

Grau en Enginyeria Informàtica de Gestió i Sistemes d’Informació

PLATAFORMA DE VISUALITZACIÓ D’ALERTES SANITÀRIES

Memòria Final

ERIK ESPUÑES JUBERÓ

TUTOR: EUGENI FERNÁNDEZ GONZÁLEZ

CURS 2020-2021



ABSTRACT

This project is focused on the development of a mobile application that warns the hospital staff when an event of their responsibility occurs. The product will be focused on two parts.

The first one will be a web server that will configure the users and the type of alert that they will receive.

The second one will be a mobile app that will be responsible for configuring and warning the user.

RESUM

Aquest treball se centra en el desenvolupament d’una aplicació de mòbil que avisi al personal d’un hospital quan succeeixi algun esdeveniment que sigui de la seva responsabilitat. El producte se centrarà en dues parts.

La primera, un servidor web que serà el que configuri els usuaris i el tipus d’alertes que puguin rebre.

La segona, una aplicació per a mòbils que serà l’encarregada de configurar l’alerta i avisar a l’usuari.

RESUMEN

Este trabajo se centra en el desarrollo de una aplicación de móvil que avise al personal de un hospital cuando suceda algún evento que sea de su responsabilidad. El producto se centrará en dos partes.

La primera, un servidor web que será el que configure los usuarios i los tipos de alertas que pueda recibir.

La segunda, una aplicación para móviles que será la encargada de configurar la alerta y avisar al usuario.

ÍNDEX

[ÍNDEX DE FIGURES V](#_Toc74671293)

[ÍNDEX DE TAULES VII](#_Toc74671294)

[CAPÍTOL 1. INTRODUCCIÓ 1](#_Toc74671295)

[CAPÍTOL 2. MARC TEÒRIC 3](#_Toc74671296)

[2.1. CONTEXT 3](#_Toc74671297)

[2.1.1. L’ERA DEL PAPER 3](#_Toc74671298)

[2.1.2. L’ERA DIGITAL 4](#_Toc74671299)

[2.2. ANTECEDENTS 5](#_Toc74671300)

[2.3. NECESSITATS DE LA INFORMACIÓ 5](#_Toc74671301)

[2.3.1. HEALTHCARE INFORMATION SYSTEM (HIS) 6](#_Toc74671302)

[2.3.2. MOTOR D’INTEROPERABILITAT 7](#_Toc74671303)

[2.3.3. PROTOCOLS ENTRE EL HIS I EL MOTOR D’INTEROPERABILITAT 7](#_Toc74671304)

[2.3.4. TECNOLOGIES PER A DESENVOLUPAR UN SERVIDOR WEB 9](#_Toc74671305)

[2.3.5. TECNOLOGIES PER A DESENVOLUPAR UNA APLICACIÓ MÒBIL 10](#_Toc74671306)

[2.3.6. TECNOLOGIES PER A ENVIAR I REBRE NOTIFICACIONS AL MÓBIL 12](#_Toc74671307)

[CAPÍTOL 3. OBJECTIUS I ABAST 13](#_Toc74671308)

[3.1. OBJECTIUS DEL CLIENT 13](#_Toc74671309)

[3.2. OBJECTIUS DEL PRODUCTE 13](#_Toc74671310)

[3.3. PÚBLIC POTENCIAL 13](#_Toc74671311)

[3.4. ABAST 13](#_Toc74671312)

[CAPÍTOL 4. METODOLOGIA 15](#_Toc74671313)

[CAPÍTOL 5. DEFINICIÓ DE REQUERIMENTS FUNCIONALS I TECNOLÒGICS 17](#_Toc74671314)

[5.1. REQUERIMENTS FUNCIONALS DEL SERVIDOR D’*INTERSYSTEMS IRIS FOR HEALTH* 17](#_Toc74671315)

[5.2. REQUERIMENTS FUNCIONALS DEL SERVIDOR WEB 17](#_Toc74671316)

[5.3. REQUERIMENTS FUNCIONALS DE L’APLICACIÓ 18](#_Toc74671317)

[5.4. REQUERIMENTS TECNOLÒGICS 18](#_Toc74671318)

[CAPÍTOL 6. DESENVOLUPAMENT 19](#_Toc74671319)

[6.1. DISSENY 19](#_Toc74671320)

[6.1.1. BASES DE DADES 19](#_Toc74671321)

[6.1.2. SERVIDOR WEB 20](#_Toc74671322)

[6.1.3. APLICACIÓ MÒBIL 29](#_Toc74671323)

[6.2. SERVIDOR IRIS 39](#_Toc74671324)

[6.2.1. CONFIGURACIÓ 39](#_Toc74671325)

[6.2.2. CREACIÓ DEL HIS 39](#_Toc74671326)

[6.2.3. CREACIÓ DE L’API 40](#_Toc74671327)

[6.2.4. PUBLICAR L’API 42](#_Toc74671328)

[6.3. SERVIDOR WEB 43](#_Toc74671329)

[6.3.1. *BACKEND* 43](#_Toc74671330)

[6.3.2. API 49](#_Toc74671331)

[6.3.3. GESTIÓ DE LES NOTIFICACIONS 51](#_Toc74671332)

[6.3.4. RESULTAT FINAL 54](#_Toc74671333)

[6.4. APLICACIÓ 56](#_Toc74671334)

[6.4.1. INICIAR SESSIÓ 57](#_Toc74671335)

[6.4.2. LLISTA D’ALERTES 58](#_Toc74671336)

[6.4.3. ALERTA 60](#_Toc74671337)

[6.4.4. PERFIL 60](#_Toc74671338)

[6.4.5. REBRE NOTIFICACIONS 62](#_Toc74671339)

[CAPÍTOL 7. CONCLUSIONS 65](#_Toc74671340)

[CAPÍTOL 8. POSSIBLES AMPLIACIONS 67](#_Toc74671341)

[8.1. ASSIGNAR ALERTAS A DIFERENTS ROLS 67](#_Toc74671342)

[8.2. *LOG* DELS VALORS DE LES ALERTES 67](#_Toc74671343)

[8.3. MILLORAR LA SEGURETAT EN L’APLICACIÓ 67](#_Toc74671344)

[CAPÍTOL 9. BIBLIOGRAFIA 69](#_Toc74671345)

# ÍNDEX DE FIGURES

[Figura 1.1: Esquema de l’arquitectura del producte. Font: Elaboració pròpia. 1](#_Toc74671347)

[Figura 2.1: Esquema de les dades que emmagatzema el *HIS* [11]. Font: *Health Information Systems - Architectures and Strategies* [10]. 6](#_Toc74671348)

[Figura 2.2: Exemple de missatge de *HL7* *v2*. Font: *HL7 in Personal Health* [15, p. 7]. 8](#_Toc74671349)

[Figura 2.3: Exemple del missatge de *HL7* *v3*. Font: *HL7 in Personal Health* [15, p. 7]. 8](#_Toc74671350)

[Figura 2.4: Exemple d’una imatge generada pel protocol *DICOM*. Font: *DICOM Basics using Java - Extracting Image Data* [16]. 9](#_Toc74671351)

[Figura 6.1: Diagrama ERD de la base de dades. 20](#_Toc74671352)

[Figura 6.2: Diagrama UML de classes del servidor web. 23](#_Toc74671353)

[Figura 6.3: Diagrama BPMN de la creació d’una institució mèdica. 24](#_Toc74671354)

[Figura 6.4: Diagrama BPMN de la gestió de les notificacions. 25](#_Toc74671355)

[Figura 6.5: Disseny de la UI de l’inici de sessió en el servidor web. 26](#_Toc74671356)

[Figura 6.6: Disseny de la UI de la pàgina d’inici en el servidor web. 27](#_Toc74671357)

[Figura 6.7: Dissenys de la UI de la llista d’institucions mèdiques en el servidor web. 27](#_Toc74671358)

[Figura 6.8: Disseny de la UI de la visualització d’una institució mèdica en el servidor web. 28](#_Toc74671359)

[Figura 6.9: Disseny de la UI de la creació d’una institució mèdica en el servidor web. 28](#_Toc74671360)

[Figura 6.10: Diagrama UML de classes de l’aplicació mòbil. 31](#_Toc74671361)

[Figura 6.11: Diagrama BPMN de l’inici de sessió en l’aplicació. 32](#_Toc74671362)

[Figura 6.12: Diagrama BPMN de la visualització de la llista d’alertes en l’aplicació. 33](#_Toc74671363)

[Figura 6.13: Diagrama BPMN de la visualització d’una alerta en l’aplicació. 34](#_Toc74671364)

[Figura 6.14: Disseny de la UI de la llista d’alertes en l’aplicació mòbil. 38](#_Toc74671365)

[Figura 6.15 Figura 6.16: Disseny de la UI de la visualització d’una alerta en l’aplicació mòbil. 38](#_Toc74671366)

[Figura 6.17: Imatge de la configuració d’una API en *Intersystems IRIS for Health*. 42](#_Toc74671367)

[Figura 6.18: *Port forwarding* que s’ha fet al router. 43](#_Toc74671368)

[Figura 6.19: Resultat final del taulell de la pàgina web. 54](#_Toc74671369)

[Figura 6.20: Resultat final de la creació d’una alerta de la pàgina web. 55](#_Toc74671370)

[Figura 6.21 Figura 6.22: Resultat final de la llista alertes de la pàgina web. 55](#_Toc74671371)

[Figura 6.23 Figura 6.24: Resultat final de la visualització d’una alerta de la pàgina web. 56](#_Toc74671372)

[Figura 6.25 Figura 6.26: Resultat final de la edició d’una alerta de la pàgina web. 56](#_Toc74671373)

[Figura 6.27: Resultat final de l’inici de sessió a l’aplicació. 57](#_Toc74671374)

[Figura 6.28: Resultat final de la llista d’alertes a l’aplicació. 59](#_Toc74671375)

[Figura 6.29: Resultat final de la visualització de l’alerta a l’aplicació. 60](#_Toc74671376)

[Figura 6.30: Resultat final del perfil a l’aplicació. 61](#_Toc74671377)

[Figura 6.31: Format de les notificacions de les tres alertes que hi ha en el servidor. 62](#_Toc74671378)

# ÍNDEX DE TAULES

[Taula 6.1: Cas d’ús d’inici de sessió en l’aplicació mòbil. 35](#_Toc74502560)

[Taula 6.2: Cas d’ús de la llista d’alertes en l’aplicació mòbil. 35](#_Toc74502561)

[Taula 6.3: Cas d’ús de la visualització d’una alerta en l’aplicació mòbil. 36](#_Toc74502562)

[Taula 6.4: Taula de les rutes de l’API 49](#_Toc74502563)

# INTRODUCCIÓ

En els anys on s’han vist marcats per una pandèmia mundial, s’ha pensat a fer una aplicació per ajudar tot el personal sanitari i millorar la tecnologia que s’usa als hospitals.

Perquè la introducció del producte a desenvolupar sigui més completa s’ha afegit un esquema (Fig. 1.1) amb tota l’arquitectura del producte.

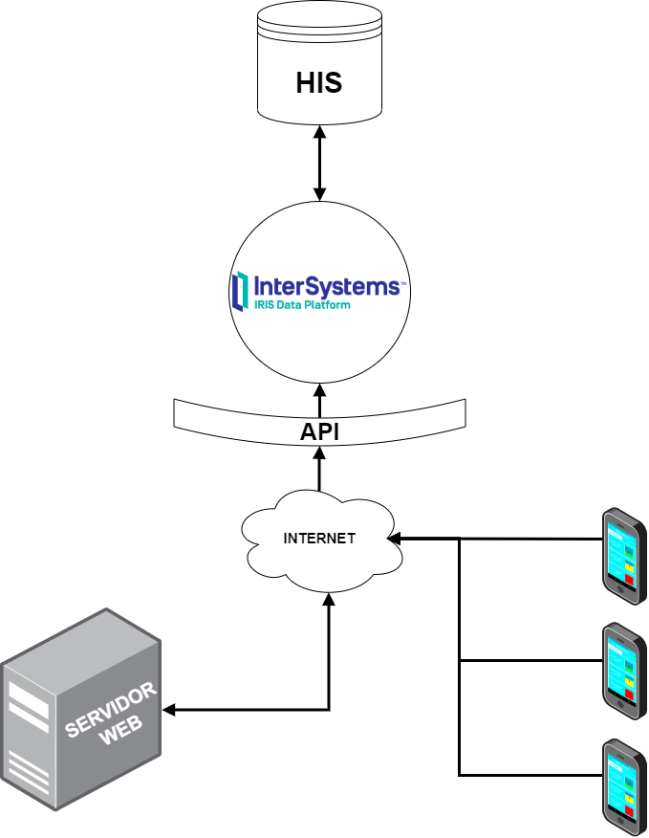


Figura .: Esquema de l’arquitectura del producte. Font: Elaboració pròpia.

El que es pretén aconseguir és desenvolupar una aplicació de mòbil que enviï alertes a l’usuari sanitari que l’estigui usant, quan succeeixi qualsevol fet que sigui de la seva responsabilitat. Per exemple, si un gerent d’un hospital necessita que l’avisin quan l’ocupació dels llits per a persones ingressades per *COVID-19* supera un valor, aquesta aplicació enviarà una alerta al mòbil del gerent perquè pugui intervenir de manera ràpida. Aquesta aplicació es comunicarà amb el motor d’interoperabilitat que usi l’hospital, en el cas d’aquest TFG, es comunicarà amb una *API* que es generarà del motor d’interoperabilitat *InterSystems IRIS for Health* [1].

Aquesta aplicació ve acompanyada d’un servidor que estarà sempre en línia i serà el que permeti configurar quin tipus d’alertes rep cada usuari de l’aplicació. Posant d’exemple el cas anterior, abans de poder rebre aquestes alertes, s’haurà de configurar que el gerent pugui rebre alertes d’ocupació de pacients de *COVID-19*.

Per acabar es necessitarà programar una *API* en un motor d’interoperabilitat, per a simular com funcionaria l’aplicació i el servidor en un entorn real. Aquest element serà el principal, ja que sense les dades del *HIS*, sistema d’informació de l’hospital, l’aplicació no podrà cap dada i per tant cap alerta funcionaria.

# MARC TEÒRIC

## CONTEXT

Aquest TFG entra en el món de la medicina. Per tant per saber bé el context del producte a elaborar s’ha de veure l’evolució que ha tingut el tractament de dades en els hospitals. Podem diferenciar clarament dues eres on la revolució informàtica va permetre el salt d’una a altra. La primera, l’era en què totes les dades dels hospitals eren a paper, on es veurà quins problemes hi havia i el motiu pel qual es va fer la transició a l’era digital.

### L’ERA DEL PAPER

La primera vegada que es va escriure un document mèdic, que es té constància, és d’un papir de l’antic Egipte del 1600 aC, on es reporten casos de tractaments de ferides de guerra.

Els documents mèdics van seguir evolucionant amb els grecs. Hipòcrates, el metge de l’Antiga Grècia més famós i considerat el pare de la medicina, va fundar la seva pròpia escola per aprendre medicina i va deixar documents escrits sobre els símptomes que tenien els pacients.

Tots aquests documents es van anar traduint a l’àrab perquè metges com al-Razi poguessin aprendre medicina en la gran era daurada islàmica entre el segle VIII i XIII. Els àrabs són els que van crear la idea de l’hospital. A més van ser els que van crear els historials mèdics, que eren escrits pels aprenents de metge sota la supervisió d’un metge titulat.

Aquests documents van seguint evolucionant en l’època de la llum a Europa. Primer a la França del segle XVII on les anàlisis de l’anatomia humana van canviar la perspectiva de la medicina. Després a inicis del segle XIII a Suècia van començar a refinar un sistema d’històries mèdiques a paper que van anar refinant fins al segle XX. [2]

A finals del segle XIX i inicis del XX, va haver-hi un canvi important, on als Estats Units es va començar a fer un historial mèdic dels pacients com es coneix en l’actualitat. Això inclou, un identificador del pacient, hàbits del pacient, malalties anteriors, historial familiar, anàlisis, entre molts altres aspectes més. Aquest historial va ser molt útil, i segueix sent-ho, per a ambulatoris o centres d’atenció primària. [3]

#### PROBLEMES DELS HISTORIALS MÈDICS EN PAPER

Amb els documents a paper sorgeixen una sèrie d’inconvenients que tenen a veure amb l’espai físic que ocupen, la cerca d’aquells documents i, també, la mobilitat que tenen.

El primer problema sorgeix amb l’espai que ocupa cada document i el temps que s’han de tenir arxivats. Si agafem dades d’un hospital universitari alemany [4], cada any ingressen entre 300.000 i 400.000 pacients. Això significa que es generen vuit milions d’impressions a paper, que equival a un volum d’un quilòmetre i mig en paper. A part del volum generat en un any, a Alemanya s’ha de guardar tot document mèdic un mínim de trenta anys. Si aquests documents fossin digitals, ocuparien entre 10 i 15 terabytes, el volum seria de menys d’un metre cúbic.

Amb aquestes dades es pot veure que emmagatzemar les dades a paper requereix un espai d’emmagatzematge molt gran, i a més requereix un manteniment humà (empleats encarregats per gestionar aquelles dades) i físic (el magatzem). El temps per poder accedir a qualsevol document emmagatzemat també és molt elevat, fent que sigui bastant ineficient haver d’anar a consultar dades en aquests historials. Per acabar, tenir la documentació en paper dificulta la interconnexió amb diferents països. Per exemple si una persona viatja a un país com el Japó i ha de ser atès en un hospital d’aquella nació, tindran molt complicat, per no dir gairebé impossible, consultar el seu historial si aquesta està emmagatzemada en paper.

El motiu principal de la transició dels documents mèdics de paper als digitals va ser perquè amb el temps les màquines dels hospitals recol·lectaven més i més dades que han de ser emmagatzemades. Per tant, era més còmode emmagatzemar-les en digital que imprimir-les [5].

### L’ERA DIGITAL

Tot i que no se sap ben bé qui va fer el primer sistema de registres mèdics en digital o electrònic, segons Dalianis [6], durant els anys 60 i 70, els suecs van construir el primer sistema de registres electrònics per un entorn mèdic. Es va fer a l’Hospital Universitari de Karolinska.

Amb el que no hi ha discrepància és amb el primer sistema utilitzat de manera general. Va ser desenvolupat per la Universitat Harvard i es deia *COSTAR* [6] (*Computer-stored ambulatory record*). Amb el temps van anar sorgint nous sistemes creats de manera interna a algun hospital o de manera pública. Més tard mentre progressaven l’electrònica aquests sistemes que estaven als hospitals es van anar fent més i més grans fins al punt de començar-se a comunicar-se entre ells fent que apareixes el termini *HIS*, *Health/Healthcare Information System*.

L’any 2005, l’OMS [7] va començar a posar normes perquè els *HIS* funcionessin de manera correcta, tinguessin la seguretat esperada i a més van predir que l’any 2015 tots els hospitals haurien de tenir-ne un. Segons Gillum [3], l’any 2011, més del 50 % dels metges van reportar que utilitzaven un *HIS*. A Amèrica del Nord el primer hospital completament digital va ser a Toronto l’any 2015.

## ANTECEDENTS

Els antecedents que es busquen en aquest apartat són aplicacions internes dels hospitals i que, preferiblement, es comuniquin amb el motor d’integració.

Durant la recerca d’aquests antecedents només es trobaven aplicacions públiques, i que per tant no són gaire útils pel producte a desenvolupar. La majoria d’aquestes són bases de dades de malalties. Un exemple d’aquestes aplicacions són *UpToDate* [8] o *Dynamed* [9].

Les aplicacions que són antecedents per l’aplicació que es farà han sigut impossible de trobar, ja que no hi ha aplicacions d’ús intern que siguin públiques.

## NECESSITATS DE LA INFORMACIÓ

Per poder entendre i desenvolupar l’aplicació s’han de saber algunes coses bàsiques del funcionament dels hospitals. Primer s’ha de saber que és el *HIS*, quins components té i perquè és tan important. Després s’ha de saber que és el motor d’interoperabilitat, ja que és el pilar fonamental de l’aplicació a fer. Tot seguit s’ha de saber quins protocols usen el *HIS* i el motor d’interoperabilitat per a comunicar-se, ja que pot ser bastant important a l’hora d’entendre algunes dades que pot rebre el producte. Per acabar es necessita saber quines tecnologies existeix per a desenvolupar servidors web, aplicacions mòbils i per a enviar notificacions als dispositius mòbils. La majoria d’aquesta informació, la que correspon als apartats 2.3.1, 2.3.2 i 2.3.3, ha estat extreta del llibre anomenat *Health Information Systems - Architectures and Strategies* [10].

### HEALTHCARE INFORMATION SYSTEM (HIS)

El gran nucli d’informació que hi ha en un hospital és el *Healthcare o Health Information System* (*HIS*). Aquest sistema és l’encarregat de guardar i processar tota la informació que genera un hospital.

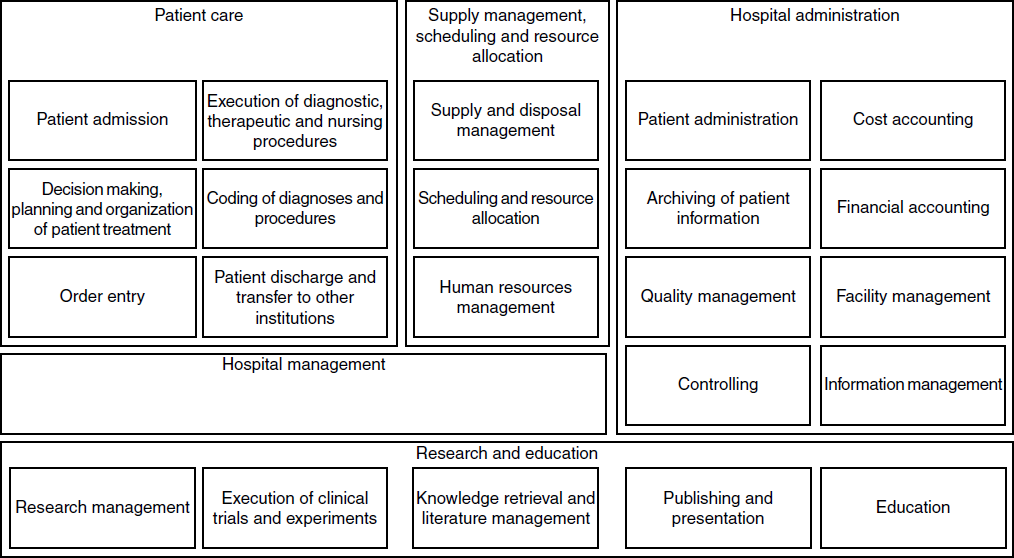


Figura .: Esquema de les dades que emmagatzema el *HIS* [11]. Font: *Health Information Systems - Architectures and Strategies* [10].

En aquest esquema, es pot veure totes les dades que tractarà el *HIS*, però pot observar-se que falten molts serveis que ofereix tant un hospital com el CAP, per exemple el servei de pediatria i de cardiologia d’entre d’altres. Tots aquests serveis que no es veuen a la imatge, formen part de *HIS*, però són un subsistema d’aquest.

En definitiva, el *HIS* es pot definir com un conjunt de subsistemes d’informació dels diferents serveis que ofereix tota institució mèdica que emmagatzema, processa i pren decisions sobre les dades rebudes.

### MOTOR D’INTEROPERABILITAT

Perquè el *HIS* pugui funcionar correctament, necessita dades d’altres hospitals per poder tenir tota la informació completa. Aquesta comunicació entre diferents *HIS* d’hospitals la proporciona el motor d’interoperabilitat que té cada institució mèdica. A més aquest motor d’interoperabilitat pot generar una *API* perquè aplicacions externes al *HIS*, com aplicacions web puguin comunicar-se. Per tant el motor d’interoperabilitat és l’encarregat de comunicar-se amb els elements externs al *HIS* que controla.

Aquest motor d’interoperabilitat és la base de la *e-health*, conjunt de tecnologies de la informació i comunicació que s’utilitza en l’entorn sanitari, ja que sense ell costaria molt fer una comunicació bona entre diferents *HIS*. A Espanya, segons un estudi fet per la *Healthcare Information and Management Systems Society* (*HIMSS*) l’any 2014 [12], el 93 % dels hospitals tenen un motor d’interoperabilitat integrat.

### PROTOCOLS ENTRE EL HIS I EL MOTOR D’INTEROPERABILITAT

Aquests protocols o estàndards que es mostraran a continuació també poden ser usats per la interoperabilitat dels *HIS*.

##### HL7

*Health Level 7* és l’estàndard més usat per a la comunicació entre el *HIS* i el motor d’interoperabilitat, com entre motors d’interoperabilitat. Aquest estàndard envia missatges a un destinatari, però no inclou imatges [13, p. 2].

Aquest protocol té dues versions, la versió 2 i la versió 3. La principal diferencia que tenen aquestes versions és que la més nova, la versió 3, genera un missatge més fàcil de llegir, gràcies a la implementació del *HL7* *RIM* (*Reference Information Model*). Per tant com es pot veure en les següents imatges, es pot veure que és molt més clar llegir un missatge de la versió 3, que de la versió 2.

El principal problema que tenen aquestes dues versions és que no són compatibles entre elles. Per tant si s’implementa alguna solució usant una de les dues versions, és difícil canviar-la [14].

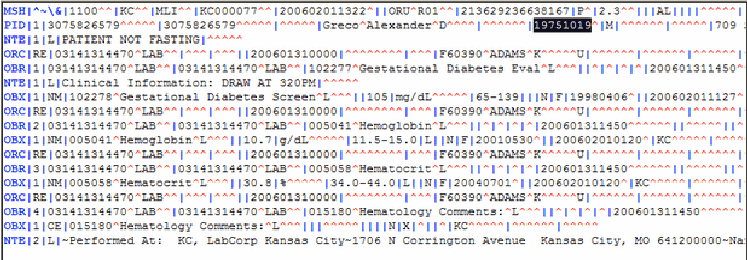


Figura .: Exemple de missatge de *HL7* *v2*. Font: *HL7 in Personal Health* [15, p. 7].

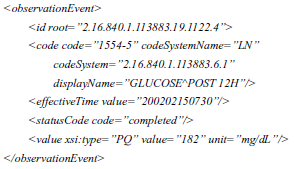


Figura .: Exemple del missatge de *HL7* *v3*. Font: *HL7 in Personal Health* [15, p. 7].

##### DICOM

L’estàndard *Digital Imaging and Communications in Medicine* (*DICOM*) és el que gestiona l’intercanvi d’imatges que genera el *HIS*. Com pot ser una radiografia, una ressonància magnètica, etc.

A diferència del *HL7*, aquest estàndard necessita fer una connexió *TCP-IP* per a enviar la imatge. Aquest estàndard codifica i comprimeix la imatge per a facilitar el transport i la seguretat.



Figura .: Exemple d’una imatge generada pel protocol *DICOM*. Font: *DICOM Basics using Java - Extracting Image Data* [16].

##### ISO/IEEE 11073

Aquest protocol, creat per la família d’estàndards internacional, és el que permet poder intercanviar senyals vitals o biomèdiques. Aquest protocol ofereix un conjunt de tecnologia per poder connectar-ho a un pacient i que funcioni sense cap configuració extra. És un protocol molt útil per al monitoratge dels pacients.

### TECNOLOGIES PER A DESENVOLUPAR UN SERVIDOR WEB

En aquest subapartat es veuran tres dels frameworks més usats per al desenvolupament de pàgines web, *Ruby on Rails*, *Spring* i *Django*. Es veurà quines característiques tenen cadascun i que els fa diferents.

La web que s’haurà de fer és relativament senzilla, només necessita que es pugui connectar a una base de dades. Ja que només servirà per configurar les alertes que puguin rebre els usuaris per departament.

#### RUBY ON RAILS

* Llenguatge de programació
* Característiques

Les webs més famoses que usen Ruby són GitHub, Airbnb i Twitch.

#### SPRING

* Llenguatge de programació
* Característiques

La web més famosa que usa Spring és Netflix.

#### DJANGO

* Llenguatge de programació
* Característiques

Les webs més famoses que usen Django són Spotify, Instagram i YouTube.

### TECNOLOGIES PER A DESENVOLUPAR UNA APLICACIÓ MÒBIL

En aquest subapartat es veuran tres dels *frameworks* més usats per al desenvolupament d’aplicacions mòbils, *React Native*, *Xamarin* i *Flutter*. Es veurà quines característiques tenen cadascun i que els fa diferents.

El requisit principal que es necessita per l’aplicació de mòbil és que ha de poder enviar notificacions. Ja que sense notificacions no es poden enviar alertes per als usuaris i no compliria el propòsit que es vol amb l’aplicació. Per tant si hi ha algun *software* que no permeti complir aquest requisit, es descartarà per complet.

A més hi ha uns requisits menys prioritaris que són que permeti l’ús d’animacions i que sigui multiplataforma, això vol dir que el codi fet a aquella aplicació funcioni tant per *iOS* com per *Android*, que són els dos sistemes operatius dominants en els sistemes mòbils amb el 99% del mercat [17].

A continuació es mostrarà un conjunt dels *frameworks* més usats per a la creació i desenvolupament d’aplicacions mòbils.

#### REACT NATIVE

L’any 2015, *Facebook* va desenvolupar *React Native* [18], és un *framework* multiplataforma, això vol dir que et permet desenvolupar una aplicació per a dispositius mòbils, ordinadors i webs.

El llenguatge per usar aquest *framework* és *JavaScript* i utilitza l’*API* adequada per a cada dispositiu. Això significa que busca ser natiu en cada plataforma que s’executa. A més gràcies a *JavaScript*, es pot veure els canvis fets al dispositiu una vegada deses el codi. A més, *React Native* és el segon repositori de *GitHub* amb més contribuïdors. De tal manera que la seva comunitat està molt viva.

*React Native* és un dels *frameworks* més populars. Així ho demostra el llistat d’aplicacions que usen aquest *framework*, entre elles hi ha *Facebook*, *Instagram*, *Skype*, *Discord*, *Tesla*.

#### XAMARIN

És un *framework* desenvolupat per *Microsoft*. *Xamarin* [19] estén de *.NET* i utilitza *C#* com el seu llenguatge principal.

Segons la seva pàgina web, *Xamarin* és un *framework* multiplataforma i natiu. D’igual manera que *React Native*, *Xamarin* és *Open Source* i té una gran comunitat darrere.

Les aplicacions més destacades que utilitzen *Xamarin* són, l’aplicació dels premis *Oscar*, *UPS* i *BBVA*.

#### FLUTTER

És un *framework* de desenvolupament d’aplicacions mòbils, web i aplicacions d’ordinadors desenvolupat per *Google*. *Flutter [19]* es desenvolupa utilitzat *Dart*. És el més nou dels dos *frameworks* comentats anteriorment.

Segons la seva pàgina web, *Flutter*, és un *framework* molt semblant als anteriors, et diuen que és fàcil de desenvolupar, fer la UI és flexible i a més et prometen un rendiment natiu per a les aplicacions.

Entre les aplicacions més conegudes que utilitzen *Flutter* hi ha *Ebay*, *Stadia* i el *New York Times* entre d’altres.

### TECNOLOGIES PER A ENVIAR I REBRE NOTIFICACIONS AL MÓBIL

En aquest subapartat es veuran els mecanismes que hi ha per a poder enviar i rebre notificacions als dispositius mòbils.

S’ha de considerar que per enviar es pugui fer des de el servidor web.

#### FIREBASE CLOUD MESSAGING

La solució més coneguda és *Firebase* [20], una plataforma pel desenvolupament d’aplicacions mòbils, de propietat de *Google*.

Aquesta plataforma té el seu sistema d’enviar notificacions, *Firebase Cloud Messaging*. Aquest sistema, segons la seva pàgina web, ofereix una solució multiplataforma per a diferents sistemes operatius. Addicionalment *Firebase* ofereix un *SDK* per a poder enviar notificacions des de un servidor web o qualsevol altra aplicació.

#### ONESIGNAL

#### EXPO PUSH NOTIFICATIONS

# OBJECTIUS I ABAST

En aquest apartat s’explicaran els objectius que té aquest treball, tant pel client com pel producte. A més s’inclourà quin és el públic objectiu i l’abast d’aquesta aplicació.

## OBJECTIUS DEL CLIENT

1. Demostrar que amb l’aplicació millora l’efectivitat de visualitzar un esdeveniment.
2. Fer una aplicació multiplataforma per arribar al màxim nombre d’usuaris.
3. Gestionar les alertes que disposin els usuaris a través d’una pàgina web.
4. Rebre notificacions quan s’ha superat el llindar del valor establert en l’alerta.
5. Integrar l’aplicació a l’hospital sense que suposi una dificultat extra pel client.

## OBJECTIUS DEL PRODUCTE

1. Dissenyar un sistema per a poder afegir hospitals i diferents motors d’interoperabilitat.
2. Tenir una seguretat adient d’una aplicació mèdica.
3. Evitar que la comunicació entre el servidor i l’aplicació mai es perdi per problemes del producte.
4. Seguir les lleis de protecció de dades dels usuaris.

## PÚBLIC POTENCIAL

Aquesta aplicació està pensada per a ser utilitzada per a qualsevol treballador d’un hospital o institució mèdica que demani accés al producte. Això vol dir que si estàs donat d’alta i el mateix hospital t’ha donat un usuari i contrasenya de l’aplicació, es podrà accedir i per tant usar-la.

## ABAST

Aquest producte està pensat per a ser utilitzat per a qualsevol hospital que vulgui usar aquesta aplicació independentment de la empresa del seu motor d’interoperabilitat.

Dit això, no vol dir que el disseny de la base de dades i del servidor no s’hagi pensat en el cas que es puguin afegir noves institucions mèdiques o usar altres empreses que disposin d’un motor d’interoperabilitat. Per tant l’abast d’aquest projecte pot ser tant “petit” com centrar-se en un únic motor d’interoperabilitat i un únic hospital, com fins a convertir-se en una aplicació que s’utilitza globalment.

# METODOLOGIA

Aquest TFG tindrà un procés cíclic en la part del producte. Com es pot saber utilitzar una metodologia *scrum* sent només un desenvolupador pot ser difícil, per tant s’usarà una petita variant. El principal objectiu d’aquesta metodologia és que en cada *sprint*, que serà d’una setmana, l’aplicació afegeixi funcionalitats sense trencar les anteriors.

En el control de versions que s’utilitzarà, hi haurà tres branques per defecte, *master*, *develop* i *documentation*. A part quan s’hagi d’afegir una funció al producte, es crearà una branca amb les sigles de la fita que li correspongui i el seu nom. Per exemple *cs\_login*, en el cas de fer la funció del servidor web d’iniciar sessió.

A més aquest projecte tindrà tres fites relacionades amb les entregues que hi ha de la documentació del projecte.

La primera ve relacionada amb l’entrega de l’avantprojecte, en aquesta fita s’haurà de dissenyar tot el producte. Aquest disseny ha de constar els diagrames *UML* de l’aplicació i del servidor, el diagrama relacional de la base de dades, el flux de l’aplicació i com es veurà visualment. Aquesta fita suposarà la transició de la preproducció del producte i l’inici de la producció.

La segona fita començarà la producció del servidor i està marcada amb l’entrega de la memòria intermèdia. Aquesta fita se separarà en tres *sprints* i, a arribar la data de la fita, s’haurà de tenir el servidor d’*InterSystems IRIS for Health* i el servidor web. Aquests tres *sprints* corresponen al servidor web i es farà a tres setmanes de la data límit de la fita. Abans de fer aquesta web es crearà el servidor d’*InterSystems IRIS for Health*. El motiu de què, la producció del servidor d’*InterSystems IRIS for Health*, no estigui tallant en *sprints* és perquè en el moment de l’elaboració d’aquest apartat no se sap quan es trigarà a fer-ho. Així que s’ha deixat una longitud de temps equivalent a set *sprints*, per evitar inconvenients al futur.

La tercera i última fita coincideix amb la data de l’entrega de la memòria final i servirà per desenvolupar l’aplicació. Aquesta última fita estarà composta de set *sprints* que aniran des de connectar-se amb el servidor web fins a fer la interfície d’usuari.

# DEFINICIÓ DE REQUERIMENTS FUNCIONALS I TECNOLÒGICS

Aquest projecte necessita separar els requeriment funcionals del servidor i de l’aplicació, ja que són bastants diferents.

## REQUERIMENTS FUNCIONALS DEL SERVIDOR D’INTERSYSTEMS IRIS FOR HEALTH

1. **Crear una base de dades per a administrar les alertes.**
2. **Crear una *API* que retorni els valors de la base de dades**
3. **Fer que la *API* pugui rebre peticions de tot internet.**

Per provar que tots aquests requeriments funcionin, farà falta un ordinador amb accés a internet i que tingui instal·lat el motor d’interoperabilitat d’*InterSystems IRIS for Health*.

## REQUERIMENTS FUNCIONALS DEL SERVIDOR WEB

1. **Iniciar sessió al servidor web.**
2. **Tancar sessió al servidor web.**
3. **Crear noves institucions mèdiques.**
4. **Crear usuaris per l’aplicació i servidor.**
5. **Crear alertes per l’aplicació.**
6. **Crear rols per l’aplicació.**
7. **Assignar o eliminar alertes als rols.**
8. **Esborrar rols.**
9. **Esborrar institucions mèdiques.**
10. **Esborrar usuaris.**
11. **Poder visualitzar la llista d’usuaris, de rols, d’alertes i d’institucions mèdiques.**
12. **Poder filtrar la llista d’usuaris, de rols, d’alertes i d’institucions mèdiques.**
13. **Poder buscar en la llista d’usuaris, de rols, d’alertes i d’institucions mèdiques.**

Per provar que tots aquests requeriments funcionin, farà falta un ordinador amb accés a internet i que pugui obrir un navegador web.

## REQUERIMENTS FUNCIONALS DE L’APLICACIÓ

1. **Iniciar sessió a l’aplicació.**
2. **Tancar sessió a l’aplicació.**
3. **Poder visualitzar la llista d’alertes per usuari.**
4. **Buscar l’alerta desitjada.**
5. **Poder canviar la contrasenya del seu compte.**
6. **Poder demanar l’eliminació de l’usuari.**
7. **Rebre notificacions quan canvi de gravetat l’alerta.**

Per provar que tots aquests requeriments funcionin, farà falta un dispositiu mòbil amb accés a internet. El sistema operatiu ha de ser *Android* o *iOS*.

## REQUERIMENTS TECNOLÒGICS

Els requeriments tecnològics per poder executar, tant l’aplicació com els servidors d’*InterSystems IRIS for Health* i web són bastant simples, en àmbit d’usuari.

Per començar el servidor d’*InterSystems IRIS for Health* només té un requeriment tecnològic. Tenir un ordinador o dispositiu que pugui instal·lar el motor d’interoperabilitat. Aquest requeriment només l’usarà el client, ja que aquest necessita poder extreure les seves dades. Així que el client disposarà d’un requeriment tecnològic més que és disposar d’un servidor físic que tingui el *HIS* i el motor d’interoperabilitat per poder accedir a les dades d’aquell hospital.

El servidor web només té un requeriment tecnològic. Tenir un ordinador o dispositiu capaç de connectar-se a una pàgina web. Preferiblement que sigui un ordinador o que tingui una pantalla en mode horitzontal per a poder visualitzar bé les dades. Encara que s’intentarà que el servidor web es pugui visualitzar en els dispositius amb pantalles més petites.

Els requeriments tecnològics de l’aplicació és tenir un dispositiu mòbil, amb el sistema operatiu *Android* o *iOS*. Aquest dispositiu hauria de disposar de bona autonomia i connexió a internet.

# DESENVOLUPAMENT

En aquesta secció s’explicarà tota la producció del producte a produir en aquest TFG. La secció s’ha separat en quatre subapartats, que corresponen a les quatre tasques principals per fer aquest producte: disseny, servidor d’*InterSystems IRIS for Health*, servidor web i aplicació.

## DISSENY

Abans de començar a la producció del producte s’ha de dissenyar alguns aspectes. L’element principal a dissenyar és la base de dades, ja que serà la que unifiqui l’aplicació i els dos servidors. A més s’ha de dissenyar bé el servidor web i l’aplicació, per això s’han elaborat uns diagrames de classe *UML* i diagrames de flux *BPMN*. A més en el producte s’ha dissenyat una *UI*, tant pel servidor web com per l’aplicació mòbil.

### BASES DE DADES

Per poder fer la base de dades s’ha pensat en el futur i per si l’aplicació creix significativament.

Així que aquest diagrama relacional (Fig. 6.1), té constància de les diferents institucions mèdiques que puguin usar l’aplicació. Per a fer això, per a cada institució mèdica es guarda la seva ubicació: el país on està situada i el seu nom, per exemple “Hospital de Mataró”. Per acabar es necessiten les dades perquè l’aplicació mòbil pugui accedir a la *API* que es generarà al seu motor d’interoperabilitat. D’aquestes dades de la *API*, només es guardarà la *URL*, és a dir, la part comuna de totes les *API* que tingui aquella institució. Per exemple, com es veurà més endavant, aquesta *URL* pot ser “https://www.tfginformatica.dev/rest/”. A més s’haurà de tenir desat per a cada institució les credencials per a poder accedir a aquella *API*.

A continuació hi ha la taula de rols, aquests rols poden ser els diferents serveis o grups de treballadors de cada institució mèdica que facin ús de les mateixes alertes. Per exemple, pot haver-hi un rol que sigui “Infermeria”. Aquest rol tindrà un conjunt d’alertes exclusives per aquest rol i uns empleats.

En la taula d’empleats, només es tindrà constància del seu rol, les seves credencials d’accés al producte i, per acabar, algunes dades personals, com pot ser el nom i cognoms.

En la taula d’alertes es necessita saber a quin rol pertany, i a continuació com poder accedir a la *URL* de la *API* especifica per aquell rol. En aquest cas ha de ser una continuació de la *URL* proporcionada per la institució mèdica. Per acabar l’alerta també ha de saber quan un valor és correcte o incorrecte i també ha de tenir dades per a l’aplicació mòbil.

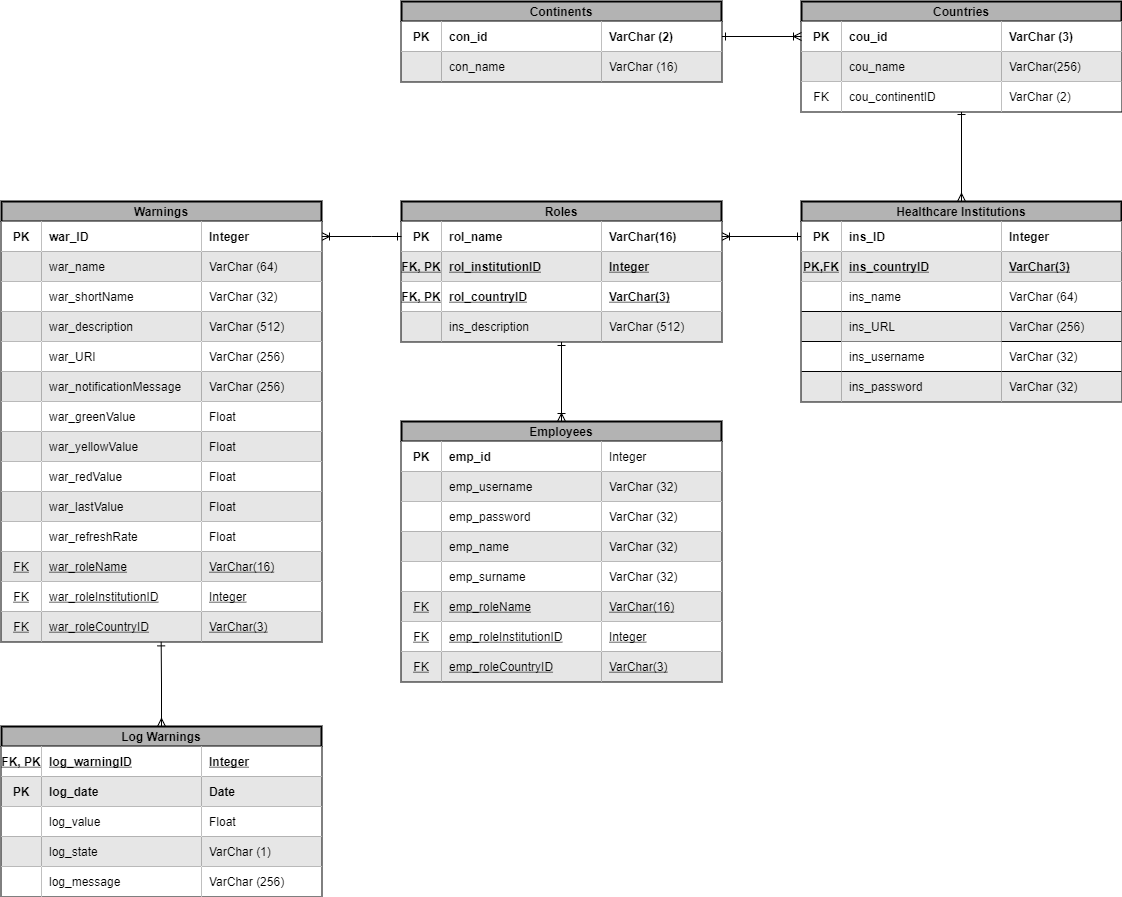
Aquesta alerta tindrà guardat un historial dels valors que ha proporcionat en el passat. 

Figura .: Diagrama ERD de la base de dades. Font: Elaboració pròpia.

### SERVIDOR WEB

El servidor estarà fet usant el *framework Spring*. El motiu d’utilitzar aquest *framework* és perquè és bastant utilitzat en el món de les pàgines web, és fàcil de trobar recursos que ajudin el desenvolupament d’un servidor web i, per acabar, utilitza el llenguatge de programació *Java*.

Per a poder programar aquest servidor, s’utilitzarà el *IDE* *IntelliJ* de *JetBrains*. Aquest *IDE* és l’escollit, ja que és un dels *IDE* més utilitzats per a programar en *Java* i a més l’empresa desenvolupadora té una gran oferta de *IDE* de diferents llenguatges que fa que canviar d’un llenguatge a un altre no suposi un canvi molt gran.

El servidor web serà l’encarregat de gestionar la configuració de l’aplicació. Això significa que serà l’únic que tindrà accés a la base de dades. Així que qualsevol canvi, o inserció d’alguna fila a la base de dades ho haurà de fer el servidor.

A més el servidor serà l’encarregat d’enviar les notificacions al mòbil dels usuaris. Per a poder fer això el servidor tindrà un *thread* per a cada alerta que hi hagi i aquest *thread* anirà informant els usuaris que estiguin assignats a aquesta alerta de si hi ha hagut un canvi de color. En el cas que hi hagi, s’enviarà una notificació a l’usuari.

#### DIAGRAMES

La part principal per a poder començar a programar és saber quin disseny i com es comportarà el servidor. Així que s’ha elaborat un diagrama de classes i un parell de diagrames *BPMN*.

##### DIAGRAMA UML DE CLASSES

Aquest diagrama mostrà com estaran comunicades les classes i a més quins mètodes i atributs tindrà.

És un diagrama de classe senzill, ja que el *framework Spring* proporciona tota la part de connectar-se a la pàgina web. D’aquesta manera només s’ha de gestionar del funcionament de la web.

Les classes que estan més amunt són les encarregades de la base de dades. La primera, *DatabaseDAO*, és la que farà les crides i la segona, *DatabaseController*, farà de façana.

Després hi ha els controladors web, el primer, *APIController*, serà el que gestioni la *API*, que serà en la que es comunicarà l’aplicació. Les dues següents seran les que gestionin la web i estan separades en les que només ensenyen una pàgina HTML, WebGetController, i les que afegeixen dades a la base de dades, WebPostController.

Després hi haurà la classe TFGApplication, que és on està el main i és qui fa executar el servidor web. Tot seguit apareix la classe NotificationController, que serà qui creí les alertes i enviï les notificacions als usuaris de l’aplicació mòbil.

Per acabar hi ha les classes del domini, on es crearan els objectes, institució mèdica (HealthcareInstitution), rol (Role), empleat (Employee) i alerta (Warning).

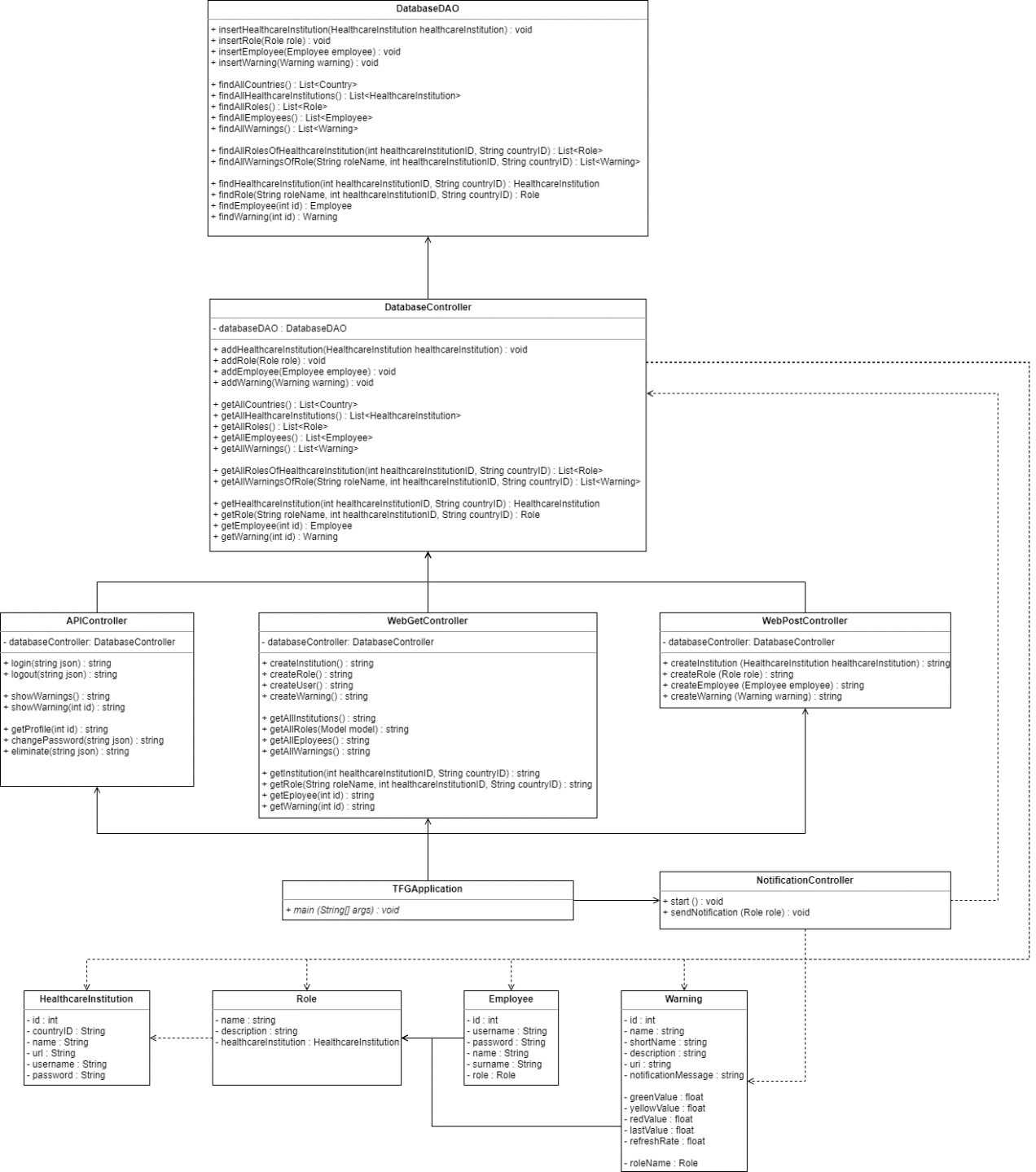


Figura .: Diagrama *UML* de classes del servidor web. Font: Elaboració pròpia.

##### DIAGRAMES BPMN

A continuació es mostraran dos diagrames de flux *BPMN*. El primer és per mostrar com funciona una creació d’una institució mèdica. S’ensenyarà només la creació, ja que per a tots els elements que utilitzen la base de dades des del servidor té el mateix funcionament.

El procés serà simple, l’usuari s’autentificarà al servidor web, a continuació, l’usuari anirà a la pàgina per a crear la institució mèdica. Introduirà les dades per a que es pugui crear aquella institució i si aquelles dades son correctes la institució mèdica es crearà.

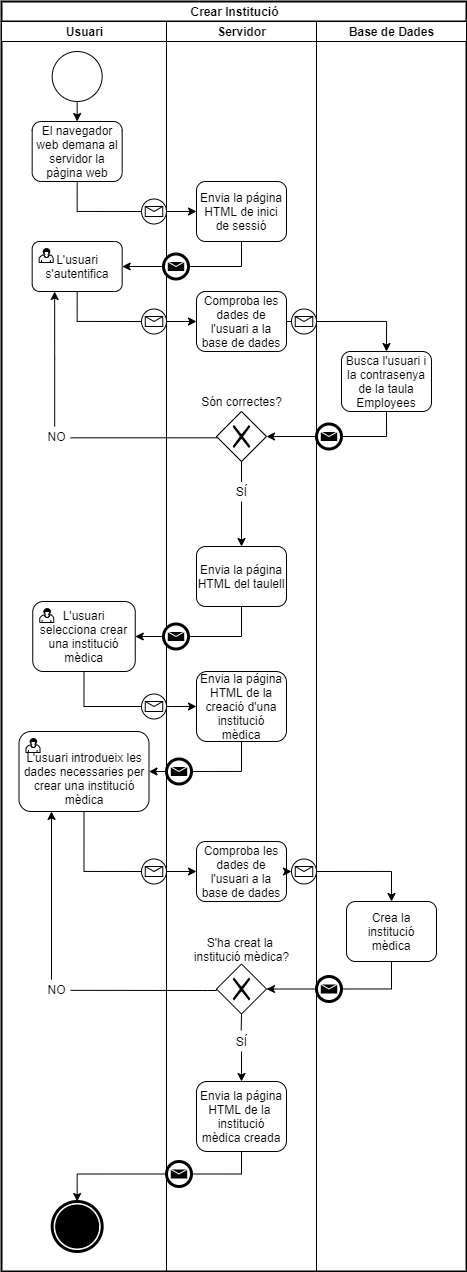


Figura .: Diagrama *BPMN* de la creació d’una institució mèdica. Font: Elaboració pròpia.

El segon diagrama és referent a com es gestionen les notificacions. La columna que es diu alerta n’hi haurà tantes alertes com alertes hi hagi a la base de dades, ja que cada una serà un *thread*.

El funcionament del *thread*, serà que individualment es demanaran les dades a la *API* que tingui assignada aquella alerta. Una vegada ja tingui el valor fara una comprovació per saber si ha d’enviar una notificació o no. Una vegada hagi acabat, el *thread* es dormirà el temps necessari i, quan hagi passat aquell temps, tornarà a fer la funcionalitat un altre cop.

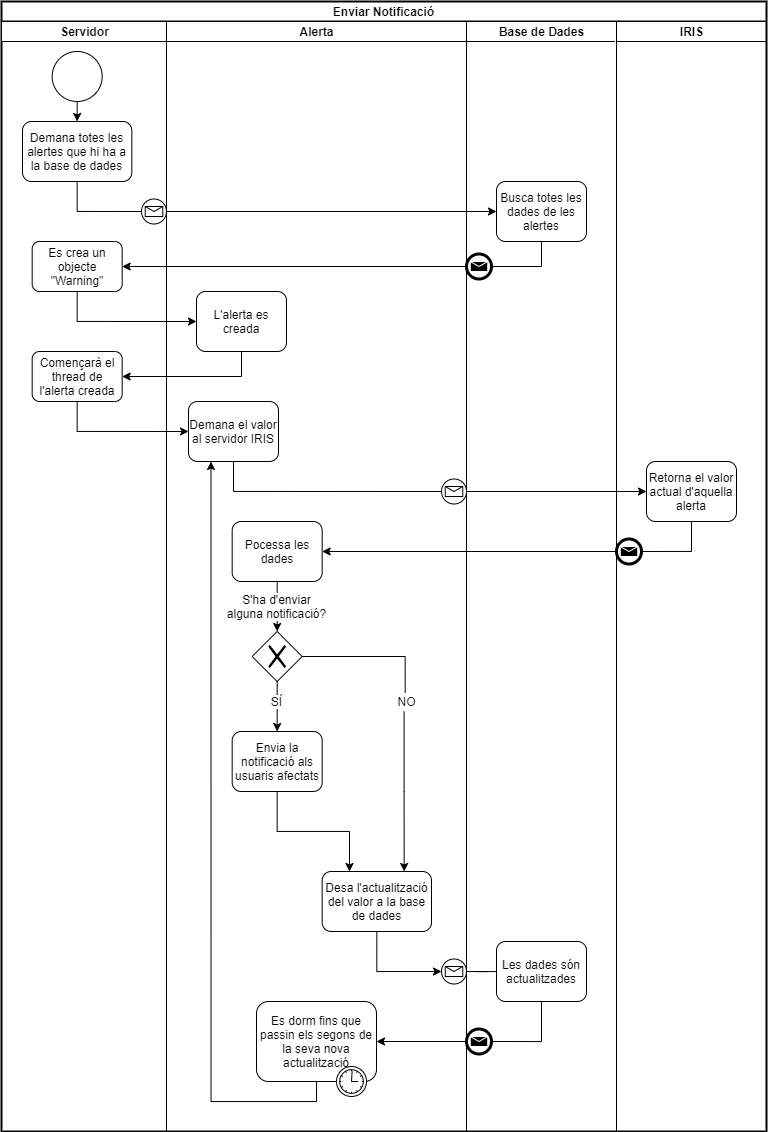


Figura .: Diagrama *BPMN* de la gestió de les notificacions. Font: Elaboració pròpia.

#### UI

Per la interfície d’usuari, *UI*, s’han pensat en colors que fossin semblants als mèdics, aquests colors, després de buscar algunes referències, s’ha optat per un blau cel i blanc. En l’aplicació també seran els mateixos colors.

Per poder donar contrast en la creació o eliminació de dades, s’ha optat pel color verd i vermell respectivament.

Per a diferenciar els camps de textos amb els textos no editables, hi haurà un rectangle blanc i el text serà en negre. Com es pot veure a la imatge següent.



Figura .: Disseny de la *UI* de l’inici de sessió en el servidor web. Font: Elaboració pròpia.

Una vegada iniciada la sessió es mostrarà la capçalera de la pàgina web, que constarà de quatre botons de navegació per a facilitar l’ús de l’aplicació. A més hi haurà un peu de pàgina amb algunes dades referent al TFG.

El següent disseny (Fig. 6.6) és de la pàgina d’inici, una vegada s’hagi iniciat sessió. S’ha pensat a posar botons per les llistes i creació.



Figura .: Disseny de la *UI* de la pàgina del taulell en el servidor web. Font: Elaboració pròpia.

A continuació es veurà com són els dissenys per llistar, crear i veure les dades individualment. Per tal de fer-ho més senzill, només es mostraran els dissenys de les institucions mèdiques.



Figura .: Dissenys de la *UI* de la llista d’institucions mèdiques en el servidor web. Font: Elaboració pròpia.

La llista d’institucions mediques (Fig. 6.7) tindrà un format de taula. A la capçalera d’aquesta taula hi haurà les eines de filtratge al costat del nom de la columna. Per diferenciar entre les files i columnes s’ha optat a fer un degradat amb un color de la mateixa família cromàtica que el blau cel, però, més fosc.

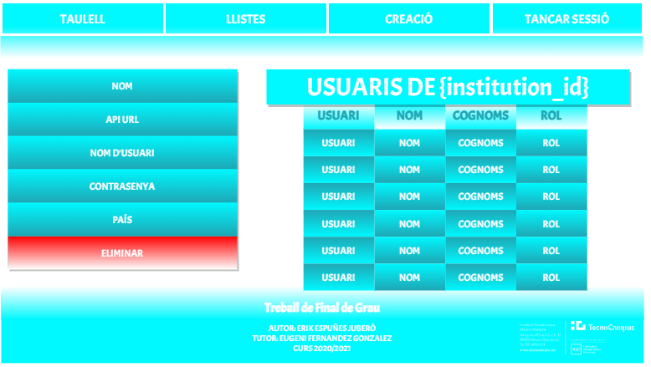


Figura .: Disseny de la *UI* de la visualització d’una institució mèdica en el servidor web. Font: Elaboració pròpia.

A continuació es mostra la visualització d’una institució mèdica (Fig. 6.8) en detall. Aquí hi haurà totes les dades necessàries i es mostraran els usuaris que pertanyen a la institució mèdica. També es podran esborrar o editar les dades de la institució que s’estigui veient.



Figura .: Disseny de la *UI* de la creació d’una institució mèdica en el servidor web. Font: Elaboració pròpia.

Per acabar hi ha el disseny de la creació d’una institució mèdica. Aquí hi haurà tots els camps necessaris per a poder crear la institució. Totes les altres creacions de dades, tindrà un aspecte molt semblant.

### APLICACIÓ MÒBIL

L’aplicació estarà feta usant el *framework React Native*. El motiu d’utilitzar aquest framework és perquè és bastant utilitzat en les majories d’aplicacions importants en l’àmbit del mòbil, com per exemple Instagram. A més és multiplataforma, això vol dir que no s’ha de fer el codi dues vegades, ja que és compatible amb iOS i Android. *React Native*, com també el seu pare *React*, utilitza el llenguatge de programació JavaScript. A més per a poder ajudar a compilar i executar en un mòbil s’ha usat el *framework* *expo*, que ofereix poder tenir l’aplicació executant-se al mòbil mentre s’està desenvolupant.

Per a poder programar aquesta aplicació, s’utilitzarà l’IDE WebStorm de JetBrains. S’ha escollit aquest IDE perquè és un IDE de l’empresa JetBrains específic per JavaScript. I com s’ha comentat anteriorment la transició de diferents IDE de la mateixa empresa no suposa un canvi molt gran, respecte a la UI i les dreceres de teclat.

L’aplicació mòbil serà l’encarregada d’avisar a l’usuari que ha succeït alguna anomalia en alguna de les alertes que té assignades. A més serà d’ús propi per l’usuari, això vol dir que ho podrà consultar sempre que vulgui.

#### DIAGRAMES

A l’hora de poder començar a programar l’aplicació, és necessari saber com estaran organitzades les classes. A més s’ha de pensar bé el flux que tindran alguns casos d’ús, que es veuran en el següent subapartat, per a fer una bona gestió dels servidors web i d’IRIS.

##### DIAGRAMA UML DE CLASSES

Aquest diagrama UML està orientat a les classes o scripts que hi haurà en el desenvolupament de l’aplicació amb *React Native.*

L’element clau d’aquest diagrama són les classes que es comunicaran amb el servidor web i el servidor d’IRIS o de l’API corresponent. S’ha pensat que per millorar la seguretat de l’aplicació i del tractament de dades, de posar el patró façana. Així que les escenes, que són les pantalles que es veuran en el mòbil, es comunicaran directament amb una classe, que és el CommunicationManager, i aquesta distribuirà o gestionarà la comunicació amb el servidor web i l’API corresponent.

En la classe Controller, pot saltar a la vista l’únic mètode anomenat waitForResponse. Aquest mètode té la funció de bloquejar aquella classe fins que no arribi una resposta de l’API. És necessari fer un bloqueig, ja que les crides a una API són asíncrones, així que no se sap quan pot trigar a arribar les dades necessàries. Així que com és un mètode comú entre WebServerController i APIController, s’ha fet que aquestes dues classes siguin filles de la classe Controller.

Per acabar les classes d’escena seran com si fossin una pàgina HTML, així que només s’encarregaran de la part gràfica de l’aplicació i de rebre les comandes directament de l’usuari. Com aquestes classes tenen una variable i un mètode en comú, s’ha pensat en què han d’estendre d’una classe abstracta, anomenada Scene, per a tenir un bon disseny de l’aplicació i per no repetir codi innecessari.

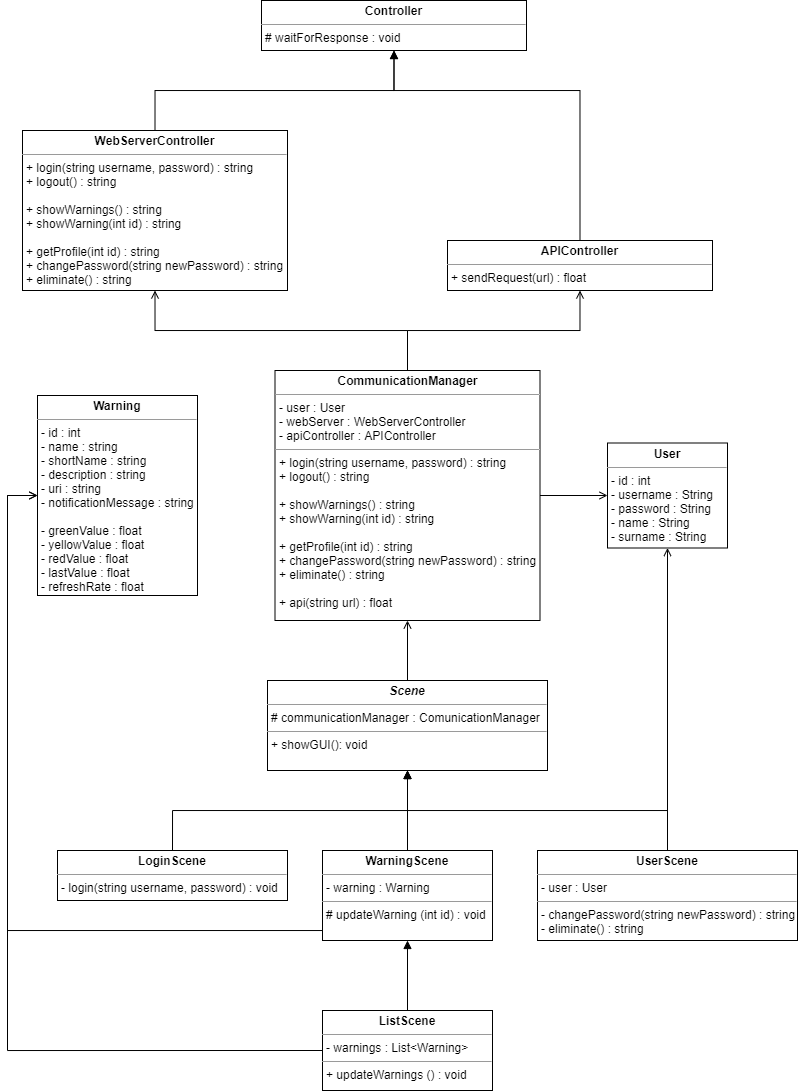


Figura .: Diagrama UML de classes de l’aplicació mòbil.

##### DIAGRAMES BPMN

A continuació es veuran tres diagrames BPMN.

El primer d’ells és per saber com funciona l’inici de sessió en l’aplicació. Aquest diagrama és comú per a tots aquells casos d’ús que no fa falta una comunicació directa amb IRIS. Per tant els casos d’ús de tancar sessió, canviar la contrasenya i demanar l’eliminació del compte seran semblants al següent diagrama.

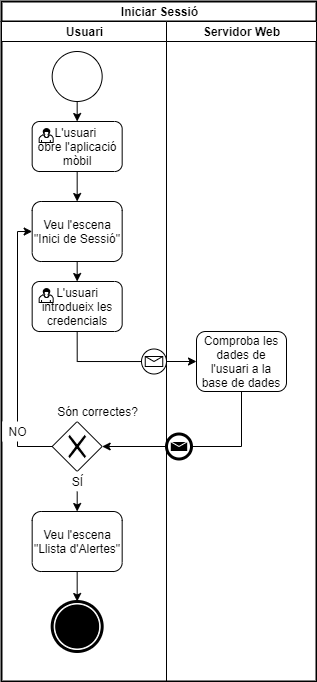


Figura .: Diagrama BPMN de l’inici de sessió en l’aplicació.

El segon diagrama és sobre la llista d’alertes. Aquesta llista, a part de comunicar-se amb el servidor web, s’ha de comunicar amb el servidor d’IRIS o de l'API que proporcioni la institució mèdica.

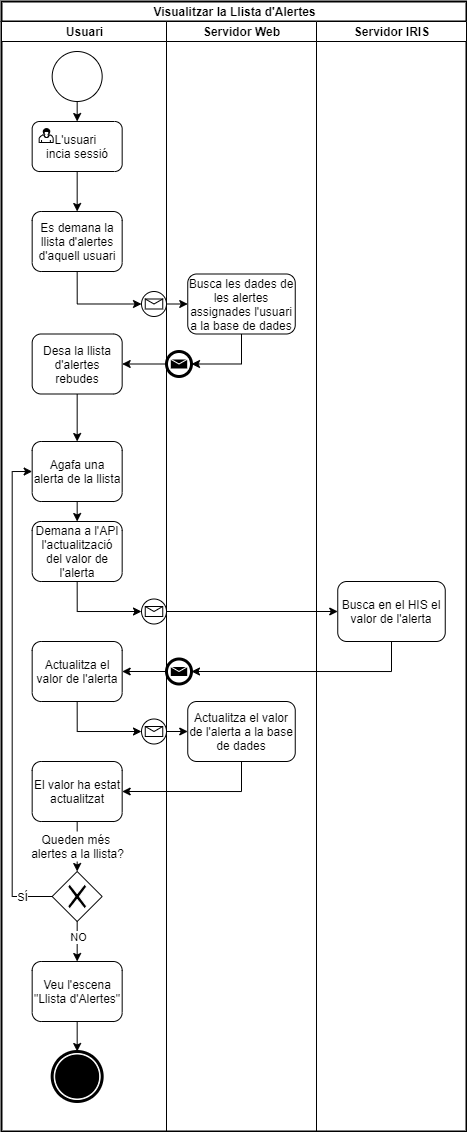


Figura .: Diagrama BPMN de la visualització de la llista d’alertes en l’aplicació.

El tercer i últim diagrama és sobre la visualització d’una alerta. És un diagrama bastant semblant a l’anterior però sense la iteració en la llista, ja que només s’actualitzarà una alerta.

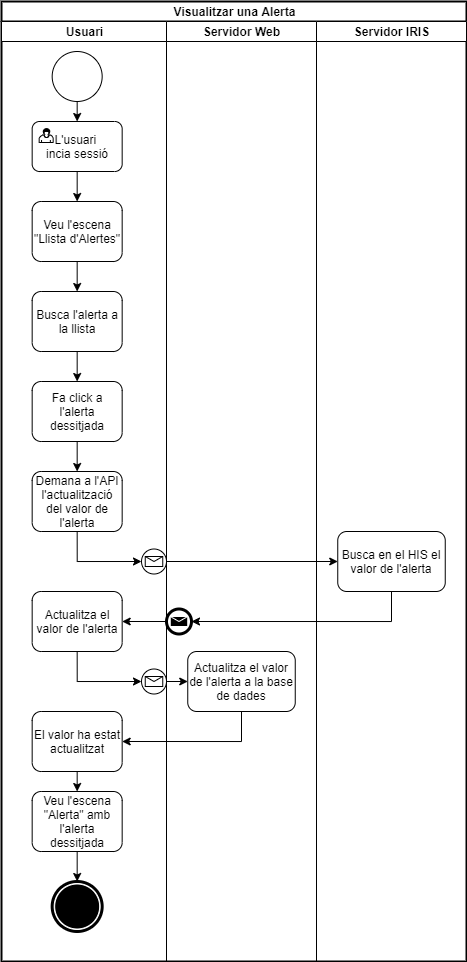


Figura .: Diagrama BPMN de la visualització d’una alerta en l’aplicació.

#### CASOS D’ÚS

En el cas de l’aplicació mòbil s’han elaborat tres casos d’ús per explicar el funcionament i el que ha de fer l’usuari per a poder utilitzar l’aplicació. Els tres casos d’ús que es veuran a continuació son les mateixes funcionalitats que s’ha vist en els tres diagrames BPMN.

|  |  |
| --- | --- |
| INICI DE SESSIÓ | |
| DESCRIPCIÓ | Aquest cas d’ús tracta de com s’inicia sessió a l’aplicació mòbil. |
| ACTOR | Qualsevol usuari amb accés a l’aplicació. |
| PRECONDICIONS | — |
| GARANTIES D’ÈXIT | — |
| FLUX BÀSIC | 1. L’usuari accedeix a l’aplicació mòbil. |
| 2. L’usuari veu l'escena “Inicia Sessió”. |
| 3. L’usuari introdueix l’usuari. |
| 4. L’usuari introdueix la contrasenya. |
| 5. L’usuari fa clic al botó d'“Iniciar Sessió”. |
| 6. L’usuari ha iniciat sessió. |
| 7. L’usuari veu l'escena “Llista d’Alertes”. |
| FLUX ALTERNATIU | 5A. La base de dades no troba el nom d’usuari. Torna al pas 2 explicant a l’usuari el motiu perquè no s’ha pogut iniciar sessió. |
| 5B. La contrasenya escrita per l’usuari no correspon a la que està a la base de dades. Torna al pas 2 explicant a l’usuari el motiu perquè no s’ha pogut iniciar sessió. |
| 5C. No s’ha pogut establir connexió amb el servidor web. Torna al pas 2 explicant a l’usuari el motiu perquè no s’ha pogut iniciar sessió. |
| 5D. No hi ha connexió amb la base de dades. Torna al pas 2 explicant a l’usuari el motiu perquè no s’ha pogut iniciar sessió. |
| REQUISITS ESPECIALS | Disposar de connexió a internet. |
| LLISTA DE TECNOLOGIA I VARIACIONS DE DADES | Telèfon intel·ligent, amb sistema operatiu Android o iOS, que disposi d’accés a internet. |

Taula .: Cas d’ús d’inici de sessió en l’aplicació mòbil.

|  |  |
| --- | --- |
| VISUALITZACIÓ DE LA LLISTA D’ALERTES | |
| DESCRIPCIÓ | Aquest cas d’ús tracta de com es visualitza la llista d’alertes, una vegada s’ha iniciat sessió, a l’aplicació mòbil. |
| ACTOR | Qualsevol usuari registrat. |
| PRECONDICIONS | Haver executat el cas d’ús “Inici de sessió”. |
| GARANTIES D’ÈXIT | — |
| FLUX BÀSIC | 1. L’usuari veu l'escena “Llista d’Alertes”. |
| FLUX ALTERNATIU | 1A. S’ha perdut la connexió amb el servidor web. Torna al pas 1 explicant a l’usuari el motiu perquè no s’ha pogut visualitzar la llista d’alertes. |
| 1B. No hi ha connexió amb la base de dades. Torna al pas 1 explicant a l’usuari el motiu perquè no s’ha pogut visualitzar la llista d’alertes. |
| 1C. No s’ha trobat cap alerta per l’usuari. Torna al pas 1 explicant a l’usuari el motiu perquè no s’ha pogut visualitzar la llista d’alertes. |
| 1D. No s’ha pogut establir connexió amb el servidor IRIS. Els valors actuals de l’alerta no estaran actualitzats. |
| REQUISITS ESPECIALS | Disposar de connexió a internet. |
| LLISTA DE TECNOLOGIA I VARIACIONS DE DADES | Telèfon intel·ligent, amb sistema operatiu Android o iOS. |

Taula .: Cas d’ús de la llista d’alertes en l’aplicació mòbil.

|  |  |
| --- | --- |
| VISUALITZACIÓ DETALLADA DE L’ALERTA | |
| DESCRIPCIÓ |  |
| ACTOR | Qualsevol usuari registrat. |
| PRECONDICIONS | Haver executat el cas d’ús “Inici de sessió”. |
| Haver executat el cas d’ús “Visualització de la llista d’alertes”. |
| GARANTIES D’ÈXIT | — |
| FLUX BÀSIC | 1. L’usuari veu l'escena “Llista d’Alertes”. |
| 1. L’usuari cerca l’alerta desitjada. |
| 1. L’usuari clica a sobre de l’alerta desitjada. |
| 1. L’usuari veu l'escena “Alerta”. |
| **FLUX ALTERNATIU** | 2A. L’usuari no troba l’alerta desitjada. Torna al pas 1. |
| 3A. S’ha perdut la connexió amb el servidor web. Torna al pas 1 explicant a l’usuari el motiu perquè no s’ha pogut visualitzar l’alerta. |
| 3B. No hi ha connexió amb la base de dades. Torna al pas 1 explicant a l’usuari el motiu perquè no s’ha pogut visualitzar l’alerta. |
| 3C. No s’ha pogut establir connexió amb el servidor IRIS. Els valor actual de l’alerta no estarà actualitzat. |
| **REQUISITS ESPECIALS** | Disposar de connexió a internet. |
| **LLISTA DE TECNOLOGIA I VARIACIONS DE DADES** | Telèfon intel·ligent amb sistema operatiu Android o iOS. |

Taula .: Cas d’ús de la visualització d’una alerta en l’aplicació mòbil.

#### UI

Per la UI de l’aplicació s’ha optat pel mateix estil que el servidor web, així que tindrà un fons blanc i els elements on hi hagi les alertes o els camps on introduir dades serà en un color blau cel.

Per a poder representar bé els valors de les alertes, s’ha pensat a fer-ho com si fos un semàfor. Així que quan es vegi el llistat d’alertes, al costat del nom de l’alerta es veurà un quadrat amb el valor i el color, si és correcte, verd, si no és un valor òptim, groc, i si és un valor perillós, vermell.

La llista d’alertes serà l’escena per defecte en l’aplicació, una vegada s’hagi iniciat correctament l’aplicació.

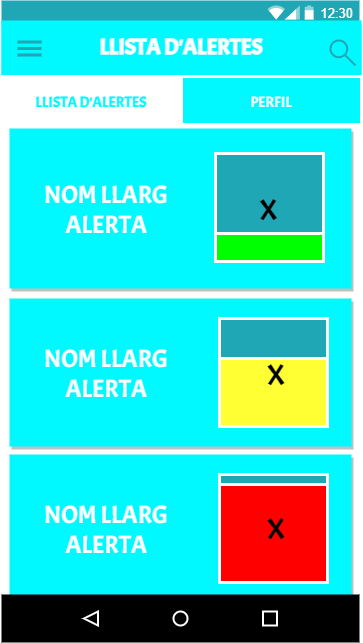


Figura .: Disseny de la UI de la llista d’alertes en l’aplicació mòbil.

Si es fa clic a sobre d’alguna alerta d’aquesta llista, es podrà veure la descripció d’aquesta alerta. A més, de nou, es podrà veure el color i el valor que té aquella alerta en aquell moment.

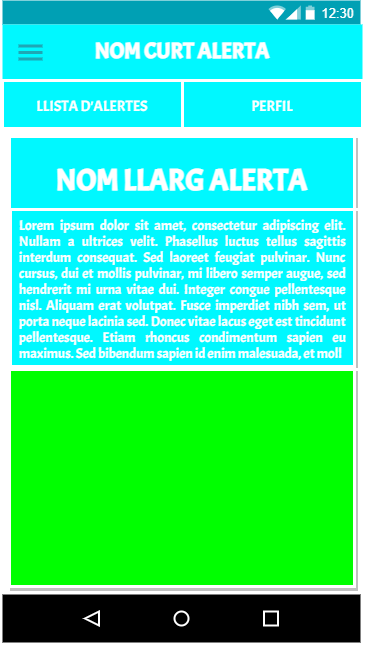


Figura . Figura .: Disseny de la UI de la visualització d’una alerta en l’aplicació mòbil.

## SERVIDOR IRIS

El primer pas en el procés de producció d’aquest TFG, és crear el servidor on estarà el motor d’interoperabilitat Intersystems IRIS for Health.

### CONFIGURACIÓ

Per poder començar a implementar el servidor d’IRIS, primer s’ha de configurar l’entorn de desenvolupament. A l’inici s’ha creat una màquina virtual amb el programa Hyper-V, que ja ve instal·lat amb el sistema operatiu Windows 10.

Aquesta màquina virtual té el sistema operatiu de Windows Server 2019. El motiu d’elegir Windows Server en comptes d’algun altre sistema operatiu que sol ser més usat per a fer servidors ha sigut per la corba d’aprenentatge. Amb el sistema operatiu Windows Server, la instal·lació del motor d’interoperabilitat IRIS for Health era bastant senzilla, ja que l’empresa et proporciona un instal·lador. El problema que hagués sorgit amb un altre sistema operatiu com pot ser Linux, és que aquesta instal·lació s’hagués hagut de fer a través d’un Docker. Això és un problema, ja que Windows, o almenys els processadors AMD, que és el que s’utilitza per elaborar aquest TFG, no permet la virtualització niuada. Així que es va optar per la solució més senzilla que era usar Windows Server 2019, amb l’instal·lador d’IRIS for Health.

### CREACIÓ DEL HIS

Per a fer el HIS, s’ha optat per fer una simulació. Això es deu a que el temps de desenvolupament per elaborar un HIS en condicions seria molt llarg, i això causaria que no donés temps a desenvolupar tant el servidor web com l’aplicació.

Així que directament s’han programat dues classes dintre d’IRIS.

La primera és com si fos l’objecte, que s’ha anomenat HIS. Aquest objecte està relacionat amb una taula en la base de dades, que s’ha afegit després de crear aquest objecte. Aquest objecte només té les variables de les tres alertes que est tenen pensades per aquest TFG. La primera és el percentatge d’ocupació dels llits d’un hospital, malalts per la COVID-19. La segona és el nombre de vacunes Pfizer per la COVID-19 que queden en el magatzem de la institució mèdica. La tercera i última alerta són el nombre d’ambulàncies que estan a l’hospital, és a dir, les ambulàncies que estan aparcades.

1. Class TFG.HIS Extends (%Persistent, %JSON.Adaptor) [ DdlAllowed ]
2. {
3. Property hisCovidBedsPercentage As %Float;
4. Property hisPfizerVaccineInStorage As %Integer;
5. Property hisAmbulancesWaiting As %Integer;
6. }

Com s’ha mencionat, després s’ha creat la taula a la base de dades. Aquesta taula té les mateixes variables que la classe i a part té un identificador. S’han generat cent mil dades diferents perquè, sempre doni un valor diferent.

### CREACIÓ DE L’API

Per a crear una API en IRIS for Health són necessari tres classes.

La primera és un JSON anomenat *spec* (Specification) i és l’encarregat de configurar les API. Això vol dir que en aquest JSON, hi haurà quina és la part de l'URI especifica per accedir a les seves dades i quins són els codis de resposta de cada API específica. A continuació hi ha un exemple per a obtenir el percentatge d’ocupació de llits amb malalts de COVID-19.

1. "/covidBed":{
2. "get":{
3. "operationId":"GetCovidBeds",
4. "produces":["application/json"],
5. "responses":{
6. "200":{"description":"Success"},
7. "404":{"description":"Not Found"},
8. "500":{"description":"Server error"}
9. }
10. }
11. }

Les següents classes s’autogeneren soles. Aquestes classes es diuen *disp* i *impl*. La classe *disp* (Dispatch), és l'encarregada de gestionar el servei REST. En el cas d’aquest TFG, s’ha deixat com està, ja que era la funció esperada. En el codi següent es pot veure com s’ha agafat el mètode GetCovidBeds, ja que els altres dos mètodes que queden són pràcticament idèntics, a excepció de la línia 6, que és la que està marcada amb verd, que canviaria la invocació del mètode.

1. ClassMethod GetCovidBeds() As %Status
2. {
3. Try {
4. Do ##class(%REST.Impl).%SetContentType("application/json")
5. If '##class(%REST.Impl).%CheckAccepts("application/json")' Do ##class(%REST.Impl).%ReportRESTError( ..#HTTP406NOTACCEPTABLE, $$$ERROR( $$$RESTBadAccepts)) Quit
6. Set response=##class(TFG.impl).GetCovidBeds()
7. Do ##class(TFG.impl).%WriteResponse(response)
8. } Catch (ex) {
9. Do ##class(%REST.Impl).%ReportRESTError( ..#HTTP500INTERNALSERVERERROR, ex.AsStatus(), $parameter("TFG.impl", "ExposeServerExceptions"))
10. }
11. Quit $$$OK
12. }

La següent i ultima classe programada per a crear l’API, és la *impl* (Implementation). Aquesta classe té la funció d'agafar les dades d’una base de dades o d’algun altre sistema que implementi el HIS. El que s’ha fet és molt simple, ja que com hi ha una sola taula, només s’ha d’accedir a una de les columnes de la taula cada vegada que es fa una crida a l’API. Així que s’ha generat un valor aleatori que va del 1 al 100.000, per a agafar una taula. Seguidament en cada mètode, GetCovidBeds, GetPfizerVaccineInStorage i GetAmbulancesWaiting, s’ha agafat la variable corresponent i s’envia en format JSON.

A continuació es mostrarà la implementació del mètode GetCovidBeds. D’igual manera que la classe *disp*, en els altres mètodes només varien en la línia 9, que és la que està marcada en color verd, i varia només l’atribut de retorn de la classe HIS.

1. ClassMethod GetCovidBeds() As %DynamicObject
2. {
3. Try {
4. Set id=$RANDOM(100000)+1
6. Set cm = ##class(TFG.HIS).%OpenId(id)
8. Do ..%SetStatusCode("200")
9. return {"value": (cm.hisCovidBedsPercentage)}
10. } Catch(ex) {
11. Do ..%SetStatusCode("500")
12. return {"errormessage": "Server error"}
13. }
14. }

Una vegada ja s’ha programat tot, només falta crear a IRIS, l’aplicació web, que seria l’API. En la següent imatge es pot veure com se li assigna un nom, que serà una part de l’URI, és a dir, l’API que s’ha fet tindrà la següent forma:

* https://www.domini.com/rest/tfg/covidBed: Pel cas dels llits del percentatge d’ocupació de llits amb malalts de COVID-19.
* https://www.domini.com/rest/tfg/pfizerStorage: Pel cas del nombre de vacunes Pfizer que queden en el magatzem.
* https://www.domini.com/rest/tfg/ambulancesWaiting: Pel cas del nombre d’ambulàncies, de la institució mèdica, aparcades.

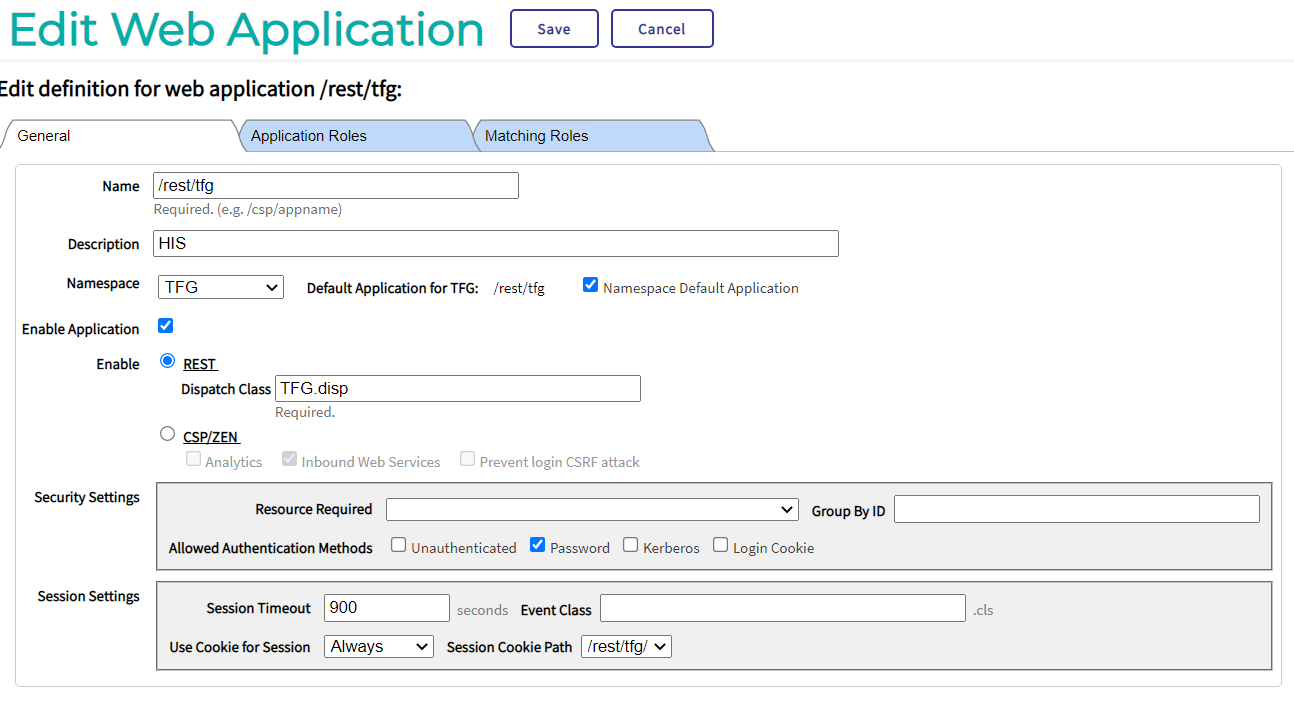


Figura .: Imatge de la configuració d’una API en *Intersystems IRIS for Health*.

Després del nom se li ha d’assignar quina classe d’enviament (*disp*) ha d’utilitzar. Per acabar només queda la seguretat i en el cas d’aquesta API, es necessitarà un usuari i contrasenya.

### PUBLICAR L’API

Per a publicar l’API del servidor d’*Intersystems IRIS for Health*, s’ha de fer que els dispositius es puguin comunicar amb el servidor a través d’internet. Per això s’ha d’habilitar el router per a que pugui encaminar un port a una IP.



Figura .: *Port forwarding* que s’ha fet al router.

Així que s’ha hagut de fer un *port forwarding*, perquè així per cada petició que arribi al router pel port 52773, el router, l’enviarà a l’adreça IP on està ubicada el servidor. Una vegada fet això es va haver de trucar a la companyia d’internet per a que obrissin el port 52773, per a que tothom pugui accedir a l’API del servidor.

## SERVIDOR WEB

El servidor web s’ha fet seguint el funcionament del *framework Spring*. Així que, com s’ha pogut veure al diagrama UML, s’han separat les classes que gestionen les crides a la pàgina web, de la base dades.

### *BACKEND*

En els següent subapartats, el *backend*, s’explicarà tot el relacionat amb les alertes, ja que al ser un codi simètric, les institucions mediques, els rols i els empleats, són pràcticament identics.

Per a fer el *backend* del servidor web, és necessiten crear tres paquets.

#### DOMINI

Primer paquet que s’ha de crear és el domini, on es guardaran les dades extretes de la base de dades en una classe Java, en aquest cas la classe *Warning*.

Aquestes classes son bastant simples, només tenen el seus atributs, dos constructors, un de buit i l’altre que inicialitza totes les dades i per acabar els seus gets i sets.

Un exemple d’una classe del domini seria la següent. Com s’ha mencionat a l’inici de l’apartat 6.3, es mostra la classe *Warning*, ja que tota l’explicació del servidor web es farà sobre com es crea i es mostren alertes.

A sobre d’alguns atributs hi ha uns missatges per comprovar i mostrar a l’usuari si l’atribut és buit o no té la mida requerida.

1. public class Warning{
2. private int id;
3. @NotEmpty(message = "No es pot deixar el nom buit")
4. @Size(min = 1, max = 128,message = "El nom ha de tenir entre 1 i 128 caràcters")
5. private String name;
6. @NotEmpty(message = "No es pot deixar el nom curt buit")
7. @Size(min = 1, max = 64,message = "El nom ha de tenir entre 1 i 64 caràcters")
8. private String shortName;
9. @NotEmpty(message = "No es pot deixar la descripció buida")
10. @Size(min = 1, max = 512,message = "La descripció ha de tenir entre 1 i 512 caràcters")
11. private String description;
12. @NotEmpty(message = "No es pot deixar la URI buida")
13. @Size(min = 1, max = 256,message = "La URI ha de tenir entre 1 i 256 caràcters")
14. private String uri;
15. @NotEmpty(message = "No es pot deixar la descripció buida")
16. @Size(min = 1, max = 256,message = "La notificació ha de tenir entre 1 i 256 caràcters")
17. private String notificationMessage;
19. private float greenValue;
20. private float yellowValue;
21. private float redValue;
22. private float lastValue;
23. private int refreshRate;
25. private Role role;
27. private String tempRoleName;
29. //CONSTRUCTOR
31. //GETS I SETS
33. }

#### PERSISTÈNCIA

El segon paquet és el de la persistència i serà la que parli amb la base de dades. Aquest paquet tindrà dues classes. La primera, la que es comuniqui a través de *queries* amb la base de dades, *DatabaseDAO*. I l’altre la que fa de façana de la classe anterior i es comunica amb els controladors webs, *DatabaseController*.

Aquesta primera classe necessitarà, a part d’uns *strings* amb les *queries*, d’un mètode que converteixi una fila de la base de dades en un objecte del tipus *Warning*. Amb aquest mètode, agafar les dades de les files de la taula *Warnings* serà bastant més senzill, ja que no s’haurà de repetir codi.

A més el *DatabaseDAO* necessita un objecte de la classe *JdbcTemplate*, ja que serà qui farà les crides a la base de dades. Per saber quina base de dades ha de comunicar-se, s’ha hagut de crear un arxiu de propietats on hi ha totes les credencials d’aquesta base de dades.

Per acabar amb aquesta classe, hi ha cinc mètodes desprès del *warningMapper*. El primer mètode és el que crearà una alerta a la base de dades. El segon, retornarà una llista amb totes les alertes que hi ha creades. El tercer mètode retornarà la fila de la taula *Warnings* que tingui l’identificador que és passa per paràmetre del mètode. Després tenim el mètode que esborrarà de la base de dades l’alerta desitjada. I per últim hi ha el mètode que modificarà les dades d’una alerta.

1. @Repository public class DatabaseDAO {
3. private final String INSERT\_WARNING = "INSERT INTO Warnings VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?);";
4. private final String FIND\_ALL\_WARNINGS = "SELECT \* FROM Warnings";
5. private final String FIND\_WARNING\_BY\_ID = "SELECT \* FROM Warnings WHERE war\_id=?";
6. private final String DELETE\_WARNING = "DELETE FROM Warnings WHERE war\_id=?";
7. private final String UPDATE\_WARNING = "UPDATE Warnings SET war\_description=?, war\_uri=?, war\_notificationmessage=?, war\_greenvalue=?, war\_yellowvalue=?, war\_redvalue=?, war\_refreshrate=? WHERE war\_id=?";
8. private final String COUNT\_WARNINGS = "SELECT COUNT(war\_id) FROM Warnings";
10. //MAPPERS
11. private RowMapper<Warning> warningMapper = (resultSet, i) -> {
12. Warning warning = new Warning(
13. resultSet.getInt("war\_id"),
14. resultSet.getString("war\_name"),
15. resultSet.getString("war\_shortname"),
16. resultSet.getString("war\_description"),
17. resultSet.getString("war\_uri"),
18. resultSet.getString("war\_notificationmessage"),
19. resultSet.getFloat("war\_greenvalue"),
20. resultSet.getFloat("war\_yellowvalue"),
21. resultSet.getFloat("war\_redvalue"),
22. resultSet.getFloat("war\_lastvalue"),
23. resultSet.getInt("war\_refreshrate"),
24. findRole(
25. resultSet.getString("war\_rolename"),
26. resultSet.getInt("war\_roleinstitutionid"),
27. resultSet.getString("war\_rolecountryid"))
28. );
29. return warning;
30. };
32. public int insertWarning(Warning warning) {
33. int id = jdbcTemplate.queryForObject(COUNT\_WARNINGS, new Object[]{}, Integer.class);
34. jdbcTemplate.update(
35. INSERT\_WARNING,
36. id,
37. warning.getName(),
38. warning.getShortName(),
39. warning.getDescription(),
40. warning.getUri(),
41. warning.getNotificationMessage(),
42. warning.getGreenValue(),
43. warning.getYellowValue(),
44. warning.getRedValue(),
45. 0,
46. warning.getRefreshRate(),
47. warning.getRole().getName(),
48. warning.getRole().getHealthcareInstitution().getId(),
49. warning.getRole().getHealthcareInstitution().getCountry().getId()
50. );
51. return id;
52. }
54. public List<Warning> findAllWarnings() {
55. return jdbcTemplate.query(FIND\_ALL\_WARNINGS, new Object[]{}, warningMapper);
56. }
58. public Warning findWarning(int id) {
59. return jdbcTemplate.queryForObject(FIND\_WARNING\_BY\_ID, new Object[]{id}, warningMapper);
60. }
62. public void deleteWarning(int id) {
63. jdbcTemplate.update(DELETE\_WARNING, id);
64. }
66. public void updateWarning(Warning warning) {
67. jdbcTemplate.update(
68. UPDATE\_WARNING,
69. warning.getDescription(),
70. warning.getUri(),
71. warning.getNotificationMessage(),
72. warning.getGreenValue(),
73. warning.getYellowValue(),
74. warning.getRedValue(),
75. warning.getRefreshRate(),
76. warning.getId()
77. );
78. }
79. }

La classe DatabaseController és bastant més senzilla que la classe anterior, ja que crida directament als mètodes de la classe anterior. A continuació es mostra com s’ha fet la classe DatabaseController.

1. @Service("DatabaseController")
2. public class DatabaseController {
4. private final DatabaseDAO databaseDAO;
6. public DatabaseController(DatabaseDAO databaseDAO) {
7. this.databaseDAO = databaseDAO;
8. }
10. public int addWarning(Warning warning) {
11. return databaseDAO.insertWarning(warning);
12. }
14. public List<Warning> getAllWarnings() {
15. return databaseDAO.findAllWarnings();
16. }
18. public Warning getWarning(int id) {
19. return databaseDAO.findWarning(id);
20. }
21. public void editWarning(Warning warning) {
22. databaseDAO.updateWarning(warning);
23. }
24. public void deleteWarning(int id) {
25. databaseDAO.deleteWarning(id);
26. }
27. }

#### CONTROLADOR WEB

Per acabar es necessita el controlador web, aquest serà l’encarregat de gestionar les crides a la pàgina web. Està separat en quatre classes, ja que cada una només gestionarà les crides HTML d’una classe del domini.

Per tant, es mirarà la classe *WarningController*, en el cas de les alertes, tindrà cinc mètodes GET i dos mètodes POST. Es començarà amb els mètodes GET.

El primer mètode retornarà la pàgina HTML per la creació d’una alerta, per crear una alerta s’ha d’afegir la llista de tots els rols fent una crida al mètode *getAllRoles* de la classe *DatabaseController*.

1. @GetMapping("/warning/create")
2. public String createWarning(Warning warning, Model model) {
3. model.addAttribute("roles", databaseController.getAllRoles());
4. return "warning/createWarning";
5. }

El segon mètode serà el que mostri la llista d’alertes. Així que per poder-ho ensenyar, a part de la pàgina HTML, fa falta la llista d’alertes, fent una crida al mètode *getAllWarnings* de la *DatabaseController*.

1. @GetMapping("/warning/all")
2. public String getAllWarnings(Model model) {
3. model.addAttribute("warnings", databaseController.getAllWarnings());
4. return "warning/showWarnings";
5. }

El tercer mètode retornarà la pàgina HTML per ensenyar les dades d’una alerta en específic. Així que s’ha de passar aquella alerta fent la crida al mètode pertinent de la *DatabaseController*.

1. @GetMapping("/warning/{id}")
2. public String getWarning(@PathVariable int id, Model model) {
3. model.addAttribute("warning", databaseController.getWarning(id));
4. return "warning/showWarning";
5. }

El quart i cinquè mètode necessita un identificador passat per paràmetre, ja que, el quart mètode, retornarà la pàgina HTML per a poder editar una alerta i el cinquè directament esborrarà l’alerta i redirigirà a l’usuari a la llista d’alertes.

1. @GetMapping("/warning/edit/{id}")
2. public String editWarning(@PathVariable int id, Model model) {
3. model.addAttribute("warning",
4. databaseController.getWarning(id));
5. return "warning/editWarning";
6. }
7. @GetMapping("/warning/delete/{id}")
8. public String deleteWarning(@PathVariable int id, Model model) {
9. databaseController.deleteWarning(id);
10. return getAllWarnings(model);
11. }
12. }

A continuació es mostrarà els POST. Aquí hi haurà dos mètodes.

El primer mètode serà l’encarregat de crear una alerta. Per això primer s’haurà de comprovar si les dades de l’alerta son correctes. En cas que hi hagi qualsevol error, ho notificarà a l’usuari. Una vegada l’alerta ha estat creada es redirigirà l’usuari a veure les dades d’aquella alerta.

1. @PostMapping("/warning/create")
2. public String createWarningPOST(@Valid Warning warning, BindingResult bindingResult, Model model, RedirectAttributes redirectAttributes)
3. {
4. if (bindingResult.hasErrors()) {
5. return createWarning(warning, model);
6. }
8. String[] role = warning.getTempRoleName().split(" - ");
9. warning.setRole(databaseController.getRole(role[2], Integer.parseInt(role[1]), role[0]));
11. warning.setId(databaseController.addWarning(warning));
13. redirectAttributes.addAttribute("id", warning.getId());
14. return "redirect:/warning/{id}";
15. }

El segon i últim mètode POST serà el que guardi els canvis quan s’editi una alerta. Així que aquest mètode, no comprovarà que les dades siguin correctes, ja que les dades que es pot editar no necessita cap tipus de comprovació. A més, de la mateixa manera que el mètode anterior, redirigirà a l’usuari per veure les dades de l’alerta editada.

1. @PostMapping("/warning/edit/{id}")
2. public String editWarningPOST(Warning warning, RedirectAttributes redirectAttributes)
3. {
4. databaseController.editWarning(warning);
6. redirectAttributes.addAttribute("id", warning.getId());
7. return "redirect:/warning/{id}";
8. }

### API

La API serà la que utilitzarà l’aplicació mòbil per a agafar les dades de la base de dades. Només hi haurà quatre rutes GET.

|  |  |  |
| --- | --- | --- |
| **RUTES** | **PARÁMETRES DE ENVÍAMENT** | **RESPOSTA** |
| login/{username}/{password} /{notificationToken} | Nom d’usuari | Dades de l’usuari |
| Contrasenya |
| *Token* de la notificació |
| /logout/{username}/{notificationToken} | Nom d’usuari | — |
| *Token* de la notificació |
| /warning/all/{roleName}/ {healthcareInstitutionID}/{countryID} | Nombre del rol | Llista d’alertes |
| ID de la institució mèdica |
| ID del país |
| /employee/change/{username}/{newPassword} | Nom d’usuari | — |
| Nova contrasenya |

Taula .: Taula de les rutes de l’API

La primera serà l’encarregada de permetre o no l’inici de sessió a l’aplicació. A més tindrà una funció extra per a la gestió de les notificacions, però es veurà en el següent apartat.

1. @GetMapping("/login/{username}/{password}/{notificationToken}")
2. public Employee login(@PathVariable String username, @PathVariable String password, @PathVariable String notificationToken) {
3. Employee employee = databaseController.getEmployee(username);
4. (...)
5. if (passwordEncoder.matches(password, employee.getPassword())) {
6. return employee;
7. } else {
8. return null;
9. }
10. }

La segona és per permetre tancar la sessió. En aquest cas només fa una funció per a les notificacions. Aquesta funció es treure el *token* de la notificació per a que l’usuari pugui parar de veure notificacions.

1. @GetMapping("/logout/{username}/{notificationToken}")
2. public boolean logout(@PathVariable String username, @PathVariable String notificationToken) {
3. (...)
4. return true;
5. }

La tercera és l’encarregada de retornar a l’usuari la llista d’alertes que té assignades. Així que si se li dona les dades del rol, li retornarà la llista completa.

1. @GetMapping("/warning/all/{roleName}/{healthcareInstitutionID}/{countryID}")
2. public List<Warning> getAllWarnings(@PathVariable String roleName, @PathVariable int healthcareInstitutionID, @PathVariable String countryID) {
3. List<Warning> allWarnings = databaseController.getAllWarningsOfRole(roleName, healthcareInstitutionID, countryID);
4. return allWarnings;
5. }

La quarta i última ruta és per a permetre l’usuari canviar de contrasenya.

1. @GetMapping("/employee/change/{username}/{newPassword}")
2. public int changePassword(@PathVariable String username, @PathVariable String newPassword) {
3. return databaseController.changePassword(username, newPassword);
4. }

Després està la que l’usuari podrà canviar la contrasenya

### GESTIÓ DE LES NOTIFICACIONS

Una vegada ja es té totes les funcionalitats del servidor programades, toca entrar en com gestionar les notificacions. Per a fer això aquest apartat s’ha de separar en tres parts.

#### THREADS

Per a fer que s’enviïn les notificacions s’ha creat una classe *WarningThread*. Aquesta classe el que farà és executar-se en paral·lel amb el servidor web i els altres t*hreads* amb l’objectiu de què automàticament demanin les dades corresponent a la adreça HTTP que correspongui.

El funcionament del t*hread* serà senzill. Sempre s’estarà executant. Primer buscarà el valor actual de l’alerta que té assignada. Comprovarà que aquella alerta ha canviat de gravetat, és a dir si hi ha el valor ha passat de verd a groc, de groc a vermell, etc. Si l’alerta ha canviat de gravetat enviarà una notificació. En cas contrari continuarà amb l’execució desant el nou valor a la base de dades. Per acabar el t*hread* s’adormirà el temps que s’hagi assignat a aquella alerta.

1. public void run() {
2. while (true) {
3. float newLastValue = notificationMonitor.getLastValueFromAPI(warning);
5. int newWarningColor = knowWarningColor(newLastValue);
6. if (newWarningColor != notificationMonitor.GREEN && newWarningColor != knowWarningColor(this.warning.getLastValue()))
7. {
8. try {
9. notificationMonitor.sendNotifications(
11. databaseController.getAllEmployeesOfRole(
12. warning.getRole().getName(),
13. warning.getRole().getHealthcareInstitution().getId(),
14. warning.getRole().getHealthcareInstitution().getCountry().getId()),
15. newWarningColor,
16. warning);
17. } catch (PushClientException e) {
18. }
19. }
20. this.warning.setLastValue(newLastValue);
21. this.databaseController.setWarningLastValue( this.warning.getId(), newLastValue);
23. try {
24. Thread.sleep(this.warning.getRefreshRate() \* 1000L);
25. } catch (InterruptedException e) {
26. }
27. }
28. }

#### SECCIÓ CRÍTICA

En aquest problema s’ha de bloquejar la secció crítica perquè l’executi un únic t*hread* a la vegada. Aquesta secció crítica té relació amb la crida a l’API de la institució mèdica, ja que si es fan dues crides alhora hi ha problemes de perduda dels missatges. Així s’ha posat un *lock* a l’inici del mètode per evitar que hi hagi més d’un t*hread* a la secció crítica. Després es comença a preparar el paquet per enviar-ho a la institució mèdica. Es necessiten les credencials per a poder accedir a aquella API i l'URL completa. Finalment només s’ha d’esperar el resultat i alliberar el *lock* perquè un altre t*hread* ho pugui utilitzar.

1. float getLastValueFromAPI(Warning warning) {
2. lock.lock();
3. float value;
5. HttpHeaders httpHeaders = new HttpHeaders();
6. String auth = warning.getRole().getHealthcareInstitution().getUsername() + ":" + warning.getRole().getHealthcareInstitution().getPassword();
7. byte[] encodedAuth = Base64.encodeBase64(  
    auth.getBytes(StandardCharsets.US\_ASCII));
8. String authHeader = "Basic " + new String(encodedAuth);
10. httpHeaders.add(HttpHeaders.AUTHORIZATION, authHeader);
12. HttpEntity<String> request = new HttpEntity<String>("", httpHeaders);
13. RestTemplate restTemplate = new RestTemplate();
14. try {
15. ResponseEntity<String> response = restTemplate.exchange(warning.getRole().getHealthcareInstitution().getUrl() + warning.getUri(), HttpMethod.GET, request, String.class);
17. ObjectMapper mapper = new ObjectMapper();
18. JsonNode actualObj = mapper.readTree(response.getBody());
19. value = (float) actualObj.get("value").asDouble();
20. } catch (Exception e) {
21. value = 0;
22. }
24. lock.unlock();
26. return value;
27. }

#### QUAN INICIAR ELS THREADS?

L’altra problemàtica arriba amb una pregunta. Quan s’han d’iniciar els *threads*?

La solució a aquesta pregunta és simple. Cada vegada que un usuari iniciï sessió a l’aplicació, primer desarà el *token* de la notificació per a poder, posteriorment, enviar les notificacions al dispositiu mòbil. Després iniciarà els *threads* que encara no hagin estat iniciats fent una crida al mètode *startThreads*. Aquest mètode agafarà totes les alertes que hi ha a la base de dades i a continuació, farà una cerca per comprovar que aquella alerta està desada en un atribut de llistes d’alertes.

1. public void startThreads() {
2. List<Warning> warnings = databaseController.getAllWarnings();
3. for (Warning warning : warnings) {
4. WarningThread thread = warningIsInThread(warning.getId());
5. if (thread == null) {
6. WarningThread newWarning = new WarningThread(warning, databaseController, notificationMonitor);
7. newWarning.start();
8. threads.add(newWarning);
9. } else {
10. thread.setWarning(warning);
11. }
12. }
13. }

### RESULTAT FINAL

Només entrar a la direcció web apareixerà el taulell de l’aplicació. En aquesta pàgina es podrà seleccionar entre crear o veure la llista d’institucions mèdiques, rols, alertes i usuaris. A més en cada pàgina hi ha un *header* que hi haurà tots els elements de navegació per a poder crear o veure el llistat dels elements de la web.



Figura .: Resultat final del taulell de la pàgina web.

A continuació, i depenent del que s’hagi escollit entre veure el llistat o crear, es veurà com es crea una alerta. D’igual manera que en tot aquest subapartat, del servidor web, només es veurà tot el relacionat amb les alertes.

En la creació d’una alerta s’haurà de posar totes les dades necessàries per a poder crear l’alerta. Si un d’aquests elements no son introduïts de manera correcta es mostrarà un error a l’usuari.

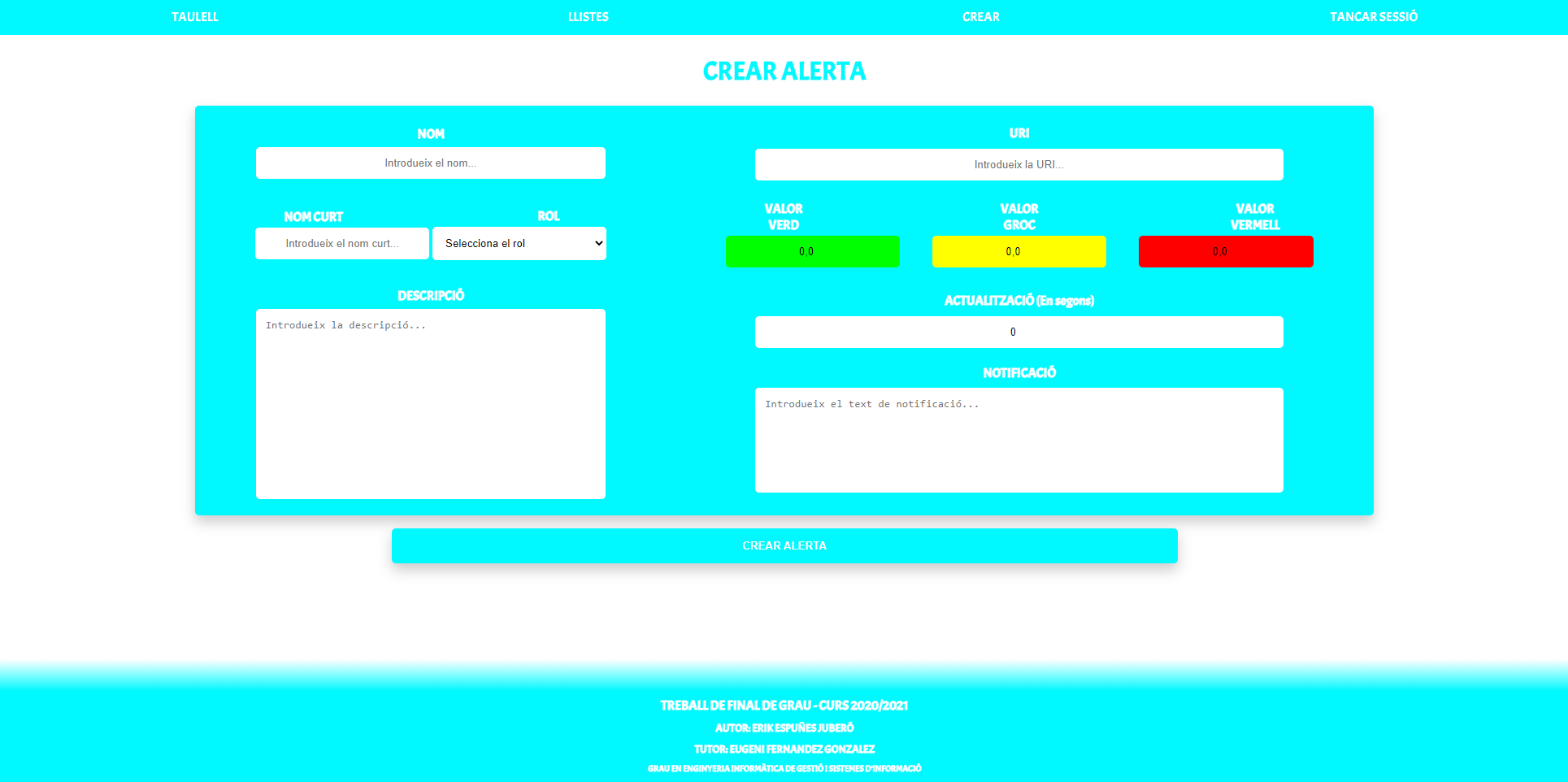


Figura .: Resultat final de la creació d’una alerta de la pàgina web.

La llista d’alertes es on hi ha mes requeriments, ja que s’ha fet que es pugui cercar en cada columna i ordenar de forma ascendent o descendent la llista en funció d’una columna.

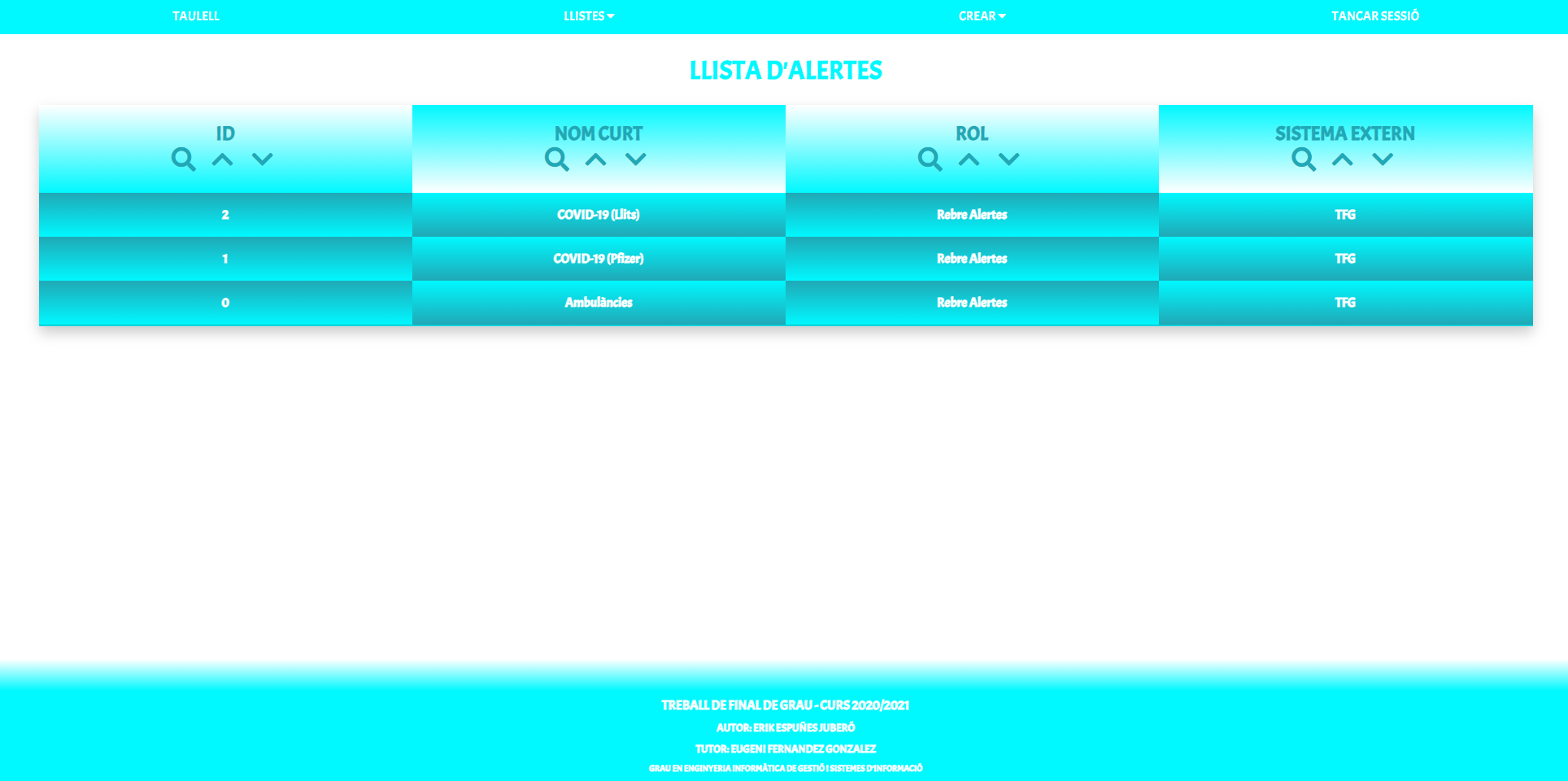


Figura . Figura .: Resultat final de la llista alertes de la pàgina web.

Una vegada es fa click a sobre d’un element de la llista s’obre la pàgina de la visualització d’una alerta. Aquí és veurà totes les dades de l’alerta, i a més és podrà editar i esborrar l’alerta.

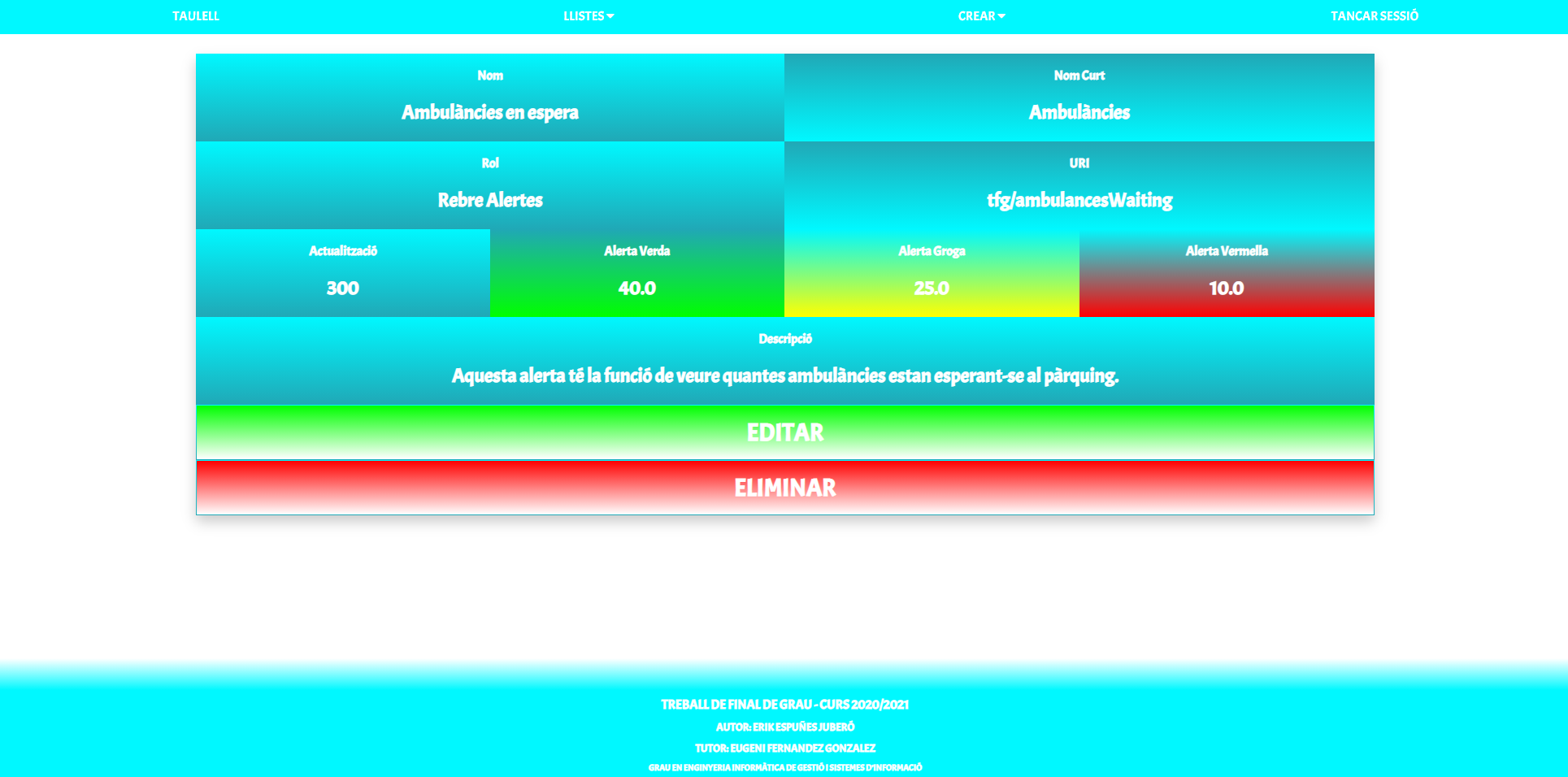


Figura . Figura .: Resultat final de la visualització d’una alerta de la pàgina web.

Editar les alertes es un disseny igual que el de la creació, a excepció que hi ha elements que no es poden variar. Aquests elements son els noms i el rol. Ja que al ser les claus primàries de l’alerta hi hauria molts problemes en canviar aquestes dades.



Figura . Figura .: Resultat final de la edició d’una alerta de la pàgina web.

## APLICACIÓ­

Una vegada es tenen els dos servidors s’ha de començar a desenvolupar l’aplicació. Per això s’han utilitzat els *frameworks* de *React Native* i *Expo*. Com s’ha vist en l’apartat 6.1.3, el motiu d’utilitzar *React Native* és per la facilitat que et dona desenvolupar una aplicació per als dos sistemes operatius més utilitzats, iOS i Android. I el motiu d’usar *Expo* és per la facilitat de poder executar l’aplicació al mòbil sense necessitat d’usar cap simulador o emulador.

### INICIAR SESSIÓ

L’inici de sessió es la primera pantalla que es mostrarà a l’aplicació. El disseny, es simple, ja que només es necessita introduir el nom d’usuari i la contrasenya. Una vegada estiguin introduïts, s’enviaran les dades al servidor. Si tot es correcte, anirà a la llista d’alertes, si hi ha alguna dada incorrecte, es mantindrà a l’inici de sessió informant que no s’ha pogut iniciar sessió.

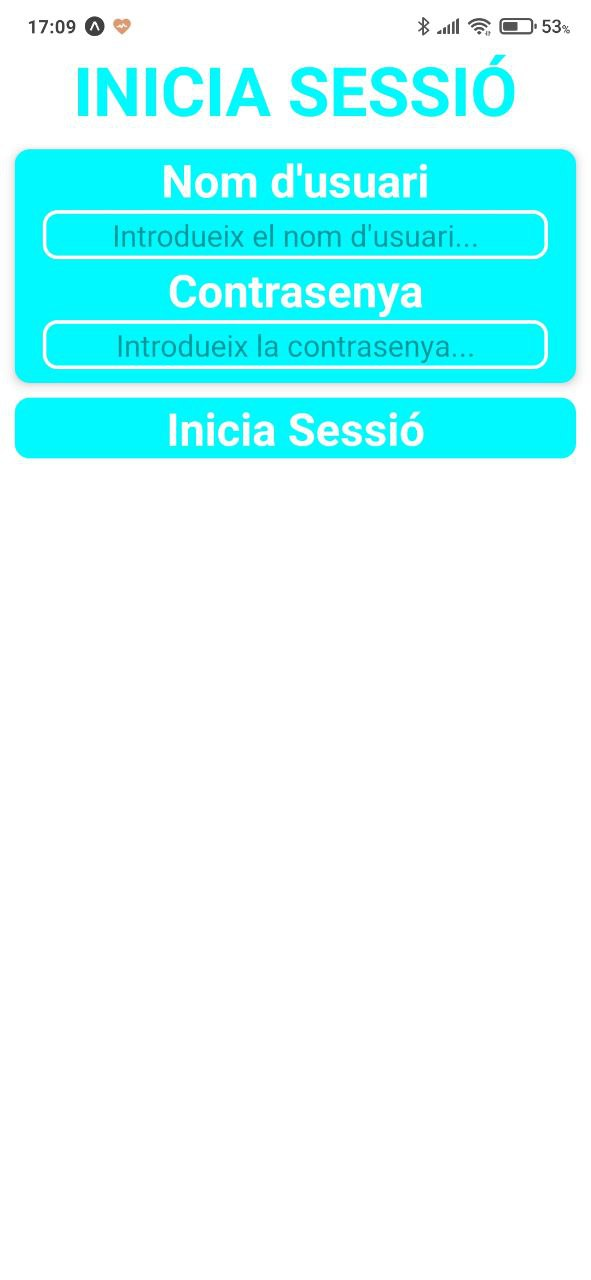


Figura .: Resultat final de l’inici de sessió a l’aplicació.

JavaScript té dos *plugins* que permeten les comunicacions HTTP, *Axios* i *fetch*. Els dos són bastants similars, és a dir que donen els mateixos resultats, però *Axios* està pensat en el desenvolupament amb Node.js. Així que, per la manera que s’utilitzen, s’usarà *Axios* o *fetch* en funció del que s’hagi d’invocar.

En el cas de l’inici de sessió s’usarà *Axios*. El que es farà és enviar la petició a l’URL que pertoqui. Si la petició no té cap error, és a dir ha pogut arribar al destinatari, es comprovarà si l’inici de sessió és correcte. Si les dades estan ben introduïdes, es desaran les dades de l’usuari i s’anirà a la llista d’alertes. Si és incorrecte, s’informarà a l’usuari de que hi ha algun error en les dades introduïdes.

1. const login = async () => {
2. setLoading(true);
3. Axios
4. .get(CommunicationManager.URL + 'login/' + username + '/' + password + '/' + CommunicationManager.token)
5. .then(function (response) {
6. setLoading(false);
7. if (response.data !== '') {
8. CommunicationManager.user = response.data
9. CommunicationManager.loggedIn = true
10. navigation.navigate("List")
11. } else {
12. alert('El nom d\'usuari o la contrasenya no són correctes.')
13. }
14. })
15. .catch(function (error) {
16. setLoading(false)  
     alert('S\'ha produit un error al servidor')
17. })
18. }

### LLISTA D’ALERTES

La funció que tindrà aquesta pantalla és de mostrar les alertes que té assignat aquell usuari. Així que mostrarà en una llista el nom, valor i color de l’alerta. Si es fa click a sobre de cada alerta es veurà la informació detallada d’aquesta.

A més si es refresca la llista d’alertes, lliscant el dit des de amunt de la pantalla cap avall, s’actualitzarà aquestes llistes. Cada vegada que es mostri la llista d’alertes o es refresqui, hi haurà una comunicació amb la URL que tingui assignada aquella alerta per obtenir el valor més recent de l’alerta.

Addicionalment també s’hauria d’actualitzar aquest valor a la base de dades del servidor web. Però com s’ha vist en el servidor d’*Intersystems IRIS for Health*, el resultat de les crides a l’API de la institució mèdica són valors aleatoris a la base de dades. Així que en el treball realitzat no s’ha implementat la funcionalitat d’actualitzar el valor de l’alerta a la base de dades.

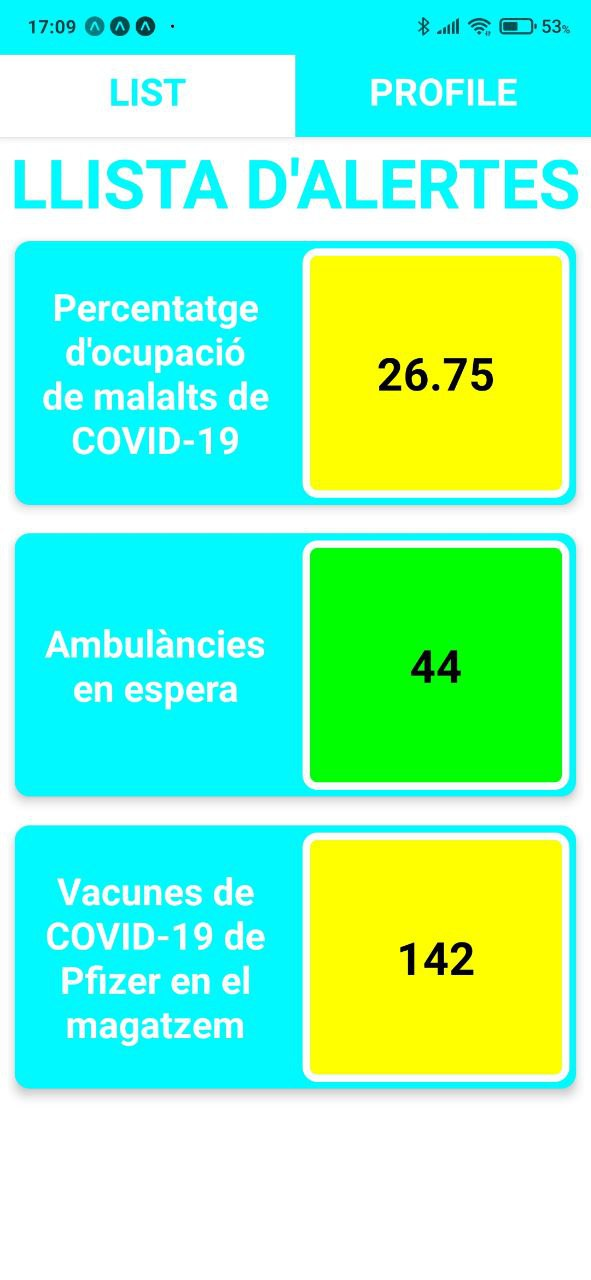


Figura .: Resultat final de la llista d’alertes a l’aplicació.

Per obtenir la llista d’alertes s’ha fet de la mateixa manera que l’inici de sessió. Si la petició no té cap error, farà una iteració de la llista d’alertes i enviarà una petició web a la URL de l’alerta per a obtenir el seu valor. Una vegada s’hagi acabat la iteració s’actualitzaran les dades mostrades a l’usuari.

1. const warningListData = async () => {
2. axios
3. .get(CommunicationManager.URL + 'warning/all/' + CommunicationManager.user.role.name + '/' + CommunicationManager.user.role.healthcareInstitution.id + '/' + CommunicationManager.user.role.healthcareInstitution.country.id)
4. .then(async function (response) {
5. setData(response.data)
6. setRefreshing(true)
7. const dataArray = Object.values(response.data)
8. for (let i = 0; i < dataArray.length; i++) {
9. const warning = dataArray[i]
10. try {
11. let response = await fetch(warning.role.healthcareInstitution.url + warning.uri, {
12. method: 'GET',
13. headers: {
14. 'Accept': '\*/\*',
15. 'User-Agent': 'tfg/0.1.0',
16. 'Content-Type': 'application/json',
17. 'Connection': 'keep-alive',
18. 'Authorization': 'Basic ' + btoa(warning.role.healthcareInstitution.username + ':' + warning.role.healthcareInstitution.password)
19. }
20. })
21. let json = await response.json()
22. warning.lastValue = json.value
23. } catch (error) {  
     }
24. }
25. setRefreshing(false)
26. setData(dataArray)
27. })
28. .catch(function (error) {
29. alert('S\'ha produit un error al servidor.')
30. })
31. }

### ALERTA

Una vegada s’ha fet click a sobre d’una alerta, o d’una notificació relacionada amb l’alerta, es mostrarà aquesta pantalla. La pantalla només contindrà informació referent a l’alerta. Així que es mostrarà el nom complert, la descripció, el valor i el color de l’alerta.

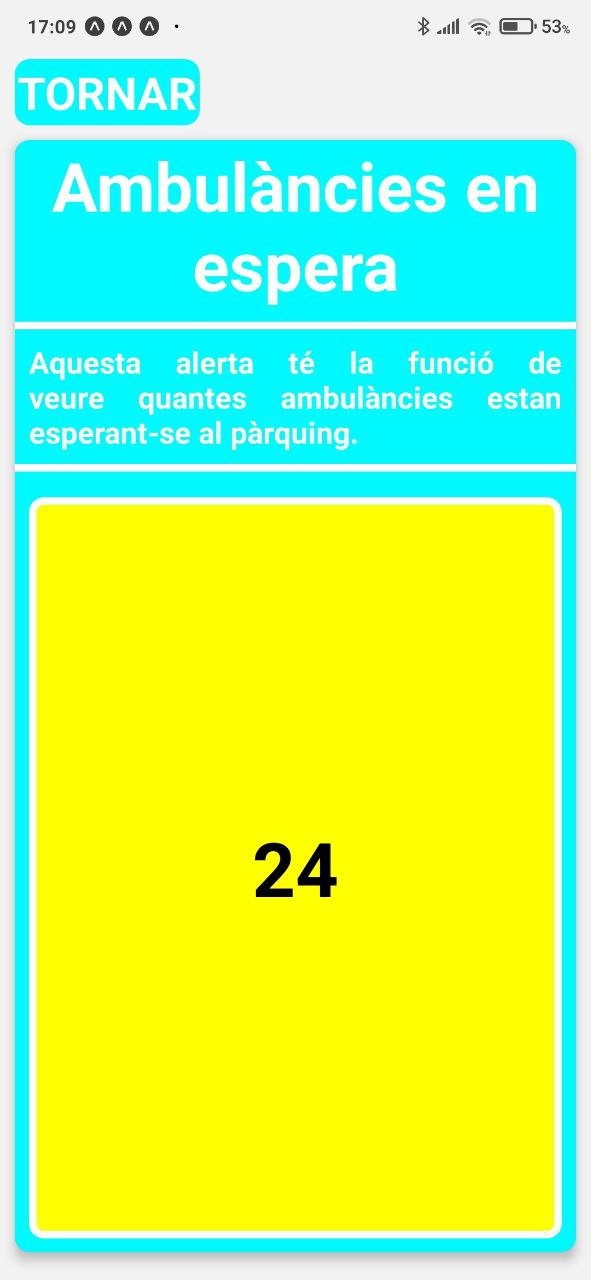


Figura .: Resultat final de la visualització de l’alerta a l’aplicació.

### PERFIL

El perfil servirà per poder veure les dades del nom d’usuari, nom i cognoms i rol de l’usuari. A més tindrà les funcionalitats de canviar de contrasenya i tancar sessió.



Figura .: Resultat final del perfil a l’aplicació.

El mètode de tancar sessió, *logout* és molt semblat al de canviar de contrasenya o els altres dos mètodes vists anteriorment. Només es comunica amb l’API de tancar sessió i si ha sigut exitós canvia l’escena a la de l’inici de sessió.

El mètode de canviar de contrasenya té alguna variació, ja que s’ha de comprovar la longitud de la contrasenya. Així que a partir d’aquí ja és un mètode típic per a la comunicació HTTP.

1. const changePassword = async () => {
2. if (password.length >= 3)
3. Axios
4. .get(CommunicationManager.URL + 'employee/change/' + CommunicationManager.user.username + '/' + password)
5. .then(function (response) {
6. if (response.data === '') {
7. alert('No s\'ha pogut canviar la contrasenya')
8. }
9. })
10. .catch(function (error) {
11. alert('S\'ha produit un error al servidor')
12. })
13. else
14. alert('La nova contrasenya ha de tenir entre 3 i 32 caracters')
15. setModalVisible(false)
16. }

### REBRE NOTIFICACIONS

El format de les notificacions seran senzilles. Tindrà el títol que informarà del color de gravetat. I el text informatiu que informarà de quina alerta ha estat la que ha enviat la notificació.

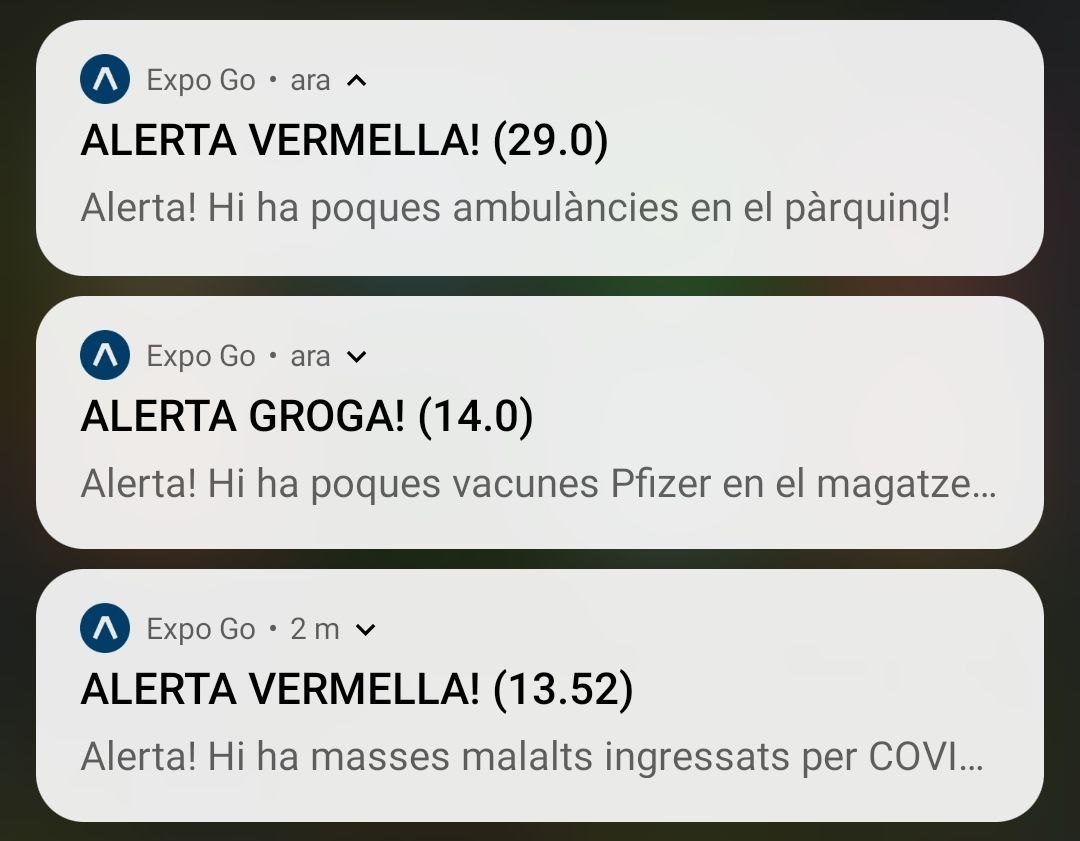


Figura .: Format de les notificacions de les tres alertes que hi ha en el servidor.

Per a poder rebre notificacions primer s’ha de generar el *token* que estarà assignat al dispositiu. Per això en el *script* que configura l’aplicació s’ha fet que inicialitzi la variable *token*, així quan es faci l’inici de sessió aquest valor ja estarà inicialitzat.

1. useEffect(() => {
2. registerForPushNotificationsAsync().then(token => savedData.token = token.replace("[", "\_").replace("]", ""));
3. (...)
4. }, []);

Un altre aspecte a valorar es que fer quan un usuari faci click a sobre d’una notificació. El que es farà es mostrar la pantalla de la informació de l’alerta. Però si l’usuari no havia iniciat sessió anteriorment, al tornar de l’alerta s’anirà a l’inici de sessió en comptes de la llista d’alertes.

1. responseListener.current = Notifications.addNotificationResponseReceivedListener(response => {
2. savedData.warning = response.notification.request.content.data;
3. navigation.navigate('Warning')
4. });

# CONCLUSIONS

En aquest apartat es valorarà el treball fet al llarg del desenvolupament d’aquest producte.

Per començar s’ha d’analitzar els objectius establerts a l’inici del desenvolupament

Per concluir es pot aformar que el desenvolupament d’aquest treball ha estat exitós, donat que s’han pogut completar tots els requeriments i s’han assolit la gran majoria dels objectius.

# POSSIBLES AMPLIACIONS

En aquest apartat s’exposaran les millores que pot tenir el producte de cara al futur.

## ASSIGNAR ALERTAS A DIFERENTS ROLS

Un element a millorar és poder assignar la mateixa alerta a diferents rols. El motiu és que per evitar la redundància de dades s’hauria d’implementar aquesta millora, ja que amb el model actual s’hauria de crear una altra alerta amb la mateixa informació.

Perquè funcionés correctament s’hauria de modificar la base de dades perquè permeti una assignació entre una alerta i diferents rols. Aquesta modificació hauria de venir a través d’una taula intermèdia entre la relació de les taules *Roles*  i *Warnings*.

## *LOG* DELS VALORS DE LES ALERTES

Un aspecte senzill d’implementar pot ser un *log* de les alertes. Aquest *log* hauria de poder mostrar a l’usuari l’evolució dels valors que ha tingut aquella alerta. A més es podria donar informació sobre quan aquella alerta té valors preocupants. Per exemple si cada dilluns el percentatge d’ocupació dels llits per malalts amb Covid-19 té valors greus, aquest sistema podria avisar, el dia anterior o abans que l’usuari inici la seva jornada laboral, d’aquest fet.

El mecanisme de funcionament podria ser senzill, un gràfic on mostra els valors en les últimes 24 hores i, fins i tot, fent una predicció de les futures hores.

## MILLORAR LA SEGURETAT EN L’APLICACIÓ

L’última ampliació que es podria fer és millorar la seguretat de l’aplicació.

Un aspecte a millorar podria ser la comunicació amb l’API, que en comptes de posar les dades en l’URL, s’hauria de posar en un JSON i així es pot encriptar el missatge.

Una altra millora en la seguretat pot ser que només els autoritzats puguin modificar les dades en el servidor web. És a dir, que per cada institució mèdica, els que tenen el rol d’administrar aquella institució, només puguin editar aquella institució mèdica.

# BIBLIOGRAFIA

[1] InterSystems, *IRIS for Health*. https://www.intersystems.com/products/intersystems-iris-for-health/, 2018. [En línea]. Disponible en: https://www.intersystems.com/products/intersystems-iris-for-health/

[2] H. Dalianis, «Chapter 2 - The History of the Patient Record and the Paper Record», en *Clinical Text Mining*, Cham: Springer International Publishing, 2018, pp. 5-13. doi: 10.1007/978-3-319-78503-5.

[3] R. F. Gillum, «From Papyrus to the Electronic Tablet: A Brief History of the Clinical Medical Record with Lessons for the Digital Age», *AJM*, p. 5, oct. 2013, doi: 10.1016/j.amjmed.2013.03.024.

[4] A. Winter, R. Haux, E. Ammenwerth, B. Brigl, N. Hellrung, y F. Jahn, «2.4 Importance of Systematic Information Management», en *Health Information Systems - Architectures and Strategies*, Second Edition., Springer, 2011, pp. 12-16.

[5] Reinhold Haux, «3.1. The 1st line: towards computer-based information processing tools», en *Health information systems — past, present, future*, 2006, p. 271. [En línea]. Disponible en: 0.1016/j.ijmedinf.2005.10.001

[6] G. O. Barnett *et al.*, «COSTAR—A computer-based medical information system for ambulatory care», *Proc. IEEE*, vol. 67, n.o 9, pp. 1226-1237, 1979, doi: 10.1109/PROC.1979.11438.

[7] «eHealth Resolution», World Health Organization, Geneva, 58th World Health Assembly. Resolution 28, may 2005. Accedido: ene. 01, 2021. [En línea]. Disponible en: https://www.who.int/healthacademy/media/WHA58-28-en.pdf

[8] Wolters Kluwer Health, *UpToDate*. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.uptodate.android

[9] EBSCO Information Services, *Dynamed*. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.ebsco.dmp&hl=en&pcampaignid=pcampaignidMKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1

[10] A. Winter, R. Haux, E. Ammenwerth, B. Brigl, N. Hellrung, y F. Jahn, *Health Information Systems: Architectures and Strategies*. London: Springer London, 2011. doi: 10.1007/978-1-84996-441-8.

[11] A. Winter, R. Haux, E. Ammenwerth, B. Brigl, N. Hellrung, y F. Jahn, «5.5 A Reference Model for the Domain Layer of Hospital Information Systems», en *Health Information Systems - Architectures and Strategies*, Second Edition., Springer, 2011, pp. 70-71.

[12] HIMSS, «Strategic Interoperability in Germany, Spain & the UK». 2014. [En línea]. Disponible en: https://www.himss.eu/file/191/download?token=goNVaM8t

[13] A. Winter, R. Haux, E. Ammenwerth, B. Brigl, N. Hellrung, y F. Jahn, «6.5.4.1 Health Level 7 (HL7) Version 2», en *Health Information Systems: Architectures and Strategies*, London: Springer London, 2011, pp. 149-151. doi: 10.1007/978-1-84996-441-8.

[14] A. Winter, R. Haux, E. Ammenwerth, B. Brigl, N. Hellrung, y F. Jahn, «6.5.4.2 Health Level 7 (HL7) Version 3», en *Health Information Systems: Architectures and Strategies*, London: Springer London, 2011, pp. 151-152. doi: 10.1007/978-1-84996-441-8.

[15] «HL7 in Personal Health System component’s integration for Mental Health Treatment».

[16] S. Subramanian, «DICOM Basics using Java - Extracting Image Data», oct. 06, 2014. https://saravanansubramanian.com/extractdicomimagedata/

[17] S. O’Dea, «Mobile operating systems’ market share worldwide from January 2012 to October 2020», *Statista*, nov. 30, 2020. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (accedido dic. 31, 2020).

[18] Facebook, *React Native*. https://reactnative.dev/, 2015. [En línea]. Disponible en: https://reactnative.dev/

[19] Microsoft, *Xamarin*. 2011. [En línea]. Disponible en: https://dotnet.microsoft.com/apps/xamarin

[20] Google, *Firebase*. 2011. [En línea]. Disponible en: https://firebase.google.com/