviant-li com a paràmetre la instància de *Session,* el llistat amb el nom dels dispositius, el nom de la propietat i el divisor del rellotge, aquests tres últims paràmetres s'obtenen del contingut del missatge.

Dins del mètode ***addSessionToMultipleConnector()*** el que es fa és recórrer els noms dels diferents dispositius rebuts i per a cada un es crida al mètode *addSessionToConnector(),* explicat en el missatge anterior, enviant-li com a paràmetre la instància de *Session* rebuda, una instància de *FesaConnectorInfo* creada amb el nom del dispositiu i la propietat i el divisor del rellotge rebuts com a paràmetre.

quit

Aquest missatge es rep abans de que es tanqui la connexió permanent amb el servidor quan l'usuari tanca el navegador i serveix per evitar un error que es produeix en el servidor Tomcat, aquest error es troba explicat a l'apartat *5.4.2*. Quan es rep aquest missatge es crida al mètode *removeSessionCompletely()* de la classe *FesaConnectorControler,* enviant-li com a paràmetre la instància de *Session* rebuda.

En el mètode ***removeSessionCompletely()*** el que es fa és cridar al mètode *removeSession()* on s'elimina de *FesaConnectorControler* tota la informació de les subscripcions d'aquesta sessió.

4.6.2 Recepció de dades d'una subscripció

La recepció de dades d'una subscripció es realitza en les instancies de tipus *FesaConnector.* Quan un dispositiu rep noves dades pels camps, de la propietat a la que fa referencia la subscripció al dispositiu, crida automàticament al mètode *receiveUpdate()* de la instància *FesaConnector* corresponent i que rep com a paràmetres una instància de *UpdateInfo*, classe creada pel CERN i que conté informació relativa a la subscripció i una instància de *DataMap.*

La classe ***DataMap*** és una classe creada pel CERN i conté tota la informació dels camps de la propietat d'una subscripció que han estat actualitzats i múltiples mètodes per accedir a aquesta informació.

Dins de ***receiveUpdate()*** es mira si no hi ha cap sessió interessada en aquesta subscripció i quan temps fa d'això, en el cas que aquest temps sigui superior al indicat a la propietat *secondsToUnsubscribe* es crida al mètode *unSubscribe()* per eliminar aquesta subscripció. En cas contrari, es crida al mètode *setData()* de la classe *FesaDataMapContainer* perquè

emmagatzemi aquestes dades, enviant-li com a paràmetres l'identificador de la subscripció i la instància de *DataMap.*

Dins del mètode ***setData()*** es guarda en el vector *connectorsDataMap*, de tipus *Map<String, FesaDataMap>,* una clau que serà l'identificador de la subscripció i estarà relacionat amb una instància de tipus *FesaDataMap* que conté la instància de tipus *DataMap* rebuda i la data en la que s'ha creat l'objecte.

4.6.3 Enviament de dades d'una subscripció

L'enviament de dades d'una subscripció activa es realitza des de el mètode *sendRecentData()* de les instancies de tipus *SessionScheduler.* Com ja s'ha comentat anteriorment, quan una sessió està preparada per rebre dades es crea dins de la instància *SessionScheduler* relacionada amb aquesta sessió un thread que cada cert temps executa aquest mètode.

El mètode ***sendRecentData()*** es defineix un vector, de tipus *ArrayList<FesaDataMessage>*, anomenat *dataMsgList*. Es recorren totes les claus del vector *fields,* de tipus *Map<String, Set<String>>*, on cada clau es correspon al identificador d'una subscripció i esta relacionada amb el llistat de camps d'aquesta subscripció dels que es vol rebre les dades i per a cada clau es fa el següent:

1. Es crida al mètode *getData()* de la classe *FesaDataMapContainer*, enviant-li com a paràmetre la clau de la iteració actual (l'identificador de la subscripció), i retorna la instància de tipus *FesaDataMap* que es correspon amb aquesta clau.
2. Es compara la data de l'ultima vegada que la instància de *SessionScheduler* va enviar dades per aquesta clau amb la data de creació de la instància *FesaDataMap,* si la data d'aquesta última es anterior a la data emmagatzemada a la instància de *SessionScheduler* no es fa res, en cas contrari, vol dir que cal enviar dades d'aquesta subscripció i es continua amb l'execució.
3. Es recuperen el llistat de camps relacionats amb l'identificador que conté el vector *fields* i es transforma el vector obtingut del tipus *Set<String>* a un vector del tipus *String[]* anomenat *list_fields_reduced.*
4. Si la longitud del vector *list_fields_reduced* és igual a la dels camps que conté la instància *DataMap* que es troba dins de la instància *FesaDataMap* a les hores es crida al mètode *createDataMsg()* enviant-li com a paràmetre la clau de la iteració actual i la instància del *DataMap que conté la instància FesaDataMap.* Aquest mètode ens retorna una instància de tipus *FesaDataMessage* i l'afegim al vector *dataMsgList.*

   En cas contrari, es crea una nova instància de *DataMap* que contindrà només la informació dels camps que volem enviar. Per fer això recorrem cada un dels camps que es troba en el vector *list_fields_reduced* i en cada iteració afegim en el nou vector de tipus *DataMap* la informació d'aquest camp que es troba dins de

la instància *DataMap* que conté la instància de *FesaDataMap.* Un cop hem recorregut tots els camps i hem afegit la informació corresponent al nou vector de tipus *DataMap*, es crida al mètode *createDataMsg()*¸enviant-li com a paràmetre la clau de la iteració actual de les subscripcions i la nova instància de tipus *DataMap.* Igual que abans aquest mètode ens retorna una instància de tipus *FesaDataMessage* i l'afegim al vector *dataMsgList.*

Un cop s'ha acabat de recórrer totes les claus del vector *fields* es mira si el vector *dataMsgList* no esta buit, en cas afirmatiu es crida al mètode *sendText(),* enviant-li com a paràmetre una instància de *FesaMultiDataMessage*.

En el mètode ***createDataMsg()*** s'encarrega de crear i retornar una instància de tipus *FesaDataMessage,* amb la informació dels camps d'una subscripció en concret. Per fer-ho primer crea un vector de tipus *List<FesaDataValue>* anomenat *fields*, acte seguit, recull el llistat amb el nom dels camps relacionats amb l'identificador de la subscripció que rep com a paràmetre i el transforma en un vector de tipus *String[]*. Es recorre els noms dels camps que conté el vector de tipus *String[]* i per a cada nom es mira si aquest existeix dins de la instància *DataMap.* Si existeix es crea una instància de tipus *FesaDataValue* i s'afegeix al vector *fields.* Un cop s'han recorregut tots els noms dels camps es crea i retorna la instància de tipus *FesaDataMessage* enviant al constructor el vector *fields*.

La classe ***FesaDataValue****,* serveix per crear l'estructura de la informació d'un camp concret, que ha estat actualitzat, per poder enviar-lo en un missatge al navegador. Per crear una instància d'aquesta classe s'envia al constructor el nom del camp, el valor d'aquest camp, com una cadena de text on s'utilitza el mètode *getValueAsString()* de la classe *DataMap,* i en el cas de que sigui de tipus vector s'envia la informació de les dimensions d'aquest camp, utilitzant el mètode *getArrayDimensions()* de la classe *DataMap*.

La classe ***FesaDataMessage*** serveix per crear l'estructura de la informació de tots els camps d'una subscripció que han estat actualitzats per poder enviar les instàncies d'aquest classe com un missatge al navegador. Per crear una instància d'aquesta classe s'envia al constructor el identificador de la subscripció i un vector de tipus *List<FesaDataValue>* amb la informació i valors de tots els camps de la subscripció que han estat actualitzats. Dins el constructor s'especifica que el missatge és de tipus "DATA", això indica al navegador que només conté les dades dels camps d'una subscripció.

La classe ***FesaMultiDataMessage*** crea l'estructura que s'enviarà com un missatge al navegador per enviar de manera simultània tots els camps que han estat actualitzats de diferents subscripcions. Per crear una instància d'aquesta classe s'envia al constructor un vector de tipus *ArrayList<FesaDataMessage>,* on en cada posició del vector i ha uns

instància de *FesaDataMessage.* Dins del constructor i es defineix que el missatge és de tipus "*DATA_MULTIPLE",* això indica al navegador que el missatge conté les dades de camps de diferents subscripcions.

4.6.4 Comunicacions via AJAX

Algunes de les comunicacions que es realitzen a la pàgina web són comunicacions AJAX. La classe del servidor que s'encarrega de gestionar aquestes comunicacions és *AjaxController* que hereta de la classe *HttpServlet* del paquet *javax.servlet.http.HttpServlet* i es defineix que el patró de la URL per accedir a aquest és "*AjaxController"*.

En aquesta classe es sobreescriuen els mètodes *doGet()* per les comunicacions de tipus GET i el mètode *doPost()* per les comunicacions de tipus POST. El codi principal d'aquesta classe es troba en *doPost(),* mentre que *doGet()* crida al mètode anterior enviant-li tots els paràmetres que ha rebut.
El mètode *doPost()* rep dos paràmetres, el primer és una instància de *HttpServletRequest* que encapsula les dades del client, i el segon de *HttpServletResponse* que encapsula la resposta cap al client. Dins d'aquest mètode es recupera el paràmetre *action* enviat pel client mitjançant el mètode *getParameter()* de la instància *HttpServletRequest* i enviant-li com a paràmetre el nom del paràmetre que volem recollir, com ja s'ha dit, en aquest cas és *action.* Depenent del valor d'aquest paràmetre es realitzaran unes coses o unes altres. El possibles valors que s'espera rebre en aquest paràmetre són:

1. **getSettingsByUser**: serveix per obtenir el llistat de Settings d'un usuari específic.
2. **getDevicesByName**: s'utilitza per consultar els dispositius que contenen part d'una cadena de text.
3. **getDeviceProperties**: serveix per obtenir les propietats d'un dispositiu específic.
4. **getPropertyFields**: serveix per obtenir els camps de la propietat d'un dispositiu específic.

En qualsevol d'aquest tipus de missatges es transforma la informació obtinguda en una cadena de text i es guarda en una variable anomenada *msg* i al final de tot es crida a "*out.print(msg)*" per enviar el missatge al client. A continuació s'explica amb més detall que succeeix quan es rep un d'aquests missatges.

getSettingsByUser

Aquest tipus de missatges es reben quan l'usuari a seleccionat un usuari en el desplegable del formulari "*Load Settings*" i ha clicat sobre el botó "*Load".* Aquest missatge contindrà el nom de l'usuari seleccionat i s'ha d'enviar al client el llistat de seccions i Settings que té aquest usuari. Per fer-ho es segueixen els passos següents:

1. Es recupera el paràmetre *user* utilitzant el mètode *getParameter()* de la instància *HttpServletRequest*.

2. Es crea una instància de *SubscriptionSettingController()* per poder obtenir, a partir del mètode *getModel(), una instància de SubscriptionSettingModel* que ens permet accedir a tota la informació dels Settings que es troben guardats en un fitxer JSON.
3. Es crida al mètode *sortSettings()* de la instància *SubscriptionSettingModel* per ordenar el llistat de Settings.
4. Si el nom no és igual a *NULL* es crida al mètode *getSettingsByUser()* de la instància de *SubscriptionSettingModel,* enviant-li com a paràmetre el nom de l'usuari. Aquest mètode recorre tots els Settings que té emmagatzemats la instància de *SubscriptionSettingModel* i mira si el seu propietari és l'usuari que s'ha rebut com a paràmetre, en cas afirmatiu, s'afegeix en un vector de tipus *ArrayList<SubscriptionSettingData>.* Un cop s'han revisat tots els Settings es retorna aquest vector.
5. Es transforma el vector retornat per *getSettingsByUser()* i es transforma en una cadena de text utilitzant "*gson.toJson()*" i es guarda aquesta cadena de text dins la variable *msg.*

getDevicesByName

Aquest missatge es rep quan l'usuari ha introduït quatre o més caràcters en el cap *Device* del formulari "*Subscription on-demand*" de la pàgina web. L'objectiu d'aquest missatge és obtenir un llistat de tots els dispositius que el seu nom contingui la cadena de text rebuda i enviar-lo al client. Per fer-ho es segueixen els passos següents:

1. Es recupera el paràmetre *deviceName* utilitzant el mètode *getParameter()* de la instància *HttpServletRequest*.
2. Es crida al mètode *getDevices()* de la classe *FesaDeviceInfo,* enviant-li com a paràmetre la cadena de text del paràmetre *deviceName.* Aquest mètode retorna el llistat complet de dispositius que el seu nom conté aquesta cadena de text.
3. Es transforma el llistat retornat per *getDevices()* i es transforma en una cadena de text utilitzant "*gson.toJson()*" i es guarda aquesta cadena de text dins la variable *msg.*

La classe ***FesaDeviceInfo*** és una classe estàtica creada per obtenir informació dels dispositius utilitzant llibreries creades en el CERN. Aquesta classe defineix la propietat *controlDeviceService* que és del tipus *ControlDeviceService* i que permet realitzar diferents consultes per obtenir informació dels dispositius.

En el mètode ***getDevices()*** *de FesaDeviceInfo* ha de rebre com a mínim un paràmetre de tipus cadena amb part del nom, també pot rebre un paràmetre per especificar la classe a la que han de pertànyer aquests dispositius i un valor booleà que indica si cal afegir el caràcter % a la consulta que realitza. El codi d'aquest mètode es troba explicat a l'apartat 5.1.3. El procediment que segueix és:

1. Mira si el nom del dispositiu no conté el caràcter % i s'ha especificat que s'afegeixi, en aquest cas modifica el text del nom del dispositiu afegint al principi i al final aquest caràcter.
2. Es crea una primera part d'una consulta SQL per fer que es compari el nom dels dispositius en majúscules amb la part del nom rebut també en majúscules.
3. Si s'ha especificat el nom d'una classe, s'afegeix a la consulta que el nom de la classe del dispositiu ha de ser igual a aquesta.
4. Es crida al mètode *getControlDevices()*, de la instància *ControlDeviceService* creada pel cern, enviant-li com a paràmetre la consulta creada anteriorment, obtenint un vector de tipus *ControlDevice* amb la informació dels diferents dispositius que cumpleixen aquesta condició.
5. Es recorrent els diferents dispositius del vector de tipus *ControlDevice* guardant el nom de cada un dispositius en un vector de tipus *String.*
6. Retorna el vector amb els noms de tots els dispositius trobats.

getDeviceProperties

Aquest missatge es rep quan l'usuari ha seleccionat un dispositiu del cap *Device* del formulari "*Subscription on-demand*" de la pàgina web. L'objectiu d'aquest missatge és obtenir un llistat de totes les propietats d'aquest dispositiu i enviar-lo al client. Per fer-ho es segueixen els passos següents:

1. Es recupera el paràmetre *deviceName* utilitzant el mètode *getParameter()* de la instància *HttpServletRequest*.
2. Es crida al mètode *getDeviceProperties()* de la classe *FesaDeviceInfo,* enviant-li com a paràmetre la cadena de text del paràmetre *deviceName.* Aquest mètode retorna el llistat complet de les propietats que té aquest dispositiu.
3. Es transforma el llistat retornat per *getDeviceProperties()* i es transforma en una cadena de text utilitzant "*gson.toJson()*" i es guarda aquesta cadena de text dins la variable *msg.*

En el mètode ***getDeviceProperties()*** *de FesaDeviceInfo* rep com a paràmetre el nom del dispositiu que es vol obtenir la informació. El codi d'aquest mètode es troba explicat a l'apartat 5.1.4. El procediment que segueix és:

1. Es crida al mètode *getControlDevice()* de la instància *ControlDeviceService* i se li envia com a paràmetre el nom del dispositiu, retornant una instància de tipus *ControDevice*, aquesta classe creada pel CERN conté la informació del dispositiu indicat.
2. Es crida al mètode *getDevicePropertyNames()* de la instància de tipus *ControDevice* per obtenir vector de tipus *String* amb totes les propietats que té aquest dispositiu.
3. Ordena alfabèticament les propietats del vector obtingut
4. Es retorna el vector que conté el llistat de propietats.

getPropertyFields

Aquest missatge es rep quan l'usuari ha seleccionat una propietat del cap Property del formulari "*Subscription on-demand*" de la pàgina web. L'objectiu d'aquest missatge és obtenir un llistat de tots els camps que conté la propietat d'un dispositiu en concret. Per fer-ho es segueixen els passos següents:

1. Es recupera el paràmetre *deviceName* utilitzant el mètode *getParameter()* de la instància *HttpServletRequest*.
2. Es recupera el paràmetre *propertyName* utilitzant el mètode *getParameter()* de la instància *HttpServletRequest*.
3. Es crida al mètode *getPropertyFields()* de la classe *FesaDeviceInfo,* enviant-li com a paràmetre la cadena de text del paràmetre *deviceName* i la del paràmetre *propertyName.* Aquest mètode retorna el llistat complet de tots els camps a la propietat del dispositiu.
4. Es transforma el llistat retornat per *getPropertyFields()* i es transforma en una cadena de text utilitzant "*gson.toJson()*" i es guarda aquesta cadena de text dins la variable *msg.*

En el mètode ***getPropertyFields()*** *de FesaDeviceInfo* rep com a paràmetre el nom del dispositiu i de la propietat que es vol obtenir la informació. El codi d'aquest mètode es troba explicat a l'apartat 5.1.5. El procediment que segueix és:

1. Es crea una instància de tipus *DevicePropertyService*. Aquesta classe ha estat creada pel CERN
2. Es crida al mètode *getControlDevice()* de la instància *ControlDeviceService* i se li envia com a paràmetre el nom del dispositiu, retornant una instància de tipus *ControDevice*, aquesta classe creada pel CERN conté la informació del dispositiu indicat.
3. Es crida al mètode *getFilterItems()* de la instància de *DevicePropertyService* enviant-li com a paràmetre el nom de la classe del dispositiu dispositiu (que s'obté cridant al mètode *getDeviceClassName()* de la instància *ControDevice*), la versió de la classe del dispositiu (aquesta s'obté cridant al mètode *getDeviceClassVersionString()* de la instància *ControDevice*) i el nom de la propietat. El mètode *getFilterItems()* retorna un vector de tipus *String* amb el nom de tots els camps la propietat.
4. Es recorre el vector amb el nom de tots els camps, i s'elimina d'aquests el començament de la cadena ja que aquest conté també el nom de la propietat i el caràcter #, guardant aquest nom reduït en un altre vector.
5. S'ordena alfabèticament aquest últim vector creat.
6. Es retorna el vector amb el llistat de noms reduït i ordenat.

5. Desenvolupament

5.1 Extractes de codi

5.1.1 Creació d'una subscripció a un dispositiu

El codi següent s'executa en el costat del servidor, forma part de la classe *FesaConnector* i mostra com es genera una subscripció a un dispositiu, que es troba en un servidor FESA, mitjançant la llibreria *JapcCommunicationCreatorNoINCA* creada pel CERN. Cada instància de *FesaConnector* esta relacionada amb un dispositiu i una propietat, de manera que només es crearan noves instancies si es sol·licita una subscripció a un dispositiu diferent o al mateix dispositiu però a una propietat diferent.

.

```
/**
 * Generates the communication and subscription to the FESA device
 *
 * @throws CommunicationException
 */
private void subscribe() throws CommunicationException {
    if(_connector == null) {
        _connector = (cern.bisw.fxexpert.comm.JapcCommunicationNoINCA)
new JapcCommunicationCreatorNoINCA().getCommunication(
                                    _connectorInfo.getDeviceName());
    }

    _subscription = _connector.subscribe(
                        _connectorInfo.getCycleName(),
                        _connectorInfo.getPropertyName(),
                        true,
                        this);
}
```

Un cop s'ha realitzat la subscripció amb el dispositiu, cada vegada que aquest emeti una senyal de que té dades noves es crida de manera automàtica al mètode *receiveUpdate()* explicat en el punt 5.1.2 d'aquest apartat.

5.1.2 Recepció de dades noves d'un dispositiu

El codi següent s'executa en el costat del servidor, forma part de la classe *FesaConnector* i representa el mètode *receiveUpdate()*. Aquest mètode és cridat automàticament quan es reben dades del dispositiu al que s'ha realitzat la subscripció.

```
/**
 * When the subscription data is updated this method is called and
receives this data.
 *
 * The first time it is executed it initializes the information about
the fields of this subscription name and type
 *
 * It is responsible to send the data received to FesaDataMapContainer
class, in case of not having any
 * subscriptions for longer time (indicated in _secondsToUnsubscribe),
the subscription will be deleted and will
```

```
 * stop receiving data.
 *
 * @param info
 * @param dataMap Contains all the data received from the subscription
update
 */
@Override
public void receiveUpdate(UpdateInfo info, DataMap dataMap) {
    if(!mutex) {
        mutex = true;

        if(_noSessions == null) {
            if(_connectorInfo.getFields() == null) {
                _connectorInfo.setFields(dataMap);
            }

            FesaDataMapContainer.setData(this.getInfo().getKey(),
                                         dataMap);
        }
        else
        {
            int seconds = (int)( new Date().getTime() -
                                    _noSessions.getTime() )/1000;

            if(seconds > _secondsToUnsubscribe) {
                unSubscribe();
            }
        }
        mutex = false;
    }
}
```

Aquest mètode el que fa és mirar si la variable *_noSessions* és igual a null, això vol dir que hi ha Sessions interessades en rebre les seves dades, de manera que si és la primera vegada que rep dades, aprofita aquestes dades per inicialitzar la informació corresponent als camps i acte seguit crida a *FesaDataMapContainer.setData()* perquè guardi les dades rebudes d'aquest dispositiu i puguin ser consultades per les sessions. En el cas que *_noSessions* no sigui igual a null, contindrà una data que indicarà la última vegada que una classe va estar interessada en les seves dades, si la diferencia entre aquest temps i el temps actual és més gran que l'indicat a la propietat *_secondsToUnsubscribe*, s'eliminarà la subscripció.

5.1.3 Obtenció llistat de dispositius que el seu nom conté una cadena de text

El codi següent s'executa en el costat del servidor, forma part de la classe *FesaDeviceInfo* i mostra com s'utilitzen les classes *ControlDeviceService* i *ControlDevice,* creades pel CERN, per obtenir un llistat de tots els dispositius que el seu nom conté la cadena de text rebuda com a paràmetre.

```
/**
 * Gets array of device names from a query string. The query string is
part of the device name with % and the class name. If the filter don't
contains a % the query will be done with '%deviceName%'.
 *
```

```
 * @param deviceName - part of the name of the control device
 * @param className - the exact class name of the control device
 * @param addPercentage - if is true and the deviceName don't have %,
it's added at the beginning and at the end of the name
 * @return
 */
public synchronized static  Vector<String> getDevices(String
deviceName, String className, boolean addPercentage) {
    Vector<String> vDevice = new Vector<String>();

    try{
        if(!deviceName.contains("%") && addPercentage)
            deviceName = "%" + deviceName + "%";

        String query = "UPPER(deviceName) like '" +
                       deviceName.toUpperCase() + "'";

        if ( (className != null) && (!className.equals("")) )
            query += " and UPPER(classname)='" +
                         className.toUpperCase() + "'";

        ControlDevice[] devices =
                    controlDeviceService.getControlDevices(
                                                query,
                                                "deviceName");

        for (ControlDevice cd : devices) {
            if(cd.getDeviceName().indexOf("0000000") < 0)
                vDevice.add(cd.getDeviceName());
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return vDevice;
}
```

5.1.4 Obtenció llistat de propietats del dispositiu

El codi següent s'executa en el costat del servidor, forma part de la classe *FesaDeviceInfo* i mostra com s'utilitzen les classes *ControlDeviceService* i *ControlDevice,* creades pel CERN, per obtenir les propietats dels dispositiu que té el mateix nom que el que es rep com a paràmetre.

```
/**
 * Gets array of device properties
 *
 * @param deviceName - the exact name of the control device
 * @return String[] of <i>property</i>
 */
public synchronized static String[] getDeviceProperties(String
deviceName) {
    String[] properties;

    try{
        ControlDevice cd;

        cd = controlDeviceService.getControlDevice(deviceName);
```

```
        properties = cd.getDevicePropertyNames();
        Arrays.sort(properties);

        return properties;

    }catch (Exception ex) {
        ex.printStackTrace();
    }

    return new String[0];
}
```

5.1.5 Obtenció llistat de camps de la propietat

El codi següent s'executa en el costat del servidor, forma part de la classe *FesaDeviceInfo* utilitzant les classes *ServiceLocator, DevicePropertiService, ControlDeviceService* i *ControlDevice,* creades pel CERN, per obtenir el llistats de camps que conté la propietat del dispositiu que rep com a paràmetres.

```
/**
 * Gets array of property fields from a specific device.
 *
 * @param deviceName - the exact name of the control device
 * @param propertyName - the exact name of the device property
 * @return Vector<String> of <i>fields</i>
 */
public synchronized static Vector<String> getPropertyFields(String
deviceName, String propertyName) {
    String[] fields;
    Vector<String> vFields = new java.util.Vector<String>();

    try{
        ControlDevice cd;
        int pL = propertyName.length()+1;
        DevicePropertyService devicePropertyService =
            ServiceLocator.getService(DevicePropertyService.class);

        cd = ServiceLocator.getService(
            ControlDeviceService.class).getControlDevice(deviceName);

        fields = devicePropertyService.getFilterItems(
                        false,
                        false,
                        cd.getDeviceClassName(),
                        cd.getDeviceClassVersionString(),
                        propertyName);

        //Removes the name of the property and the # on each field
        for (String f : fields) {
            vFields.add(f.substring(pL, f.length())) ;
        }

        Collections.sort(vFields);

    }catch (Exception ex) {
        ex.printStackTrace();
    }
```

```
    return vFields;
}
```

5.1.6 Compressió de dades

Aquest codi es troba dins del fitxer app.js i és executat en el costat del client i és cridat quan donem format a les dades de tipus vector i el nombre d'elements que contenen és més gran o igual a la constant LENGTH_TO_COMPRESS_*DATA* amb la intenció de reduir la quantitat de les dades. Els mètodes que es poden cridar des de fora de la instància *ReduceArray* són:

- *reduceY:* Aquest mètode s'utilitza quan només coneixem els valor que equivaldrien a l'eix de les Y, o dit d'una altre manera, els valors corresponents al eix de les X equivaldria a l'índex de la posició que ocupa cada valor dins de l'array.
- *reduce:* Aquest mètode s'utilitza quan tenim dos vectors, un que conté el índex corresponent a cada valor i el segon conté els valors per a cada un d'aquests índexs.

Aquests mètodes, a part de rebre com a paràmetres els vectors originals, també reben l'amplada del contenidor o element on s'han de mostrar les dades, si la longitud del nombre d'elements que contenen els vectors supera en quatre l'amplada del contenidor s'aplicarà la compressió. L'objectiu és eliminar tots aquells punts que per l'alta concentració de dades en un espai reduït no són útils perquè queden amagades per altres valors. Els dos mètodes retornen un array de dues dimensions, on la primera conté els valors de tots els índex de les dades i la segona els valors corresponents.

```
var ReduceArray = function(){
    var get_index_start_from_xmin = function(xIn, xMin) {
        var ind_start = -1;

        for (var i = 0; i < xIn.length; i++) {
            if (xIn[i] >= xMin && ind_start == -1) {
                ind_start = i;
                i = xIn.length;
            }
        }

        return ind_start;
    };

    var get_index_end_before_xmax = function(xIn, xMax) {
        var ind_end = -1;

        for (var i = xIn.length - 1; i >= 0; i--) {
            if (xIn[i] <= xMax && ind_end == -1) {
                ind_end = i;
                i = 0;
            }
        }

        return ind_end;
```

```
    };

    var get_pixel_x_index_from_value_array = function(width_screen,
x_value, min_x_value_on_screen, max_x_value_on_screen) {
        return Math.trunc( width_screen * (x_value -
min_x_value_on_screen) / (max_x_value_on_screen -
min_x_value_on_screen) );
    };

    /**
     * Public methods
     */
    return {

        /**
         *
         *
         *  @param {array}  yIn -> the original values
         *  @param {string} width_scene -> value that it will be used
to make the reduction
         *
         *  @return {array}  -> returns an array 2D, the first
position has the indexes of each value, the second one has the values
         */
        reduceY: function(yIn, width_scene){
            var length = yIn.length;
            var xin = new Array(length);

            for(var i = 0; i < length; i++) {
                xin[i] = i;
            }

            return this.reduce(xin, yIn, 0, length, width_scene);
        },

        /**
         *
         *
         *  @param {array}  xIn ->
         *  @param {array} yIn ->
         *  @param {int} xMin ->
         *  @param {int} xMax ->
         *  @param {float} width_scene ->
         *
         *  @return {array} -> returns an array 2D, the first position
has the indexes of each value, the second one has the values
         */
        reduce: function(xIn, yIn, xMin, xMax, width_scene){

            //output data
            var xOut = [];
            var yOut = [];

            //No compression if width null or greater than the array
size

            if (width_scene == 0 || xIn.length < (4 * width_scene)) {
                return [xIn,yIn];
            }
```

```
            var size_per_pixel = parseInt(1 + xIn.length /
width_scene);

            var ind_start = get_index_start_from_xmin(xIn, xMin);
            var ind_end   = get_index_end_before_xmax(xIn, xMax);

            if (ind_start == -1) {
                //return copyToResultData(xOut, yOut, width_scene);
                return [xIn,yIn];
            }

            var xOut_tmp = [];
            var yOut_tmp = [];

            var xstart = ind_start;

            //Select data (compressed) per pixel 0-w
            var index_merged = 0;
            var index_tmp = 0;
            var index_out = 0;

            var factor = (xMax - xMin) / width_scene;

            for (var i = 0; i < width_scene; i++) {

                var v_xMergedToPixel = new Array(size_per_pixel);
                var v_yMergedToPixel = new Array(size_per_pixel);

                index_merged = 0;

                for (var k = xstart; k < ind_end; k++) {

                    var xscreen =
get_pixel_x_index_from_value_array(width_scene, xIn[k], xMin, xMax);

                    if (xscreen == i) {
                        v_xMergedToPixel[index_merged] = xIn[k];
                        v_yMergedToPixel[index_merged] = yIn[k];
                        index_merged = index_merged +1;
                    }
                    else {
                        xstart = k;
                        k = ind_end;
                    }

                }

                //No points found then add MAX_VALUE to be processed
afterward
                if (index_merged== 0) {
                    xOut_tmp[index_tmp] = (xMin + i * factor);
                    yOut_tmp[index_tmp] = Number.MAX_VALUE;
                    index_tmp = index_tmp +1;
                }
                //Only 1 points added
                else if (index_merged == 1) {
                    xOut_tmp[index_tmp] = v_xMergedToPixel[0];
                    yOut_tmp[index_tmp] = v_yMergedToPixel[0];
                    index_tmp = index_tmp +1;
                }
                //Only 2 points added
```

```
                else if (index_merged == 2) {
                    xOut_tmp[index_tmp] = v_xMergedToPixel[0];
                    yOut_tmp[index_tmp] = v_yMergedToPixel[0];
                    index_tmp = index_tmp +1;

                    xOut_tmp[index_tmp] = v_xMergedToPixel[1];
                    yOut_tmp[index_tmp] = v_yMergedToPixel[1];
                    index_tmp = index_tmp +1;
                }
                //More than 2 points then just add the min and max
                else if (index_merged > 2) {

                    var yminloc = Number.MAX_VALUE;
                    var ymaxloc = -Number.MAX_VALUE;

                    var imin = -1;
                    var imax = -1;

                    // console.log("size merged : "+index_merged);

                    for (var j = 0; j < index_merged; j++) {

                        if (yminloc > v_yMergedToPixel[j]) {
                            yminloc = v_yMergedToPixel[j];
                            imin = j;
                        }

                        if (ymaxloc < v_yMergedToPixel[j]) {
                            ymaxloc = v_yMergedToPixel[j];
                            imax = j;
                        }
                    }

                    if (imin < imax) {

                        if (v_yMergedToPixel[imin] !=
v_yMergedToPixel[0]) {

                            xOut_tmp[index_tmp] = v_xMergedToPixel[0];
                            yOut_tmp[index_tmp] = v_yMergedToPixel[0];
                            index_tmp = index_tmp+1;
                        }

                        xOut_tmp[index_tmp] = v_xMergedToPixel[imin];
                        yOut_tmp[index_tmp] = v_yMergedToPixel[imin];
                        index_tmp = index_tmp+1;
                        xOut_tmp[index_tmp] = v_xMergedToPixel[imax];
                        yOut_tmp[index_tmp] = v_yMergedToPixel[imax];
                        index_tmp = index_tmp+1;

                        if (v_yMergedToPixel[imax] !=
v_yMergedToPixel[index_merged - 1]) {
                            xOut_tmp[index_tmp] =
v_xMergedToPixel[index_merged - 1];
                            yOut_tmp[index_tmp] =
v_yMergedToPixel[index_merged - 1];
                            index_tmp = index_tmp+1;
                        }

                    }
                    else if (imin > imax) {
```

```
                        if (v_yMergedToPixel[imax] !=
v_yMergedToPixel[0]) {
                            xOut_tmp[index_tmp] = v_xMergedToPixel[0];
                            yOut_tmp[index_tmp] = v_yMergedToPixel[0];
                            index_tmp = index_tmp+1;
                        }

                        xOut_tmp[index_tmp] = v_xMergedToPixel[imax];
                        yOut_tmp[index_tmp] = v_yMergedToPixel[imax];
                        index_tmp = index_tmp+1;

                        xOut_tmp[index_tmp] = v_xMergedToPixel[imin];
                        yOut_tmp[index_tmp] = v_yMergedToPixel[imin];
                        index_tmp = index_tmp+1;

                        if (v_yMergedToPixel[imin] !=
v_yMergedToPixel[index_merged - 1]) {
                            xOut_tmp[index_tmp] =
v_xMergedToPixel[index_merged - 1];
                            yOut_tmp[index_tmp] =
v_yMergedToPixel[index_merged - 1];
                            index_tmp = index_tmp+1;
                        }
                    }
                    else {
                        xOut_tmp[index_tmp] = v_xMergedToPixel[imin];
                        yOut_tmp[index_tmp] = v_yMergedToPixel[imin];
                        index_tmp = index_tmp+1;
                    }
                }
            }

            //First point to be added if nothing found
            if (yOut_tmp[0] == Number.MAX_VALUE) {

                if (ind_start != 0) {

                    //New points added = interpolation
                    var valy = yIn[ind_start - 1] +
                        (yIn[ind_start] - yIn[ind_start - 1]) *
                            (xOut_tmp.get(0) - xIn[ind_start - 1]) /
                                (xIn[ind_start] - xIn[ind_start - 1]);

                    xOut_tmp[0] = valy;
                }
            }

            //Check whether we have pixel with no points (MAX_VALUE)
            for (var i = 1; i < xOut_tmp.length; i++) {

                ind_end = -1;

                if (yOut_tmp[i] == Number.MAX_VALUE) {

                    for (var j = i; j < xOut_tmp.length; j++) {

                        if (yOut_tmp[j] != Number.MAX_VALUE) {
                            ind_end = j;
                            j = xOut_tmp.length;
                        }
```

```
                    }

                    if (ind_end != -1) {}
                    else {

                        //No points at the end then remove (no need)
                        for (var j = i; j < xOut_tmp.length; j++) {
                            xOut_tmp.splice(xOut_tmp.length - 1, 1);
                            yOut_tmp.splice(yOut_tmp.length - 1, 1);
                        }

                        i = xOut_tmp.length;
                    }
                }
            }

            //Remove unused points
            for (var i = 0; i < xOut_tmp.length; i++) {

                if (yOut_tmp[i] != Number.MAX_VALUE) {
                    xOut[index_out] = xOut_tmp[i];
                    yOut[index_out] = yOut_tmp[i];
                    index_out = index_out+1;
                }
            }

            return [xOut, yOut];
        }
    }
}();
```

5.1.7 Creació de paleta de colors per les dades

Aquest codi es troba dins del fitxer app.js i és executat en el costat del client.

Primer de tots es mostra el codi de la classe *colorPalette*, per aquesta classe s'han deixat només els mètodes més rellevants. La funció de *colorPalette* és crear i emmagatzemar les diferents paletes de colors que es necessitin i retornar per un valor rebut el color corresponent per a una paleta especifica. Els colors creats per les diferents paletes tenen el format RGB i cada color consisteix en un vector on la primera posició conté el valor equivalent al color vermell, la segona al color verd i la última al color blau.

Per crear una paleta de colors ho fa, en el mètode *createPalette()*, a partir de la següent informació:

- *numColors*: total de colors que ha de contenir la paleta
- *zeroColor*: vector que conté el color equivalent al valor 0.
- *scale*: escala de possibles valors
- *base*: vector que conté els vectors amb els colors base que s'utilitzaran per crear la nova paleta de colors.

Per retornar un color s'ha de cridar el mètode *getColor*() enviant-li un valor entre 0 i 1 i el nom de la paleta de la que es vol rebre el color.

D'aquesta manera per retornar el color el que fa és multiplicar aquest valor rebut, pel nombre total de colors que té la paleta i s'utilitza la part entera del valor obtingut per accedir a la posició de la paleta que conté el color corresponent al valor rebut.

```
var colorPalette = function(){
    //Contains the used palettes with all the colors defined
    var palettes = {},
        // When one palette doesn't exist we use the default palette
        defaultPalet = "df",
        // Contains the general information about the available
palettes
        palettesData = {};

    /**
     * if there is data about the name of the color palette that you
receive as a parameter,
     * you will create the corresponding color palette from that data.
If it doesn't
     *  exist, create the default palette with the palette name
received.
     *
     *  @param {string} paletteName -> the name of the palette to be
created
     */
    var createPalette = function(paletteName) {
        let paletteData = {};
        let colors = [];

        if(palettesData[paletteName]===undefined){
            console.log("The selected color palette doesn't exist. A
default palette will be used");
            paletteData = palettesData[defaultPalet];
        } else {
            paletteData = palettesData[paletteName];
        }

        for (let i = 0, fi = (paletteData.numColors +1); i < fi; i++)
        {
            var alpha = i / paletteData.numColors;

            var index = 0;

            let scaleLength = paletteData.scale.length;
            for (var j = 0; j < scaleLength; j++) {
                if (alpha < paletteData.scale[j]) {
                    index = j-1;
                    break;
                }
                else index = (paletteData.defaultIndex === undefined)
                                ? (scaleLength -2)
                                : paletteData.defaultIndex;
            }

            var ratio = (alpha - paletteData.scale[index]) /
                        (paletteData.scale[index+1] -
                               paletteData.scale[index]);

            var r = paletteData.base[index][0] + ratio *
(paletteData.base[index+1][0] - paletteData.base[index][0]);
```

```
            var g = paletteData.base[index][1] + ratio *
(paletteData.base[index+1][1] - paletteData.base[index][1]);
            var b = paletteData.base[index][2] + ratio *
(paletteData.base[index+1][2] - paletteData.base[index][2]);

            colors[i] = [r, g, b];

        }

        palettes[paletteName] = colors.slice();
        colors = null;
    };//END createPalette = function(paletteName)

    /**
     * Public methods
     */
    return {

        /**
         * Returns the color corresponding to the value received
         *
         *  @param {float}  value -> is a value between 0 and 1
         *  @param {string} paletteName -> name of the palette whose
color you want to obtain
         *
         *  @return {array} an array that contains the RGB values
         */
        getColor: function(value, paletteName){
            let paletteData =  this.getPaletteData(paletteName);

            if (palettes[paletteName]===undefined)
                createPalette(paletteName);

            return
palettes[paletteName][Math.trunc(paletteData.numColors*value)]
        },
        /**
         * Returns the data about the paletteName received
         *
         *  @param {string} paletteName -> the name of the palette
         *
         *  @return {object} if this name exist returns this data,
otherwise returns the default palette data
         */
        getZeroColor : function(paletteName) {
            if (palettesData[paletteName]===undefined)
                return palettesData[defaultPalet].zeroColor;

            return palettesData[paletteName].zeroColor;
        }

        ,getPalette : function(paletteName){
            if (palettes[paletteName]===undefined)
                createPalette(paletteName);

            return palettes[paletteName]

        },

        /**
         * Returns the data about the paletteName received
```

```
         *
         *  @param {string} paletteName -> the name of the palette
         *
         *  @return {object} if this name exist returns this data,
otherwise returns the default palette data
         */
        getPaletteData : function(paletteName) {

            if (palettesData[paletteName]==undefined)
                return palettesData[defaultPalet];

            return palettesData[paletteName];
        },

        getData : function(paletteName){
            if (palettesData[paletteName]==undefined)
                return palettesData[defaultPalet];

            return palettesData[paletteName];

        }
    };
}();
```

5.2 Test

S'han fet probes de visualització i d'utilització en diferents navegadors. En les versions d'escriptori s'han fet probes amb Chrome, Firefox, FireFox Developer Edition i Safari. En les versions mòbils s'han fet probes amb Chrome, FireFox, Safari i amb el navegador dels dispositius Samsung, aquest últim s'ha testejat perquè els telèfons d'empresa que distribueix el CERN actualment als tècnics són d'aquesta marca.

També s'han fet probes de rendiment del servidor amb 60 sessions obertes, cada una amb múltiples subscripcions actives a l'hora i amb freqüències d'actualització de dades a 10Hz, on el servidor no ha tingut cap problema en subministrar totes aquestes dades.

En el front-end s'ha probes per detectar problemes de rendiment amb la actualització de dades, amb aquestes probes s'han detectat i intentat optimitzar al màxim els problemes comentats a l'apartat 6 amb les taules i la creació de les imatges. Amb aquestes probes també s'ha vist que els navegadors funcionen amb un bon rendiment actualitzant el contingut de 20 blocs a una freqüència de 10Hz amb tots els tipus de visualització de dades, exceptuant si tot són taules amb moltes dades on comença a produir-se un petit retard en el temps de resposta del navegador i en l'actualització dels continguts degut als problemes de reflow dels navegadors. També s'ha detectat que, en aquests moments, el navegador que ofereix un rendiment més elevant a l'hora d'actualitzar el contingut sense que apareguin problemes és Google Chrome (versió 71), seguit de FireFox Developer Edition (versió 65).

Un total de 10 persones de diferents grups, han estat provant Fesa Web. Las respostes que hem obtingut han estat molt satisfactòries, algunes de les quals són:

- La velocitat d'obrir una pàgina web és molt superior al temps que es tarda en executar les aplicacions Java actuals, especialment en màquines virtuals.
- Gran facilitat d'us i claredat en la visualització de les dades.
- El poder connectar-se des de la GPN i des de el mòbil o qualsevol dispositiu amb connexió a la xarxa, facilitarà molt el poder consultar els efectes de les modificacions de les configuracions dels dispositius, ja que les aplicacions actuals exigien utilitzar un ordinador i que estigues connectat a la TN.
- El rendiment aconseguit és molt bo tractant-se d'una pàgina web.

5.3 Versions

En aquest apartat es comenten las versions més rellevants de totes les que s'han anat creant al llarg del desenvolupament de Fesa Web.

5.3.1 Versió 1.0

En aquesta primera versió, el front-end era completament funcional, permetent realitzar subscripcions sota demanda o subscripcions per Setting i mostrant la informació de cada camp en un bloc permetent modificar la vista de les dades segons el tipus d'aquestes. Els diferents tipus de visualització de les dades eren: com un únic valor, només pels tipus de dades escalars, com una taula, com una gràfica o com una imatge.

En aquesta primera versió la distribució de classes del front-end era molt més senzilla que la de la versió final representada a la Figura 1. A la Figura 26 es pot veure el Diagrama UML del front-end d'aquesta primera versió.

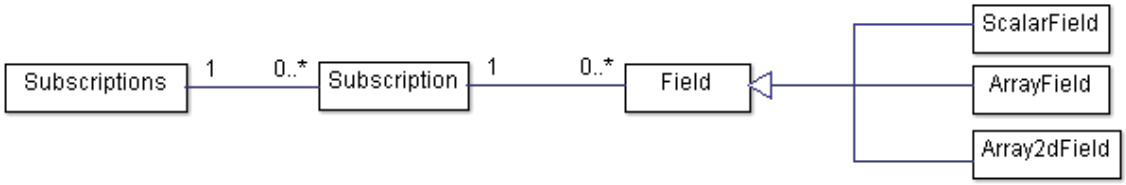

Figura 26 Diagrama UML del front-end versió 1.0

En el front-end les instancies de *FesaConnector* emmagatzemaven la informació de les sessions que volien rebre les dades de la subscripció a la que fan referencia i eren les encarregades d'enviar-lis aquestes dades.

5.3.2 Versió 2.0

En el front-end es va afegir la possibilitat de filtrar les dades que es volen visualitzar a partir de la configuració dels Settings, permetent en el cas de vectors d'una sola dimensió consultar el valor d'una posició exacte. En els vectors de dues dimensions, es pot seleccionar que només es vol

consultar les dades de tota una columna, d'una fila o només un valor d'una posició exacta del vector.

En el back-end a la versió 1.0 existia el problema que quan dues instàncies de *FesaConnector* enviaven a l'hora dades a una mateixa sessió, es produïa un error en el servidor Tomcat. Per evitar aquest problema es va modificar el back-end creant la classe *FesaDataMapContainer* per guardar les últimes dades rebudes per les instàncies de la classe *FesaConnector* juntament amb la data en la que s'han rebut.

Es va modificar la classe *FesaConnector* de manera que deixes de tenir constància de les sessions que volien rebre informació dels camps de la subscripció a la que fan referencia les seves instàncies i fer que cada vegada que reben dades les enviïn a la classe *FesaDataMapContainer*.

Per últim és va crear la classe *SessionScheduler* on cada instància d'aquesta classe conté una sessió (comunicació persistent entre el client i el servidor). Aquestes instàncies s'encarreguen d'enviar els missatges a la sessió que contenen, emmagatzemar la informació de les subscripcions que té actives aquesta sessió i els camps dels que vol rebre informació i consulten a la classe *FesaDataMapContainer* si hi ha noves dades i enviar-les en un sol missatge a la sessió.

5.3.3 Versió 3.0

En aquesta versió es va afegir la possibilitat de fer que un Setting executes altres Settings existents i la possibilitat d'incloure consultes en la definició el Setting per fer que aquest Setting fes referencia a múltiples dispositius. Per això va ser necessari reestructurar pràcticament totes les classes del front-end i el funcionament dels blocs, ja que ara un sol bloc pot mostrar informació d'un mateix camp de múltiples dispositius. La nova estructura del front-end queda representat el en diagrama UML de la *Figura 1.* Fins ara eren els les instancies dels blocs les que contenien i gestionaven la informació rebuda de les subscripcions, en aquesta versió és la classe *db* qui s'encarrega de gestionar aquestes dades i els blocs només saben que han de mostrar la informació d'un camp concret, d'un o de múltiples dispositius.

El back-end també va necessitar una petita reestructuració per poder realitzar múltiples subscripcions a la vegada a partir d'un sol Setting.

5.4 Bugs

5.4.1 Tomcat i enviament missatges simultanis a una Sessió

A la primera versió de Fesa Web les subscripcions creades (instàncies de *FesaConnector*), tenien el llistat de sessions que volien rebre les actualitzacions i dels camps en els que estan interessades d'aquesta subscripció, de manera que cada subscripció s'encarregava d'enviar les dades a les sessions interessades. Això provocava que en ocasions dues instancies enviessin dades a una mateixa sessió a l'hora via

WebSocket i Tomcat no està preparat per permetre això i provocava un error cada vegada que això succeïa generant problemes en la recepció de les dades en els clients.

Per solucionar aquest problema i aprofitant que també es volia poder personalitzar la freqüència amb la que es tenien que rebre noves dades, en les subscripcions realitzades per Settings, es va crear la classe estàtica *FesaDataMapContainer* que s'encarrega d'emmagatzemar, les últimes dades rebudes de totes les subscripcions existents, juntament amb un timestamp del moment en el que s'han rebut, de manera que, siguin les pròpies sessions les que sol·licitin a aquesta classe totes les dades dels camps de les subscripcions corresponents a la sessió, i que el seu timestamp sigui superior al de l'ultima vegada que es van obtenir noves dades, d'aquesta manera és la pròpia sessió la que s'encarrega d'enviar-se en un sol missatge totes les dades que han estat actualitzades.

5.4.2 Tomcat tancament d'una sessió quan una subscripció esta activa

Un altre problema que es va trobar amb el servidor Tomcat, relacionat també amb el WebSocket, era que quan el client tancava el navegador, o una de las pestanyes que tingues la plana de Fesa Web carregada, amb una o més subscripcions actives, es produïa l'error següent en el servidor Tomcat:

"*The WebSocket session [0] has been closed and no method (apart from close()) may be called on a closed session*".

Buscant informació per internet es va descobrir que per resoldre aquest problema, s'ha de detectar, mitjançant JavaScript, l'esdeveniment de tancament del navegador per enviar un missatge via WebSocket al servidor per informar-lo de que el client esta tancant el navegador. Enviant aquest missatge just abans de que és tanqui la comunicació permanent entre el client i el servidor via WebSocket, s'evita l'error que es produïa quan la sessió del WebSocket crida al mètode close().

5.4.3 Creació de taules

Aquest punt ha estat un dels que més problemes ha generat al llarg del desenvolupament de Fesa Web pels problemes de *reflow*[3] que es produeixen al crear/modificar taules amb un nombre considerable de cel·les.

Inicialment es va decidir utilitzar el plugin *DataTable*[4] per ser un dels més utilitzats a l'hora de treballar amb taules. La intenció d'utilitzar-lo a Fesa

[3] Procés mitjançant el qual els navegadors tornen a calcular las posicions i formes dels elements presents en el document amb l'objectiu de tornar a renderitzar-los total o parcialment. Aquest procés es realitza cada vegada que s'afegeix, elimina, modifica un element, quan es modifica alguna regla CSS o modifica el tamany de la finestra del navegador, etc. Mentre dura aquest procés la finestreta del navegador queda bloquejada.

[4] Plugin especialitzat en donar format a taules i automatització de funcionalitats com: paginació, filtratge de dades, ordenació per columnes, etc. (https://clusterize.js.org/).

Web era per poder crear taules amb un scroll horitzontal i vertical, mantenint la capçalera i la primera columna, quan fos necessari, sempre visibles. El primer problema que es va detectar va ser que DataTable era molt lent al crear i actualitzar el contingut de les taules, produint bloquejos a la plana web. Per evitar això es va provar de crear manualment la taula i acte seguit aplicar-li el format amb DataTable i anar actualitzant el contingut de la taula directament amb JavaScript, sense utilitzar el plugin DataTable, això va agilitzar bastant els problemes de temps al actualitzar les dades de les taules però encara és produïen bloquejos importants a l'hora de crear tot el contingut de la plana quan es tenien que crear múltiples taules. Per millorar aquests temps encara més, o simular un millor rendiment, es van fer probes per fer que la creació del codi HTML de les taules fos asíncron mitjançant:

- *Promises*: que s'encarregaven de crear el contingut i un cop creat afegir-lo o substituir a la plana web.
- *Web Workers*: que rebien les dades amb les que es tenia que crear la taula i a partir d'aquestes creaven el codi HTML de la taula equivalent i el retronaven al codi principal que l'havia cridat per afegir-lo o substituir a la plana web.

Després de realitzar múltiples probes amb els navegadors Firefox, Firefox Developer Edition i Google Chrome, es va detectar que el millor rendiment s'obtenia amb la utilització de Web Workers. Amb la solució original, la pàgina és quedava bloquejada i al cap d'una estona apareixien tots els blocs creats, amb aquesta solució aconseguíem que la creació de tot el contingut HTML dels diferents blocs no quedi bloquejada i que tots els blocs de ràpida creació apareguin enseguida i que les taules vagin apareixent progressivament.

Un cop implementada aquesta solució, es va intentar millorar el rendiment en la creació i actualització de les taules utilitzant *Clusterize*[5], enlloc de DataTable, obtenint una millora de rendiment notable. El problema que es va trobar amb Clusterize era que quan creava la capçalera de la taula estàtica, i s'actualitzaven les dades de la taula i aquestes modificaven l'amplada de la taula, l'amplada de les capçaleres no es modificava provocant que les columnes de la capçalera no quedessin alineades amb les columnes de les dades. Aquest mateix problema es produïa en el cas explicat anteriorment, amb DataTable, al actualitzar les dades de les taules sense utilitzar el plugin per realitzar l'actualització del contingut de la taula. Així que finalment es va decidir prescindir de la utilització de plugins i crear directament les taules amb JavaScript i sense tenir capçaleres, ni columnes estàtiques a l'espera d'aconseguir trobar una solució més eficient.

[5] Plugin per maximitzar el rendiment en la creació de taules amb una quantitat elevada de files, creant només un nombre determinat de files, enlloc de generar la taula sencera, i quan es necessita realitzar un desplaçament vertical per veure la resta de valors, el que fa és actualitzar el contingut de les files prèviament creades. (https://datatables.net/)

5.4.4 Problema amb els valors MAX i MIN al crear colors per les imatges

En els Settings existeix la opció d'especificar el valor màxim i mínim de les dades que es rebran, aquest valors són utilitzats en les gràfiques per que l'eix de les Y tingui una dimensió fixa i en el cas de les imatges aquest valors són utilitzats a l'hora de crear les paletes de colors. Quan es va desenvolupar aquest opció es va produir una reducció dràstica del rendiment a l'hora de crear la imatge, dificultant aconseguir l'actualització de les imatges a una freqüència de 10Hz o bloquejant el navegador al tenir que crear les imatges d'històric de valors amb vectors de més de 500.000 punts. En el moment d'implementar aquesta funcionalitat, també es va fer una reestructuració de tot el funcionament de la creació de les paletes de colors per permetre generar de diferents tipus de paletes de colors segons l'elecció de l'usuari, i es va pensar que el problema es devia a aquesta reestructuració.

Després de realitzar múltiples probes sobre quina part del codi era on es produïa la pèrdua de rendiment, és va detectar que en realitat el problema venia de les propietats que contenien el valor MAX i MIN. Aquests valors es reben del servidor en format JSON i en conseqüència, com una cadena de text i, al guardar-les en la propietat pertinent en JavaScript, aquestes es guardaven automàticament com a cadena de text, de manera que, cada vegada que es tenia que crear una imatge utilitzant el contingut d'aquestes propietats, el navegador tenia que realitzar una conversió de tipus de cadena de text a valor numèric i aquesta conversió de tipus era la perjudicava el rendiment de la creació d'imatges.

Per solucionar aquest problema el que es va fer va ser aplicar la conversió de tipus en el moment de rebre aquesta informació i així emmagatzemar-la directament en format numèric. Amb això és va aconseguir recuperar el rendiment original a l'hora de crear les imatges.

5.4.5 Problema amb versió mòbil del navegadors Safari i de Samsung

En aquest dos casos s'ha trobat un problema al utilitzar preventDefault() per anular el funcionament per defecte dels enllaços en els menús dels diferents blocs que mostren els valors dels camps de les subscripcions realitzades. El problema que hi havia era que quan es clicava sobre aquests enllaços es realitzava l'acció indicada, però també t'enviava a l'inici de la pàgina.

Per solucionar aquest problema el que s'ha fet ha estat detectar quan la pàgina s'està carregant en un dispositiu mòbil i afegir codi JavaScript a l'etiqueta HREF de l'enllaç, substituint el seu valor típic, el caràcter "#", pel codi "javascript: return false;"; d'aquesta manera garantim que s'anul·la el comportament per defecte de l'enllaç.

6. Conclusions

Aquest treball m'ha ajudat a millorar molt els meus coneixements de Java i JavaScript. En el cas de Java la meva experiència amb aquest llenguatge era mínima i aquest treball m'ha servit per comprendre les complexitats d'aquest llenguatge de programació i sentir-me molt a gust treballant amb ell. En el cas de JavaScript, el meu nivell de coneixement era bastant bo, però a partir dels problemes de rendiment detectats en algunes situacions, vaig tenir que investigar bastant sobre com optimitzar el codi i em va permetre aprendre moltes coses sobre com afectava al rendiment les diferent maneres d'escriure un mateix codi. També va suposar tot un repte el desenvolupament de codi per la compressió de dades o la creació de paletes de color i transformació de valors en color, ja que mai havia fet cap cosa similar i en aquests dos punts el meu supervisor em va ser de gran ajuda.

Estic molt content del resultat obtingut, ja que la versió final de Fesa Web realitza moltes més coses de les que inicialment s'havia plantejat. Algunes d'aquestes funcionalitats tenen a veure amb les opcions de configuració que permetent els Setting, com són: la possibilitat de definir consultes per obtenir la informació de múltiples dispositius, poder comparar els valors de dos camps d'una mateixa propietat, fer que un Setting pugui executar altres Settings existents, permetre als usuaris crear els seus pròpies Settings, etc.

Hi ha algunes característiques que han quedat pendents de desenvolupar però la intenció és tenir-les enllestides a curt termini i són;

- Quan en un bloc del front-end s'està visualitzant una dada com una gràfica, tenir una opció per poder modificar la representació de les dades entre línies, barres o punts.
- Quan en un bloc del front-end s'està visualitzant una dada com una imatge, tenir una opció per modificar la paleta de colors d'aquesta imatge entre les paletes existents.
- Seleccionar en la configuració dels Settings un rang de dades personalitzats en vectors d'una o més dimensions. Això permetria simular un zoom sobre una part de les dades o eliminar la visualització d'informació innecessària o poc rellevant en els extrems laterals o superiors de les dades.
- Implementar l'autentificació Single Sing-On (SSO) que s'utilitzen en el CERN per garantir que només els usuaris del CERN accedeixen a Fesa Web.

7. Bibliografia

https://wikis.cern.ch/display/FESA3/Step+1+Creating+FESA+Class
https://devcode.la/tutoriales/elemento-canvas-responsive/
https://www.tutorialspoint.com/java/java_multithreading.htm
https://beginnersbook.com/2013/03/multithreading-in-java/
https://www.geeksforgeeks.org/multithreading-in-java/
https://jonathanmelgoza.com/blog/observadores-en-java/
https://www.concretepage.com/java/example-observer-observable-java
https://www.oracle.com/technetwork/es/articles/java/procesamiento-streams-java-se-8-2763402-esa.html
https://javarevisited.blogspot.com/2013/02/swap-two-numbers-without-third-temp-variable-java-program-example-tutorial.html
https://www.sitepoint.com/multi-threading-javascript/
https://caniuse.com/#search=worker
https://www.htmlgoodies.com/html5/tutorials/introducing-html-5-web-workers-bringing-multi-threading-to-javascript.html
https://stackoverflow.com/questions/30036/javascript-and-threads?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
https://www.w3.org/TR/workers/
https://developer.mozilla.org/es/docs/Web/Guide/Performance/Usando_web_workers
https://www.html5rocks.com/es/tutorials/workers/basics/
https://html.spec.whatwg.org/multipage/workers.html
https://www.html5rocks.com/en/tutorials/workers/basics/
https://developers.google.com/web/fundamentals/primers/service-workers/?hl=es-419
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Funciones/Arrow_functions
https://es.stackoverflow.com/questions/4681/qu%C3%A9-es-en-javascript
https://www.humanlevel.com/articulos/desarrollo-web/optimizacion-de-javascript.html#Marcar_el_JavaScript_como_asincrono
https://howchoo.com/g/mmu0nguznjg/learn-the-slow-and-fast-way-to-append-elements-to-the-dom
https://www.learnsteps.com/javascript-increase-performance-by-handling-dom-with-care/
https://www.todojs.com/como-medir-el-rendimiento-de-un-codigo-javascript/
https://www.sitepoint.com/10-ways-minimize-reflows-improve-performance/
https://desalasworks.com/article/javascript-performance-techniques/
https://dev.opera.com/articles/efficient-javascript/?page=2
https://leftshift.io/4-javascript-optimisations-you-should-know
https://www.todojs.com/14-claves-para-optimizar-codigo-javascript/
https://yeisondaza.com/herencia-funcional-en-javascript/
https://www.adictosaltrabajo.com/2018/04/30/es6-javascript-variables-y-constantes/

http://www.vissit.com/ajustar-el-tamano-de-texto-segun-su-contenedor/
https://netbeans.org/kb/docs/javaee/maven-entapp.html
http://artfun.pw/post/76.html
http://ewaldertl.blogspot.com/2015/01/maven-indexing-in-netbeans-filled-up-my.html
https://www.mkyong.com/maven/how-to-create-a-web-application-project-with-maven/
https://www.byteslounge.com/tutorials/java-ee-html5-websocket-example
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/HomeWebsocket/WebsocketHome.html
https://github.com/websockets/ws/wiki/Websocket-client-implementation-for-auto-reconnect
https://msdn.microsoft.com/es-es/communitydocs/win-dev/os/manejador-de-procesos-los-threads