Social CMR

Antoni Joan Morlà Vanrell
Grau en Multimèdia
Enginyeria Web

Professor**: Carlos Casado Martinez**
Consultor**: Jordi Ustrell Garrigos**
Juny 2019

Títol del Treball:	Social CMR
Nom de L'autor:	Antoni Joan Morlà
Nom del Consultor:	Jordi Ustrell Garrigos
Nom del Professor:	Carlos Casado Martinez
Data de Lliurament:	*06/2019*
Àrea del Treball:	*Enginyeria Web*
Idioma del Treball:	*Català*
Paraules clau:	*Pàgina Web, Xarxa social, Xat.*
Resum del Treball	

L'objectiu d'aquest treball, és la creació d'una xarxa social per a distints ajuntaments. Aquesta eina proveeix d'un sistema de comunicació àgil i transparent entre els distints elements que formen una localitat.
Per una banda, els administradors i moderadors de l'ajuntament, podran crear noves publicacions, les quals podran ser seguides i comentades pels veïnats de la localitat. Un altre punt important consistirà amb la implementació d'un xat en temps real, on per exemple; els veïnats podran comunicar-se entre ells i podran consultar dubtes a l'ajuntament, sense necessitat d'haver d'apropar-si.
Per això, es desenvolupen un front-end i un Back-end per a construir aquest WebApp dinàmica. Pel front-end, emprarem Javascript i Angular 7 mentre que per a desenvolupar el Back-end construïm un API RESTful amb Node.Js i MongoDB. Per a poder treballar amb un sistema de rutes HTTP, s'emprarà ExpressJS.

Abstract

The goal of this work is the creation of a social network for a City Hall. This tool will provide a system of agile and transparent communication between the different elements that form a locality.
On the one hand, the administrators and moderators of the city council, will be able to create new publications, which can be followed and commented by the neighborhoods of the town. Another important point will be the implementation of a chat in real time, where for example; the neighbors will be able to communicate with each other and they will be able to consult doubts in the city council, without needing to approach it.
For this reason, we develop a front-end and a back-end to build this dynamic webApp. For the front end, we will use Javascript and Angular 7 while developing the Back-end we will build a RESTful API with NodeJS and MongoDB. In order to work with an HTTP path system, ExpressJs will be used.

Índex

Índex de figures

Índex de taules

1. Introducció

1.1 Context i justificació del treball

Social CMR neix gràcies a la necessitat de proveir als distints ajuntaments interessats, d'un eina per a poder millorar la transparència d'aquests front als seus veïnats. D'aquesta manera, els veïnats de la localitat, estaran informats en tot moment del que passa a la localitat.

Amb això, els ajuntaments compten amb una eina per a guanyar-se la confiança dels seus veïnats i que alhora aquests tinguin una sensació de que tenen un ajuntament més transparent. No menys important, es la implementació d'un xat a temps real. Aquest fet, ve motivat per les llargues cues que es fan a les portes dels ajuntaments, on més de la meitat dels integrants, podrien resoldre els seus dubtes per mitjà d'un xat.

El WebApp es composa d'una aplicació que permet als usuaris fer publicacions i que altres usuaris les puguin seguir, deixar de seguir, comentar, denunciar, etc. Per altra banda, també s'integrarà el xat abans esmentat, de tal manera que un usuari es pugui comunicar tant per missatgeria privada com per mitjà d'una sala de xat oberta amb la resta d'integrants.

L'accés a aquesta WebApp està disponible en una primera fase per mitjà de gairebé qualsevol navegador web actual. No obstant, en posteriors versions, també es trobarà disponible tant per a dispositius Android com per a dispositius IOS en format aplicació.

1.2 Objectius del treball

L'objectiu principal del treball es crear un WebApp responsive per a la comunicació àgil entre els usuaris registrats a l'aplicació. El lloc WebApp creat, compleix amb les especificacions següents:

- El sistema ha de ser capaç de permetre la concurrència d'un nombre elevat d'usuaris.
- Cal mantenir una comunicació àgil entre tots els clients que fan ús del sistema en un determinat instant.
- El lloc estarà allotjat a un servidor que compleixi amb els estàndards mínims de seguretat. En una posterior fase de producció, també comptarà amb un sistema de protecció front a desastres (*DR*) i un sistema de còpies de seguretat diari (*Backup*).
- Cal que el sistema estigui adaptat a les directrius de l'actual reglament europeu en matèria de protecció de dades (*RGPD*).
- Cal garantir la seguretat de les dades pròpies de cada usuari, emmagatzemant les credencials d'usuari solament de manera xifrada.
- El WebApp s'ha de visualitzar correctament en els distints dispositius des de els quals es permeti l'accés (*responsive*).
- El lloc ha de permetre la visualització de l'històric de publicacions filtrat per usuari.
- Els usuaris han de disposar d'un formulari d'accés prèvia sol·licitud a l'ajuntament.
- S'han de garantir mètodes per a suspendre temporal o definitivament l'accés dels usuaris que no compleixin amb la normativa bàsica d'us. Els administradors del lloc, han de disposar de permisos per a suprimir publicacions de qualsevol usuari del sistema.

1.3 Enfocament i mètode seguit

S'ha plantejat la realització d'una aplicació que permeti a qualsevol usuari, estar connectat constantment a la seva localitat i així conèixer l'ultima hora d'aquesta. De la mateixa manera, aquests podran plantejar qüestions a l'ajuntament, les quals seran públiques i així permetran que qualsevol altre usuari registrat, pugui llegir-les en el moment que tingui el mateix problema o pregunta.

2. Arquitectura del Sistema

2.1 Client

Per a poder establir i gestionar la connexió permanent amb el servidor, empram WebSocket. D'aquesta manera existeix una connexió persistent entre el client i el servidor, i totes dues parts poden començar a enviar dades en qualsevol moment.

Per a gestionar la informació que es passa per mitjà de l'explorador Web, hi haurà un controlador, el qual s'encarregarà de comunicar les peticions HTTP que qualsevol usuari realitza al servidor. Depenent de la resposta obtinguda per part del servidor, aquest controlador gestionarà contingut amb el sistema gestor de base de dades per mitja de WebSocket.

Amb l'ús de WebSocket, qualsevol de les dues parts (client, servidor) poden començar l'enviament de dades en qualsevol moment, sense necessitat d'haver de fer cap mena de sincronització entre ells. Aquesta utilitat, es farà servir per a portar a terme l'enviament de missatges al xat entre els usuaris.

2.2 Servidor

Per a la fase de desenvolupament, aquest projecte s'ha implementat sobre un servidor IIS(*Internet Information Services*) amb la seva versió *Express*, el qual es troba allotjat a un equip amb el sistema operatiu Windows 10.

Aquest equip disposa de les següents especificacions:

Model	IGGUAl PSIPCH336
Caixa	Micro ATX
Sistema Operatiu	Microsoft Windows 10 Pro
Processador	Intel Core i5-7500 3.4Ghz 6MB
Memòria RAM	8GB DDR4 2400MHz (1 x 8GB)
Disc Dur	120GB SSD SATAIII (Kingston)

Taula 1: Arquitectura del Sistema - Equip Servidor

Per a portar a terme el desenvolupament, s'ha facilitat un compte FTP a l'equip de desenvolupament per tal que aquest pugui accedir als directoris del projecte i així poder modificar els distints documents. Per poder accedir a la línia de comandes de l'equip servidor, s'ha habilitat un accés remot per mitjà de SSH.

Tot seguit, es mostra de manera esquematitzada, un breu resum de la constitució de l'equip servidor:

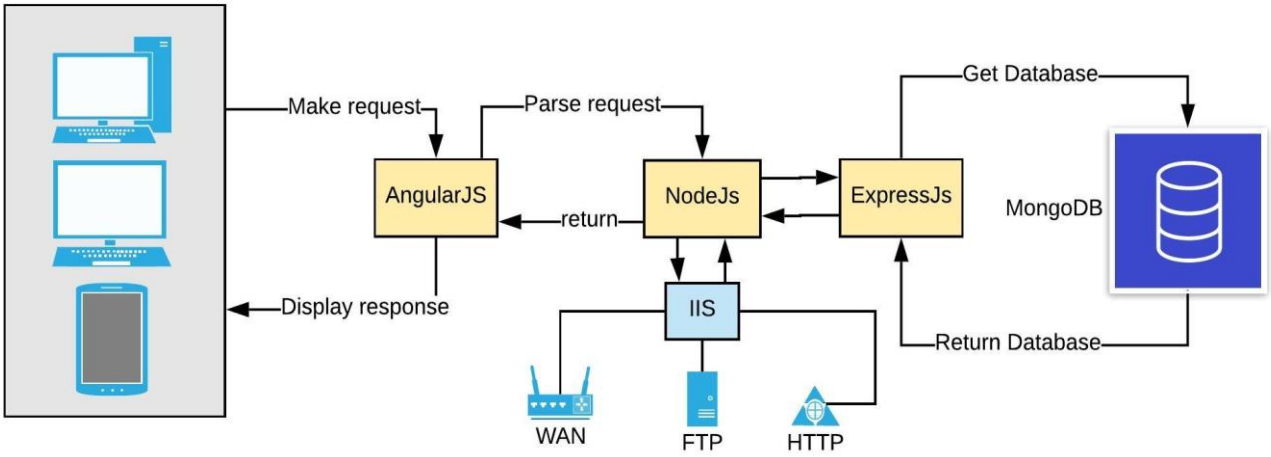


Figura 1: Arquitectura del Servidor.

2.3 Bases de dades

Per a la realització d'aquest projecte, s'ha emprat MongoDB, la qual és una base de dades de documents que ofereix una gran escalabilitat i flexibilitat, i un model de consultes i indexació avançat.

MongoDB emmagatzema dades en documents JSON flexibles, és a dir, cada document pot contenir diferents camps i les estructures de dades es poden anar modificant. El model de documents concorda amb els objectes del codi de l'aplicació, la qual cosa facilita treballar amb dades.

Les consultes ad hoc, la indexació i l'agregació en temps real permeten accedir a les dades i analitzar-les amb gran eficàcia. MongoDB és una base de dades distribuïda, per la qual cosa, proporciona una elevada flexibilitat per a futures modificacions del projecte. Cal destacar que és una base de dades de codi obert i d'ús gratuït.

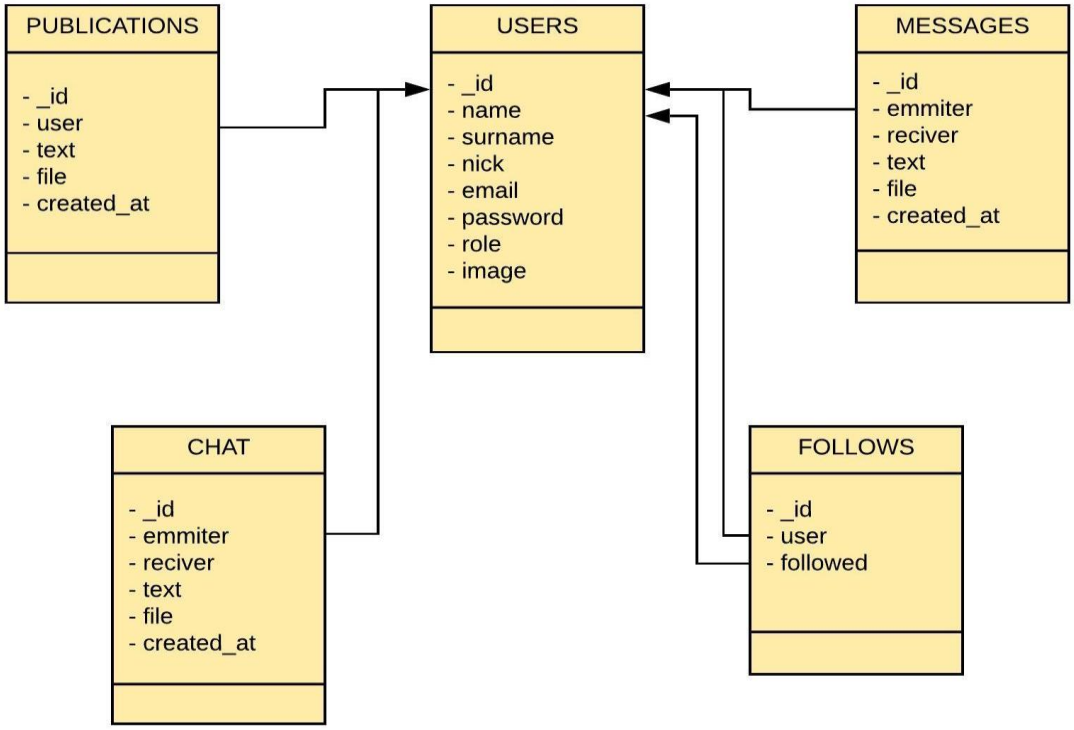


Figura 2: Model de base de dades.

2.4 API Back-end

Per portar a terme el desenvolupament d'aquest projecte, s'ha optat per la creació d'un API Backend mitjançant Node.js i MongoDB. Aquest Api s'adaptarà al patró de disseny Model-Vista-Controlador (MVC), ja que d'aquesta manera l'estructura té una major organització i així es facilitaran les possibles tasques d'evolució i manteniment que es puguin portar a terme en un futur.

Pel que fa a l'estructura de L'API, aquesta s'organitza per mitjà de directoris (models, controladors, serveis, rutes, pujades i middlewares.) i arxius.

El punt d'entrada a L'API, és un arxiu anomenat 'index.js' el qual està situat al directori arrel del projecte. Aquest arxiu porta a terme la connexió amb la base de dades i també requereix un altre arxiu anomenat 'app.js', el qual s'encarrega de requerir 'Express.js' per a la gestió de rutes així com de fer una trucada a l'arxiu de rutes que pertany a cada entitat de la base de dades.

app.routing.ts | user.service.ts | user.guard.ts | auth.js

```
'use strict' // per a poder emprar els derrers standards de js
var jwt = require('jwt-simple');  // carreguem la llibreria jwt
var moment = require('moment'); // carreguem la llibrería per a la gestió de dates
var secret = 'Tjoan1989.';

exports.ensureAuth =   function(req,res,next){  // generem el middelware d'autenticació
        if(!req.headers.authorization){
            return res.status(403).send({message: 'La petició no disposa de cap Token autorizat'});
        }

        var token = req.headers.authorization.replace(/['"]+/g,''); // eliminam les cometes del token
            try{
                var payload = jwt.decode(token,secret);
                if(payload.exp <= moment.unix()){   //comprovam la caducitat del token
                        return res.status(401).send({message: 'El token ha caducat'});
                }

            }catch(ex){
                return res.status(404).send({message: 'Token no vàlid'});

            }


        req.user = payload;  // obtenim les dades d'usuari conectat
        next(); // seguim l'execució del programa
}

```

app.routing.ts | user.service.ts | user.guard.ts | auth.js | jwt.js

```
'use strict' // per a poder emprar els derrers standards de js
var jwt = require('jwt-simple');  // carreguem la llibreria jwt
var moment = require('moment'); // carreguem la llibrería per a la gestió de dates
var secret = 'Tjoan1989.';

exports.createToken = function(user){
    var payload = {
        sub: user._id,
        name: user.name,
        surname: user.surname,
        nick: user.nick,
        email: user.rol,
        image: user.image,
        iat: moment().unix(), // guarda el current timestamp de la data actual
        exp: moment().add(30,'days').unix // expiració del token
    }
    return jwt.encode(payload, secret);
}
```

Les rutes pertanyents a cada model (Entitat), es troben al directori de rutes (Ex: api/routes/user.js) i aquestes s'encarreguen de donar l'URL i un mètode a les diferents funcions pertanyents a cada controlador (Ex: api/controllers/user.js).

Els controladors s'encarreguen de gestionar la lògica de negoci del seu model corresponent (Ex: api/model/user.js), així com també s'encarreguen de gestionar la comunicació entre els models i les vistes d'usuari.

Cal destacar que gairebé totes les rutes, és troben protegides per un middleware d'autenticació (api/middlewares/auth.js) per mitjà del seu mètode 'ensureAuth'. Aquest middleware requereix les següents llibreries: 'Jwt i momment'.

Per tal de donar una major seguretat al token de sessió, guardem les dades d'usuari dintre d'una variable anomenada 'payload', la qual emmagatzema un objecte Json amb les dades de l'usuari. Paral·lelament creem un altra variable anomenada 'secret' i posteriorment s'aplica un ***' jwt.encode(payload, secret)'***, de tal manera que es fa pràcticament impossible que algú pugui suplantar el token d'un altre usuari.

A continuació es descriuen de forma detallada i organitzada per entitats, tots els mètodes i funcions corresponents a l'API .

- ***jwt.encode(payload, secret)*** *És un sistema de seguretat, utilitzat per facilitar el procés d'autenticació d'usuaris a l'aplicació. Atès que un atacant pot obtenir de vegades la clau pública, l'atacant pot modificar l'algorisme a la capçalera a HS256 i després utilitzar la clau pública de RSA per signar les dades. Per això la clau pública(payload) es codificada antes de ser enviada a la capçalera amb la clau privada (secret)*

2.4.1 User

saveUser - Aquesta funció ens serveix per a donar d'alta un nou usuari. Primer de tot, s'obtenen els paràmetres arribats per POST i les guardem a un objecte anomenat 'user'. Aquest mètode només permetrà l'alta d'usuari si els paràmetres arribats per POST són els següents (user.name, user.surname, user.nick, user.role i user.password). Aquest mètode aplica un control per a no permetre usuaris duplicats.

loginUser - Aquesta funció ens serveix per a dur a terme el login d'un usuari. Primer de tot cerquem si aquest usuari i contrasenya concorden amb qualque usuari de la base de dades. Si les dades són correctes, es genera i retorna un token per a permetre l'accés a l'aplicació per part de l'usuari. Si per altra banda les dades no són correctes, es retorna un error 404.

getUser - Funció que retorna un objecte JSON amb les dades de l'usuari que s'ha passat per paràmetres a la URL. Aquesta funció fa una trucada a la funció asíncrona *followThisUser* per tal de retornar si l'usuari rebut per paràmetres segueix a un usuari en concret.

followThisUser - Aquesta funció asíncrona comprova si l'usuari identificat segueix a un usuari en concret, per si no és així poder-lo seguir. Aquesta funció rep els paràmetres *identity_user_id* i *user_id*.

updateUser - Funció per a actualitzar les dades d'un usuari. Aquesta funció comprova si l'usuari a modificar es l'usuari connectat. Posteriorment, s'actualitzen les dades desitjades per mitjà de la funció de callback '*findByIdAndUpdate'*.

getUsers - Aquesta funció retorna un llistat d'usuaris paginats per mitjà de mongoose-pagination. S'ha aplicat un filtre d'ordenació (sort) a la propietat '_id'. Per altra banda, també es retorna el total d'usuaris existents a la base de dades així com el nombre de pàgines d'usuaris que hi ha.

followingUsersId - Aquesta és una funció asíncrona per a obtenir un array d' usuaris seguits per un usuari així com dels usuaris que segueixen a un usuari(usuari connectat)

getCounters / getCountFollows - Aquestes són dues funcions que fan la funció de gestionar un comptador d'usuaris als quals segueix un usuari i d'usuaris que segueixen al mateix usuari.

uploadImage - notUploadUserImage - Aquest conjunt de funcions i mètodes, serveixen per a la gestió controlada de pujades d'imatges de perfil d'usuari.

getImageFile - Aquesta funció retorna la imatge del perfil d'usuari.

2.4.2 Follow

saveFollow - Aquesta funció serveix per a la gestió de seguiments (follow). Primer de tot recull els paràmetres arribats per POST i posteriorment es fa una instància per a portar a terme un nou seguiment. Per a obtenir les dades de l'Id de l'usuari que segueix (follower), s'empra el mètode *req.params.sub*.

deleteFollow - Aquesta funció serveix per a eliminar un seguiment (follow). Primer de tot, la funció obté l'Id de l'usuari connectat i mitjançant paràmetres s'indica l'usuari al qual es deixarà de seguir per mitjà del mètode *remove.*

getFollowingUsers - Aquesta funció retorna un llistat dels usuaris als quals segueix l'usuari connectat de manera paginada, a no ser que s'indiqui un usuari específic mitjançant l'URL**.** Un cop identificat l'usuari s'estableix el nombre d'usuaris que contindran cada pàgina, així com el total d'usuaris seguits i el nombre de pàgines que contindrà el llistat.

getFollowedUsers - Aquesta funció retorna un llistat dels usuaris els quals segueixen a l'usuari connectat de manera paginada, a no ser que s'indiqui un usuari específic mitjançant l'URL**.** Un cop identificat l'usuari s'estableix el nombre d'usuaris que contindran cada pàgina, així com el total d'usuaris seguits i el nombre de pàgines que contindrà el llistat.

2.4.3 Publication

savePublication - Aquesta funció serveix per a donar d'alta una nova publicació. Primer de tot es recullen les dades de l'usuari connectat i posteriorment es comprova si ha arribat una cadena de text com a paràmetre. Si tot es correcte ,es crea un nou objecte de publicació seguint el model '*PublicationSchema'*.

getPublications - Aquest mètode permet llistar les publicacions dels usuaris als quals l'usuari identificat segueix(follow) de manera paginada. Primer de tot es crea un array d'usuaris als quals es segueix (follows_clean []). Posteriorment es cerquem totes les publicacions dels usuaris pertanyents a l'array follows_clean i es retornen les publicacions existents juntament amb l'objecte de l'usuari per mitjà de *populate.*

getPublication - Aquesta funció retorna la publicació passada com a paràmetres a la URL per mitjà de la funció *'Publication.findById()'*.

deletePublication - Aquesta funció serveix per a eliminar una publicació. Primer de tot, la funció obté l'Id de l'usuari connectat i mitjançant paràmetres s'indica l'Id de la publicació que es vol eliminar per mitjà del mètode *remove.*

uploadImage - notUploadUserImage - Aquest conjunt de funcions i mètodes, serveixen per portar a terme la gestió controlada de pujades d'imatges de adjuntes a una publicació.

getImageFile - Aquesta funció retorna la imatge del perfil d'usuari.

2.4.4 Message

saveMessage - Aquesta funció serveix per a donar d'alta un nou missatge. Primer de tot es recullen les dades de l'usuari connectat i posteriorment es comprova si han arribat com a paràmetre tant una cadena de text com l'usuari receptor (receiver). Si tot és correcte, es crea un nou objecte de missatge seguint el model '*MessageSchema'*

getRecivedMessages - Aquesta funció retorna un llistat dels missatges rebuts de forma paginada**.** Un cop identificat l'usuari, es porta a terme una cerca de tots els missatges dels quals l'usuari identificat n'és receptor per mitjà de *Message.find ({receiver: userId}*).

getEmmitMessages - Aquesta funció retorna un llistat dels missatges enviats de forma paginada**.** Un cop identificat l'usuari, es porta a terme una cerca de tots els missatges dels quals l'usuari identificat n'és emissor per mitjà de *Message.find ({emmitter: userId}*).

getUnreadMessages - Aquesta funció retorna el nombre de missatges rebuts els quals no han estat llegits**.** Un cop identificat l'usuari, es porta a terme una cerca de tots els missatges no llegits dels quals l'usuari identificat n'és receptor per mitjà de *Message.countDocuments({receiver: userId, viewed: 'false'})*.

setReadMessage - Aquesta funció serveix per a actualitzar el flag (viewed) d'un missatge i canviar-li el valor de false a true.

2.4.5 Xat

Per portar a terme el xat en temps real, s'ha utilitzat WebSockets, ja que ens proporciona un canal de comunicació full-duplex a través d'una única connexió TCP, de manera que els usuaris no han de fer peticions HTTP addicionals per enviar i rebre missatges. Una connexió WebSocket es manté oberta i d'aquesta manera redueix considerablement la latència. Aquest protocol ens ha permès que els Xats s'actualitzin sense necessitar cap mena d'interacció per part de l'usuari.

2.5 Client Angular

Pel que fa a la part del client, s'ha optat per l'ús del framework Angular. Aquest s'adaptarà al patró de disseny Model-Vista-Controlador (MVC), ja que d'aquesta manera l'estructura té una major organització i així es facilitaran les possibles tasques d'evolució i manteniment que es puguin portar a terme en un futur.

Pel que fa a l'estructura del client, aquesta s'organitza per mitjà de directoris (app (components, messages, models, services), assets, environments) i arxius.

El punt d'entrada al client, és un arxiu anomenat 'app.component.html' el qual està situat al directori arrel del projecte. Aquest arxiu té la seva lògica de negoci a l'arxiu 'app.component.ts'. Aquest últim és carregat juntament amb 'app.routing.ts' dintre de l'arxiu 'app.module.ts'

A continuació es detallen algunes de les parts més importants de l'estructura del client Angular.

- package.json: Aquest fitxer conté la llista de dependències que es necessiten.

- src /app/styles.css: Aquest fitxer conté el CSS global disponible a tota l'aplicació.

- src / main.ts: Aquest és el fitxer principal que inicia l'aplicació angular ja que AppModule és arrencat aquí.

- src / index.html: aquest és el primer fitxer que s'executa al costat de main.ts quan es carrega la pàgina.

- src / app / app.module.ts: és el fitxer on es defineixen tots els components, proveïdors i mòduls. Sense definir-los aquí, no es poden utilitzar en cap altre lloc del codi.

- src / app / app.component.html: Aquest és el component principal de l'aplicació angular i tots els altres components estan presents en aquest component. src / app / app.component.ts és la lògica d'aquest component, i src / app / app.component.css és el CSS d'aquest component. Aquest mateix component no fa cap lògica important, però actua com a contenidor per a altres components.

- dist: aquest directori és on estan presents els fitxers construïts. TypeScript es converteix bàsicament a JavaScript i els fitxers resultants s'emmagatzemen aquí després de 'minificar-se'. Com que els exploradors web només entenen JavaScript, és necessari convertir TypeScript a JavaScript abans de desplegar el codi .

3. Plataforma de desenvolupament

3.1 Software

Per al desenvolupament de social CMR, s'han emprat una sèrie de tecnologies i eines de programari, sent totes elles de codi lliure.

- **Node.Js**, És l'entorn de desenvolupament emprat per al costat del servidor.
- **MongoDB** ha estat emprat com a sistema gestor de bases de dades.
- **Angular** Ha estat emprat com a Framework Javascript de desenvolupament del frontend.

3.2 Altres

Per a la visualització del projecte en temps de desenvolupament, es farà servir l'explorador web 'Firefox Developer Edition', ja que aquest ens preveu d'una sèrie d'eines per tal d'obtenir una major agilitat a l'hora de desenvolupar.

Entre els avantatges que aquest explorador ens ofereix, podem destacar algunes de les següents.

- **Consola Web:** Ens permet veure missatges de registre (logs) en una pàgina web i també interactuar amb la pàgina emprant JavaScript.
- **Inspector de pàgina**: Ens permet revisar i modificar el contingut HTML i CSS de la pàgina.
- **Depurador JavaScript**: Ens permet examinar i modificar el contingut JavaScript que està executant-se en una pàgina.
- **Monitor de xarxa**: Ens permet veure les sol·licituds de xarxa fetes quan una pàgina està carregada.

- **Inspector d'emmagatzematge**: Ens permet Inspeccionar cookies, emmagatzematge local i de sessió present a una pàgina.

Aquestes són només algunes de les funcionalitats més importants que ens dóna aquest explorador

Per al desenvolupament, es fa servir l'editor de text *Visual Studio Code* juntament amb el plug-in *SimpleFTP* per a la sincronització remota amb el directori on s'allotja el projecte.

4. Planificació del Treball

Per a portar a terme el desenvolupament, estructurarem les diferents etapes com a fites a seguir.

	Nombre de dies	Inici	Fi
PAC 1	**12**	**20/02/2019**	**04/02/2019**
Definició del Projecte	4	20/02/2019	24/02/2019
Objectius del projecte	2	25/02/2019	26/02/2019
Investigació sobre les tecnologies a emprar	3	27/02/2019	01/03/2019
Redacció del document	3	02/03/2019	04/03/2019

Taula 2: Fites de planificació PAC 1

	Nombre de dies previsió	Inici	Fi
PAC 2	**28**	**06/03/2019**	**04/02/2019**
Aprenentatge Node.Js i Angular 7	28	**06/03/2019**	**03/04/2019**
Estructura de continguts	5	06/03/2019	11/03/2019
Disseny de Base de dades	3	12/03/2019	15/03/2019
Desenvolupament Back-end	20	15/03/2019	03/04/2019

Taula 3: Fites de planificació PAC 2

	Nombre de dies previsió	Inici	Fi
PAC 3	**36**	**04/04/2019**	**12/05/2019**
Aprenentatge Node.Js i Angular 7	**36**	**04/04/2019**	**12/05/2019**
Implementació API Back-end	11	04/04/2019	15/04/2019
Formularis (registre + Login)	8	16/04/2019	24/04/2019
Timeline	4	25/04/2019	29/04/2019
Seguiment d'usuaris	2	30/04/2019	31/04/2019
Missatgeria privada	7	01/05/2019	07/05/2019
Redacció del document	4	08/05/2019	12/05/2019

Taula 4: Fites de planificació PAC 3

	Nombre de dies	Inici	Fi
PAC 4	**26**	**13/05/2019**	**10/06/2019**
Implementació del Xat	14	13/05/2019	27/05/2019
Correcció d'errors	2	28/05/2019	29/05/2019
Fase de test	5	30/05/2019	04/06/2019
Redacció del document	5	05/06/2019	10/06/2019

Taula 5: Fites de planificació PAC 4

5. APIs i llibreries externes.

Per al desenvolupament de Social CMR s'han emprat una sèrie d'APIs i llibreries externes, les quals es detallen a continuació.

5.1 Back-end

Express.js - És un framework que serveix per a treballar amb el protocol HTTP (rutes, controladors, middlewares,etc..). Per a instal·lar-ho, entrem al directori "SocialCMR/Api" i executem la següent comanda "npm install express --save".

bcrypt - És una llibreria que ens proveeix d'un sistema de xifrat per a contrasenyes segures. Per a instal·lar-ho, entrem al directori "SocialCMR/Api" i posteriorment executem la següent comanda: "npm install bcrypt-nodejs --save".

body-parse - És una llibreria que ens permet convertir els JSON consumits per l'Api a un objecte Javascript. Per a instal·lar-ho, entrem al directori "Social CMR /Api" i executem la següent comanda: "npm install body-parser --save".

connect_multiparty - És una llibreria que ens proveeix d'un sistema per a gestionar la pujada de fitxers al servidor. Per a instal·lar-ho, entrem al directori "SocialCMR/Api " i executem la següent comanda: "npm install connect-multiparty --save".

mongoose - És una llibreria que ens proveeix d'un sistema ORM per a gestionar els mètodes de MongoDB. Per a instal·lar la llibreria, ens situem al directori "SocialCMR/Api" i executem la següent comanda: "npm install mongoose --save".

JWT - És una llibreria que ens proveeix d'un sistema per a la gestió de Tokens de d'autenticació. Per a instal·lar-ho, primer entrem al directori "SocialCMR/Api" i posteriorment executem la següent comanda: "npm install --save jwt-simple".

moment - És una llibreria que ens proveeix d'un sistema per a la gestió de dates. Per a instal·lar-ho, entrem al directori "SocialCMR/Api" i executem la següent comanda "npm install moment --save".

mongoose pagination - És una llibreria que ens proveeix d'un sistema per a la paginació de resultats de mongoDB. Per a instal·lar-ho, entrem al directori "SocialCMR/Api" i executem la següent comanda "npm install save mongoose-pagination"

5.2 Front-end

Angular CLI és l'eina de línia de comandes estàndard per a crear, depurar i publicar aplicacions Angular.

jQuery és una llibretera multiplataforma de Javascript que permet simplificar la manera d'interactuar amb els documents HTML, manipular l'arbre DOM, manejar esdeveniments, desenvolupar animacions i agregar interacció amb AJAX

AJAX és una tècnica de desenvolupament web per a crear aplicacions interactives que s'executen en el navegador dels usuaris mentre es manté la comunicació asíncrona amb el servidor en segon pla.

Bootstrap és un framework té com a objectiu facilitar el desenvolupament web ja que permet crear de forma senzilla webs de disseny responsiu.

HTML5 Es tracta d'un llenguatge de marques per a crear el layout d'una pàgina web, així com donar format al seu aspecte.

6. Usabilitat / UX

6.1 Perfils d'usuari

Social CMR disposa de cinc tipus d'usuaris (Administrador, Entitat, Empresa, Veïnat i convidat). Aquests usuaris, tindran uns rols i permisos distints quan a l'ús de la xarxa social es refereix.

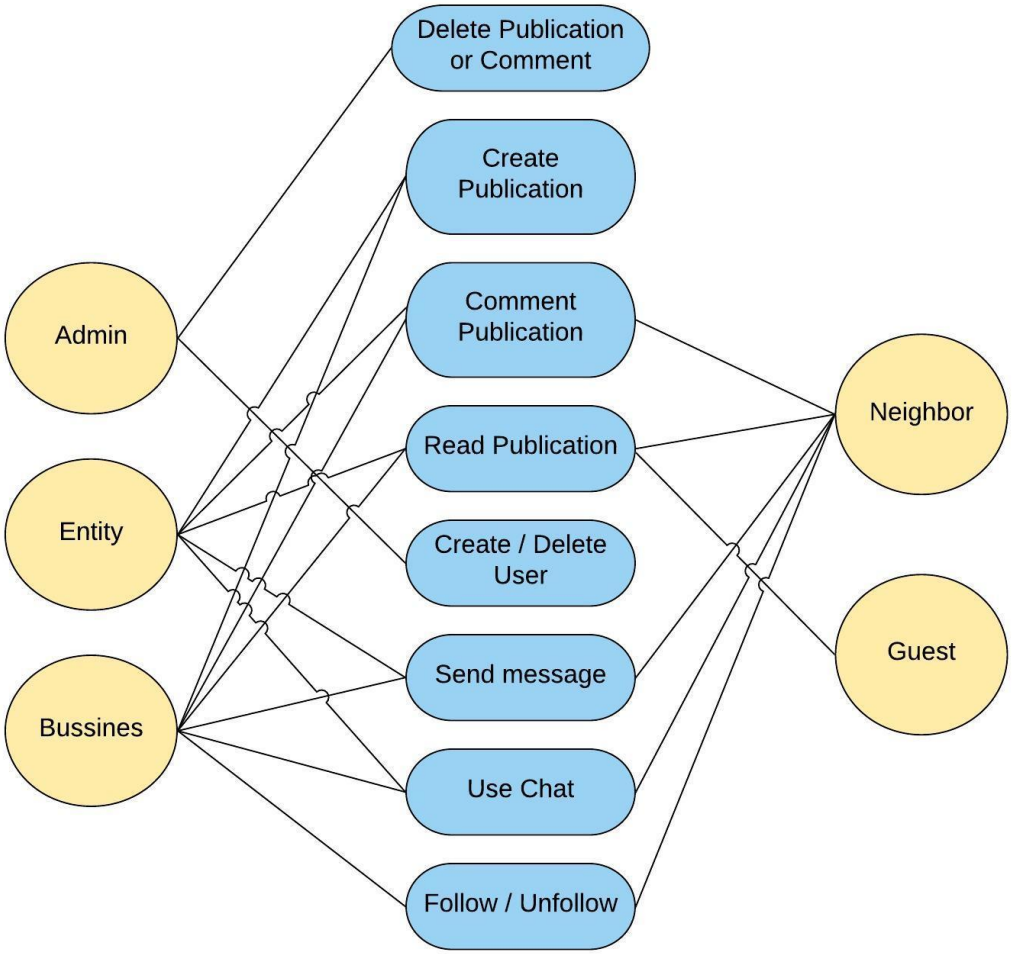


Figura 3: Diagrama de casos d'ús.

L'usuari Administrador actuarà com a moderador i podrà eliminar publicacions i comentaris de qualsevol usuari, així com suspendre l'activitat d'un usuari/Entitat/Empresa per ús inadequat del sistema.

Les entitats són les diferents organitzacions públiques del poble (Ex; Associació de Joves, Associació de gent gran, Batllia, Ajuntament, etc.), les quals poden fer publicacions sobre les coses que van passant pel poble o projectes a realitzar. Aquests projectes podran ser votats i comentats pels usuaris de tipus Veïnat i els resultats de la votació seran públics.

Les empreses seran les organitzacions que pertanyen a l'associació d'empresaris del poble, les quals podran donar-se a conèixer per mitjà de publicacions al seu perfil d'usuari.

Els veïnats seran les persones físiques amb padró vigent a l'ajuntament del poble en qüestió. Aquest tipus d'usuari no podrà fer publicacions, sinó que tan sols podrà fer comentaris i votacions a les publicacions d'Entitats i empreses.

La xarxa social disposarà d'un sistema de missatgeria privada per tal que les diferents parts que componen la xarxa social es puguin comunicar de forma àgil entre si. Per a poder enviar/rebre un missatge privat, cal que emissor/receptor tinguin una relació d'amistat dintre de la xarxa social (per a establir aquesta relació, cal que els dos usuaris se segueixin mútuament "Estil Instagram").

Per defecte, quan un usuari és una alta nova segueix de forma predeterminada a totes les Entitats públiques de la localitat, no obstant aquest pot reclinar el seguiment.

6.2 Especificació dels casos d'ús

Generar una nova publicació

Actors: Entitat pública, Empresa local.

Descripció: L'usuari accedeix al sistema i mitjançant un formulari escriu una nova publicació, on hi pot adjuntar contingut multimèdia i enllaços.

Condició prèvia: L'usuari ha de disposar d'un dispositiu connectat a la xarxa, així com d'un usuari registrat.

Curs de l'esdeveniment

Actor	Sistema
1. L'usuari accedeix a Social CMR	
	2. Es genera una connexió entre el navegador de l'usuari i el servidor.
	3. El sistema analitza si el navegador de l'usuari disposa d'un Token d'autenticació correcte. En el cas de que les dades siguin correctes, s'obre la plana d'inici d'aquest usuari.
4.Lusuari clica sobre la icona de 'Nova publicació'	
	5. El sistema mostra el formulari per a la creació de noves publicacions.
6. L'usuari introdueix les dades pertinents de la publicació, així com adjunta els arxius o links pertinents.	
	6. El sistema guarda la publicació i aquesta es mostrada al perfil de l'usuari i també al timeline dels usuaris que segueixen a l'usuari el qual ha creat la publicació.

Cas alternatiu

2a. No es pot establir la connexió entre l'usuari i el servidor, ja sigui perquè el servidor no es troba disponible o bé perquè l'usuari no disposa de connexió a la xarxa.
2b. Es notifica a l'usuari de que hi ha hagut un error de connexió amb el servidor.
3a. El Token d'autenticació és incorrecte. S'informa l'usuari que hi ha hagut un error.
6a. El sistema no pot emmagatzemar la publicació. S'informa l'usuari que hi ha hagut un error

Taula 6: Esdeveniment: Generar una publicació

Alta/Baixa d'Usuaris i publicacions

Actors: Administrador.

Descripció: L'usuari accedeix al sistema i mitjançant un formulari dóna d'alta/baixa un usuari.

Condició prèvia: L'usuari ha de disposar d'un dispositiu connectat a la xarxa, així com d'un usuari registrat amb permisos d'administrador.

Curs de l'esdeveniment

Actor	Sistema
1. L'usuari accedeix a Social CMR	
	2. Es genera una connexió entre el navegador de l'usuari i el servidor.
	3. El sistema analitza si el navegador de l'usuari disposa d'un Token d'autenticació correcte. En el cas de que les dades siguin correctes, s'obre la plana d'inici d'aquest usuari.
4.Lusuari clica sobre la icona de 'Nova publicació'	
	5. El sistema llista els usuaris així com ensenya una icona del tipus 'Baixa d'usuari' al costat de cada un dels usuaris llistats. El sistema també proporciona un enllaç al formulari d'alta d'usuaris.
6. L'usuari gestiona l'alta i baixa d'usuaris així com també l'eliminació de comentaris i publicacions que no s'adapten a les normes bàsiques d'ús de l'aplicació.	

Cas alternatiu

2a. No es pot establir la connexió entre l'usuari i el servidor, ja sigui perquè el servidor no es troba disponible o bé perquè l'usuari no disposa de connexió a la xarxa.
2b. Es notifica a l'usuari de que hi ha hagut un error de connexió amb el servidor.
3a. El Token d'autenticació és incorrecte. S'informa a l'usuari de que hi ha hagut un error.
6a. El sistema no pot gestionar alguna alta o baixa sol·licitada. S'informa l'usuari que hi ha hagut un error

Taula 7: Esdeveniment: Alta/Baixa d'usuaris i publicacions

Follow / Unfollow

Actors: Entitat pública, Empresa local, veïnat de la localitat.

Descripció: L'usuari accedeix al sistema i mitjançant la llista d'usuaris pot seguir/deixar de seguir a un altre usuari del sistema.

Condició prèvia: L'usuari ha de disposar d'un dispositiu connectat a la xarxa, així com d'un usuari registrat.

Curs de l'esdeveniment

Actor	Sistema
1. L'usuari accedeix a Social CMR	
	2. Es genera una connexió entre el navegador de l'usuari i el servidor.
	3. El sistema analitza si el navegador de l'usuari disposa d'un Token d'autenticació correcte. En el cas de que les dades siguin correctes, s'obre la plana d'inici d'aquest usuari.
4.Lusuari introdueix el nom de l'usuari a seguir / deixar de seguir.	
	5. El sistema ensenya la fitxa d'usuari a seguir/ deixar de seguir. Si l'usuari el qual s'ha llistat, ja es seguit per l'usuari connectat, apareix una icona amb el text 'deixar de seguir'. Si per el contrari no es així, el text que apareix a l'Icona es 'seguir'

Cas alternatiu

2a. No es pot establir la connexió entre l'usuari i el servidor, ja sigui perquè el servidor no es troba disponible o bé perquè l'usuari no disposa de connexió a la xarxa.
2b. Es notifica a l'usuari de que hi ha hagut un error de connexió amb el servidor.
3a. El Token d'autenticació és incorrecte. S'informa l'usuari de que hi ha hagut un error.
6a. El sistema no pot gestionar el Follow/ unfollow sol·licitat. S'informa a l'usuari de que hi ha hagut un error

Taula 8: Esdeveniment: Follow/Unfollow

Xat i Missatgeria

Actors: Entitat pública, Empresa local, veïnat de la localitat.

Descripció: L'usuari accedeix al sistema i mitjançant l'eina de missatgeria o l'eina de Xat en directe i es comunica amb altres usuaris.

Condició prèvia: L'usuari ha de disposar d'un dispositiu connectat a la xarxa, així com d'un usuari registrat.

Curs de l'esdeveniment

Actor	Sistema
1. L'usuari accedeix a Social CMR	
	2. Es genera una connexió entre el navegador de l'usuari i el servidor.
	3. El sistema analitza si el navegador de l'usuari disposa d'un Token d'autenticació correcte. En el cas que les dades siguin correctes, s'obre la plana d'inici d'aquest usuari.
4.Lusuari accedeix al mòdul de missatgeria / Xat.	
	5. El sistema llista els usuaris disponibles per a poder establir una conversació en temps real. Si hi ha usuaris amb els quals hi ha establerta una amistat que estan connectats, el sistema mostra un llistat de tots aquests usuaris i dóna l'opció d'iniciar una nova conversació.
	6. Si hi ha usuaris no connectats amb els quals es manté una amistat, el sistema els llista a l'apartat d'usuaris no connectats.
7. L'usuari estableix una nova conversació amb l'usuari desitjat.	

Cas alternatiu

2a. No es pot establir la connexió entre l'usuari i el servidor, ja sigui perquè el servidor no es troba disponible o bé perquè l'usuari no disposa de connexió a la xarxa.
2b. Es notifica a l'usuari de que hi ha hagut un error de connexió amb el servidor.
3a. El Token d'autenticació és incorrecte. S'informa l'usuari de que hi ha hagut un error.
7a. Si hi ha algun error en la tramesa de missatges, el sistema ho notifica a l'usuari.

Taula 9: Esdeveniment: Xat i Missatges

6. 3 Prototips

Social CMR és una aplicació web SPA (Single Page Application) on s'hi van carregant els continguts als diferents marcs, sense necessitat d'haver de recarregar tota la plana cada cop que es vol consultar contingut nou.

A continuació es detalla el prototip Lo-Fi (Baixa fidelitat) per tal de donar a conèixer l'estructura de les diferents planes d'una manera esquematitzada.

6.3.1 Visió general Lo-Fi de la pàgina de Login.

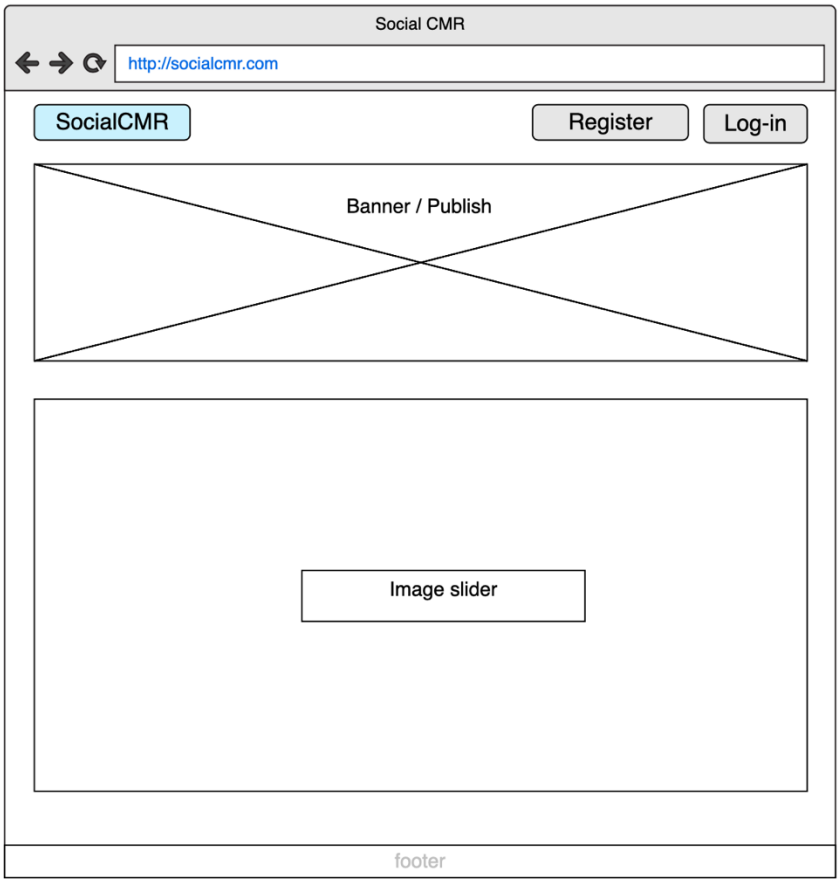


Figura 4: Prototip de les pàgines d'inici i de registre.

Pàgina d'inici:

- Logotip de Social CMR al backgroud de la pàgina.
- Baner publicitari de L'ajuntament.
- Botó d'accés i botó de Sobre nosaltres.

Pàgina de registre:

- Formulari de registre per donar d'alta un nou usuari.

6.3.2 Visió general de la plana home.

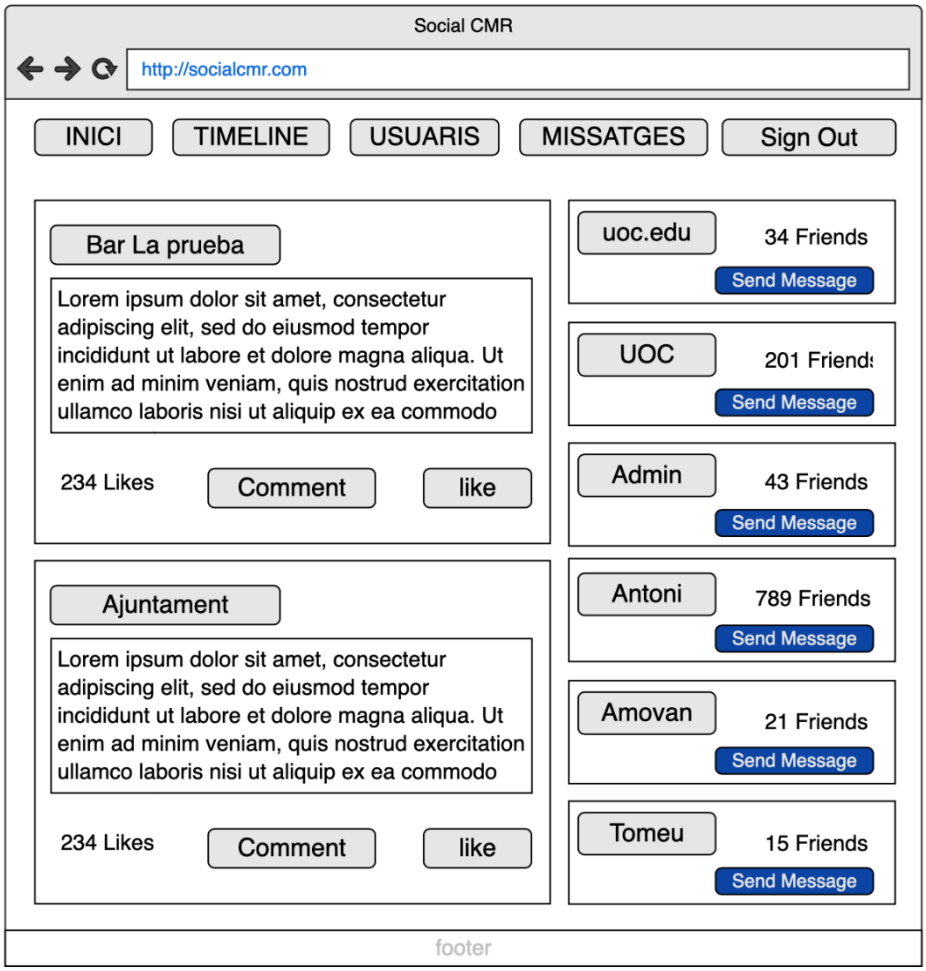


Figura 5: Prototip de la pàgina d'inici un cop l'usuari ha estat logat.

En aquesta pàgina es visualitza el contingut corresponent a la pàgina d'inici d'un usuari. Primer de tot hi apareix un camp de cerca d'usuaris per a poder visualitzar el perfil d'un usuari. També hi podem trobar un botó de tancament de sessió.

A la part esquerra es mostra un llistat de les històries d'usuari dels usuaris seguits. Aquest llistat es troba filtrat de tal manera que les primeres històries que hi apareixen són les més recents.

A la part dreta es mostra un bloc que conté un llistat dels usuaris amb els quals hi ha una conversació establerta. A la part superior d'aquest bloc es llisten primer de tot els usuaris (amb els quals hi ha una relació d'amistat) connectats.

6.3.3 Visió general d'un llistat d'usuaris.

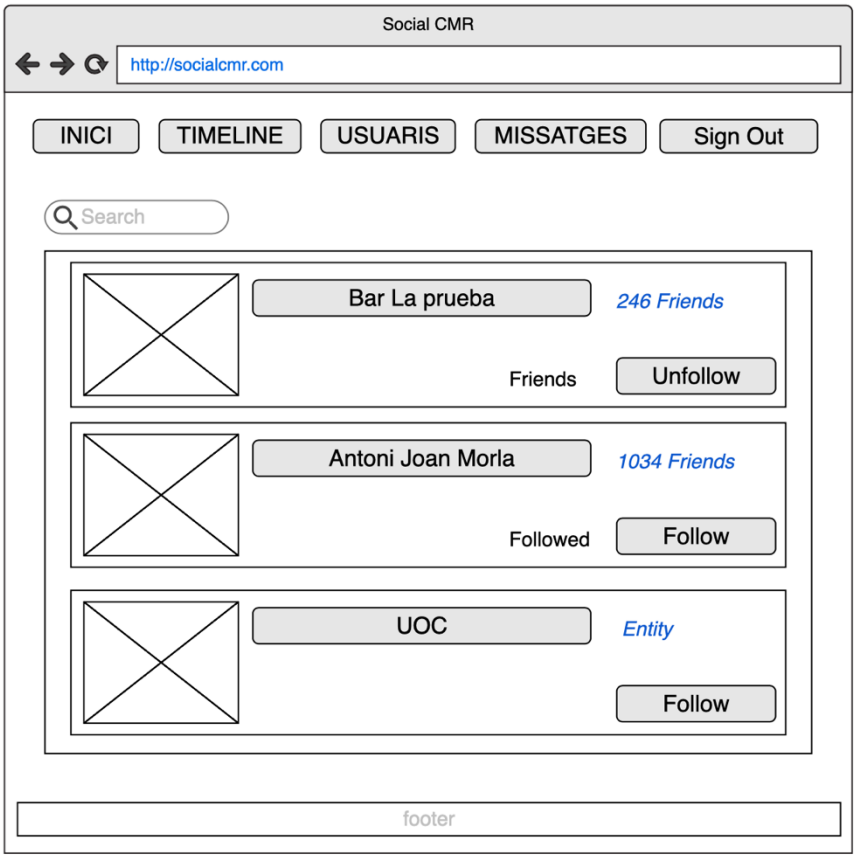


Figura 6: Prototip d'un llistat d'usuaris..

En aquesta pàgina es visualitza el contingut corresponent a la pàgina d'usuaris. Primer de tot hi apareix un camp de cerca d'usuaris per a poder visualitzar les

coincidències de forma dinàmica. També hi podem trobar un botó de tancament de sessió.

El cos principal es compon d'un sol bloc, on hi apareix un llistat dels resultats coincidents amb la cerca. Els resultats es mostren en forma de fitxa d'usuari. Cada una d'aquestes fitxes ensenya una foto de l'usuari, així com el seu nom, els amics que té i un botó per a seguir o deixar de seguir (Follow/Unfollow) aquest usuari.

6.3.4 Visió general del timeline.

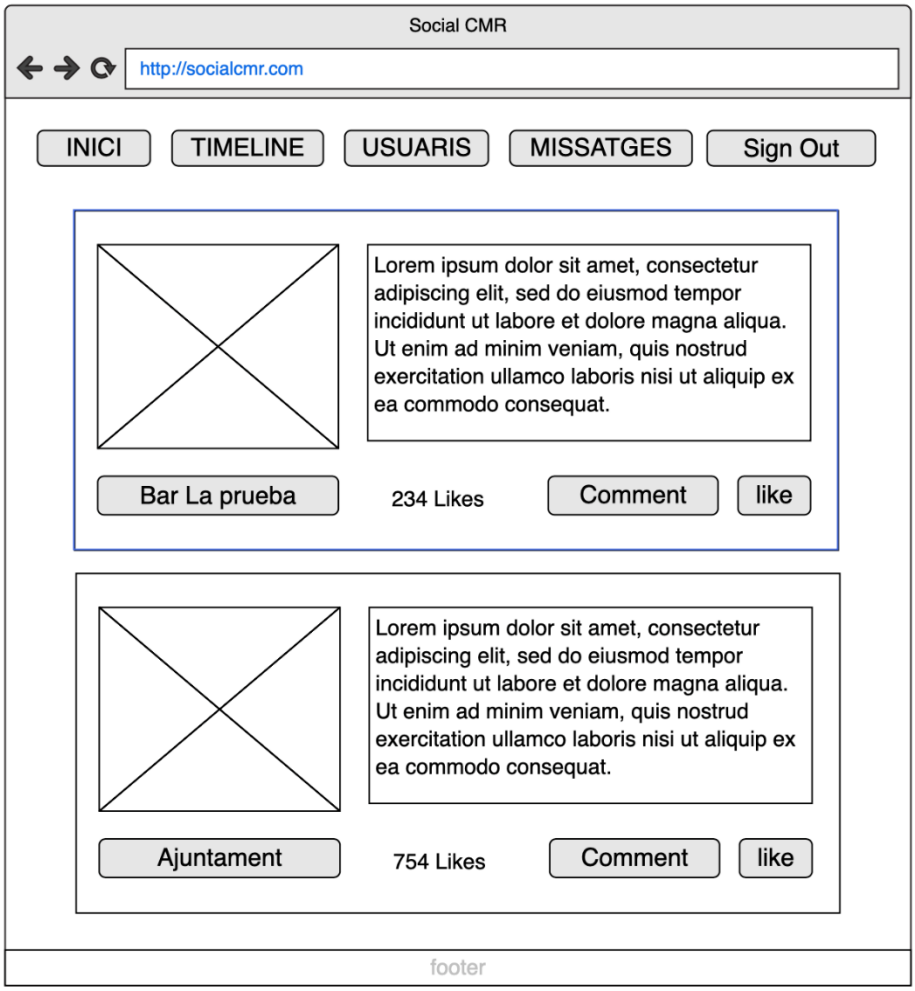


Figura 7: Prototip del timeline..

En aquesta pàgina es visualitza el contingut corresponent a les històries d'usuari. filtrades de mes modernes a més antic.

El cos principal es compon d'un sol bloc, on hi apareix un llistat dels resultats. Els resultats es mostren en forma de fitxa d'usuari. Cada una d'aquestes fitxes ensenya una foto de l'usuari, així com el seu nom, els 'likes' de la publicació, un botó per a donar un 'like' i un altre per a escriure un comentari a la publicació.

7. Seguretat

Tal com s'especifica en el Punt 2.4, l'Api Backend està protegida per mitjà d'un middleware d'autenticació per tal que només els usuaris registrats puguin accedir a les diferents rutes d'aquesta.

Pel que fa al Front i per tal que el WebApp estigui protegit i les diferents parts que el componen només puguin ser visitades per determinats tipus d'usuaris, hem fet ús dels anomenats guards. El principal avantatge que ens dona l'ús dels guards, es que aquests tenen una gran escalabilitat i poden ser emprat baix les condicions que es vulguin.

En el nostre cas, només comprovem si l'usuari es troba identificat i si aquest pot accedir a un determinat complement per mitjà de CanActivate. El guard ens retorna 'true' o 'false' indicant així si un determinat usuari pot accedir a un determinat contingut.

El primer que s'ha fet, es posar l'etiqueta 'Injectable()'. Posteriorment es crida al mètode CanActivate y fem la comprovació del login. Si l'usuari no està identificat, el redirigim a la pàgina de login per mitjà de router.navigate.

```
import {Injectable} from '@angular/core';
import {Router, CanActivate} from '@angular/router';
import {UserService} from './user.service';
@Injectable()
export class UserGuard implements CanActivate{
  constructor(
      private _router:Router,
      private _userService:UserService
    ){}
  canActivate(){
    let identity = this._userService.getIdentity();
    if(identity  && (identity.role == 'role_user' || identity.role == 'Role_user')){
      return true
    }else{
      this._router.navigate(['login']);
      console.log('identity.role');
      return false;
    }
  }
}
```

8. Instruccions d’instal·lació/implantació

A aquest apartat es recull la documentació necessària per a realitzar la instal·lació del sistema, així com dels requisits previs per a poder portar-ho a terme.

8.1 Requisits previs

A aquest apartat s'inclou la documentació de tots els requeriments previs a l'inici del procés d'instal·lació.

Com a requisits previs a la instal·lació destaquen:

- Equip servidor amb el sistema operatiu (Windows 10)

- Descàrrega del fitxer comprimit socialCMR.zip.

- Instal·lació de MongoDB (https://www.mongodb.com/download-center/community)

- Instal·lació de RoboMongo

- Instal·lació de Node.js (https://nodejs.org/es/download/)

El projecte ja incorpora les diferents llibreries necessàries, pel que no fa falta portar a terme la descàrrega de cap d'elles.

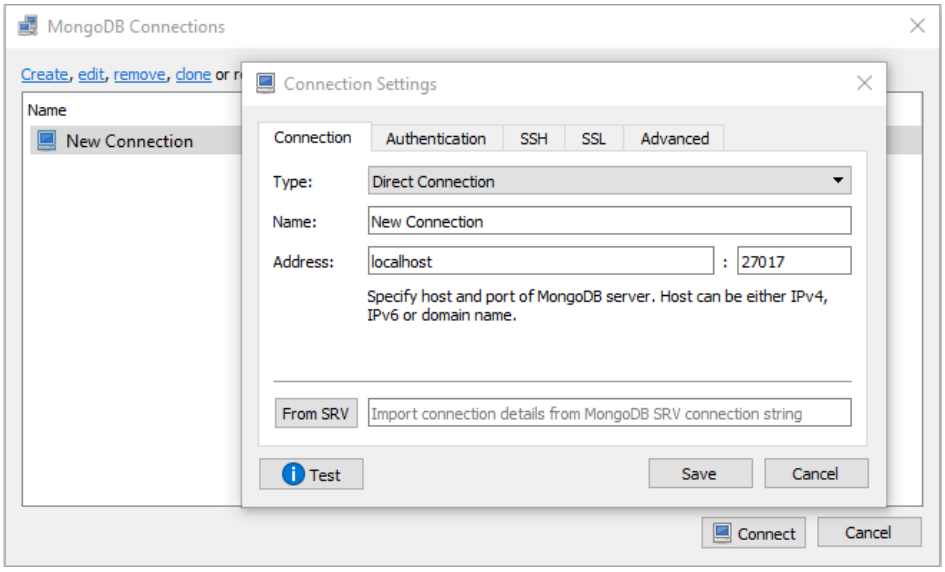
8.2 Verificació de requisits:

Per a verificar que Node JS es troba instal·lat podem executar la següent comanda:
node --v
Per a verificar que MongoDB es troba instal·lat podem executar la següent comanda:
mongod --v

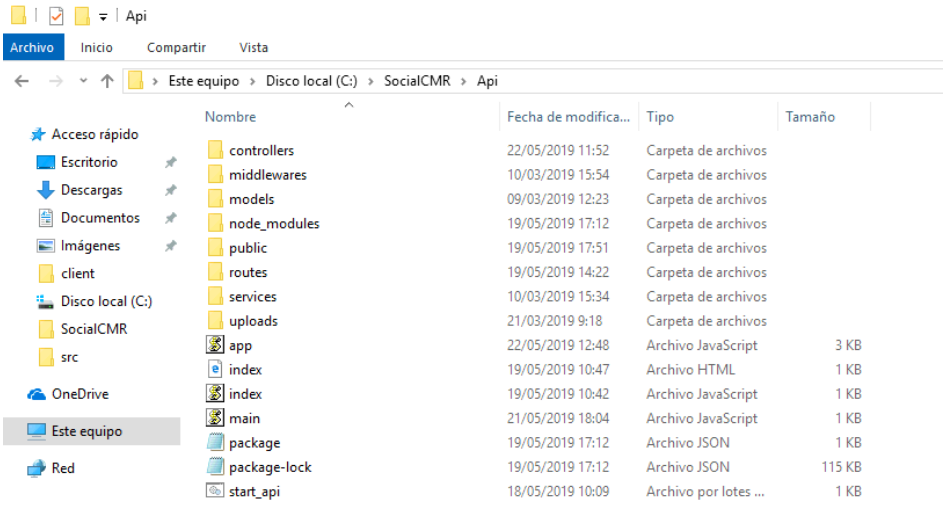
8.3 Procediment d'instal·lació

Per portar a terme la instal·lació de SocialCMR en local és necessari seguir les següents passes.

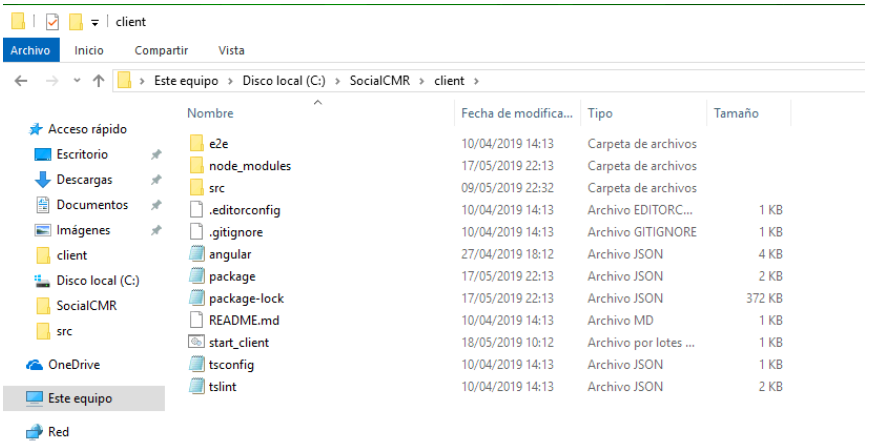
1. Descomprimir el projecte al directori desitjat.
2. Crear una base de dades anomenada social_cmr a Robomongo al port 27017.



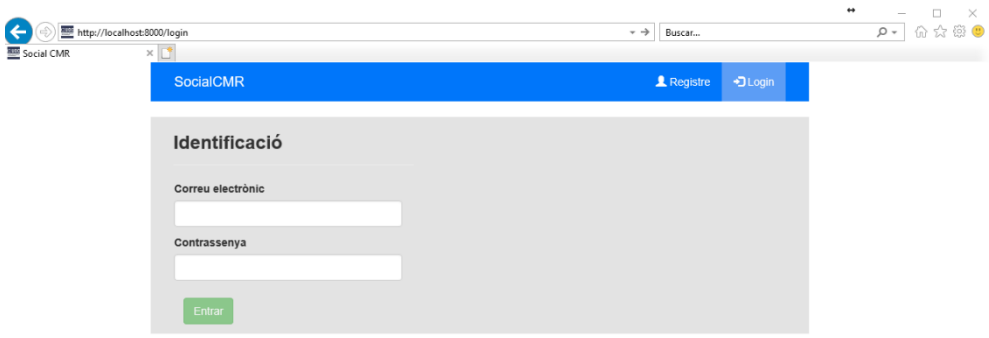
3. Un cop descomprimit el projecte, només queda executar el servidor Node (Per això, entrem al directori SocialCMR/Api i executem 'start_api.bat')



4. Per engegar Angular; entrem al directori SocialCMR/Client i executem ‘start_client.bat’.



5. Per últim, podem accedir al projecte des de l'explorador a la següent ruta: http://localhost:8000



9. Instruccions d'ús

El sistema permet la creació de nous usuaris, així com l'edició dels usuaris ja existents.

A continuació es resumeixen d'una manera esquemàtica les funcionalitats principals de Social CMR.

1) Accés a SocialCMR

    a. Registrar un nou usuari
    b. Accés amb les credencials d'un usuari ja existent

2) Plana d'inici

    a. Timeline
        i. Afegir publicació.
        ii. Eliminar publicació del mateix usuari.
        iii. Veure publicacions dels usuaris seguits.
    b. Usuaris
        i. Seguir un nou usuari.
        ii. Deixar de seguir un usuari.
        iii. Afegir publicació.
        iv. Cercar usuaris.
        v. Accedir al perfil d'un usuari.
    c. Missatges
        i. Veure missatges rebuts.
        ii. Veure missatges enviats.
        iii. Enviar un nou missatge.
    d. Informació d'usuari
        i. Veure el perfil de l'usuari connectat.
        ii. Editar perfil d'usuari.
        iii. Tancar sessió.

Pel que fa a la fase de desenvolupament s'han aplicat els permisos que té un usuari al Timeline depenent del cognom d'un usuari, ja que no hi ha cap funció al servei d'edició d'un usuari que permeti modificar el rol d'aquest. Quan un usuari es registra a l'aplicació té de forma predeterminada un rol d'usuari del tipus 'Role_user', ja que el que es vol és que sigui el propi administrador qui defineixi un usuari com a 'ADMIN' o com a 'ENTITAT'.
Per tal de poder provar l'escriptura de publicacions en fase de desenvolupament, s'ha canviat la comprovació de (identity.role) a (identity.surname) per tal que en la fase de correcció es pugui apreciar la diferència de permisos entre tipus d'usuari, només canviant el cognom de l'usuari.

Els tres tipus d'usuari que hi ha són:

- ADMIN
- VEINAT
- ENTITAT

** basta amb canviar el cognom de l'usuari per veure de forma reactiva els canvis al Timeline,*

10. Projecció a futur

L'aplicació desenvolupada constitueix la primera versió per a una futura posta en producció d'un sistema més complet, el qual permeti la creació d'agendes compartides i la subscripció dels usuaris a esdeveniments creats per altres usuaris amb rol 'Entitat'.

Per altra banda, les versions futures, comptaran amb un disseny molt més treballat, el qual farà molt més agradable l'estància a la plana web a l'usuari.

Altres millores propostes per a la versió actual:

- Aplicar correctament el *responsive* del web-app.
- Millorar la seguretat del formulari, aplicant doble validació al camp 'password'.
- Corregir el formulari de pujada d'imatges de perfil.
- Cal permetre que un usuari pugui bloquejar un seguidor.
- Cal millorar la missatgeria, perquè així els missatges es vegin en mode 'conversació'.

11. Conclusions

El projecte Social CMR ha tingut dues fases ben diferenciades. La primera d'elles ha estat la fase d'elecció del projecte, ja que de temps enrere tenia la idea de fer un sistema de gestió i visualització d'imatges del tipus biblioteca multimèdia (PHP + Js). A causa de la no recomanació per part del professor vaig decidir desenvolupar una 'mini' xarxa social, ja que era la idea que més em va agradar de les proposades a l'aula de l'assignatura.

La segona fase va ser la d'estudi dels llenguatges i tecnologies amb els quals ha estat desenvolupada l'aplicació web. Aquesta sens dubte va ser una de les tasques més difícils i estressants per a mi, ja que quan vaig començar a desenvolupar, havia d'anar endavant i enrere per a corregir errors en el codi, els quals s'anaven produint molt sovint, per culpa de la inexperiència amb els llenguatges emprats.

Tot i això, aquest projecte m'ha sigut molt enriquidor en l'àmbit personal, ja que he après com funciona Node.js, Angular, Express i MongoDB. Cal destacar que gràcies a les diferents fonts consultades he pogut aprendre el necessari per seguir amb aquest projecte, ja que sense elles segurament hagués desistit.

En definitiva, crec que l'elecció del projecte ha fet que em compliqués la vida i em portés molta més feina de la que en realitat m'hauria emportat desenvolupar un projecte amb PHP, ja que és el llenguatge amb el qual tinc més experiència. No obstant això, crec que m'ha ajudat a descobrir un món el qual mai havia tingut l'oportunitat de conèixer.

12. Codi font (extracte del controlador d'usuaris)

12.1 - Controlador d'usuaris

A aquest apartat es detalla el codi font del controlador d'usuari de l'Api Backend (Api/controllers/user.js), ja que si be es cert que totes les parts de l'aplicació son importants per a formar un conjunt, aquesta es la part més rellevant del sistema.

A aquest arxiu si troben les funcions i mètodes que permeten portar a terme accions tan importants com: Alta d'un nou usuari, Identificació/Login d'un usuari, edició d'un usuari, obtenció d'usuaris seguits/seguidors, pujada d'imatges de perfil, restricció d'usuaris duplicats, etc.

```
'use strict' //Ens permet emprar les derreres característiques de Javascript
var bcrypt = require('bcrypt-nodejs'); // requerim bcrypt per al xifrat de contrasseñes
var User = require('../models/user');// carreguem el model d'usuari
var Follow = require('../models/follow'); // carreguem el model de seguiment
var Publication = require('../models/publication'); // carreguem el model de publicacions

var jwt = require('../services/jwt'); // carrega de token
var mongoosePaginate = require('mongoose-pagination'); // carreguem mongoose paginate
var fs = require('fs');// importam file system
var path = require('path');// importem la llibreria per a treballar amb la ruta dels fitxers

// funció per a donar d'alta un usuari
function saveUser(req,res){
    var params = req.body; // recollim els parametres arribats per POST i els guardem
    var user = new User();
    // Si els parametres obtinguts son correctes
    if(params.name && params.surname && params.nick && params.email && params.password){
        // doncs afegim les dades a l'objecte d'usuari
        user.name = params.name;
        user.surname = params.surname;
        user.nick = params.nick;
        user.email = params.email;
        user.role = params.role;
        user.image = 'null';

        // apliquem un control per a no permetre usuaris duplicats
        User.find({ $or: [

            {nick: user.nick.toLowerCase()}, // cerquem si aquest nick ja existeix
            {email: user.email.toLowerCase()} // cerquem si aquest mail ja existeix

            ]}).exec((err,users) =>{ // apliquem una funció de callback
                if(err) return res.status(500).send({message: 'Hi ha agut un error, torna a intentar-ho!!'});//si hi
ha error, doncs..
                if(users && users.length >= 1){
                     return res.status(500).send({message: 'Usuari no disponible'});
                }else{
```

```
                    // guardem la contrassenya encriptada
                    bcrypt.hash(params.password,null,null, (err,hash) =>{
                        user.password = hash;
                        user.save((err,userStored) =>{
                            if(err) return res.status(500).send({message: 'Hi ha agut un error, torna a intentar-
ho'});//si hi ha error, doncs..
                            if(userStored){
                                    return res.status(200).send({user: userStored}); // retornem l'objecte de
l'usuari desat
                            }else{
                                    return res.status(404).send({message: 'Hi ha hagut un error, torna a intentar-
ho'})
                            }
                        });// guardem l'usuari
                    });// xifram la contrassenya
                }
            });
    }else{
        return res.status(200).send({  // si hi ha alguna mancança al formulari, doncs...
            message: 'Hi ha camps no omplerts correctament!!'
        })
    }
}
// funció per a dur a terme el login d'usuari
function loginUser(req,res){
    var params = req.body;
    var email = params.email;
    var password = params.password;
    User.findOne({email: email}, (err, user) =>{
            if(err) return res.status(500).send({message: 'Error del sistema'})
            if(user){
                bcrypt.compare(password, user.password, (err, check) =>{ // cerquem si aquest usuari i contrassenya
concorden
                    if(check){//si el check es correcte, retornem les dades d'usuari

                            if(params.gettoken){// si es rep un token, doncs..
                                // generem i retornem el token
                                return res.status(200).send({
                                        token: jwt.createToken(user)
                                });
                            }else{ // si son dades planes, doncs..
                                user.password = undefined;// d'aquesta manera evitem que es retorni e valor de la pwd
a l'objecte json

                                res.status(200).send({user});// retornem les dades d'usuari
                            }
                    }else{
                        //si hi ha error, retornen un 404
                        res.status(404).send({message:'Usuari o contrassenya incorrectes'});
                    }
                })
            }else{
                // si hi ha error, retornem un 404
```

```
                res.status(404).send({message:'Usuari o contrassenya incorrectes!'});
            }
    })
}



//obtenir les dades d'un usuari
function getUser(req, res) {
    var userId = req.params.id;

    User.findById(userId, (err, user) => {
        if (!user) return res.status(404).send({message: "User Not Found."});
        if (err) return res.status(500).send({message: "Request Error."});

        followThisUser(req.user.sub, userId).then((value) => {
            return res.status(200).send({
                user,
                following: value.following,
                followed: value.followed
            });
        });
    });
}
//retornan llistat d'usuaris paginats
function getUsers(req,res){
    var identity_user_id = req.user.sub; // obtenim id de l'usuari connectat
    var page = 1; // cream variable page
    if(req.params.page){
        page = req.params.page; // obtenim la página de l'usuari connectat
    }
    var itemsPerPage = 5; // definim la quantitat d'elements a mostrar a cada página
    User.find().sort('_id').paginate(page, itemsPerPage, (err,users,total) =>{  // obtenim els usuaris de la base de
dades ordenats per ordre de creació
        if(err) return res.status(500).send({message: 'Hi ha agut un error, torna a intentar-ho!!'}); //si hi ha
error feim un return
        if(!users) return res.status(404).send({message: 'No hi ha usuaris disponibles'});
        followingUsersId(identity_user_id).then((value)=>{
            return res.status(200).send({
            total_users:total,
            pages: Math.ceil(total/itemsPerPage), // feim el redondeig per a treure les pagines
            users_following: value.following,
            users_follow_me: value.followed,
            items_per_page: itemsPerPage,
            users
        });
        })
    } )
}

// cream una funció asícrona per a obtenir un array de usuaris seguits i usuaris que ens segueixen (usuari connectat)

async function followingUsersId(user_id){
```

```
        // guardem un array d'usuaris els quals estic seguint
        var following = await Follow.find({"user":user_id}).select({'_id':0,'_v':0,'user':0});
        var followed = await Follow.find({"followed":user_id}).select({'_id':0,'_v':0,'followed':0});
        var followings_clean = []; //obtenim el llistat sense els camps que no necessitem
            following.forEach((follow) =>{
                followings_clean.push(follow.followed);

            });
        var followeds_clean = []; //obtenim el llistat sense els camps que no necessitem
            followed.forEach((follow) =>{
                followeds_clean.push(follow.user);
            });
        return{
            following: followings_clean,
            followed: followeds_clean,
        }
}

// cream funció per a la pujada d'imàtges de perfil d'usuari
function uploadImage(req,res){

    var userId = req.params.id;
    if(userId != req.user.sub){ // comprovem si l'usuari a modificar es l'usuari conectat
         return notUploadUserImage(res, file_path, {message:'No disposes de permís per a modificar aquest
usuari!!'});
    }
    if(req.files){

        var file_path = req.files.image.path; // obtenim la ruta de l'arxiu
        var file_split =file_path.split('\\');// separem el path en directoris i subdirectoris
        var file_name = file_split[2]; //obtenim el nom de l'arxiu
        var ext_split = file_name.split('\.');//separem per a obtenir l'extensio
        var file_ext = ext_split[1];//obtenim el tipus d'extensio




        //gestionem el control d'extensions permeses
        if(file_ext == 'png' || file_ext == 'jpg' || file_ext == 'jpeg' || file_ext == 'gif'){
            //actualitzam la imàtge d'usuari
            User.findByIdAndUpdate(userId, {image: file_name}, {new:true} , (err,userUpdated) =>{
                // si la petició no retorna l'objecte actualizat, doncs..
                if(err) return res.status(500).send({message: 'Hi ha agut un error, torna a intentar-ho!!'});
                // Si no s'ha rebut l'actualització de l'usuari..
                if(!userUpdated) return res.status(404).send({message: 'Hi ha agut un error, torna a intentar-
ho!!'});
                // si tot ha anat be, doncs retorna l'objecte actualitzat
                return res.status(200).send({user: userUpdated});
            })
        }else{
            // no permetem extensions no indicades al condicional
             return notUploadUserImage(res,file_path, 'Extensió no permesa!!');
        }
```

```
    }else{
        return res.status(200).send({message: 'Hi ha hagut un problema!!'});
    }
}

// creem una funció per a no permetre la pujada d'arxius

function notUploadUserImage(res,file_path,message){
    fs.unlink(file_path, (err)=>{
        return res.status(200).send({message: message});
    })
}


//retornem l'imatge del perfil d'un usuari
function getImageFile(req,res){


    var image_file = req.params.imageFile;
    var path_file = './uploads/users/' + image_file;

    //comprovem si existeix la imàtge d'usuari
    fs.exists(path_file, (exists) =>{

        if(exists){
            res.sendFile(path.resolve(path_file)); //rebem el directori i l'arxiu
        }else{
            //sino ens retorna un missatge d'error
            res.status(200).send({message: "No s'ha trobat cap coincidència"});
        }
    })


}
```

12.2 - Model d'usuari

A aquest apartat és detalla l'esquema d'usuari a la base de dades (Api/models/user.js).

```
'use strict' //Ens permet emprar les derreres característiques de Javascript
var mongoose = require('mongoose');// carreguem mongoose per a poder treballar amb mongoDB
var Schema = mongoose.Schema; // definim un nou schema
var UserSchema = Schema({ // generem l'Schema d'usuari
     name: String,
     surname: String,
     nick: String,
     email: String,
     password: String,
     role: String,
     image: String
})
module.exports = mongoose.model('User', UserSchema); // exportem la configuració del model user
```

12.3 - Rutes del controlador d'usuari

Aquest és l'arxiu que gestiona les rutes de L'Api pertanyents al controlador d'usuari. Hi podem trobar tant mètodes de lectura(get), escriptura(post) i d'actualització(put). En una següent fase del projecte, s'hi afegiran mètodes del tipus 'Delete'; per tal de que els administradors puguin eliminar usuaris del sistema, per així adequar-se a la GDPR.

Aquest arxiu es troba a la ruta següent: (Api/routes/user.js).

```
'use strict' //Ens permet emprar les derreres característiques de Javascript

var express = require('express'); // carreguem express
var UserController = require('../controllers/user'); // carreguem el controlador d'usuari
var api = express.Router(); // carreguem el router d'express
var md_auth = require('../middlewares/auth'); //requerim el middleware d'autenticació
var multipart = require('connect-multiparty'); // carreguem la llibrería multiparty
var md_upload = multipart({uploadDir:'./uploads/users/'}); // creem middleware de pujades
api.get('/home', UserController.home);// definim la ruta
api.get('/prova',md_auth.ensureAuth, UserController.prova); // definim la ruta
api.post('/register',UserController.saveUser);
api.post('/login',UserController.loginUser);
api.get('/user/:id',md_auth.ensureAuth,UserController.getUser);
api.get('/username/',md_auth.ensureAuth,UserController.getUsername);
api.get('/users/:page?',md_auth.ensureAuth,UserController.getUsers);
api.put('/update-user/:id',md_auth.ensureAuth,UserController.updateUser);
api.post('/upload-image-user/:id',[md_auth.ensureAuth, md_upload],UserController.uploadImage);
api.get('/get-image-user/:imageFile',md_auth.ensureAuth,UserController.getImageFile);
api.get('/counters/:id?',md_auth.ensureAuth,UserController.getCounters);
module.exports = api; // exportam la configuració
```

12.4 - Vista del component d'usuaris

Aquest és l'arxiu que gestiona la vista del component 'users' del client Angular. Aquest arxiu conté els mètodes encarregats de donar la lògica a l'arxiu 'users.component.html', el qual ens ensenya un llistat d'usuaris per mitjà d'un Scroll infinit a mesura que l'usuari arriba al final de la pàgina.

Per poder portar a terme aquest 'Scroll Infinit' i tenint en compte que l'Api ens retorna els usuaris de forma paginada, es va fent una concatenació dels resultats obtinguts amb cada paginació amb els usuaris ja existents dintre de la variable users (this.users)

Aquest arxiu es troba a la ruta següent: (client/src/app/users/user.component.ts).

```
getUsers(page,adding=false){
            this._userService.getUsers(page).subscribe(
                response => {
                      if(!response.users){
                        this.status = 'error';
                      }else{
                        this.total = response.total;
                        //this.users = response.users;
                        this.pages = response.pages;
                        this.itemsPerPage = response.items_per_page;
                        this.follows = response.users_following;
                         if(!adding){  // afegim noves publicacions a l'array amb un botó    infinit
                          this.users = response.users;
                      }else{
                          var a = this.users;
                          var b = response.users;
                          this.users = a.concat(b);
                          //feim scroll automatic amb jquery
                            $("html,body").animate({scrollTop:$('html').prop("scrollHeight")},600);
                      }
                        if(page > this.pages){
                          this._router.navigate(['/users',1]);
                        }
                      }
                },
                error =>{
                  var errorMessage = <any>error;
                  console.log(errorMessage);
                      if(errorMessage != null){
                        this.status = 'error';
                      }
                }
            );
        }
```

```
    // si ja no hi ha mes publicacions no mostram el boto
    public no_more = false;
    viewMore(){
          this.page+=1;
         if(this.page == this.pages){
            this.no_more = true;
          }
        this.getUsers(this.page, true)

    }
       refresh(){
          window.location.reload()
      }


@HostListener('window:scroll', ['$event'])
          onWindowScroll() {
        if ((window.innerHeight + window.scrollY) == document.body.scrollHeight)
                    console.log('Scroll Okk');
                    this.page+=1;
                if(this.page == this.pages){
                   this.no_more = true;
                 }
               this.getUsers(this.page, true)
         }
          }
```

12.5 - Servei d'usuaris del client Angular

Aquest és l'arxiu que gestiona el servei d'usuaris amb l'Api. Aquest servei ens permet tenir una comunicació bidireccional entre l'Api i el Client.

Aquest arxiu es troba a la ruta següent: (client/src/app/services/user.service.ts).

```
//definim els serveis i els injectem a una altre classe
import {Injectable} from '@angular/core';
//importem httpClient per a poder fer peticions ajax i HttpHeaders per a poder enviar capçaleres
import {HttpClient, HttpHeaders} from '@angular/common/http';
//importem l'observable per a poder recollir les respostes de l'api
import {Observable} from 'rxjs/observable';
//importem el model d'usuari
import {User} from '../models/user';
//importam la connexió a l'api
import {Global} from './global';
//definim el decorador injectable per a poder ser utilitzat desde qualsevol classe del client
@Injectable()
export class UserService{
  public url:string;
  public identity;
  public token;
  public stats;
  //creem el constructor per a la connexió
  constructor( public _http: HttpClient ){
    this.url = Global.url;
  }
  register(user: User): Observable<any>{
    let params = JSON.stringify(user);
    let headers = new HttpHeaders().set('Content-Type','application/json');
    return this._http.post(this.url + 'register', params, {headers:headers});
  }


  signup(user, gettoken = null): Observable<any>{

    if(gettoken != null){
      user.gettoken = gettoken;
    }
    let params = JSON.stringify(user);
    let headers = new HttpHeaders().set('Content-Type','application/json');
    return this._http.post(this.url + 'login', params, {headers:headers});
  }


  //consultam les dades de localStorage per poder extreure les dades d'usuari
  getIdentity(){
    //convertim la variable identity a un objecte de javascript
    let identity = JSON.parse(localStorage.getItem('identity'));
```

```
    if(identity != "undefined"){
      this.identity = identity;
    }else{
      this.identity = null;
    }
    return this.identity;

  }

  gettoken(){
    //convertim la variable token a un objecte de javascript
    let token = localStorage.getItem('token');
    if(token != "undefined"){
       this.token = token;
    }else{
      this.token = null;
    }
      return this.token;
    }
    getStats(){
        let stats = JSON.parse(localStorage.getItem('stats'));
        if(stats != "undefined"){
          this.stats = stats;
        }else{
          this.stats = null;
        }
        return this.stats;
    }

    getCounters(userId = null): Observable<any>{
      let headers = new HttpHeaders().set('Content-Type', 'application/json').set('Authorization',
this.gettoken());

      //comprovam si hi ha usuari autenticat
      if(userId != null){
              return this._http.get(this.url + 'counters/' + userId, {headers:headers});
      }else{
              return this._http.get(this.url + 'counters/',{headers:headers});
      }
    }
    updateUser(user: User): Observable<any>{
      let params = JSON.stringify(user);//convertim els parametres a objecte
      let headers = new HttpHeaders().set('Content-Type', 'application/json').set('Authorization',
this.gettoken());//decodificam el token
      return this._http.put(this.url + 'update-user/' + user._id, params,{headers:headers});

    }
    getUsers(page = null): Observable<any>{
        let headers = new HttpHeaders().set('Content-Type', 'application/json').set('Authorization',
this.gettoken());//decodificam el token
        return this._http.get(this.url + 'users/'+ page,{headers:headers});
    }
```

13. Bibliografia

- https://www.udemy.com/course/master-en-javascript-aprender-js-jquery-angular-nodejs-y-mas/
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/JSON%20Web%20Token
- https://momentjs.com/
- https://devcode.la/tutoriales/middlewares-en-nodejs/
- https://www.udemy.com/desarrollar-una-red-social-con-javascript-angular-y-nodejs-mongodb/
- https://www.c-sharpcorner.com/article/implement-infinite-scrolling-using-angular-6/
- https://www.youtube.com/watch?v=c8n6JsQuX2A
- https://ng-bootstrap.github.io/#/components/pagination/overview
- https://www.udemy.com/socket-io-solutions/
- https://carlosazaustre.es/como-crear-una-api-rest-usando-node-js/
- https://getbootstrap.com/docs/4.3/utilities/
- https://api.jquery.com/