Raquel Acedo i Miriam Conde

Introducció

Bon dia, som la Raquel Acedo i la Miriam Conde. Avui us presentarem una aplicació que incorpora un xat i un fòrum informatiu, entre altres coses, vinculada a un institut de Sant Andreu de la Barca. Aquesta aplicació és una aplicació mòbil que serveix tant per iOS com Android i s'anomena APPALAU, acord amb la web ja existent de l'institut. Per tal de poder-la fer servir tant per iOS com per Android l'hem programada amb la plataforma IONIC.

Tecnologies

IONIC és un framework open source. Té un conjunt d'eines que faciliten el desenvolupament i les proves. Ho podem desenvolupar per múltiples plataformes amb el mateix codi. IONIC adapta el seu disseny segons la plataforma utilitzada. Angular és qui realitza la lògica de validació entre pantalles i la manipulació d'HTML.

Amb IONIC, crear un projecte es fa a partir de la línia de comandes, juntament amb la creació de les diferents pàgines, components o serveis que es poden necessitar.

Per poder començar a programar hem hagut de fer diverses instal·lacions i aprendre a programar en diferents llenguatges nous per a nosaltres. Ara us farem una pinzellada de les tecnologies que es necessiten i dels llenguatges que hem utilitzat.

Respecte a les tecnologies, aquestes són algunes de les que es recomanaven instal·lar. S'ha de tenir un navegador per fer les proves a l'emulador i un entorn de programació com és el **Visual Studio Code**. Per fer proves amb Android hem instal·lat l'**Android Studio** i en cas de voler fer proves amb iOS hagués estat necessari instal·lar l**'XCode**. Per a que IONIC i Apache Cordova funcionin, s'ha d'instal·lar el JDK versió 8 i el Git. Per la base de dades hem utilitzat **IONOS**, una plataforma no gratuïta que et dóna l'oportunitat de crear pàgines web amb facilitat i fer bases de dades.

Angular CLI és una eina de línia de comandes que facilita la creació, generació, execució, testing i deploy per aplicacions web i mòbil. Ens facilita molt el procés d'inici de qualsevol aplicació amb Angular, ja que en pocs minuts ofereix l'esquelet d'arxius i carpetes que es necessitaran, juntament amb una quantitat d'eines ja configurades.

Per fer les proves en dispositius físics es pot fer servir tant Cordova com Capacitor.

Apache Cordova és un marc de desenvolupament mòbil de codi obert. Li permet utilitzar tecnologies web estàndard per al desenvolupament multiplataforma. Els plugins són una part integral de l'ecosistema de Cordova i proporcionen una interfície perquè Cordova i els components natius es comuniquin entre si. Això li permet invocar codi natiu des de JavaScript.

Pel que fa a **Capacitor** és un entorn d'execució natiu multiplataforma que facilita la construcció d'aplicacions web modernes que s'executen nativament a iOS, Android i Web.

Node.js és un entorn en temps d'execució multiplataforma de JavaScript que ens permet executar en el servidor, de manera asíncrona, dissenyat per crear aplicacions de xarxa escalables.

Raquel Acedo i Miriam Conde

Llenguatges

Pel que fa als llenguatges, hem hagut d'aprendre a programar en Angular i TypeScript, principalment. Per fer la connexió a la base de dades hem utilitzat el format client-servidor amb PHP i per les funcions d'estil hem utilitzat SASS, el qual és molt semblant a CSS.

En la imatge, podem veure com a partir d'Angular, SASS, HTML i altres llenguatges, podem crear aplicacions multiplataformes que serveixen tant per iOS, Windows i Android, entre d'altres.

Angular és l'encarregat de donar forma al projecte, ja que s'estructura en mòduls i components, formant una estructura d'arbre.

Angular com a llenguatge es caracteritza per l'estructura d'etiquetes amb la possibilitat d'afegir filtres i directives.

A més, utilitza Databinding per a què els valors de les variables de JavaScript es vegin o s'actualitzin al template d'HTML.

- Als exemples es mostren les diferents formes de referir-nos a les variables:

  Amb dobles claus mostrem el valor d'una variable a l'HTMLl: quan fiquem dobles claus de info.data es mostrarà el valor guardat en aquesta variable.

- Amb corxets passem dades del ts a l'HTMLl. Amb parèntesis passem dades de l'HTML al ts. A l'exemple, la funció ngModel que serveix per associar una variable a un camp, apareix amb parèntesis i corxets generant bidireccionalitat. En el cas de la funció ngSubmit entre parèntesis envia informació al ts.

- Una altra qüestió important és l'existència de funcions i decoradors propis. Així, la funció ngOnInit permet executar un codi en el moment de carregar la pàgina, a l'exemple crida a la funció per carregar informació.

- Una de les funcionalitats dels decoradors és permetre el pas de variables entre arxius HTML. A l'exemple, amb Input s'indica que es rep el valor de fileAbsences i amb Output s'envia dates. Aquestes variables apareixen a l'HTML amb corxets.

Les etiquetes que més hem utilitzat són:

- L'etiqueta **hi** amb i de l'1 al 3 defineix els encapçalats o apartats que formen el document. La mida varia, sent h1 la més gran i h3 la més petita.

- La **p** és l'etiqueta que delimita el paràgraf i s'utilitza tal com es pot veure a l'exemple de dalt.

- L'etiqueta **div** significa divisió i s'utilitza per crear seccions o agrupar continguts, apareix en l'últim exemple.

- **br** és una etiqueta que apareix en el primer exemple i s'utilitza per fer un salt de línia bàsic.

- L'etiqueta **span** és un contenidor de text genèric i serveix per aplicar estil al text o agrupar elements en línia.

- **strong** és apropiada per marcar amb especial èmfasi les parts més importants d'un text. En el primer exemple veiem que volem donar pes en dues parts.

- La **b** és l'etiqueta que indica que el text ha de ser representat en negreta, de la tipografia que s'estigui utilitzant. La veiem en el segon exemple i malgrat que sembla el mateix que l'etiqueta strong, la diferència és que aquesta només indica l'estil visual del text.

- **form** és una etiqueta que representa un formulari que conté controls interactius que permeten a un usuari enviar informació a un servidor web.

- L'etiqueta **img** representa una imatge, és a dir, un recurs extern que pot ser incrustat en el cos d'un document.

TypeScript és un llenguatge de programació construït per sobre de JavaScript. De manera que tot el codi escrit en JavaScript és vàlid per TypeScript. Però el contrari no és cert. És a dir, com els navegadors no entenen TypeScript, cal compilar-lo a JavaScript abans d'utilitzar-lo en un navegador.

Com podem veure en l'exemple de la dreta, un programa **TypeScript** es compon de mòduls, funcions, variables, declaracions i expressions. Tot fitxer ts comença amb les importacions de les classes externes que es necessitaran, seguidament un decorador. En aquest cas, el decorador utilitzat és el Component, el qual indica el nom de l'etiqueta que s'ha d'usar per utilitzar aquest component en un fitxer HTML, el fitxer HTML al que està directament relacionat i els fitxers d'estil als que pot accedir a més d'aquells globals. Per poder fer servir aquesta classe en una altra, hem de fer l'exportació de la classe. Les funcions es diferencien en dos tipus, en les funcions anònimes com és el cas del mètode getProfile i les funcions com les coneixem fins ara, com és el cas de loadProfile. Les funcions anònimes no especifiquen un nom.

Algunes de les diferències entre JavaScript i TypeScript apareixen en el conjunt d'exemples de l'esquerra. Per exemple en la primera comparació, podem veure que en el TypeScript s'ha d'indicar el tipus de variable, això també succeeix en la segona comparació dins de la definició de la funció. En la tercera comparació, en TypeScript també afegim el tipus, obtenint així la màxima potència del TypeScript, amb tots els ajuts i la detecció d'errors.

PHP és un llenguatge de programació destinat a desenvolupar aplicacions i pàgines web, afavorint la connexió entre els servidors i la interfície d'usuari.

Algunes de les característiques sintàctiques de PHP són:

- Per iniciar un arxiu PHP s'escriu: símbol de menor, interrogant i php; per finalitzar el document s'escriu interrogant i símbol de major. Aquesta etiqueta indica on comença i finalitza la interpretació del codi.

- Les variables porten el símbol de dòlar davant seguit del nom de la variable. En els exemples podem veure de diferents tipus.

- echo json_encode($var) retorna la representació JSON del valor donat. En l'exemple, torna el de la variable ret. Amb el echo, aconseguir que es mostri per pantalla.

- Permet la definició de classes i funcions mitjançant la paraula function seguida del nom de la funció i els paràmetres, que podrem referenciar des d'altres arxius PHP. A l'exemple es mostra la definició de la funció validarUser, com instanciar la classe sql_server i cridar a la funció executeQuery d'aquesta classe.

- Al segon exemple es fa servir el mètode header per enviar capçaleres sense format, en aquest cas amb el paràmetre Access Control Allow Origin per donar accés a l'aplicació.

- La variable $_GET emmagatzema la informació que es rep de l'URL a través del mètode GET provinent del formulari. En l'exemple, creem una variable $idUser que emmagatzemarà la informació de camp 'idUser'.

Una de les característiques més interessants de MySQL és que permet recórrer bases de dades multiusuari a través del web i en diferents llenguatges de programació que s'adapten a diferents necessitats i requeriments.

A la imatge, podem veure l'estructura de taules que hem utilitzat per a la base de dades on s'emmagatzemarà tota la informació necessària. Per accedir a la base de dades, s'han de fer instruccions SQL específiques en MySQL, les quals nosaltres sol·licitem en PHP. L'aplicació de servidor respondrà amb la informació sol·licitada que rebrà l'usuari.

A l'exemple es veu una sentència de selecció emmagatzemada a la variable $sql: "SELECT … FROM … WHERE"

Els pre-processadors **CSS** són eines per als desenvolupadors de llocs web, que permeten traduir un codi de fulls d'estil no estàndard, específic del pre-processador en qüestió, a un codi CSS estàndard, comprensible pels navegadors. Tenim dues sintaxis, la sintaxi SASS i la sintaxi SCSS que és la que nosaltres hem utilitzat.

Nosaltres hem utilitzat bàsicament la definició de funcions d'estil a partir de la sintaxi.nom de la funció i entre claus les declaracions de tipus universals. En la primera funció podem veure la sintaxi var i entre parèntesis posem el nom de la variable, en aquest cas ion color primary, aquesta variable de color prové de l'arxiu variable.scss que més endavant comentarem, però que bàsicament té definits els colors bàsics de l'aplicació. A més, com a característica rellevant trobem l'!important. Això indica que sense tenir en compte el que abans hi havia s'ha d'executar sempre aquesta variable. Per acabar, podem veure les diferents maneres de fer referència colors, la ja esmentada variable, els colors amb nom o amb l'RGB.

Implementació

En aquest esquema podem veure l'estructura del projecte utilitzada, en forma d'arbre, i les interaccions amb la base de dades i les notificacions push emeses.

1) Abans de començar

Per crear un projecte, des d'una terminal executem: `ionic start APPALAU blank`. Aquesta comanda crea un projecte d'IONIC anomenant APPALAU a la carpeta on estiguem situats amb una plantilla buida. Un cop creat, ens movem a la carpeta del projecte on llancem el servidor fent `ionic serve`.

Amb això, a l'obrir el projecte al Code trobem:

➔ **index.html**: És el punt d'entrada d'una aplicació web i ajuda a IONIC a què aquestes aplicacions es puguin desplegar en la web.

- ➔ **global.scss**: En aquest fitxer afegim funcions que volem aplicar globalment als diferents templates d'HTML, com és el cas de les funcions per la informació o els colors que utilitzarem al xat.

- ➔ **theme/variable.scss**: inclou la definició dels colors primary, sencondary i terciary d'entre d'altres per al tema clar i fosc, de manera que es fa extensiu a tota l'aplicació. A més de característiques pròpies d'Android i iOS.

- ➔ **assets**: carpeta on es troben tots els recursos visuals com imatges i icones que utilitzarem.

- ➔ **environments**: carpeta que inclou dos fitxers de variables de configuració global de l'aplicació.

- ➔ **app**:

    - ◆ app-routing.module.ts: Aquest és l'arxiu principal de rutes, quan creem cada pàgina, automàticament s'indica el path que ha de seguir en aquest fitxer.

    - ◆ app.component.html/ts: És el primer arxiu a carregar després que el sistema d'enrutament s'hagi validat.

    - ◆ app.module.ts: Carrega components i mòduls útils per a totes les pàgines.

A partir d'aquesta estructura inicial s'han de definir la resta d'elements des de la línia de comandes mitjançant l'estructura `ionic g` o `ionic generate` seguit d'allò que volem crear tal com mostrem tot seguit.

- ◆ components: per crear un component fem `ionic g c components/absences --spec=false` i, per a englobar tots els components creem un mòdul: `ionic g m components/components --spec=false`

    - *absences.component.ts/html/scss*: són els tres fitxers que es creen en executar la primera comanda. En aquest cas, es crea un component, és a dir, una part d'una pantalla principal.

    - *components.module.ts:* Aquest fitxer engloba tots els components que creem, els declara i els exporta per tal de poder utilitzar-los a les pàgines

- ◆ interfaces/interface.ts: en aquest fitxer creem les interfícies i classes que després utilitzarem a les diferents pàgines. Per crear-ho fem: `ionic g interface interfaces/interface --spec=false`

- ◆ pages/home: per crear una pàgina fem: `ionic g page pages/home --spec=false`

    - *home-routing.module.ts:* habilita el path per entrar en aquesta pantalla i indica, en cas d'haver-hi tabs, els paths per cada nova subpàgina.

    - *home.module.ts:* Indiquem els components que utilitzarem en aquesta pàgina sense que l'app.module.ts tingui la referència de tots els components creats a totes les pàgines.

    - *home.page.ts/html/scss:* implementació de la pàgina, allò que volem que succeeixi en aquesta pantalla. Podem crear variables, propietats i mètodes al .ts que seran cridats des de l'HTML, el qual és la interfície visual on crearem els components. Les funcions d'estil que només serveixin per a aquesta pàgina es poden posar en aquest fitxer de SCSS.

- ◆ services: per crear un servei fem: `ionic g service services/data`

- *data.service.ts:* Aquest fitxer engloba totes les funcions encarregades de fer les consultes a la base de dades.
- *GlobalVar.ts:* la variable global és l'encarregada de guardar les dades d'autenticació quan s'inicia sessió per poder fer les consultes a la base de dades.
- *push.service.ts:* Aquest servei és l'encarregat de rebre les notificacions push i fer el tractament d'aquestes.

2) Base de Dades

Per accedir a la base de dades, fem una connexió de l'aplicació amb el backend d'aquesta, que està format per arxius PHP i MySQL. A continuació mostrarem l'exemple de connexió per accedir a la Informació.

Dins del projecte, creem un servei, on definirem totes les funcions de consulta que s'enviaran al PHP. Mitjançant la funció `http.get` fem una crida a l'arxiu PHP amb les variables que enviem a l'URL. En el cas de la Informació, la funció getInfo afegeix a l'URL base l'arxiu PHP que ha de cridar juntament amb l'usuari i el token que es propi de cada usuari i controla la validesa de la connexió.

L'arxiu info.php s'encarrega de facilitar l'accés a la base de dades i retornar la informació a l'aplicació. Primer llegeix les variables que arriben a l'URL, comprova que siguin correctes amb la funció validarUser de sql_client, posteriorment crea la sentència SQL que es vol enviar a la base de dades i mitjançant la funció enviar_consulta de sql_client es fa l'accés i s'obté la informació demanada. Finalment, si no hi ha cap problema, retorna un json amb l'informació demanda.

Com hem vist, hi ha dos accessos a l'sql_client. En primer lloc, la funció validarUser recull l'identificador d'usuari i el token per fer la validació. Per fer-la, accedeix a la base de dades per consultar les dades referents a l'usuari i mitjançant la funció SHA-1, les encripta i les compara amb el token.

El segon accés és una consulta a la base de dades a través de la classe sql_server retornant la informació desitjada o un missatge d'error en cas de produir-se.

L'arxiu sql_server.php inclou la classe sql_server, que realitza l'accés a la base de dades i retorna la resposta mitjançant la variable info que inclou el número de files, els camps, les dades i l'estat, que ens indica si la consulta ha estat correcta.

La funció que realitza les consultes és executeQuery que rep com a paràmetre la sentència SQL que volem executar en la base de dades. Primer de tot realitza la connexió, llença la sentència mitjançant la instrucció query de la classe MySQL i si rep dades, les prepara per retornar-les o retorna el missatge d'error en cas contrari.

sql_server fa l'accés a MySQL, en aquest cas implementat mitjançant una base de dades en IONOS, la imatge mostra la taula corresponent a Informacions. Es defineix amb l'identificador de la informació, idInformacio, l'identificador d'usuari que redacta la informació, la data, el tema i el contingut.

A l'esquerra de cada diapositiva podem veure la pàgina o component creat.

3) LOGIN

Per crear la pàgina de Login hem fet: `ionic g page pages/login --spec=false`, on el fitxer spec és un de prova i l'indicar false estem dient que no l'instal·li.

Tal com ja hem explicat, quan parlàvem del llenguatge TypeScript, el fitxer comença amb les importacions i el decorador. Tot seguit s'implementa la classe amb la definició de les variables user i error que faràn l'intercanvi de dades entre l'HTML i el ts i el constructor que defineix les variables que es faran servir. En aquest cas la funció ngOnInit només s'ha d'esmentar, però no d'implementar, ja que no es requereix cap acció en carregar la pàgina. La funció que sí què és necessària és la funció login, la qual és un mètode asíncron per tal de garantir que finalitzi abans de canviar de pàgina. Aquesta funció fa una crida al mètode login de la classe data service, rebent una promesa amb les dades de connexió i inicialitza la variable global que es farà servir en tot el projecte.

Al fitxer SCSS creem una funció d'estil perquè la imatge mostrada a la pàgina tingui la mida indicada.

Al fitxer HTML és on estructurem la pàgina i indiquem què ha de sortir i com. Per fer la capçalera d'aquesta pàgina hem creat un component fent `ionic g c components/header-login --spec=false` on indiquem el color de la capçalera, la imatge i el títol que volem que es mostri amb les etiquetes d'Angular ion-header, ion-img i ion-title. Seguidament, afegim la imatge amb l'etiqueta d'HTML img i la funció d'estil creada amb anterioritat i creem el formulari que recollirà les dades d'usuari introduïdes amb les etiquetes form i ion-input. El botó per fer el login el fem amb ion-button. Per acabar, pel peu de pàgina de totes les pàgines utilitzades hem creat un component fent `ionic g c components/footer --spec=false` on creem el peu amb ion-footer i li indiquem el color.

4) HOME

Aquesta és la pàgina d'inici, d'on surten totes les altres pàgines. L'aplicació està basada en una estructura de tabs, les quals són pàgines que aniran canviant segons la selecció feta.

Per poder crear aquesta estructura, a l'HTML hem creat en primer lloc un component per a la capçalera que servirà per a quasi totes les altres pàgines. En aquest component indiquem el color de la capçalera, la imatge i el títol que volem que es mostri amb les etiquetes d'Angular ion-header, ion-img i ion-title, tal com hem fet amb header del login. En aquest cas, a més, hem afegit un botó per a tancar sessió i així tornar a la pantalla de login amb ion-button. Per fer la navegació a l'altra pàgina, a l'arxiu ts hem creat un Controlador de Navegació i, dins una funció asíncrona, hem indicat amb el mètode navigateRoot(), el path del login, buidant, així mateix, la variable global. D'aquesta manera, podem tornar-nos a connectar amb un altre usuari sense trepitjar informació.

Com hem comentat a l'inici, en aquest fitxer hem d'indicar els paths de les pàgines filles, per tal de poder navegar entre les diferents opcions dels tabs. Com a pantalla principal, creem el path per defecte a informació.

Ara començarem a parlar de les subpàgines dels tabs, començant per perfil.

5) PROFILE

Per guardar les dades del perfil hem creat una classe Profile definida al fitxer interface.ts on podem veure els diferents atributs que aquesta té, es tracta de les dades bàsiques de l'alumne i un array amb els usuaris que té associats.

A la funció ngOnInit carreguem el perfil amb la funció loadProfile. Aquesta funció és l'encarregada de fer una crida al mètode getProfile de la classe data.service, revent una promesa amb les dades del perfil i inicialitza la variable profile que s'utilitzarà per carregar i editar el perfil.

Al fitxer SCSS creem una funció d'estil per a què el nom de l'alumne s'escrigui en negreta i en majúscules independentment de com estigui escrit a la base de dades.

A les pàgines que surten directament de la pàgina d'inici no cal fer capçalera, ja que es mostra la creada a la pàgina home. Per posar la foto de l'alumne, creem un avatar amb ion-avatar i fem un if per saber si l'alumne en qüestió té imatge o no. Pel nom, tal com he dit, utilitzem la funció d'estil creada i es recull de la variable creada a l'arxiu ts. Per mostrar les dades personals, les de grup i les de contacte hem creat tres components. A cadascun d'ells, els hi passem amb un binding la informació que necessiten mostrar.

En el cas de les dades personals i les dades de grup, fem servir una targeta amb ion-card i etiquetes HTML per mostrar la informació.

En el cas de les dades de contacte, hem creat una estructura amb ion-grid i ion-card per diferenciar la visualització en pantalles de mida diferent. Dins de l'ion-card introduint un ngFor per crear targetes específiques per cada usuari definit.

Per poder editar el perfil, hem creat un botó amb només una icona a la part esquerra del peu de pàgina. Quan cliquem aquest botó, s'executa la funció editProfile del fitxer ts.

Aquesta funció és asíncrona i el que fa és crear un modal, és a dir, una subpàgina, i li passa amb el componentProps la variable profile amb totes les dades que necessita. Un cop tanquem el modal amb el botó de voler guardar les dades, recollim la informació que hem introduït i actualitzem l'usuari amb un accés a base de dades amb la funció updateUser de la classe data service i recarreguem el perfil perquè es mostrin els canvis.

6) EDIT PROFILE

Les variables creades en aquest arxiu ts tenen el decorador Input per tal de poder recollir les dades de l'HTML del Profile.

A la funció ngOnInit carreguem l'usuari amb la funció loadUser. Aquesta funció és l'encarregada de fer una crida al mètode getUser de la classe data service, revent una promesa amb les dades de l'usuari i inicialitza la variable user2 que s'utilitzarà per mostrar les dades d'aquest usuari a l'HTML.

El fitxer HTML comença amb la creació d'un nou component de capçalera on indiquem el color de la capçalera i el títol que volem que es mostri amb les etiquetes d'Angular ion-header i ion-title. A més, hem afegit un botó per tancar sense guardar els canvis, de manera que en clicar al botó 'Surt' s'executa la funció closeEdit. Aquesta funció està definida a l'arxiu ts del component, on creem al constructor un Controlador del modal i el tanquem sense retornar res amb el mètode dismiss.

Seguidament, hem creat un formulari amb l'etiqueta form per recollir les dades introduïdes per l'usuari a editar. En cas del codi postal i de l'email, hem introduït un patró que en cas de no complir-se no deixa desar els canvis. És aquí on utilitzem la variable users, a l'ngModel per recollir allò que han introduït, i la variable user2 per mostrar les dedes de contacte com a placeholder. La resta de placeholders es mostren a partir de les dades que li hem enviat des de la pàgina del perfil.

Per acabar, hem creat un botó a peu de pàgina per tancar el modal guardant els canvis. Al clicar el botó s'executa la funció acceptChanges.

Aquesta funció s'encarrega de tancar el modal amb el mètode dismiss i reenvia la variable users a través del camp users al perfil. Aquest tractarà aquestes dades tal com hem explicat amb anterioritat.

7) CHAT-HOME

Es tracta d'una pàgina amb les matèries que fa l'alumne i la possibilitat d'accedir a un xat per cada una d'elles.

Al constructor recuperem les imatges guardades a la carpeta assets i les assignem a la variable images per mostrar-les amb un ion-avatar al costat de cada matèria.

Tot seguit, la funció ngOnInit s'encarrega de cridar a la funció loadChatHome que mitjançant la funció getChatHome de dataService s'encarrega de recuperar les assignatures que cursa l'alumne de la base de dades i el nombre de missatges no llegits del corresponent xat.

A l'HTML, mitjançant l'estructura ion-item, ngFor i l'acció (clic) mostren la llista de matèries amb les dades pròpies de cada matèria. En el cas dels missatges no llegits fem servir ion-chip i ngIf per destacar aquest número quan hi hagi missatges sense llegir.

L'acció "clic" ens permet definir cada línia com un objecte que podem clicar i així accedir al xat concret mitjançant la funció openChat. Es tracta d'una funció asíncrona que crea un modal i li passa amb el componentProps l'assignatura i les imatges. Un cop tanquem el modal amb el botó de tornar enrere recarreguem la pàgina d'inici del xat.

8) CHAT

Es tracta del xat de la matèria, on la família i el professor poden enviar-se missatges.

La lògica de la programació continguda a l'arxiu ts, ens mostra el procés de càrrega i enviament de missatges. Primer de tot definim un listener en mode "escolta" amb pushService per si arriba un push poder recarregar els missatges. Aquesta càrrega de missatges es realitza mitjançant la funció loadMsg que a la seva vegada cridarà a la getMsg de dataService que, com ja hem comentat diverses vegades, retornarà una promesa amb la informació procedent de la base de dades per tal de carregar els missatges a l'arxiu HTML.

L'estructura de l'arxiu HTML és una mica diferent de la resta de pàgines. En aquest cas definim la capçalera directament a la pàgina per tal d'introduir la imatge i el nom de l'assignatura i un botó per tornar enrere, implementat a l'arxiu ts amb dismiss sense retornar dades.

El contingut dels missatges l'implementem amb un nou component "app-message". Aquest component, recull l'array de missatges a la variable messages i mitjançant un ngFor afegeix un nou element del component "msg" per cada missatge. Aquest component recull cada missatge i el mostra amb una targeta, donant el nom de la persona que l'envia, la data i l'hora d'enviament. Per tal de fer més visible el xat, els missatges apareixen per colors segons l'usuari propietari i els nostres apareixen carregats a l'esquerra. Això ho fem mitjançant un ion-grid i la definició de classes de SCSS.

Finalment, a l'arxiu HTML del xat, hem creat un peu de pàgina amb la possibilitat d'enviar un nou missatge al xat amb l'estructura form, ion-input i ion-button, que cridarà a la funció send, encarregada de cridar a la funció sendMsg de dataServices que enviarà el missatge al PHP corresponent per tal que ho afegeixi a la base de dades i generi el push corresponent.

9) INFORMATION

Es tracta d'una pàgina que mostra informació de caire general generada des del centre. Seguint una estructura molt similar a la del xat, l'arxiu ts preveu la carregada de les notícies i la definició d'un listener

de pushService per poder recarregar la informació si la pàgina estava oberta. La càrrega de les notícies la realitzem amb la funció loadInformation que cridarà al mètode getInfo de la classe data service, rebent una promesa amb les notícies pertinents a aquell alumne i inicialitza la variable infos.

Seguint l'estructura explicada al xat, l'arxiu HTML crida al component infos, passant-li les informacions, que rebrà el component mitjançant el corresponent Input. Al seu torn, el component Infos, fa servir una estructura ion-grid, ion-row, ion-col i el bucle ngFor per cridar al component info passant-li una notícia concreta que rebrà a la variable info definida amb el seu corresponent Input. Així, serà aquest darrer component el que mostrarà la notícia fent servir un ion-card. Per millorar la part estètica, en aquest cas hem definit unes funcions específiques a global.scss

10) Push

Per realitzar el xat i informació, definim un sistema d'alertes (push) mitjançant OneSignal i Firebase. Per poder implementar els push de OneSignal primer de tot hem hagut d'instal·lar en el projecte els plugins de Cordova-oneSignal executant `ionic cordova plugin add onesignal-cordova-plugin`.

L'arxiu push.service.ts recull la configuració i implementació del servei del OneSignal-Firebase. A la configuració iniciem el servei mitjançant la funció startInit, passant-li les APIKeys corresponents. Posteriorment, definim el comportament del terminal al rebre el push. En aquest cas, decidim que sigui una notificació.

Finalment, el comportament de l'aplicació en rebre o obrir la notificació. En el nostre cas, hem decidit que si l'aplicació no està oberta, en clicar sobre la notificació només obrirà l'aplicació. En el cas en que l'aplicació estigui oberta, definint un Listener (Event Emitter) per tal que si ens trobem a la pàgina de xat o d'informació aquesta recarregui els missatges.

En el cas de la pàgina del xat, en el constructor de la classe posem el Listener en escolta i si rep el push farà la crida al loadMsg.

Quan s'envia un missatge, el codi php cridarà la funció generar_push de sql_client. Aquesta funció accedeix a la base de dades per cercar els identificadors de OneSignal dels usuaris als quals ha d'arribar el missatge i genera un push per aquests identificadors mitjançant una crida al mètode curl, primer l'iniciem amb curl_init, establim les seves opcions amb curl_setopt, on definim l'http header amb les APIKeys, el tractament que ha de fer el servidor i els camps que ha de rebre (variable fields), finalment l'executem amb curl_exec i tanquem la connexió amb curl_close. Això enviarà un push als diferents usuaris donats d'alta.

11) DOCUMENTATION

Per guardar les dades de la documentació hem creat una interfície Document definida al fitxer interface.ts on podem veure els diferents atributs que aquesta té, els quals són strings.

A la funció ngOnInit carreguem les notícies de nova documentació amb la funció loadDocs. Aquesta funció és l'encarregada de fer una crida al mètode getDocs de la classe data service, revent una promesa amb les notícies de documentació pertinents a aquell alumne i inicialitza la variable "documents".

En aquest cas, l'HTML crea una targeta per a cada notícia amb un for. Les funcions d'estil utilitzades són les mateixes que per la informació. Per a totes les targetes es mostra la data a la qual és emesa, que

tracta aquesta documentació, un missatge comú indicant que hi ha un nou document a entregar al correu, el professor que ho necessita i la data de venciment.

12) FILE

En aquesta classe hem definit diferents variables per guardar les categories que s'utilitzaran la variable "categories" és un array d'string que guarda els strings tal com els rebem de la base de dades. La variable category inicialitzada amb absències que enregistrarà quina és l'opció seleccionada. I la variable nomCategories que s'inicialitza al constructor amb el nom correcte per mostrar-ho per pantalla.

A la funció ngOnInit carreguem les absències de l'alumne amb la funció loadFile. Aquesta funció és l'encarregada de fer una crida al mètode getFile de la classe data service, revent una promesa amb aquesta part de l'expedient i inicialitza la variable file.

Al fitxer HTML creem una capçalera amb un segment desplaçable amb les tres categories. Per canviar de categoria hem creat la funció changeFile.

Aquesta funció registra a la variable category la categoria que ha estat clicada i carrega la part seleccionada de l'expedient.

Per mostrar cada part de l'expedient, és a dir, les absències, les observacions i les incidències hem creat un component per a cadascuna, comprovant primer quina és la categoria seleccionada i passant-li la informació que necessita.

Ara explicarem cada component d'un en un.

13) ABSENCES

Encara ser un component, aquest fitxer també comença amb els imports, el decorador component i la implantació de la classe amb la definició de variables i del constructor. Les funcions definides en aquest component són aquelles que realitzen funcions cridades des de l'HTML.

L'estructura bàsica d'aquest component és una taula on s'indica la data on s'ha absentat l'alumne de classe i les sis hores possibles indicant amb una J la falta justificada, amb F la falta sense justificar i amb R el retard. La taula l'hem pintada de dos colors segons si l'índex era parell o imparell i per la capçalera hem creat una funció d'estil pròpia. Aquestes funcions les hem definides al global.scss. Per tal de justificar una falta, hem creat un botó que crida la funció justify.

Aquesta funció és asíncrona i el que fa és crear un popover, és a dir, una pàgina de mida reduïda sobre la pàgina principal i li passa amb el componentProps les dates que es poden justificar. Aquestes dates les trobem amb la funció selectDates que fa un recorregut de cerca i retorna la variable "dates". A més, la funció justify recarrega les absències fent la crida al mètode getFile de la classe data service.

14) JUSTIFY

Les variables creades en aquest fitxer porten el decorador Input per poder recollir la informació des d'un fitxer HTML. La variable justification recull la data seleccionada i el text de la justificació.

Per poder enregistrar la data i el text de la justificació creem un formulari. Per mostrar les dates fem un for i per seleccionar-les un selector. Per guardar els canvis hem creat el botó Acceptar i per tancar sense desar, el botó Cancel·lar. Aquests criden les funcions accept i cancel respectivament.

Ambdues funcions utilitzen el mètode dismiss del controlador popover que hem creat al constructor i, en cas d'acceptar, retorna la variable justification al component Absences.

15) COMMENTS

La variable fileComments definida amb el decorador Input és l'encarregada de rebre les dades des de la pàgina File.

Les observacions tindran el mateix estil que la informació, per això s'utilitzen les mateixes funcions d'estil. Creem una targeta per a cada observació amb la data, la tipificació, és a dir, si és un negatiu o positiu, el professor que l'ha posat i, en cas d'haver-hi, un comentari o detall.

16) INCIDENTS

La variable fileIncidents definida amb el decorador Input és l'encarregada de rebre les dades des de la pàgina File.

Les incidències tindran el mateix estil que la informació, i per això s'utilitzen les mateixes funcions d'estil. Creem una targeta per a cada observació amb la data, la tipificació, és a dir, si és una amonestació lleu, greu o molt greu, el professor que l'ha posat i, en cas d'haver-hi, un comentari o detall.

17) Al acabar de programar

Un cop hem acabat de programar i fer les proves a l'emulador del navegador podem fer proves en un dispositiu físic. Com hem comentat, es poden fer tant amb Cordova com Capacitor, però nosaltres ho hem fet amb Cordova. Per fer-ho funcionar s'han de seguir els següents passos:

1. Integrar Cordova al projecte fent `ionic integrations enable cordova --add`. Aquesta comanda ens modifica l'arxiu config.xml.
2. Modificar l'arxiu **config.xml** adaptant la versió, l'id, el nom de l'aplicació i altres atributs propis.
3. Preparar l'estructura del projecte amb `ionic cordova prepare android`. Aquesta comanda crea l'estructura platform/android. En cas que no funcionés degut a posibles modificacions a l'android studio, hem d'esborrar aquesta carpeta i tornar a repetir el pas 3.
4. Compilem el projecte android amb `ionic cordova build android`.
5. Posem el projecte en mode execució amb `ionic cordova run android -l --external`. Aquesta instrucció facilita una connexió mitjançant IP:port a l'aplicació.

Si tot funciona, l'aplicació està llesta per generar l'APK. Per generar-la s'han de seguir les següents passes:

1. Crear l'APK unsigned amb `ionic cordova build android --release` que genera el fitxer app_release_unsigned.apk que trobarem a la carpeta release d'android.
2. A android Studio fem build -generated signed APK per firmar-la. És molt important guardar els fitxers i les contrasenyes en un lloc segur, ja que qualsevol modificació que vulguem fer de l'aplicació caldran aquestes dades d'accés.

Un cop hem fet això, ens hem de donar d'alta al play store per a que tothom la pugui descarregar. Com aquesta és una versió de prova per l'institut, encara no l'hem donat d'alta.

Raquel Acedo i Miriam Conde

Resultat final

Ara, veurem una demostració de l'aplicació en funcionament.

- Iniciem sessió amb un usuari de prova amb les nostres dades.
- Podem veure que la pàgina principal és la d'Informació. Aquesta està ordenada de més nou a més antic. Quan es pengi una nova informació, arribarà una notificació al dispositiu.
- A documentació trobem, de més nou a més antic, la notificació de nova informació al correu, per quan és i per a què.
- A expedient tenim tres opcions. Comencem per absències. En aquesta pàgina trobem la taula amb les faltes d'assistència i la possibilitat de justificar una falta. En aquest cas, justifiquem la falta de primera hora del 12 d'abril.
- La pestanya d'observacions mostra els negatius i positius de l'alumne i, en la d'incidències les amonestacions que té i el motiu.
- Al perfil, podem veure la foto que té l'alumne i el nom en primer lloc. A més, podem veure les dades personals, les de grup i les de contacte.
- Hi ha la possibilitat de fer canvis en algunes de les dades esmentades. Es pot sortir sense guardar canvis o acceptant-los.
- Per últim, tenim el xat. A la pàgina principal podem veure les diferents assignatures de l'alumne on pot accedir i conversar amb el professor de l'assignatura i el tutor. Cada membre del xat té un color diferent per poder identificar-los. Quan hi ha un nou missatge s'envia una notificació push.
- Per tancar sessió, hem de clicar el botó Surt i podem tornar a iniciar sessió amb el mateix usuari o un altre.

Conclusions

Fins aquí el nostre projecte, on hem aconseguit crear una senzilla aplicació mòbil per a un institut, permetent així tenir a l'abast la informació de l'alumne en qualsevol moment. A més a més, també es té la possibilitat de poder comunicar-se amb els professors, justificar faltes, veure les entregues i assabentar-se de les últimes notícies gràcies al fòrum informatiu.

No obstant això, es podria perfeccionar i fer que la documentació pogués entregar-se des de la mateixa aplicació, sent una millora que es podria aplicar en futures actualitzacions.

Gràcies per la vostra atenció.