ROBOCHAT

Final angular project documentation

José Martín Bruno
DAM - 2 - A
Interface development
2018-2019

INDEX

Github link: https://github.com/jmb463/RoboChat

INTRODUCTION

In this document we will explain in detail our newest project called Robochat, this a angular based project (along with the main mark languages such as HTML and CSS).Angular is a platform that makes it easy to build applications with the web, we will also use SCSS (Sassy CSS), and Bootstrap 4, this tools will help us making a professional project without investing much time in the basics.

As mentioned before SCSS it's a “better CSS” because SCSS is closer to the syntax of a programming language than CSS, and it helps programmers build their stylesheets.

And bootstrap helps in the design of the HTML, adds more professionalism and like SCSS, makes the code easier and more understandable.

In conclusion, we use this technologies because they help us making our project professional and it reduces a lot of code, and a lot of hours.

So now focusing in the project itself, robochat is a project that includes a login, signup pages, we will use the online backend Google Firebase so we can make users and they work properly, we will also have a chat page that will be divided in 2 parts, one being the message part, having the users name, their profile picture, and a message and the other part uses firebase as well to extract all the chats stored in the firebase (this point will be explain later in the 2.4 part of the document).

To put all the code together we will use the program Visual Studio Code.

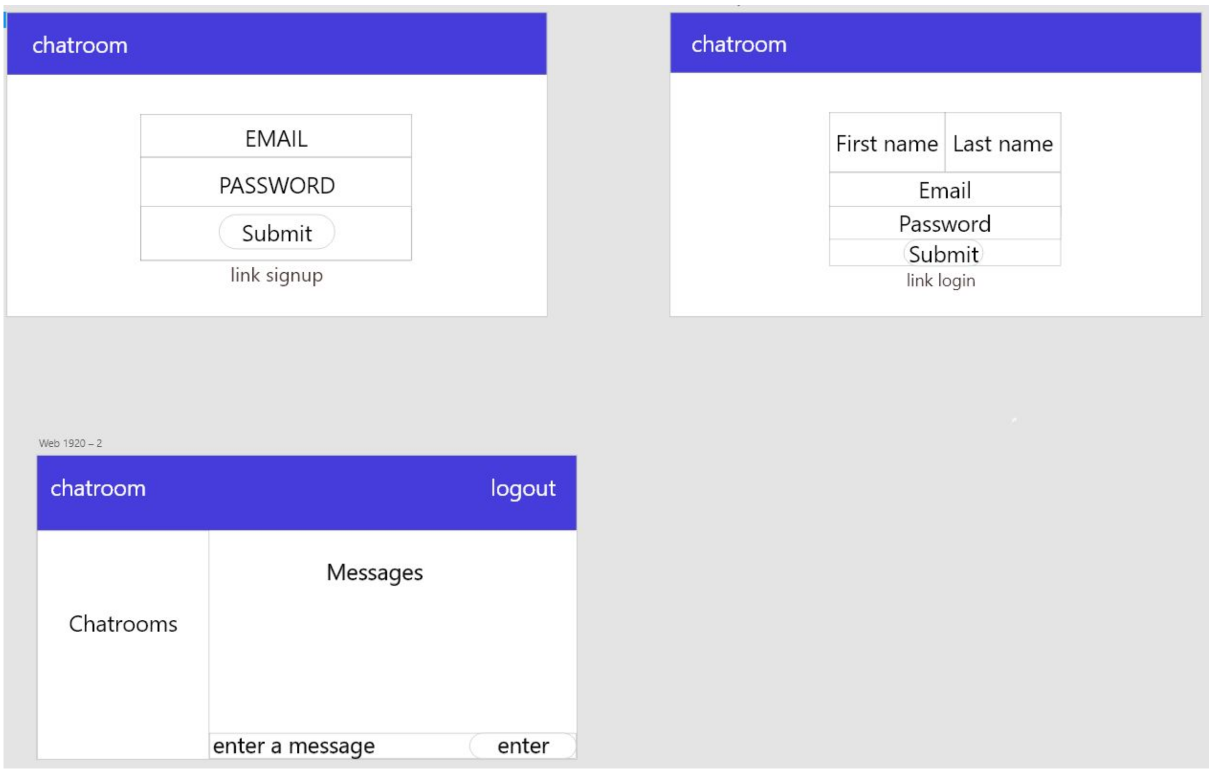
And finally we will be using node.js and git, both of this programs installation will not be shown in this document.

DESIGN AND IMPLEMENTATION

Design factors

We used the recommended colors from our project manager, we created some models on how the page should look and how its distributed, always looking to be mobile first, so we used Adobe XD to make our design.

This is one of the pages



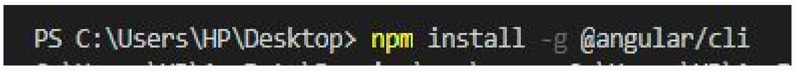
Screenshot 0: Adobe XD design

So before doing anything we need to open our Visual Studio Code and open a new terminal, we can do this on terminal → new terminal.

Screenshot 1: new terminal

Before creating the login form, we need to install angular CLI (angular command line interface), this is the main step on building our angular app, angular CLI is used to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

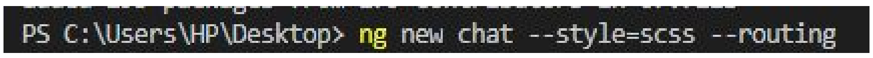
To install angular CLI globally (-g) we will use the following command.



```
PS C:\Users\HP\Desktop> npm install -g @angular/cli
```

Screenshot 2: angular cli installation

Now we will need to initialize our angular web-project, we can do that with the command



```
PS C:\Users\HP\Desktop> ng new chat --style=scss --routing
```

Screenshot 3: new angular project

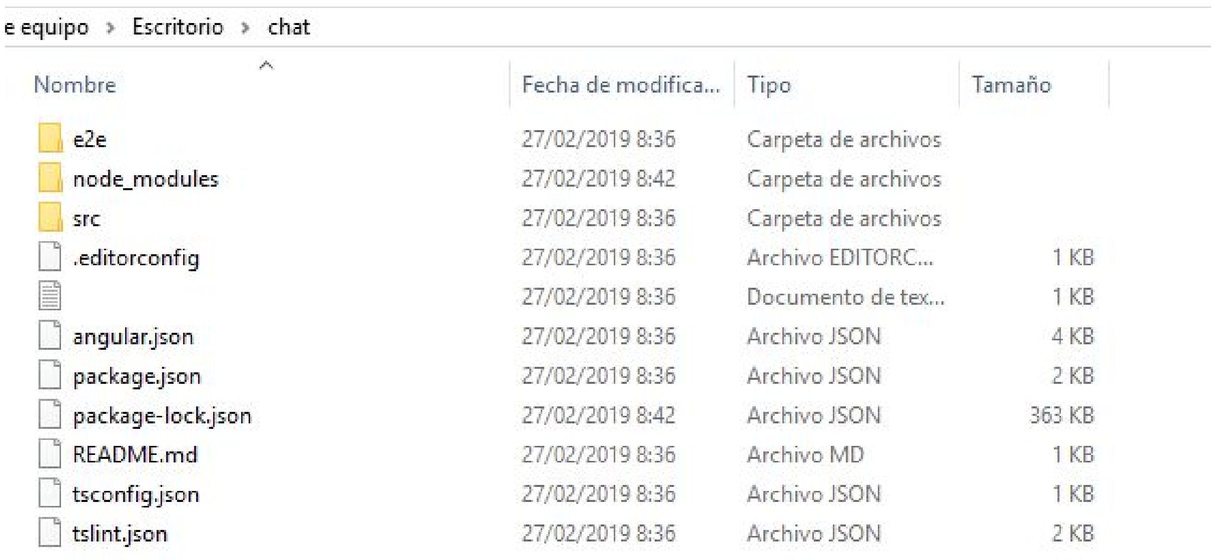
All of angular commands are runnable by using ng, so we create a new angular project called chat, and then we put the style on SCSS, if we don't do this piece of the command we would need to specify in the installation, and --routing means that we will be installing a routing module.

Angular apps are modular and Angular has its own modularity system called *NgModules*. NgModules are containers for a block of code dedicated to an application. They can contain components, service providers, and other code files whose scope is defined by the containing NgModule, we will talk about components and services later in the document.

This installation will take some time depending on the computer, because it generates the basic angular project and the folder, in my case since my route was on desktop, so my chat folder will be there.

Screenshot 4: chat folder

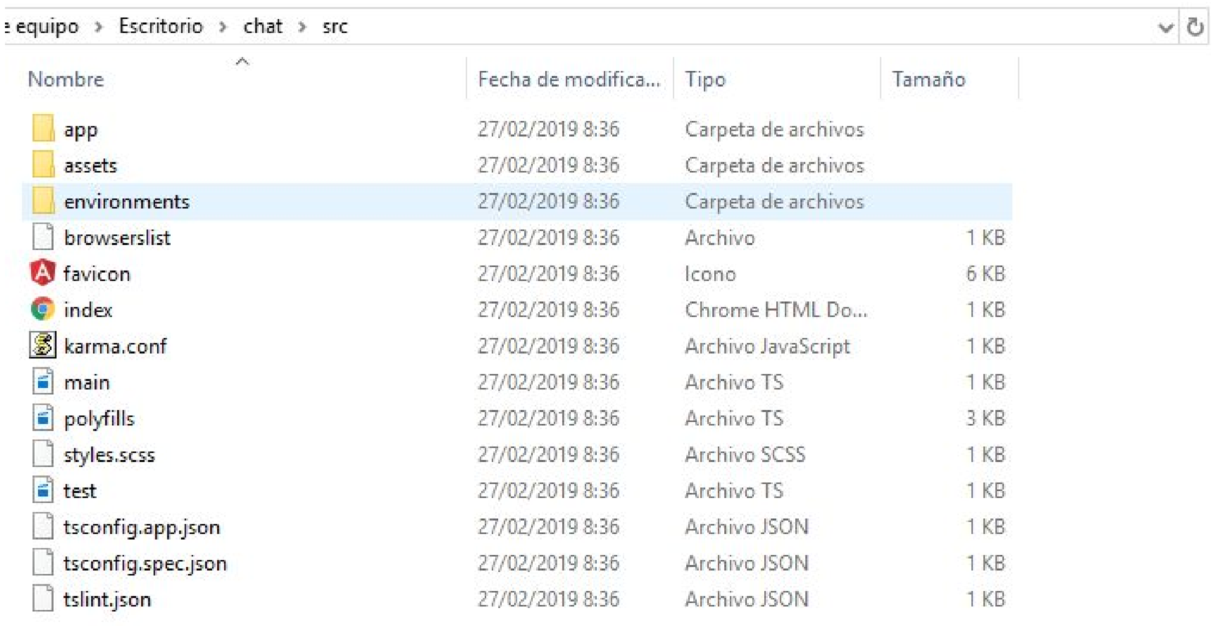
We can see that angular already created some documents, we will mainly use the src folder in which our main project is located.



e equipo › Escritorio › chat

Nombre	Fecha de modifica...	Tipo	Tamaño
e2e	27/02/2019 8:36	Carpeta de archivos	
node_modules	27/02/2019 8:42	Carpeta de archivos	
src	27/02/2019 8:36	Carpeta de archivos	
.editorconfig	27/02/2019 8:36	Archivo EDITORC...	1 KB
	27/02/2019 8:36	Documento de tex...	1 KB
angular.json	27/02/2019 8:36	Archivo JSON	4 KB
package.json	27/02/2019 8:36	Archivo JSON	2 KB
package-lock.json	27/02/2019 8:42	Archivo JSON	363 KB
README.md	27/02/2019 8:36	Archivo MD	1 KB
tsconfig.json	27/02/2019 8:36	Archivo JSON	1 KB
tslint.json	27/02/2019 8:36	Archivo JSON	2 KB

Screenshot 5: inside of chat folder



e equipo › Escritorio › chat › src

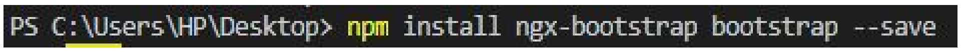
Nombre	Fecha de modifica...	Tipo	Tamaño
app	27/02/2019 8:36	Carpeta de archivos	
assets	27/02/2019 8:36	Carpeta de archivos	
environments	27/02/2019 8:36	Carpeta de archivos	
browserslist	27/02/2019 8:36	Archivo	1 KB
favicon	27/02/2019 8:36	Icono	6 KB
index	27/02/2019 8:36	Chrome HTML Do...	1 KB
karma.conf	27/02/2019 8:36	Archivo JavaScript	1 KB
main	27/02/2019 8:36	Archivo TS	1 KB
polyfills	27/02/2019 8:36	Archivo TS	3 KB
styles.scss	27/02/2019 8:36	Archivo SCSS	1 KB
test	27/02/2019 8:36	Archivo TS	1 KB
tsconfig.app.json	27/02/2019 8:36	Archivo JSON	1 KB
tsconfig.spec.json	27/02/2019 8:36	Archivo JSON	1 KB
tslint.json	27/02/2019 8:36	Archivo JSON	1 KB

Screenshot 6: src folder

So this is what the src contains a folder called app, that we will be creating all of our project , environments that we will use later to connect into firebase and the index that we will need to change later so we can see all of our modifications, and of course our main styling file styles.scss, this is the main style of the page.

But before doing anything we need to install the other main library, in the beginning we said that this project is based on Angular, SCSS and Bootstrap, so we need to install the angular version of bootstrap, since we already installed the other two.

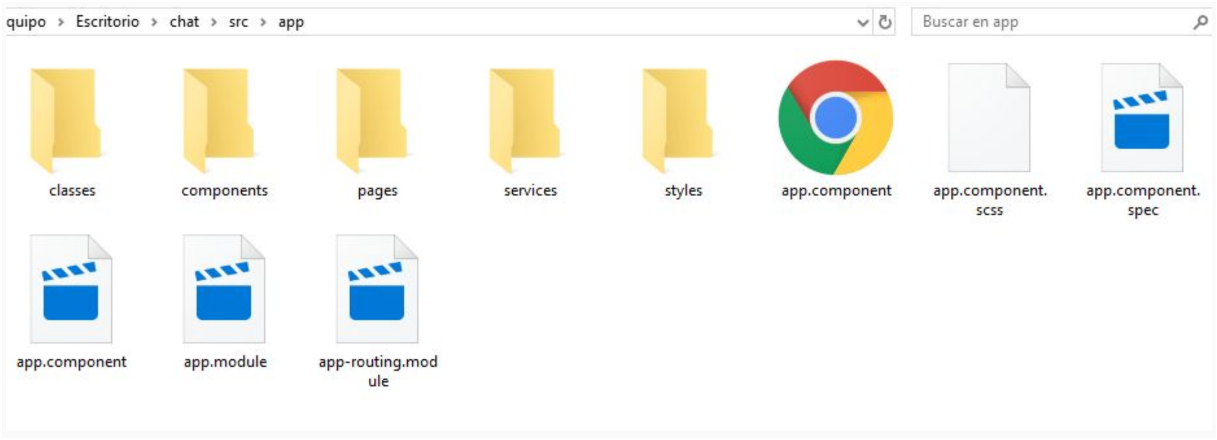
To install angular bootstrap we will use the following command:



```
PS C:\Users\HP\Desktop> npm install ngx-bootstrap bootstrap --save
```

Screenshot 7: install ngx-bootstrap

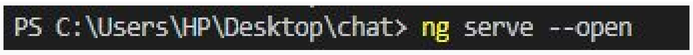
So now that we have installed bootstrap for angular we will create 5 folders on which will store all of our components and services (this folder will be situated inside the app folder).



Screenshot 8: list of the new folders

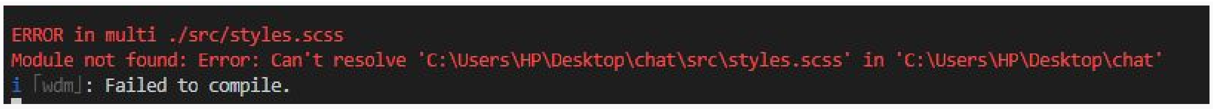
Since we want everything to be organized we will move the styles.scss to the styles folder.

If we try to execute the web-project using the command:



```
PS C:\Users\HP\Desktop\chat> ng serve --open
```

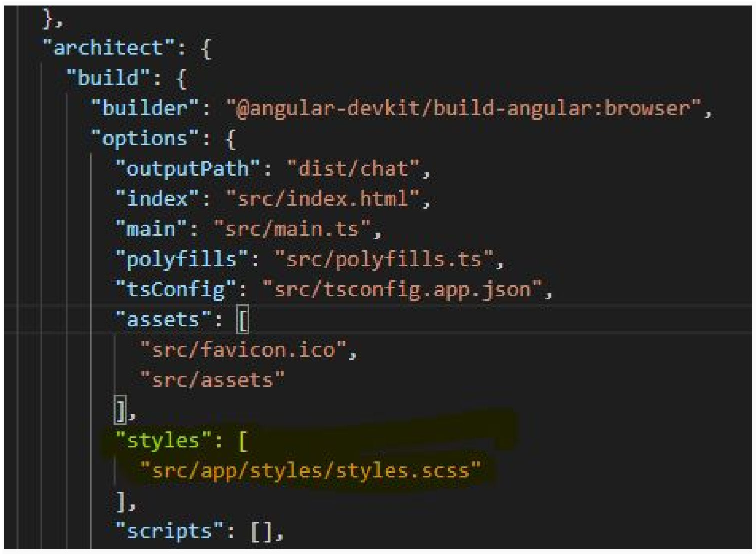
Screenshot 10: ng serve --open



```
ERROR in multi ./src/styles.scss
Module not found: Error: Can't resolve 'C:\Users\HP\Desktop\chat\src\styles.scss' in 'C:\Users\HP\Desktop\chat'
i ｢wdm｣: Failed to compile.
```

Screenshot 9: Error

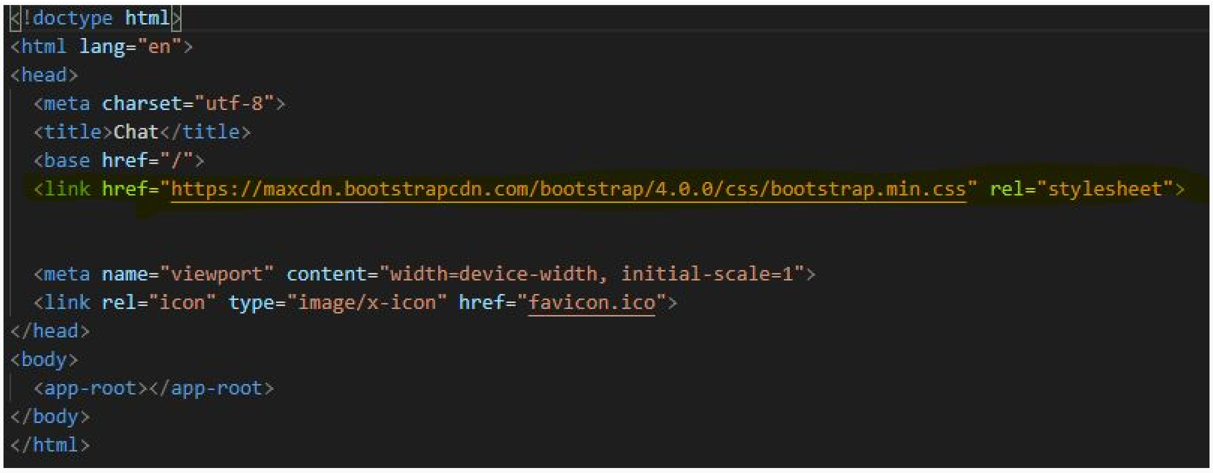
An error pops up, that's due to the fact that we changed the styles.scss from his place, so now we need to go to angular.json in the main chat folder and change the route of the style.



```
},
"architect": {
  "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {
      "outputPath": "dist/chat",
      "index": "src/index.html",
      "main": "src/main.ts",
      "polyfills": "src/polyfills.ts",
      "tsConfig": "src/tsconfig.app.json",
      "assets": [
        "src/favicon.ico",
        "src/assets"
      ],
      "styles": [
        "src/app/styles/styles.scss"
      ],
      "scripts": [],
```

Screenshot 10: changing route

If we open the project we will see that the default page don't have bootstrap applied, so we need to go to the index.html in the src folder and put the bootstrap 4 link.



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Chat</title>
  <base href="/">
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet">


  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

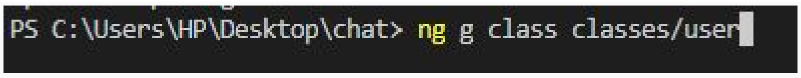
Screenshot 11: bootstrap link

Before getting into the main code we need to generate some elements like services, components, enums and guards, so now we will briefly explain what this elements are.

- Services: Service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.
- Components: Components are the fundamental building blocks of Angular applications. They display data on the screen, listen for user input, and take action based on that input.

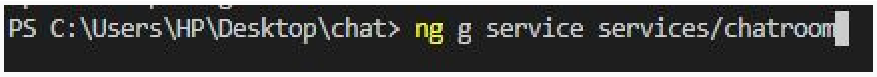
- Enums: Enum allows you to specify a possible property value from a predefined set of values using meaningful names, instead of the numeric constants that are usually used in this case.
- Guards: Guards are interfaces which can tell the router whether or not it should allow navigation to a requested route.

To generate all of this elements we need to use the command ng generate or ng g as an abbreviation, let’s see some examples:



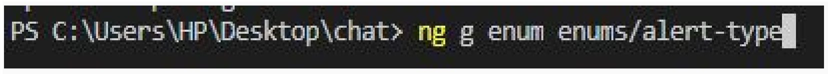
```
PS C:\Users\HP\Desktop\chat> ng g class classes/user
```

Screenshot 12: generate a class



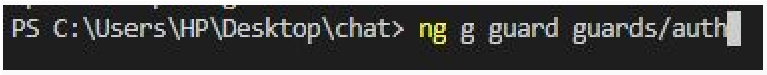
```
PS C:\Users\HP\Desktop\chat> ng g service services/chatroom
```

Screenshot 13: generate a service



```
PS C:\Users\HP\Desktop\chat> ng g enum enums/alert-type
```

Screenshot 14: generate a enum



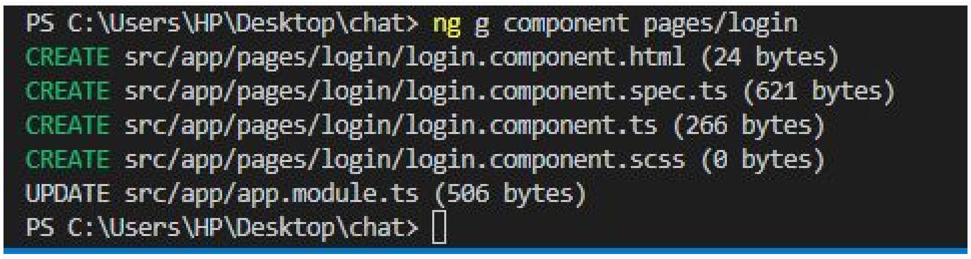
```
PS C:\Users\HP\Desktop\chat> ng g guard guards/auth
```

Screenshot 15: generate a guard

2.1 PAGES

Login page

First we need to generate the component login with the following command, as seen before, we will use this as the ng g component example.

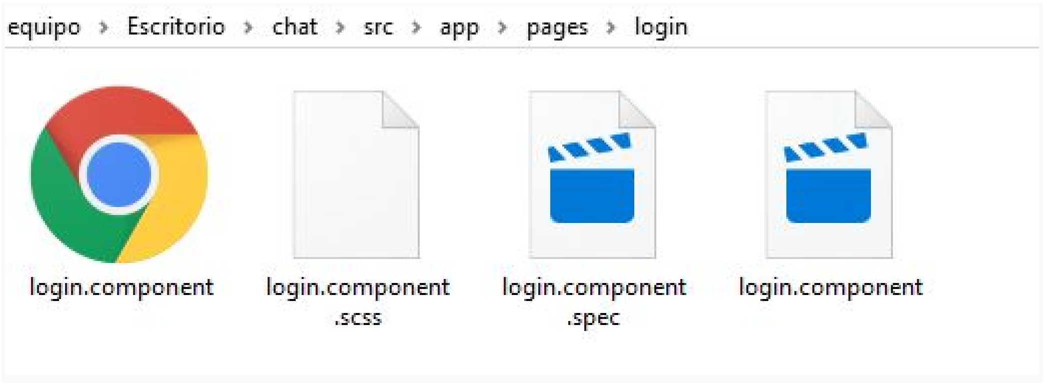


```
PS C:\Users\HP\Desktop\chat> ng g component pages/login
CREATE src/app/pages/login/login.component.html (24 bytes)
CREATE src/app/pages/login/login.component.spec.ts (621 bytes)
CREATE src/app/pages/login/login.component.ts (266 bytes)
CREATE src/app/pages/login/login.component.scss (0 bytes)
UPDATE src/app/app.module.ts (506 bytes)
PS C:\Users\HP\Desktop\chat> 
```

Screenshot 16: create login component

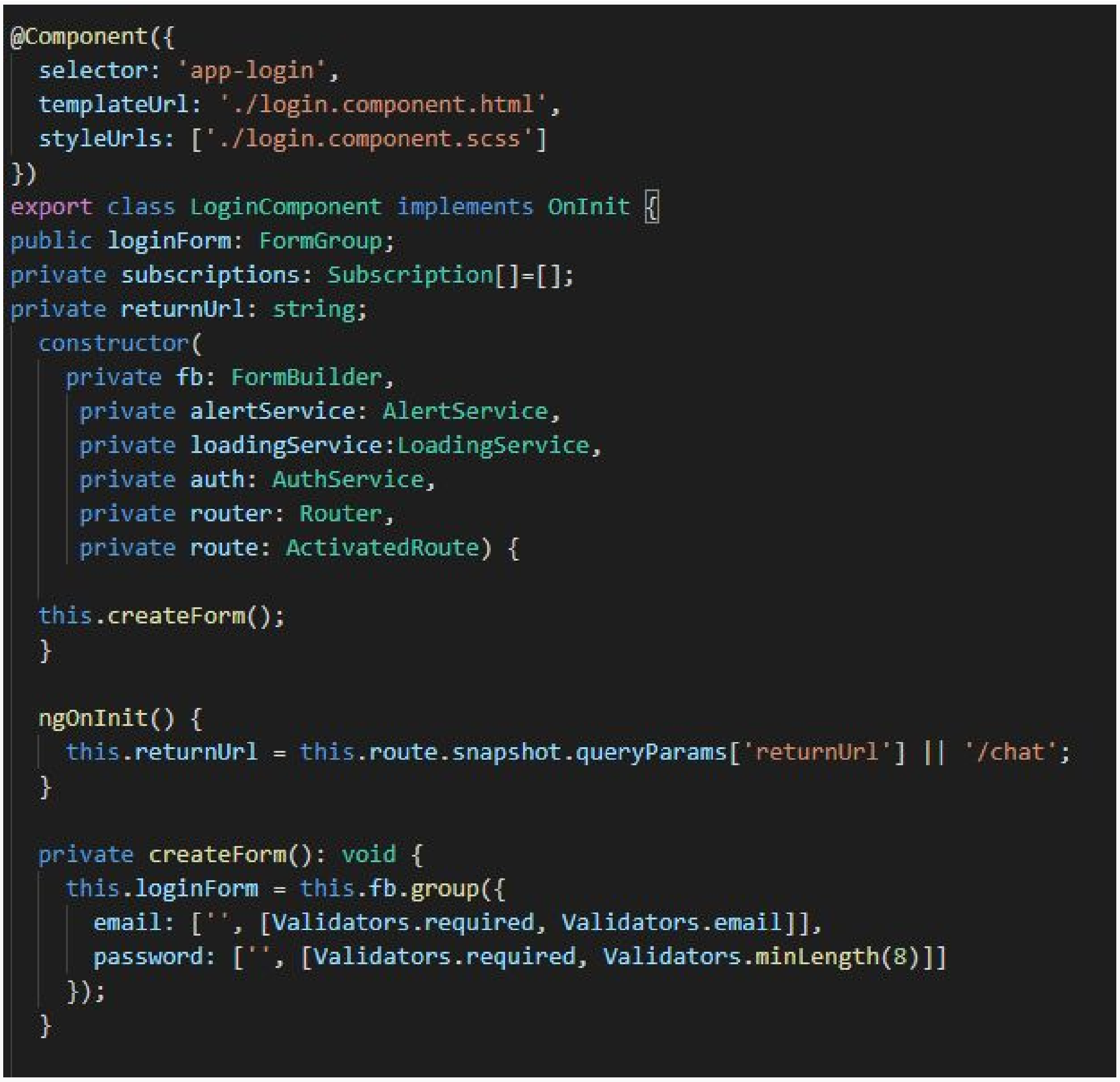
if we go to the folder we see that angular created the component html and ts (which is a variation of JavaScript for angular).

So we see that components generates us a HTML document a SCSS document and two typeScript documents, the document is the only generable element that does this, in the other elements such as the services it only generates two typeScripts.



Screenshot 17: created login documents

Now that we have our login component lets explain how the code works starting from the typeScript.



```
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
public loginForm: FormGroup;
private subscriptions: Subscription[]=[];
private returnUrl: string;
  constructor(
    private fb: FormBuilder,
     private alertService: AlertService,
     private loadingService:LoadingService,
     private auth: AuthService,
     private router: Router,
     private route: ActivatedRoute) {

  this.createForm();
  }

  ngOnInit() {
    this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/chat';
  }

  private createForm(): void {
    this.loginForm = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.minLength(8)]]
    });
  }
```

Screenshot 18: Login.TS part I

In @Component we have:

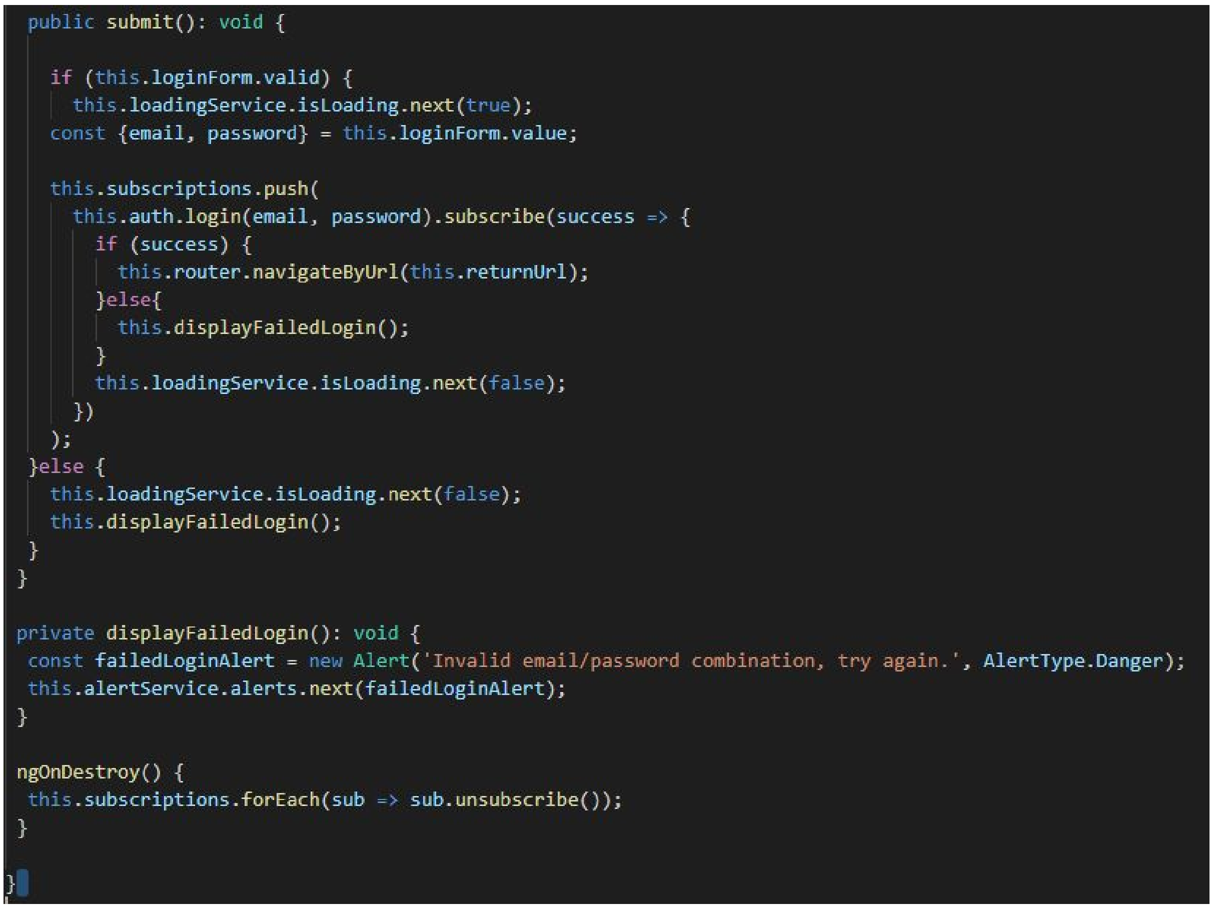
- Selector: This indicates the name of the component, for example if we wanted to use this template later in our code we would need to put <app-login></app-login> (this example would show us the template)
- TemplateUrl: This indicates the template* which is a HTML document.
- StyleUrl: This indicates the route of the SCSS file.

*Templates are simply HTML working with Angular on the proper rendering of the component within the application.

Then we have our class called LoginComponent on the constructor we have created a FormBuilder that will add our form functionality (explained better in its method), both services that provides an alert that shows up if the login fails, and a loading screen that shows up as well in the same circumstance as the login, then we have the auth service that search in the backend and verifies that the authentication is correct and we have the router and route that we use to get routes and navigate through them.

First method is the ngOnInit, that is a callback method that is invoked immediately after the default change detector has checked the directive's data-bound properties for the first time, and before any of the view or content children have been checked. It is invoked only once when the directive is instantiated. In this method we set our returnUrl string to the route of the chat page.

The second method is the createForm, this method will create a form using the FormBuilder and we set the e-mail and the password to be required and we put on a restriction that the password has to be at least 8 characters and the e-mail is based on the google parameters this means that there has to be an @ and a dot in the email.

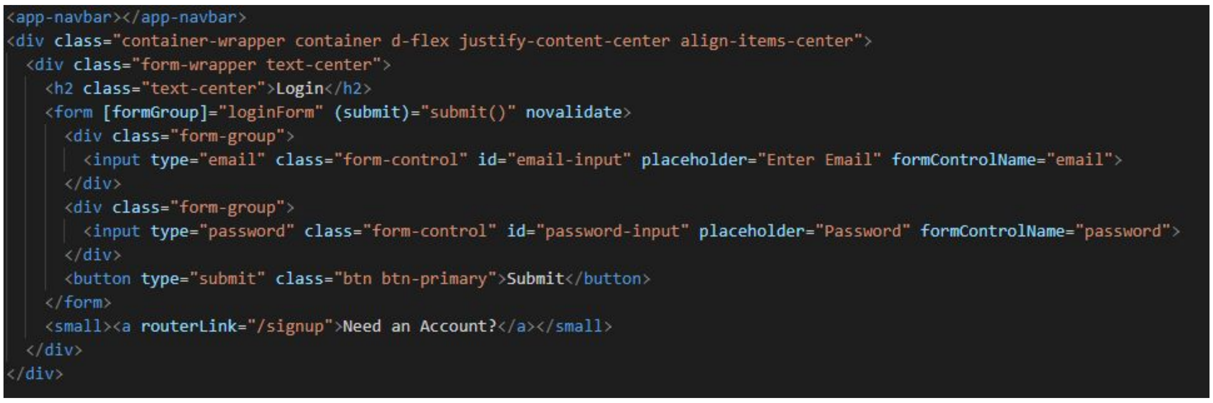


```
public submit(): void {

  if (this.loginForm.valid) {
    this.loadingService.isLoading.next(true);
  const {email, password} = this.loginForm.value;

  this.subscriptions.push(
    this.auth.login(email, password).subscribe(success => {
      if (success) {
        this.router.navigateByUrl(this.returnUrl);
      }else{
        this.displayFailedLogin();
      }
      this.loadingService.isLoading.next(false);
    })
  );
 }else {
  this.loadingService.isLoading.next(false);
  this.displayFailedLogin();
 }
}

private displayFailedLogin(): void {
 const failedLoginAlert = new Alert('Invalid email/password combination, try again.', AlertType.Danger);
 this.alertService.alerts.next(failedLoginAlert);
}

ngOnDestroy() {
 this.subscriptions.forEach(sub => sub.unsubscribe());
}

}
```

Screenshot 19: login.ts part II

Then we have the submit method that checks if the the login is valid then it will appear the chat, if its not valid it will show a message saying that the login failed (the message will have a red background as defined in the AlertType enum) and it will also show the typical loading screen defined in the loadingService.

To conclude with this component TS we have the onDestroy method that it's A callback method that performs custom clean-up, invoked immediately after a directive, pipe, or service instance is destroyed.



```
<app-navbar></app-navbar>
<div class="container-wrapper container d-flex justify-content-center align-items-center">
  <div class="form-wrapper text-center">
    <h2 class="text-center">Login</h2>
    <form [formGroup]="loginForm" (submit)="submit()" novalidate>
      <div class="form-group">
        <input type="email" class="form-control" id="email-input" placeholder="Enter Email" formControlName="email">
      </div>
      <div class="form-group">
        <input type="password" class="form-control" id="password-input" placeholder="Password" formControlName="password">
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
    </form>
    <small><a routerLink="/signup">Need an Account?</a></small>
  </div>
</div>
```

Screenshot 20: login HTML

So we have our html, first we put our navbar at the top calling the component selector (like said before), then we create a div that works as a container and inside this div we have our form wrapper. Inside the form-wrapper we put a h2 with the title of the component and we create a form with the element form.

This form is a ReactiveFormModule, this reactive form provide a model-driven approach to handling form inputs whose values change over time.

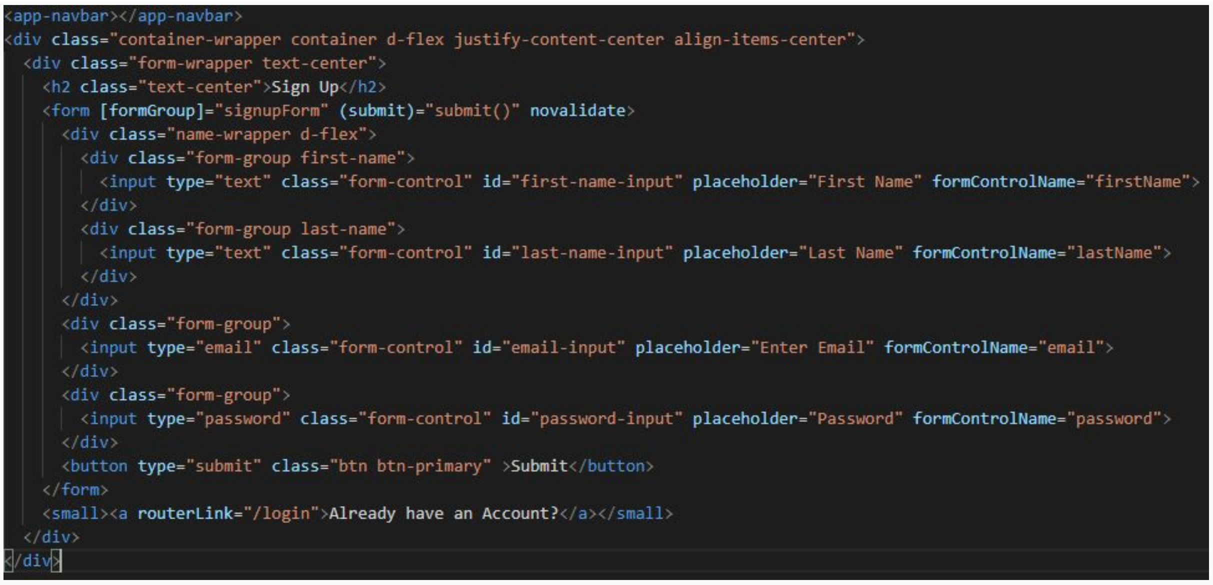
So when the form is submitted it will call the function submit that we have seen earlier in the typeScript.

After we create our form, we are going to create 2 form groups, one being the email input and the other one being the password input.

Below this two form groups we find the submit button and a link that sends us to the signup page.

Signup page

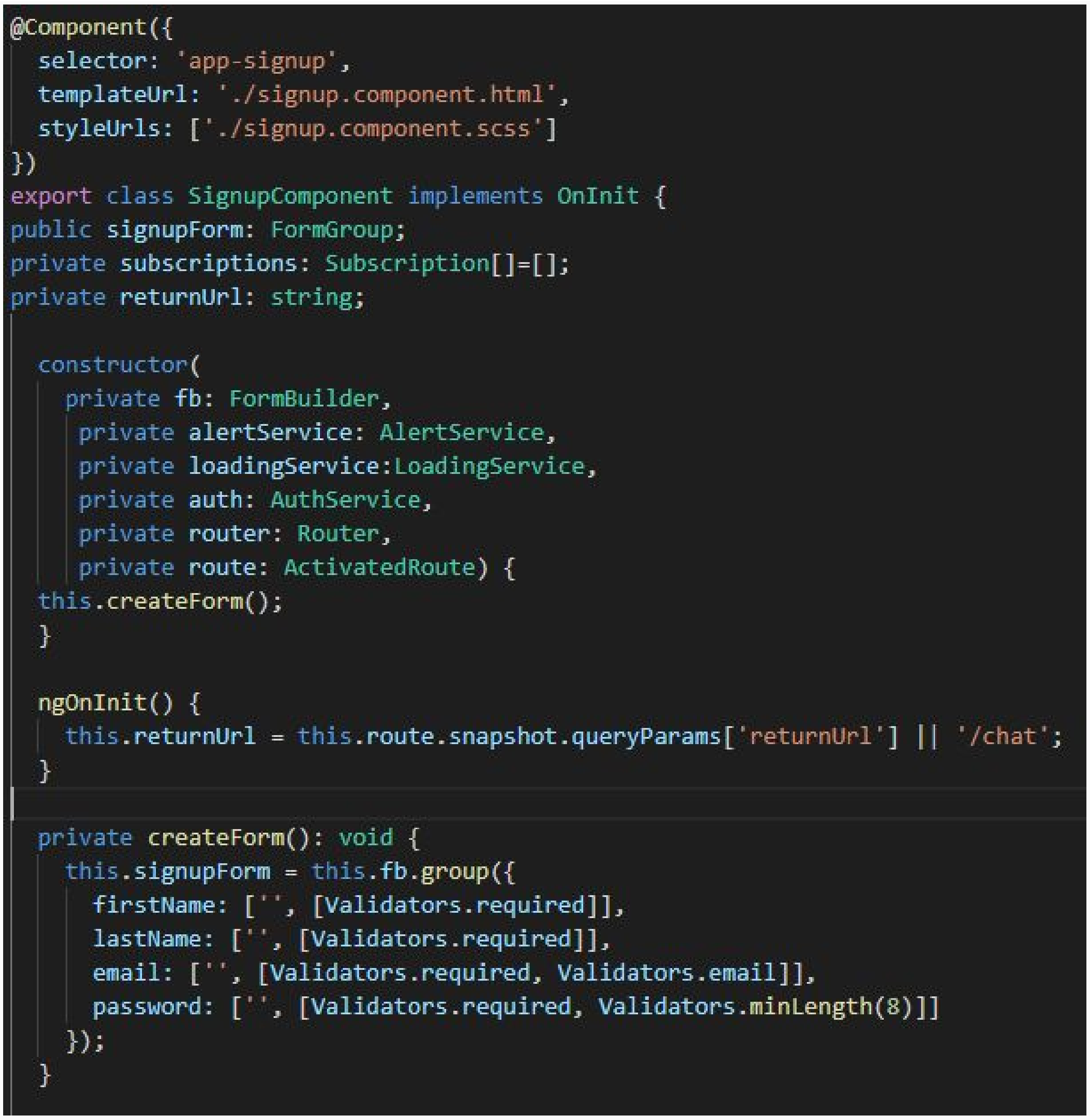
The signup page is very similar to the login page, changing the number of inputs.



```html
<app-navbar></app-navbar>
<div class="container-wrapper container d-flex justify-content-center align-items-center">
  <div class="form-wrapper text-center">
    <h2 class="text-center">Sign Up</h2>
    <form [formGroup]="signupForm" (submit)="submit()" novalidate>
      <div class="name-wrapper d-flex">
        <div class="form-group first-name">
          <input type="text" class="form-control" id="first-name-input" placeholder="First Name" formControlName="firstName">
        </div>
        <div class="form-group last-name">
          <input type="text" class="form-control" id="last-name-input" placeholder="Last Name" formControlName="lastName">
        </div>
      </div>
      <div class="form-group">
        <input type="email" class="form-control" id="email-input" placeholder="Enter Email" formControlName="email">
      </div>
      <div class="form-group">
        <input type="password" class="form-control" id="password-input" placeholder="Password" formControlName="password">
      </div>
      <button type="submit" class="btn btn-primary" >Submit</button>
    </form>
    <small><a routerLink="/login">Already have an Account?</a></small>
  </div>
</div>
```

Screenshot: 21 signup HTML

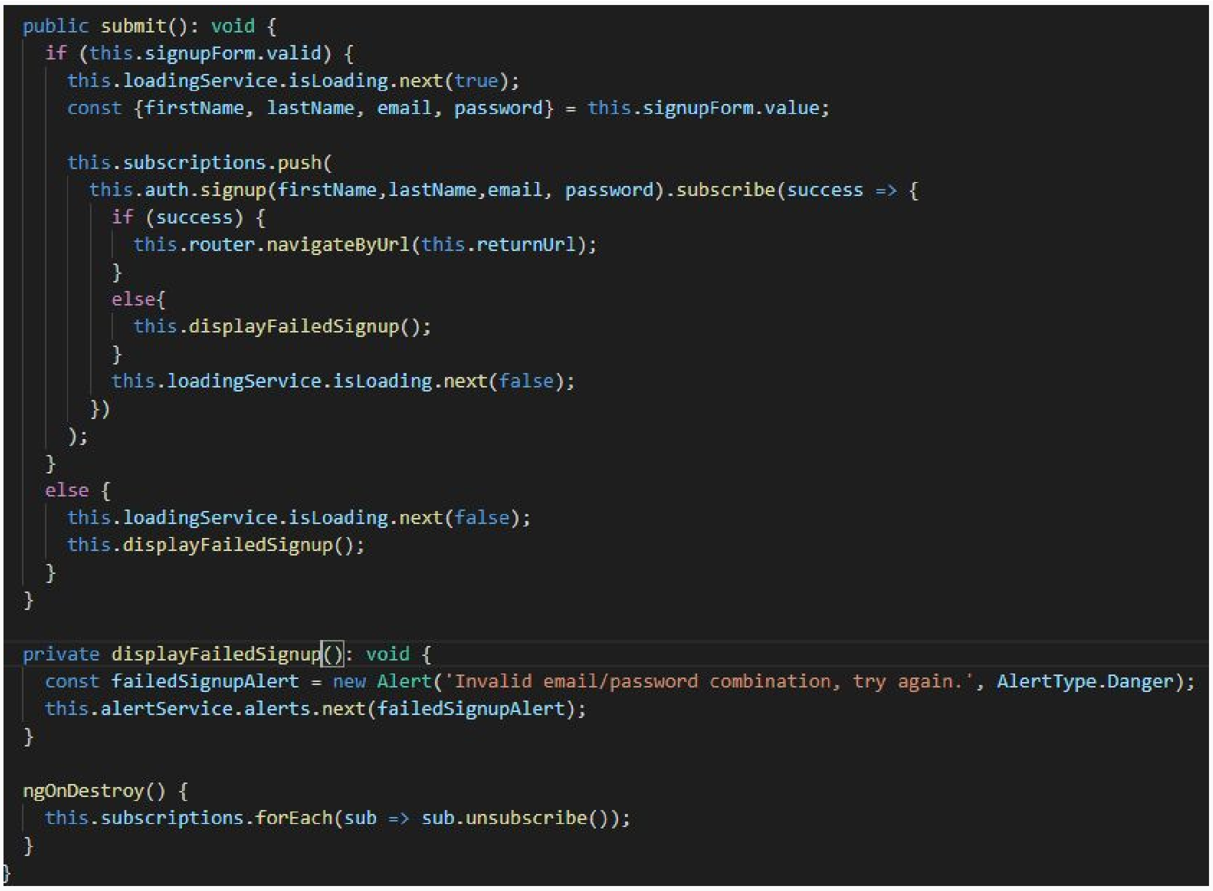
In this screenshot we see how similar it is to the loginForm, the only thing that changes in the HTML compared to the other one is the number of the form and the number of inputs/form groups.



```
@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.scss']
})
export class SignupComponent implements OnInit {
public signupForm: FormGroup;
private subscriptions: Subscription[]=[];
private returnUrl: string;

  constructor(
    private fb: FormBuilder,
     private alertService: AlertService,
     private loadingService:LoadingService,
     private auth: AuthService,
     private router: Router,
     private route: ActivatedRoute) {
  this.createForm();
  }

  ngOnInit() {
    this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/chat';
  }

  private createForm(): void {
    this.signupForm = this.fb.group({
      firstName: ['', [Validators.required]],
      lastName: ['', [Validators.required]],
      email: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.minLength(8)]]
    });
  }
```

Screenshot 22: Signup.ts part I

We see this is also similar to the login typeScript, we create our variables the same way as before, the only thing changing in the createForm method is the added values being the firstName and the lastName values.



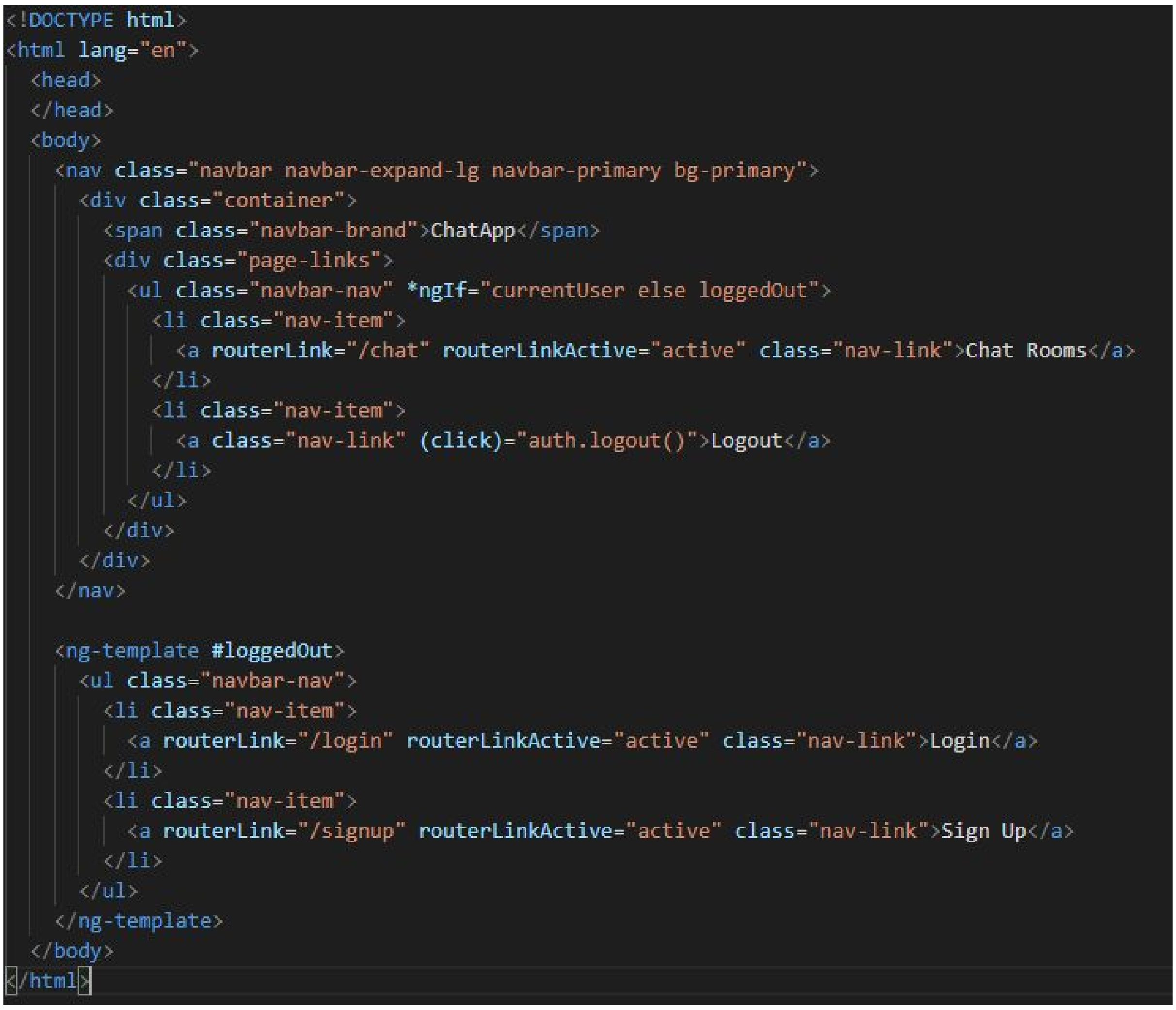
```
public submit(): void {
  if (this.signupForm.valid) {
    this.loadingService.isLoading.next(true);
    const {firstName, lastName, email, password} = this.signupForm.value;

    this.subscriptions.push(
      this.auth.signup(firstName,lastName,email, password).subscribe(success => {
        if (success) {
          this.router.navigateByUrl(this.returnUrl);
        }
        else{
          this.displayFailedSignup();
        }
        this.loadingService.isLoading.next(false);
      })
    );
  }
  else {
    this.loadingService.isLoading.next(false);
    this.displayFailedSignup();
  }
}

private displayFailedSignup(): void {
  const failedSignupAlert = new Alert('Invalid email/password combination, try again.', AlertType.Danger);
  this.alertService.alerts.next(failedSignupAlert);
}

ngOnDestroy() {
  this.subscriptions.forEach(sub => sub.unsubscribe());
}
}
```

Screenshot 23: Signup.ts part II

As compared with the login its the same code and functionality changing a couple of constant names and adding the firstName and lastName.

Navbar Component

The navbar component is a component that contains the navbar used in every page, it also includes the functionality that if the user is not on the chatroom page it will appear a different content that if the user is on the chatroom page.

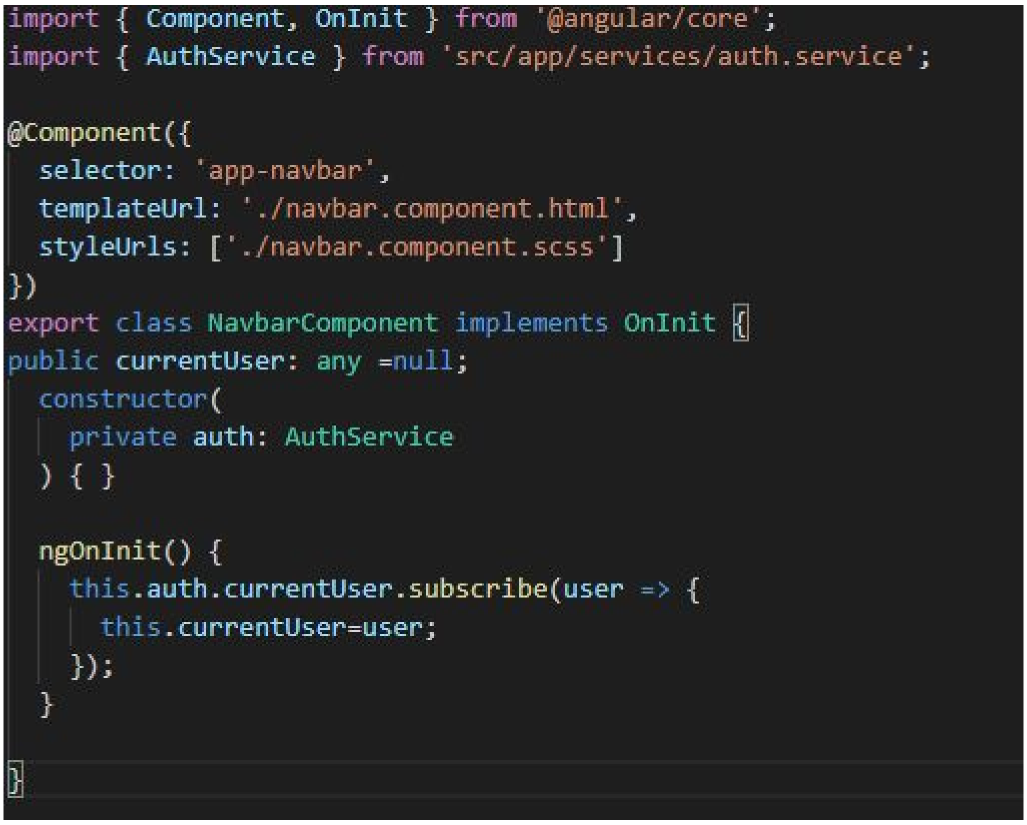


```html
<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body>
    <nav class="navbar navbar-expand-lg navbar-primary bg-primary">
      <div class="container">
        <span class="navbar-brand">ChatApp</span>
        <div class="page-links">
          <ul class="navbar-nav" *ngIf="currentUser else loggedOut">
            <li class="nav-item">
              <a routerLink="/chat" routerLinkActive="active" class="nav-link">Chat Rooms</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" (click)="auth.logout()">Logout</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>

    <ng-template #loggedOut>
      <ul class="navbar-nav">
        <li class="nav-item">
          <a routerLink="/login" routerLinkActive="active" class="nav-link">Login</a>
        </li>
        <li class="nav-item">
          <a routerLink="/signup" routerLinkActive="active" class="nav-link">Sign Up</a>
        </li>
      </ul>
    </ng-template>
  </body>
</html>
```

Screenshot 24: navbarComponent HTML

In this component we have 2 different options, if the user is logged in it will show a navbar that will have a link to log-out and a link to the chatroom page itself, if the user is logged out meaning its on the login page or the signup page it will appear the link to the login page and the link to the signup page.

Detailing on the code we see some containers a navbar-brand that will be shown on the left of the navbar and if the currentUser (explained in the TS) is not loggedOut then it will show those navbar items, if we click in the logout link it will accede to the auth service and the logout method. If it is loggedOut then it will show the other 2 links.



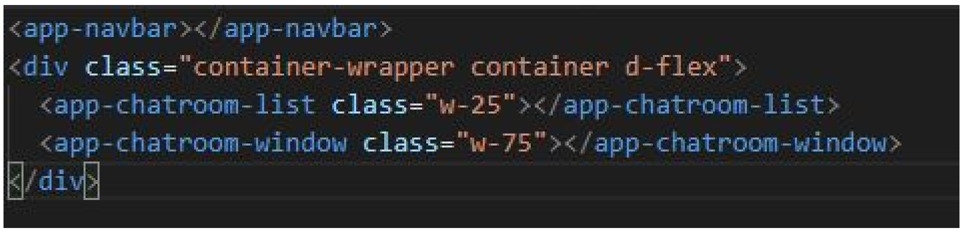
```
import { Component, OnInit } from '@angular/core';
import { AuthService } from 'src/app/services/auth.service';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.scss']
})
export class NavbarComponent implements OnInit {
public currentUser: any =null;
  constructor(
    private auth: AuthService
  ) { }

  ngOnInit() {
    this.auth.currentUser.subscribe(user => {
      this.currentUser=user;
    });
  }

}
```

Screenshot 25: navbar component TS

In the NavbarComponent class we have our constructor with the auth service that when the component is invoked it will set our currentUser to the user from the service.

Chat component

In this chat component we will add the navbar and we will divide the page in two, one being 25% of the screen that will be located the chatroom list and the other one that will occupy 75% of the screen and will display the chatroom window that shows the chat with the profile pictures, the message and the name.

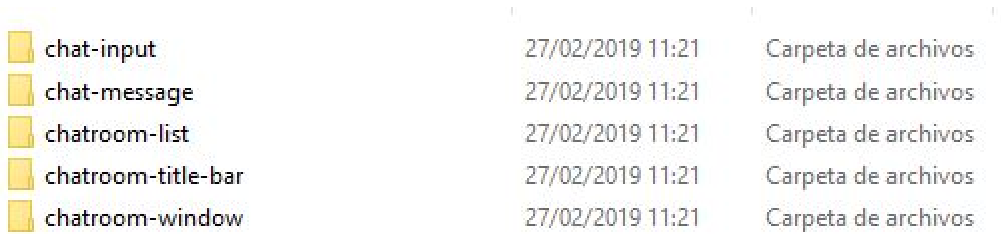


```
<app-navbar></app-navbar>
<div class="container-wrapper container d-flex">
  <app-chatroom-list class="w-25"></app-chatroom-list>
  <app-chatroom-window class="w-75"></app-chatroom-window>
</div>
```

Screenshot 26: chat component HTML

Chat Components

In this folder called components inside chat we will have located the components for the chatroom page, those being the chatroom-list and chatroom-window that we have seen in the previous screenshot and added to that we have:

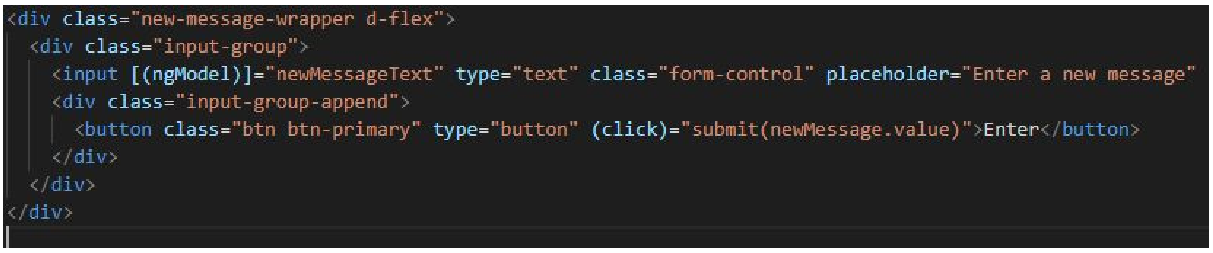


```
chat-input            27/02/2019 11:21    Carpeta de archivos
chat-message          27/02/2019 11:21    Carpeta de archivos
chatroom-list         27/02/2019 11:21    Carpeta de archivos
chatroom-title-bar    27/02/2019 11:21    Carpeta de archivos
chatroom-window       27/02/2019 11:21    Carpeta de archivos
```

Screenshot 27: chat folder

Chat input

The chat input component features a small input with a button that is located in the 75% of the screen and at the bottom of it

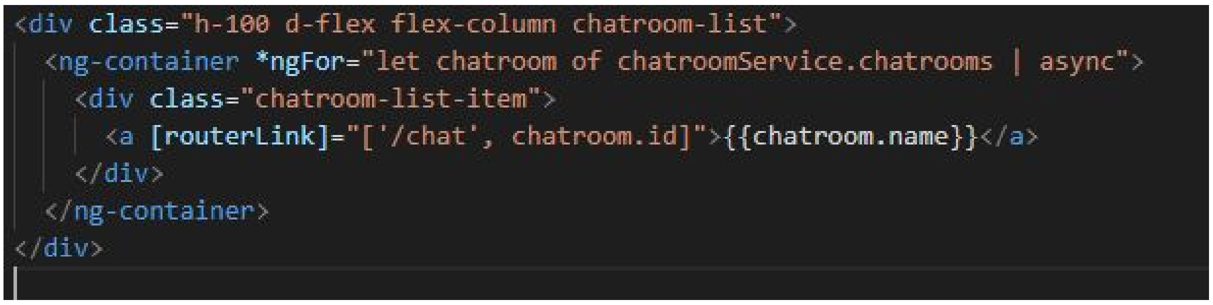


```
<div class="new-message-wrapper d-flex">
  <div class="input-group">
    <input [(ngModel)]="newMessageText" type="text" class="form-control" placeholder="Enter a new message"
    <div class="input-group-append">
      <button class="btn btn-primary" type="button" (click)="submit(newMessage.value)">Enter</button>
    </div>
  </div>
</div>
```

Screenshot 28: chat input

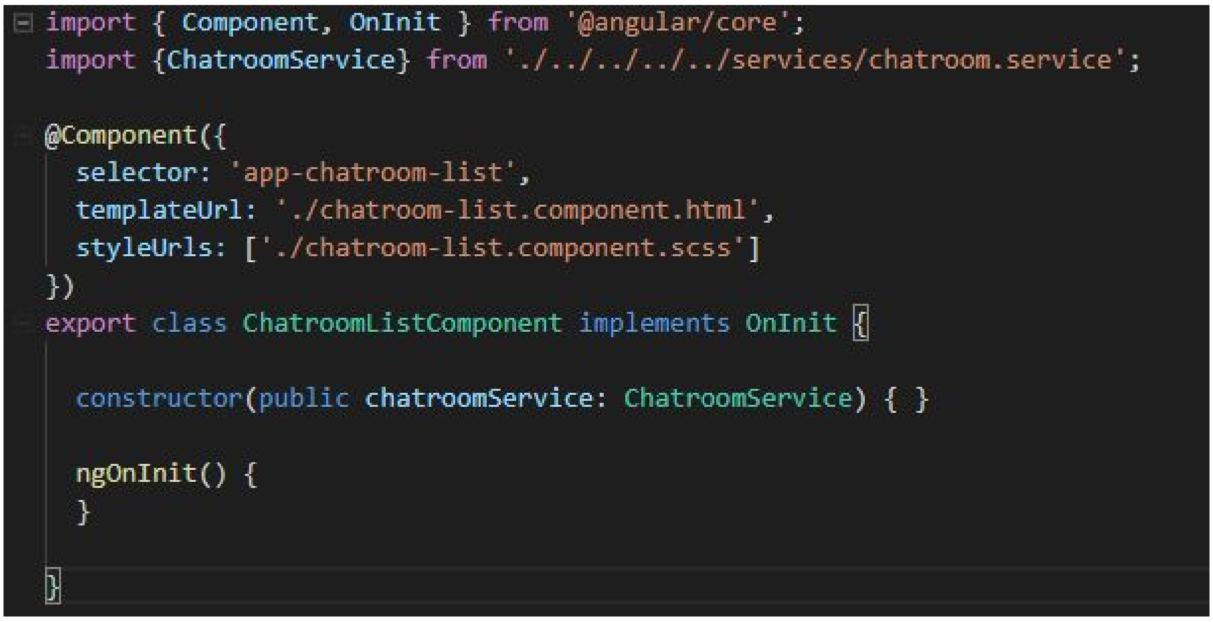
Chatroom list

The chatroom list component, is a component that shows each chatroom thats in the database (explained better in the chatroom service section). The HTML file uses an angular for, that creates a variable named chatroom of the service and we cast the name of it thanks to the double key {{}}.



```
<div class="h-100 d-flex flex-column chatroom-list">
  <ng-container *ngFor="let chatroom of chatroomService.chatrooms | async">
    <div class="chatroom-list-item">
      <a [routerLink]="['/chat', chatroom.id]">{{chatroom.name}}</a>
    </div>
  </ng-container>
</div>
```

Screenshot 29: chatroom list HTML



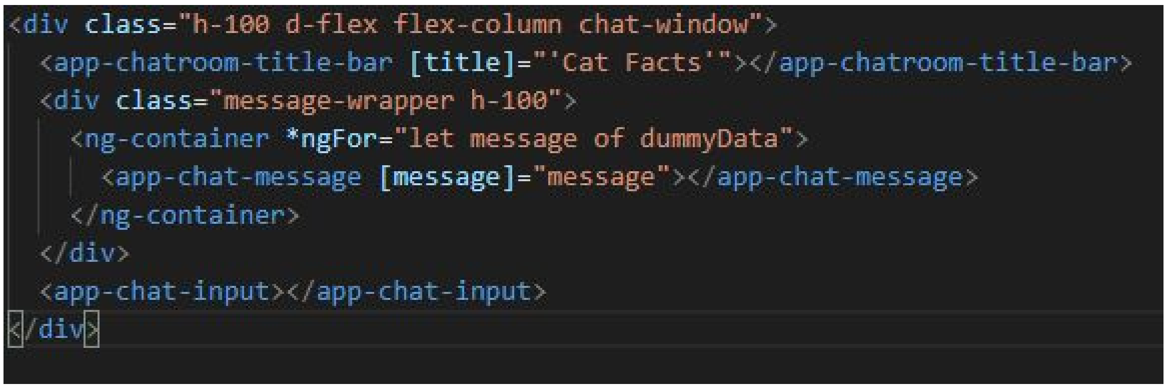
```
import { Component, OnInit } from '@angular/core';
import {ChatroomService} from './../../../../services/chatroom.service';

@Component({
  selector: 'app-chatroom-list',
  templateUrl: './chatroom-list.component.html',
  styleUrls: ['./chatroom-list.component.scss']
})
export class ChatroomListComponent implements OnInit {

  constructor(public chatroomService: ChatroomService) { }

  ngOnInit() {
  }

}
```

Screenshot 30: chatroom list TS

In the typeScript we add the chatRoomService in the constructor, this public chatroomService is used in the ngFor located in the HTML, this will call to the variable chatrooms that has all the name of the chatrooms stored in the database.

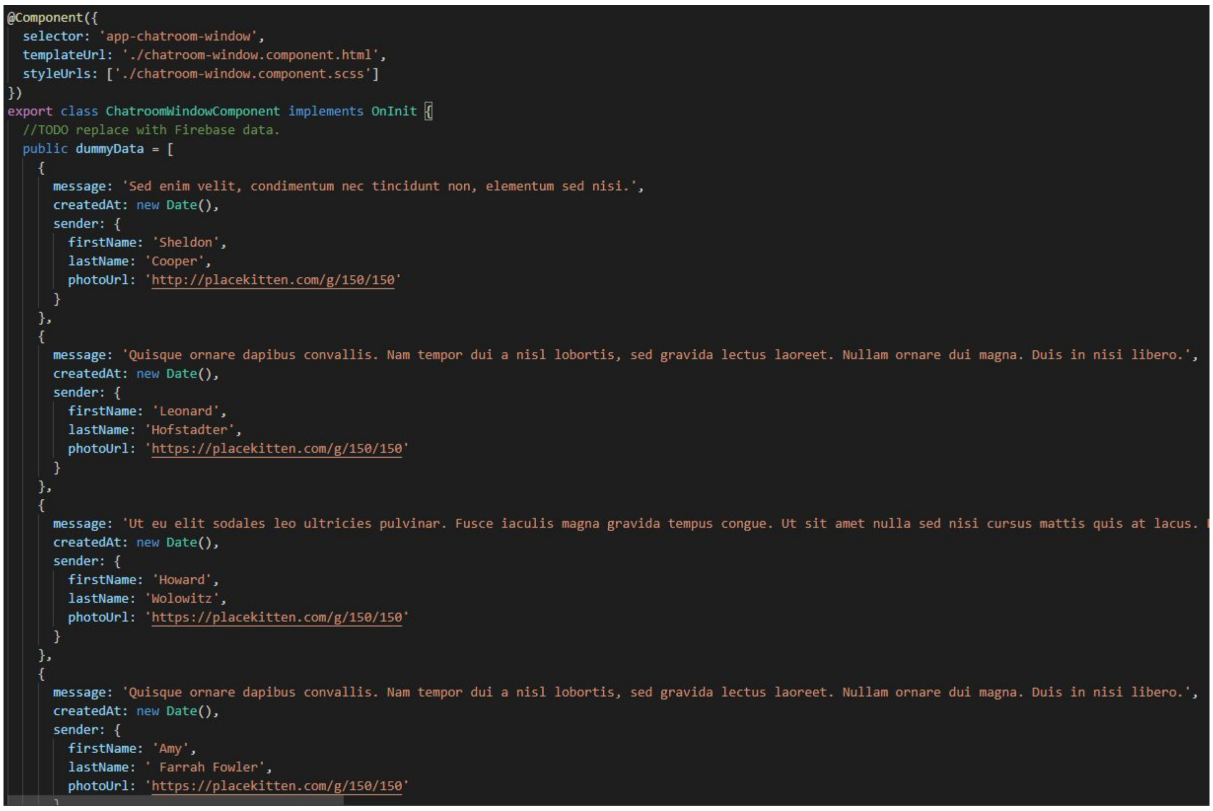
Chatroom window component

In this component we would have all the message from each chat, but due to lack of time this only shows some hard coded messages in the typeScript.



```
<div class="h-100 d-flex flex-column chat-window">
  <app-chatroom-title-bar [title]="'Cat Facts'"></app-chatroom-title-bar>
  <div class="message-wrapper h-100">
    <ng-container *ngFor="let message of dummyData">
      <app-chat-message [message]="message"></app-chat-message>
    </ng-container>
  </div>
  <app-chat-input></app-chat-input>
</div>
```

Screenshot 31: chatroom window component

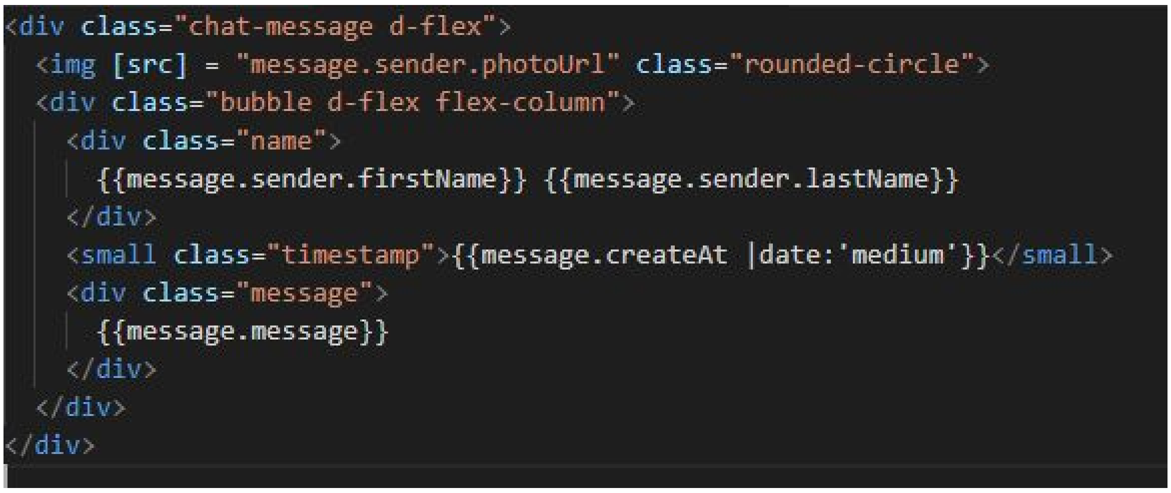


```
@Component({
  selector: 'app-chatroom-window',
  templateUrl: './chatroom-window.component.html',
  styleUrls: ['./chatroom-window.component.scss']
})
export class ChatroomWindowComponent implements OnInit {
  //TODO replace with Firebase data.
  public dummyData = [
    {
      message: 'Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.',
      createdAt: new Date(),
      sender: {
        firstName: 'Sheldon',
        lastName: 'Cooper',
        photoUrl: 'http://placekitten.com/g/150/150'
      }
    },
    {
      message: 'Quisque ornare dapibus convallis. Nam tempor dui a nisl lobortis, sed gravida lectus laoreet. Nullam ornare dui magna. Duis in nisi libero.',
      createdAt: new Date(),
      sender: {
        firstName: 'Leonard',
        lastName: 'Hofstadter',
        photoUrl: 'https://placekitten.com/g/150/150'
      }
    },
    {
      message: 'Ut eu elit sodales leo ultricies pulvinar. Fusce iaculis magna gravida tempus congue. Ut sit amet nulla sed nisi cursus mattis quis at lacus.
      createdAt: new Date(),
      sender: {
        firstName: 'Howard',
        lastName: 'Wolowitz',
        photoUrl: 'https://placekitten.com/g/150/150'
      }
    },
    {
      message: 'Quisque ornare dapibus convallis. Nam tempor dui a nisl lobortis, sed gravida lectus laoreet. Nullam ornare dui magna. Duis in nisi libero.',
      createdAt: new Date(),
      sender: {
        firstName: 'Amy',
        lastName: ' Farrah Fowler',
        photoUrl: 'https://placekitten.com/g/150/150'
```

Screenshot 32: Chatroom window hardcoded messages.

Chatroom message component

This is the component that is used in the chatroom window component to get the messages casted from the message class.



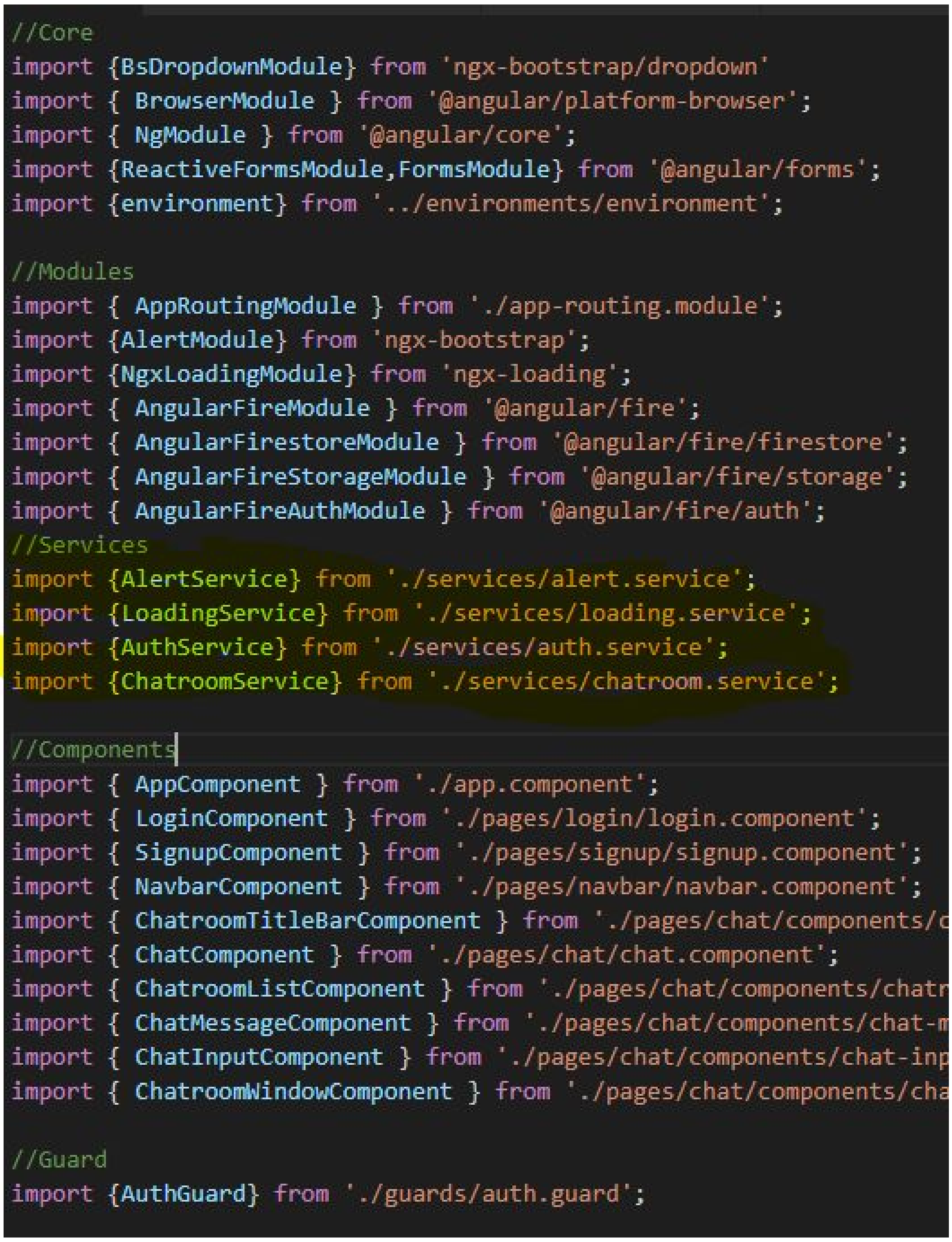
```
<div class="chat-message d-flex">
  <img [src] = "message.sender.photoUrl" class="rounded-circle">
  <div class="bubble d-flex flex-column">
    <div class="name">
      {{message.sender.firstName}} {{message.sender.lastName}}
    </div>
    <small class="timestamp">{{message.createAt |date:'medium'}}</small>
    <div class="message">
      {{message.message}}
    </div>
  </div>
</div>
```

Screenshot 33: chatroom message

2.2 Services

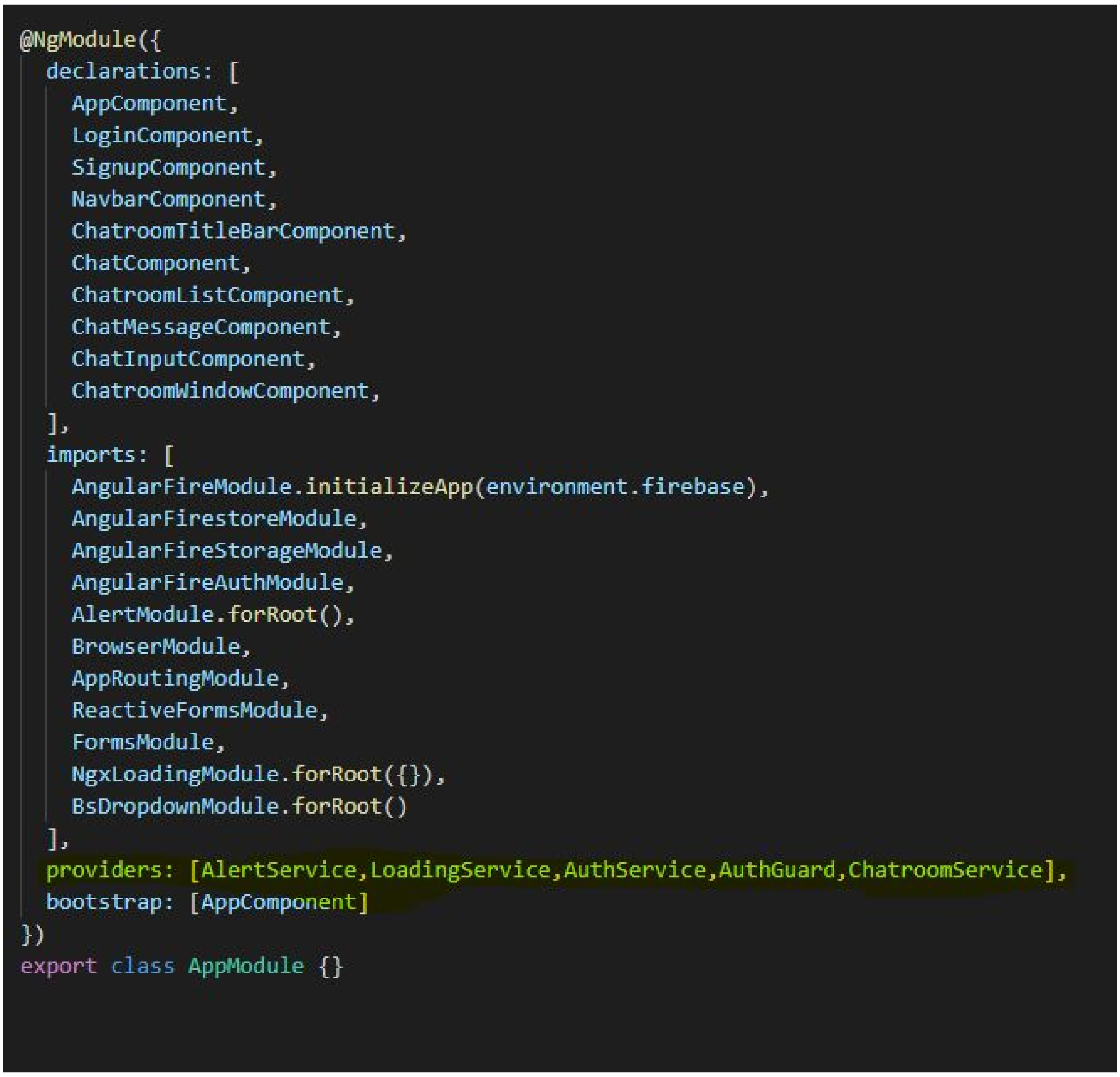
In this folder we find all the services like said before in the services there's only two typeScripts instead of a template.

We have need to declare all of our modules in the node modules file and set them in the section providers.



```
//Core
import {BsDropdownModule} from 'ngx-bootstrap/dropdown'
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {ReactiveFormsModule,FormsModule} from '@angular/forms';
import {environment} from '../environments/environment';

//Modules
import { AppRoutingModule } from './app-routing.module';
import {AlertModule} from 'ngx-bootstrap';
import {NgxLoadingModule} from 'ngx-loading';
import { AngularFireModule } from '@angular/fire';
import { AngularFirestoreModule } from '@angular/fire/firestore';
import { AngularFireStorageModule } from '@angular/fire/storage';
import { AngularFireAuthModule } from '@angular/fire/auth';
//Services
import {AlertService} from './services/alert.service';
import {LoadingService} from './services/loading.service';
import {AuthService} from './services/auth.service';
import {ChatroomService} from './services/chatroom.service';

//Components
import { AppComponent } from './app.component';
import { LoginComponent } from './pages/login/login.component';
import { SignupComponent } from './pages/signup/signup.component';
import { NavbarComponent } from './pages/navbar/navbar.component';
import { ChatroomTitleBarComponent } from './pages/chat/components/c
import { ChatComponent } from './pages/chat/chat.component';
import { ChatroomListComponent } from './pages/chat/components/chatr
import { ChatMessageComponent } from './pages/chat/components/chat-m
import { ChatInputComponent } from './pages/chat/components/chat-inp
import { ChatroomWindowComponent } from './pages/chat/components/cha

//Guard
import {AuthGuard} from './guards/auth.guard';
```

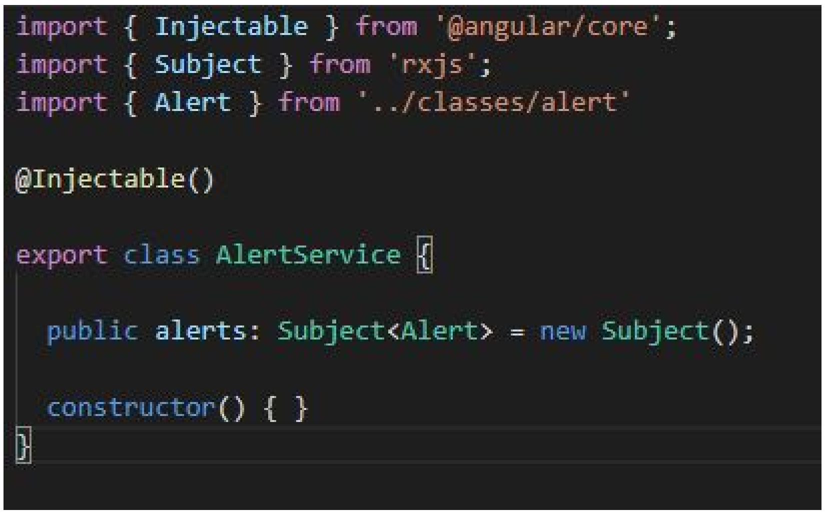
Screenshot 34: node modules service part I



```
@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    SignupComponent,
    NavbarComponent,
    ChatroomTitleBarComponent,
    ChatComponent,
    ChatroomListComponent,
    ChatMessageComponent,
    ChatInputComponent,
    ChatroomWindowComponent,
  ],
  imports: [
    AngularFireModule.initializeApp(environment.firebase),
    AngularFirestoreModule,
    AngularFireStorageModule,
    AngularFireAuthModule,
    AlertModule.forRoot(),
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    FormsModule,
    NgxLoadingModule.forRoot({}),
    BsDropdownModule.forRoot()
  ],
  providers: [AlertService,LoadingService,AuthService,AuthGuard,ChatroomService],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Screenshot 35: node modules service part II

Alert Service

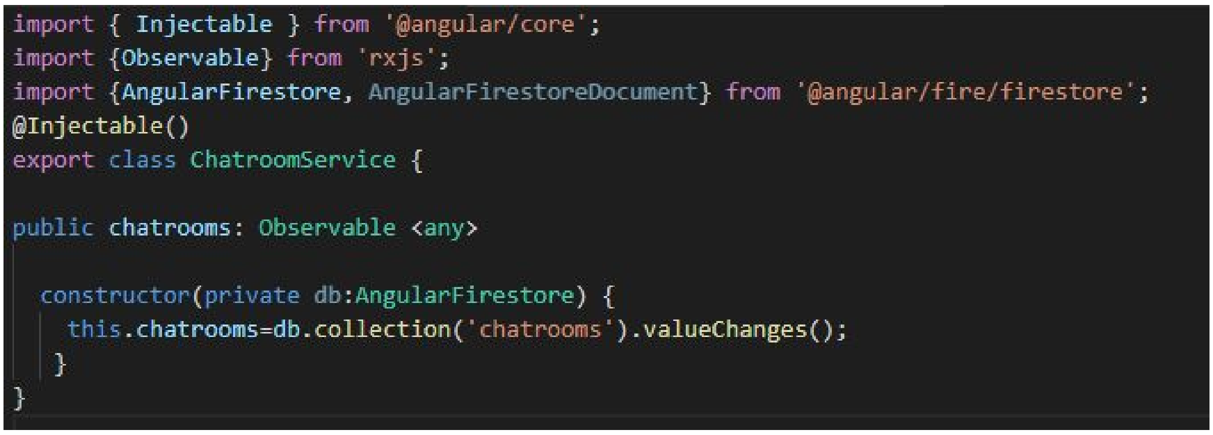


```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';
import { Alert } from '../classes/alert'

@Injectable()

export class AlertService {

  public alerts: Subject<Alert> = new Subject();

  constructor() { }
}
```

Screenshot 36: alert service

This alert service uses the class alert that will be explained in its own section, the alert will use a subject which it allow us to subscribe to one or more Observables, one of Subject functionalities is the ability to observe an event and send a message.

Chatroom Service

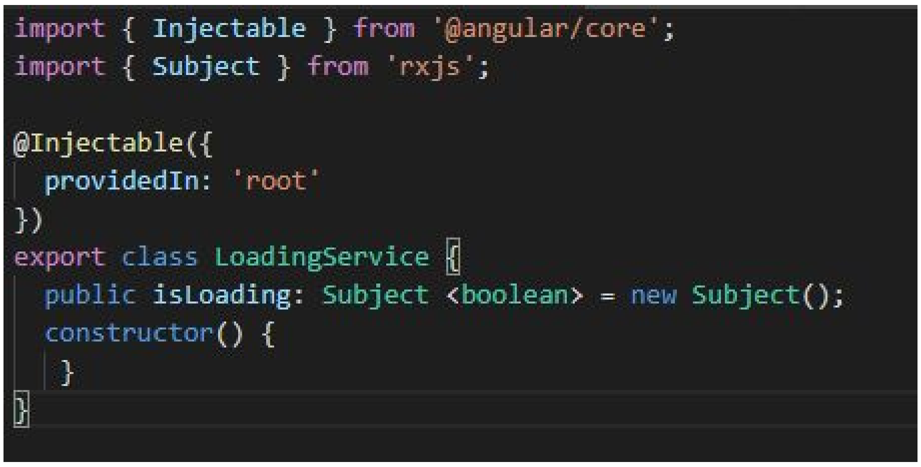


```
import { Injectable } from '@angular/core';
import {Observable} from 'rxjs';
import {AngularFirestore, AngularFirestoreDocument} from '@angular/fire/firestore';
@Injectable()
export class ChatroomService {

public chatrooms: Observable <any>

  constructor(private db:AngularFirestore) {
    this.chatrooms=db.collection('chatrooms').valueChanges();
  }
}
```

Screenshot 37: chatroom service

So as we said before in the chatroom components, here in the service we recieve all the data from the collection chatrooms, and we set all of that in an observable, this observable as said before is similar to a listener and it's also used to connect our service with the component, for example, we used the functionality Router, this router is an observable that every time a URL is created the observable also creates one.

Loading Service

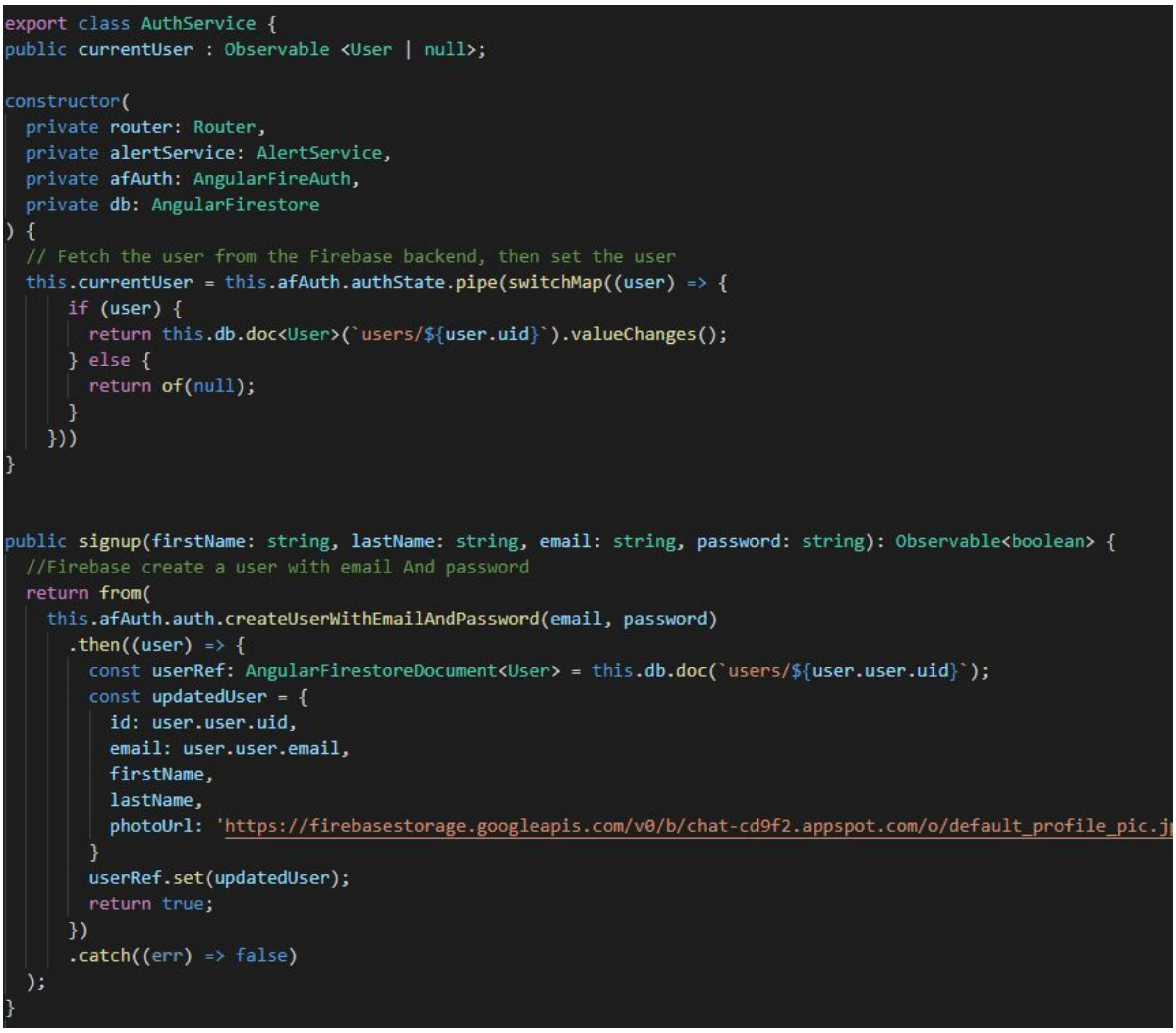


```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class LoadingService {
  public isLoading: Subject <boolean> = new Subject();
  constructor() {
   }
}
```

Screenshot 38: loading service

In this service we use the class Subject from rxjs again, this will be used in other components to check if the load has completed or not.

Auth Service

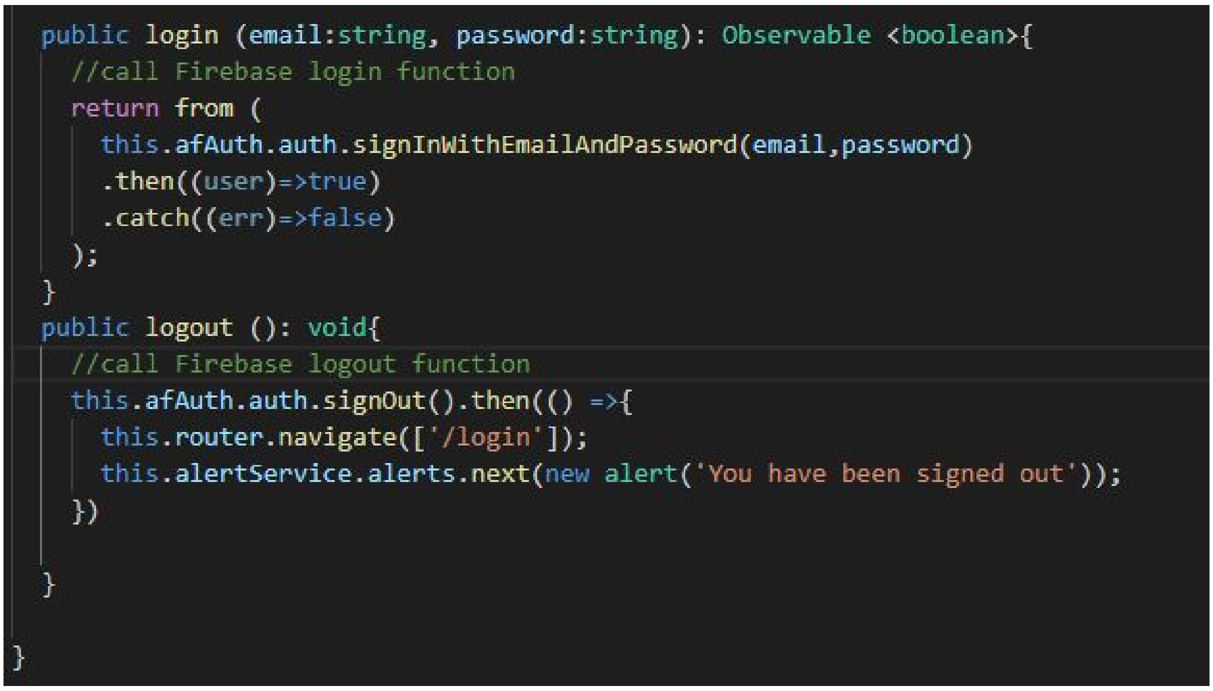


```
export class AuthService {
public currentUser : Observable <User | null>;

constructor(
  private router: Router,
  private alertService: AlertService,
  private afAuth: AngularFireAuth,
  private db: AngularFirestore
) {
  // Fetch the user from the Firebase backend, then set the user
  this.currentUser = this.afAuth.authState.pipe(switchMap((user) => {
      if (user) {
        return this.db.doc<User>(`users/${user.uid}`).valueChanges();
      } else {
        return of(null);
      }
    }))
}


public signup(firstName: string, lastName: string, email: string, password: string): Observable<boolean> {
  //Firebase create a user with email And password
  return from(
    this.afAuth.auth.createUserWithEmailAndPassword(email, password)
      .then((user) => {
        const userRef: AngularFirestoreDocument<User> = this.db.doc(`users/${user.user.uid}`);
        const updatedUser = {
          id: user.user.uid,
          email: user.user.email,
          firstName,
          lastName,
          photoUrl: 'https://firebasestorage.googleapis.com/v0/b/chat-cd9f2.appspot.com/o/default_profile_pic.j
        }
        userRef.set(updatedUser);
        return true;
      })
      .catch((err) => false)
  );
}
```

Screenshot 39: Auth service part I

In the auth service we will be creating the functionalities of everything related to our database, this being the signup, the login and the logout, let's start with the first line we set our currentUser, that if we remember is an string using a pipe, A pipe takes in data as input and transforms it to a desired output.

In the signup method we'll create our users into the database by using the method createUserWithEmailAndPassword from the library angularFire2 and we set every id,email with our class User (Explained in its own section).



```
  public login (email:string, password:string): Observable <boolean>{
    //call Firebase login function
    return from (
      this.afAuth.auth.signInWithEmailAndPassword(email,password)
      .then((user)=>true)
      .catch((err)=>false)
    );
  }
  public logout (): void{
    //call Firebase logout function
    this.afAuth.auth.signOut().then(() =>{
      this.router.navigate(['/login']);
      this.alertService.alerts.next(new alert('You have been signed out'));
    })

  }

}
```

Screenshot 40: auth service part II

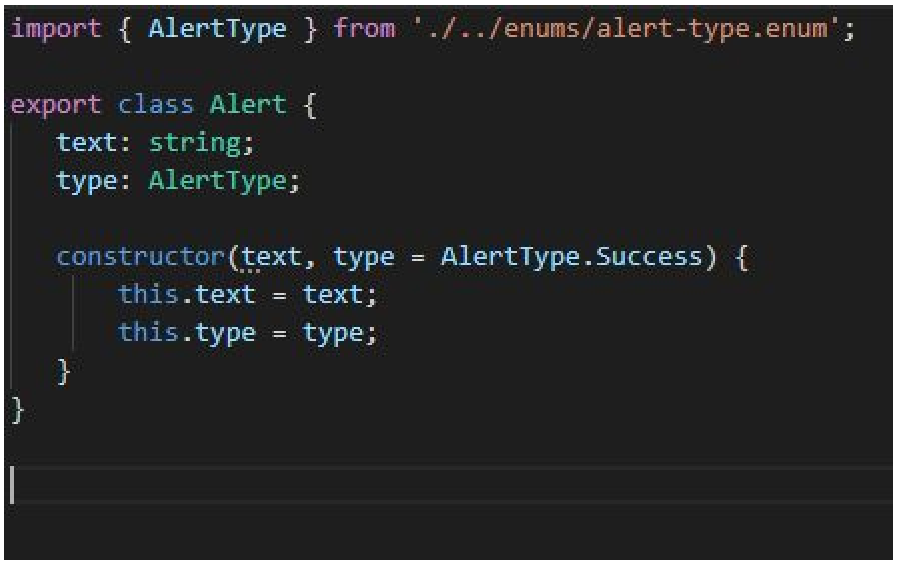
In the login method we receive the email and the password and we use the method signInWithEmailAndPassword to log us into the database.

And finally the logout method will sign out using the method signOut from the library angularFire2 and it will navigate into our login page using the route and will show a message using the alert service.

2.3 Classes

In this folder we will be storing all of our classes TS, this will be used to set all of our values into variables that will be used as well in the other elements.

Alert class

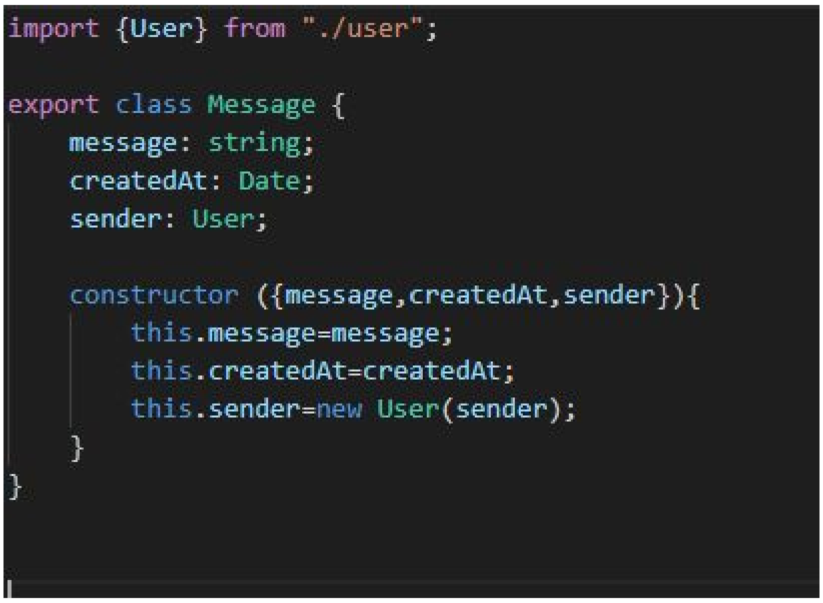


```
import { AlertType } from './../enums/alert-type.enum';

export class Alert {
  text: string;
  type: AlertType;

  constructor(text, type = AlertType.Success) {
    this.text = text;
    this.type = type;
  }
}
```

Screenshot 41: alert class

When an Alert it’s created it will need a text and a type, this type is extract from the alert-type enum and its set success as default, meaning that if we don't put a type it will show a green alert.

Message class

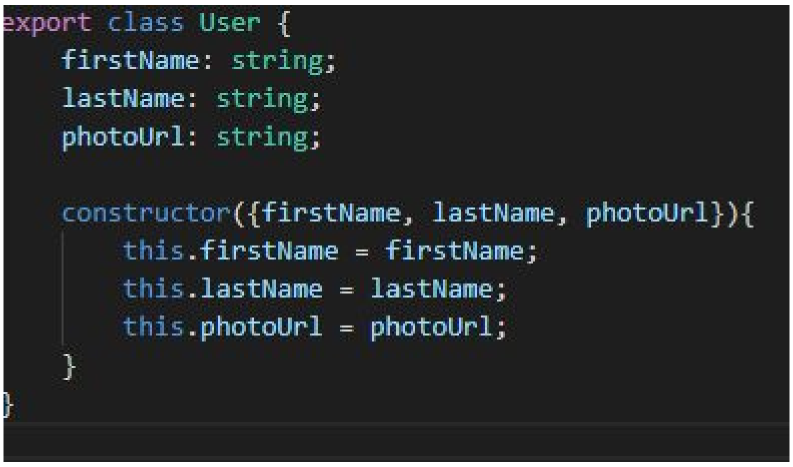


```
import {User} from "./user";

export class Message {
    message: string;
    createdAt: Date;
    sender: User;

    constructor ({message,createdAt,sender}){
        this.message=message;
        this.createdAt=createdAt;
        this.sender=new User(sender);
    }
}
```

Screenshot 42: Message class

This message creates a new message and put the user from the user class.

User class

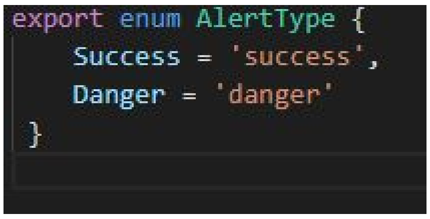


```
export class User {
    firstName: string;
    lastName: string;
    photoUrl: string;

    constructor({firstName, lastName, photoUrl}){
        this.firstName = firstName;
        this.lastName = lastName;
        this.photoUrl = photoUrl;
    }
}
```

Screenshot 43: user class

The user class creates a new user, this class will be used in a lot of components to create its users.

2.4 Enums

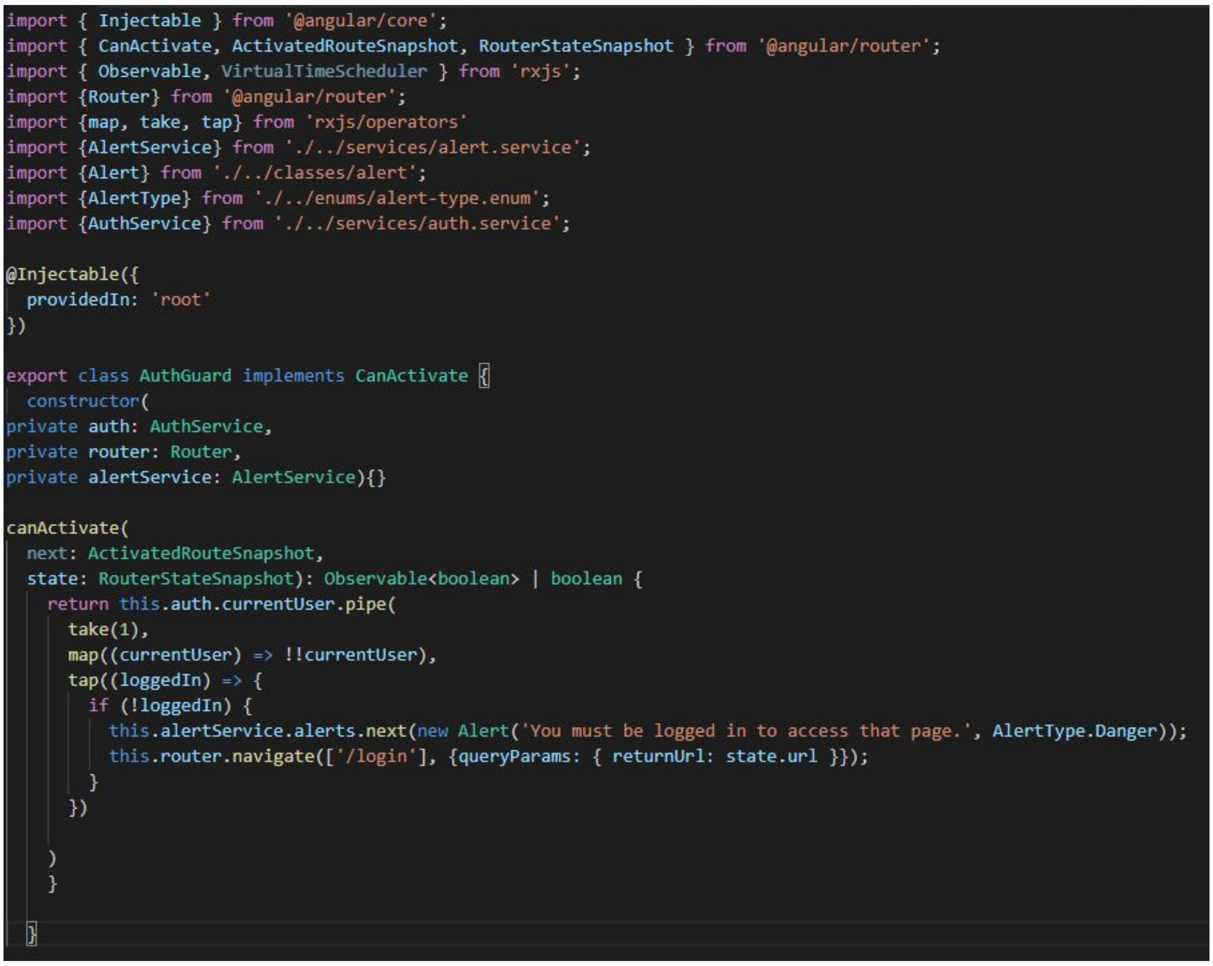


```
export enum AlertType {
    Success = 'success',
    Danger = 'danger'
 }
```

Screenshot 44: alert type

In this enum we will be setting the values of the alert, if its danger the alert will appear red, and if it’s success it will appear green.

2.5 Guards



```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
import { Observable, VirtualTimeScheduler } from 'rxjs';
import {Router} from '@angular/router';
import {map, take, tap} from 'rxjs/operators'
import {AlertService} from './../services/alert.service';
import {Alert} from './../classes/alert';
import {AlertType} from './../enums/alert-type.enum';
import {AuthService} from './../services/auth.service';

@Injectable({
  providedIn: 'root'
})

export class AuthGuard implements CanActivate {
  constructor(
private auth: AuthService,
private router: Router,
private alertService: AlertService){}

canActivate(
  next: ActivatedRouteSnapshot,
  state: RouterStateSnapshot): Observable<boolean> | boolean {
    return this.auth.currentUser.pipe(
      take(1),
      map((currentUser) => !!currentUser),
      tap((loggedIn) => {
        if (!loggedIn) {
          this.alertService.alerts.next(new Alert('You must be logged in to access that page.', AlertType.Danger));
          this.router.navigate(['/login'], {queryParams: { returnUrl: state.url }});
        }
      })

    )
    }


  }
```

Screenshot 45: auth.guard

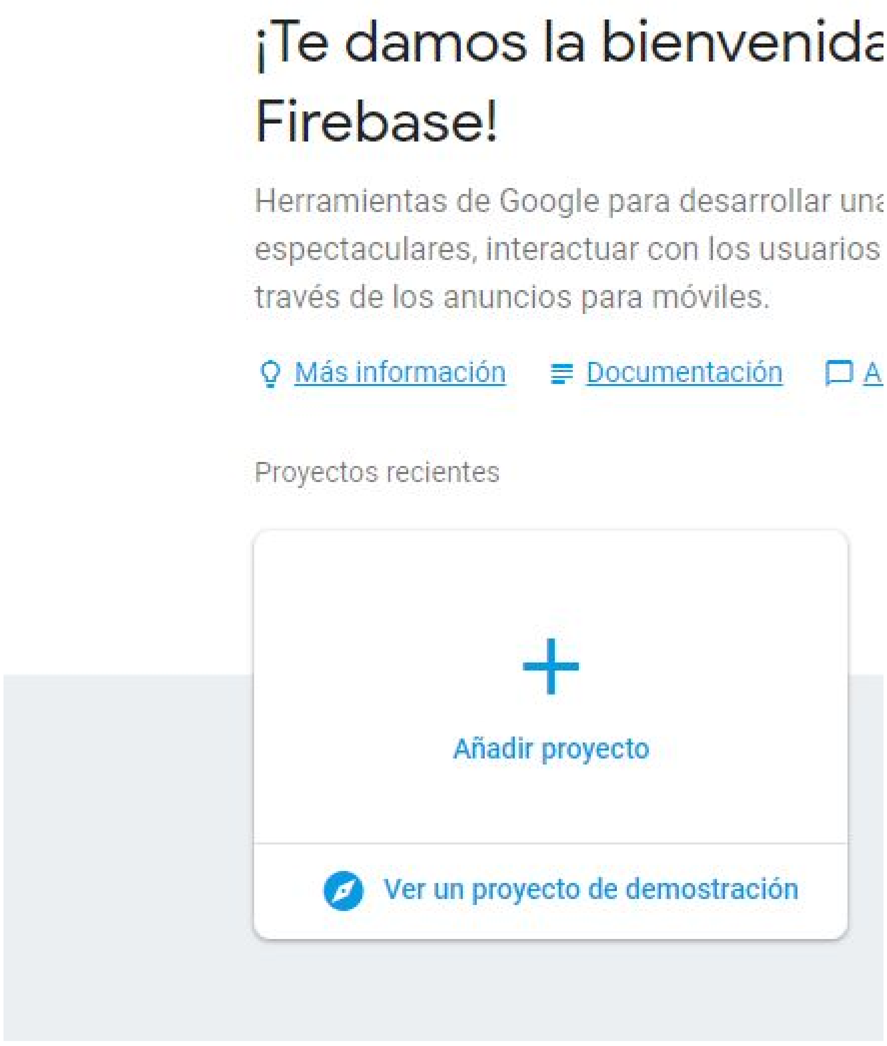
We already explained what a guard is, but resuming, guards are interfaces which can tell the router whether or not it should allow navigation to a requested route.

if the user is not logged in and tries to go to the chat it will redirect the user to the login page and it will show an alert .

2.6 Firebase

In this section we will be showing step by step how to create a firebase account and how to use it.

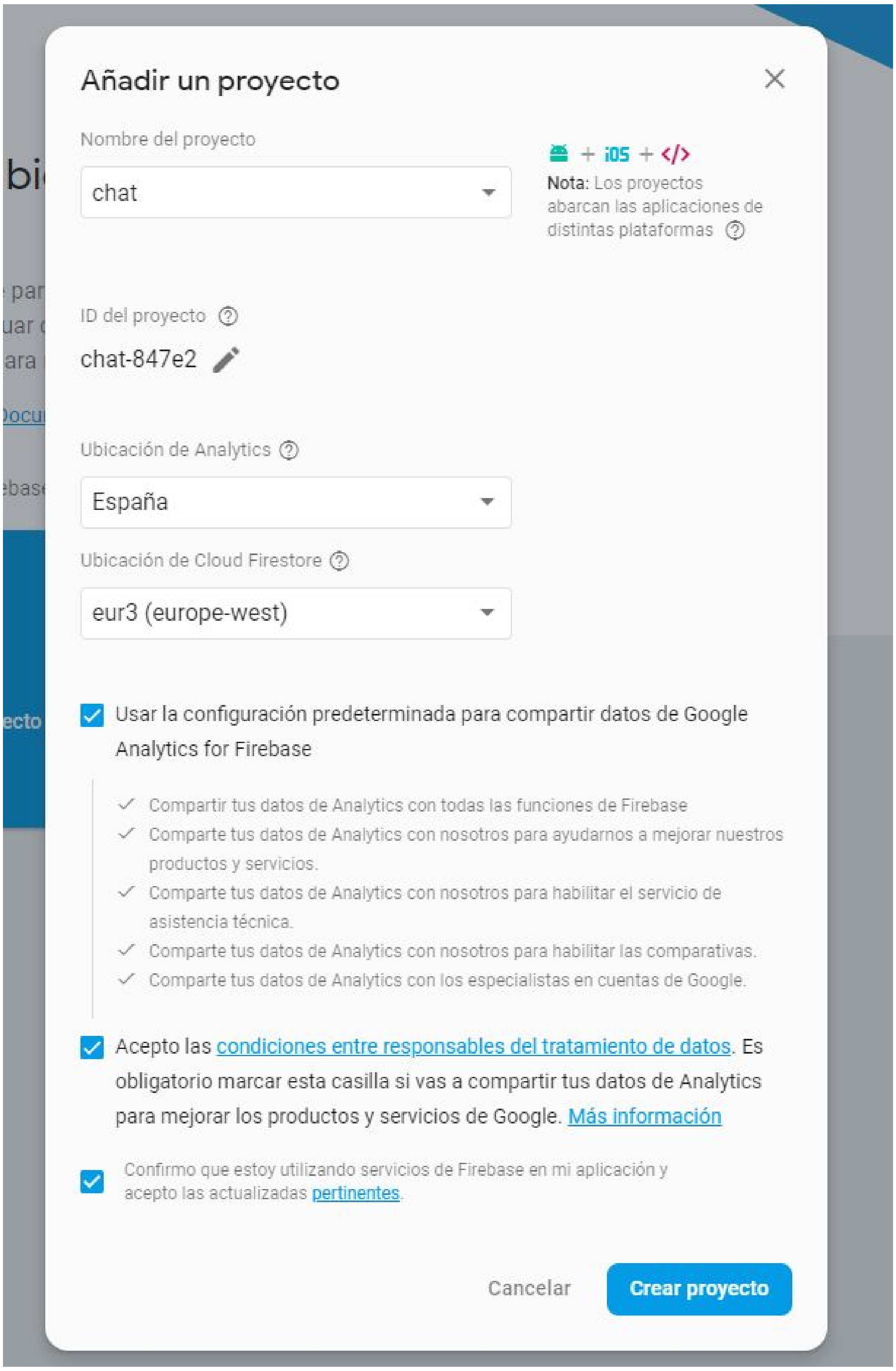
Create a project in firebase:



¡Te damos la bienvenida
Firebase!

Herramientas de Google para desarrollar una
espectaculares, interactuar con los usuarios
través de los anuncios para móviles.

Screenshot 46: Google firebase


Añadir un proyecto

Nombre del proyecto: chat

Nota: Los proyectos abarcan las aplicaciones de distintas plataformas

ID del proyecto: chat-847e2

Ubicación de Analytics: España

Ubicación de Cloud Firestore: eur3 (europe-west)

✓ Usar la configuración predeterminada para compartir datos de Google Analytics for Firebase

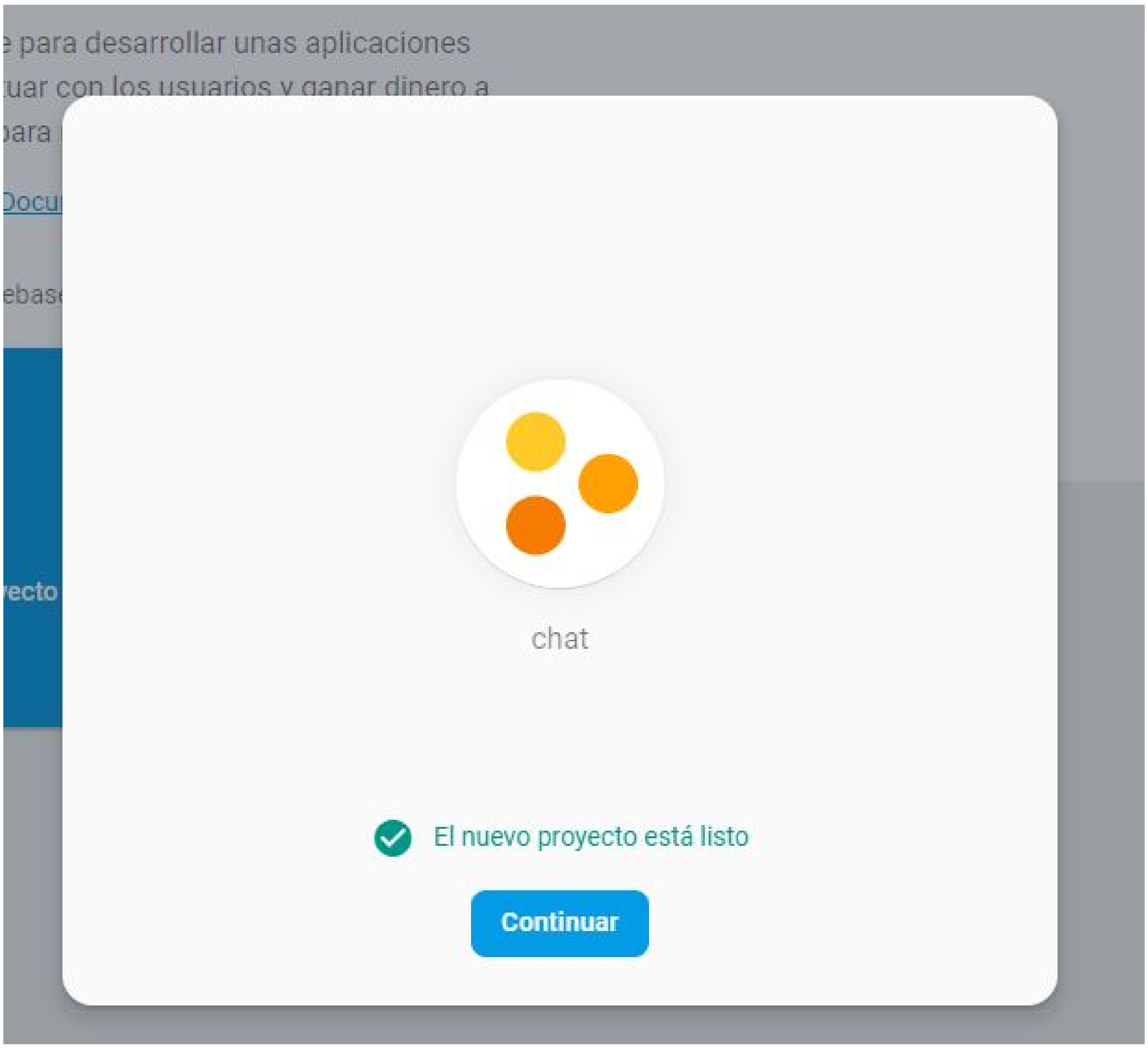
- ✓ Compartir tus datos de Analytics con todas las funciones de Firebase
- ✓ Comparte tus datos de Analytics con nosotros para ayudarnos a mejorar nuestros productos y servicios.
- ✓ Comparte tus datos de Analytics con nosotros para habilitar el servicio de asistencia técnica.
- ✓ Comparte tus datos de Analytics con nosotros para habilitar las comparativas.
- ✓ Comparte tus datos de Analytics con los especialistas en cuentas de Google.

✓ Acepto las condiciones entre responsables del tratamiento de datos. Es obligatorio marcar esta casilla si vas a compartir tus datos de Analytics para mejorar los productos y servicios de Google. Más información

✓ Confirmo que estoy utilizando servicios de Firebase en mi aplicación y acepto las actualizadas pertinentes.

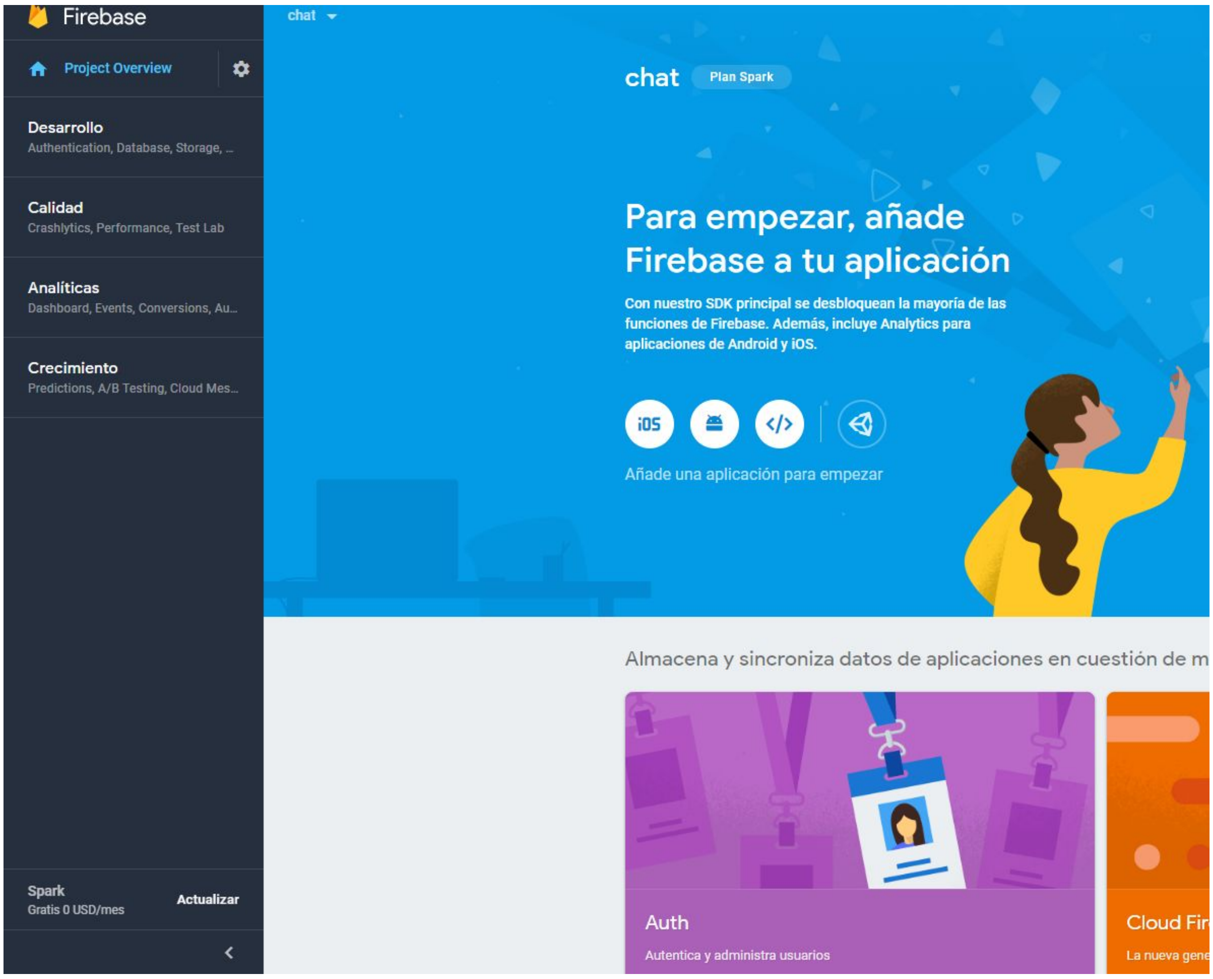
Screenshot 47: firebase new project

We need to change the location of analytics to spain and the location for cloud firestore to eu3 like in the screenshot.


e para desarrollar unas aplicaciones
uar con los usuarios y ganar dinero a
para
Docu
ebas
ecto
chat
El nuevo proyecto está listo
Continuar

Screenshot 48: chat ready

After the project is created we will instantly get redirected to the home page of our project.


Firebase
chat
Project Overview
Desarrollo
Authentication, Database, Storage, ...
Calidad
Crashlytics, Performance, Test Lab
Analíticas
Dashboard, Events, Conversions, Au...
Crecimiento
Predictions, A/B Testing, Cloud Mes...
Spark
Gratis 0 USD/mes
Actualizar
chat
Plan Spark

Para empezar, añade Firebase a tu aplicación

Con nuestro SDK principal se desbloquean la mayoría de las funciones de Firebase. Además, incluye Analytics para aplicaciones de Android y iOS.

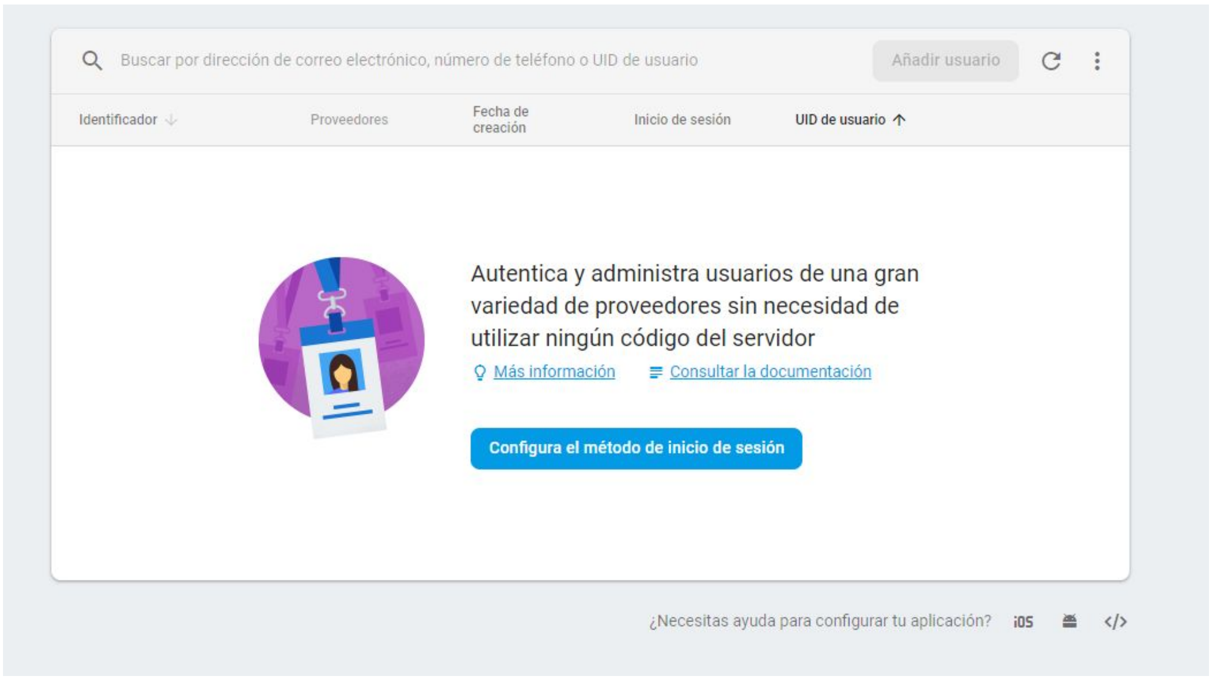
iOS
</>
Añade una aplicación para empezar

Almacena y sincroniza datos de aplicaciones en cuestión de m

Auth
Autentica y administra usuarios
Cloud Fir
La nueva gene

Screenshot 49: firebase main page

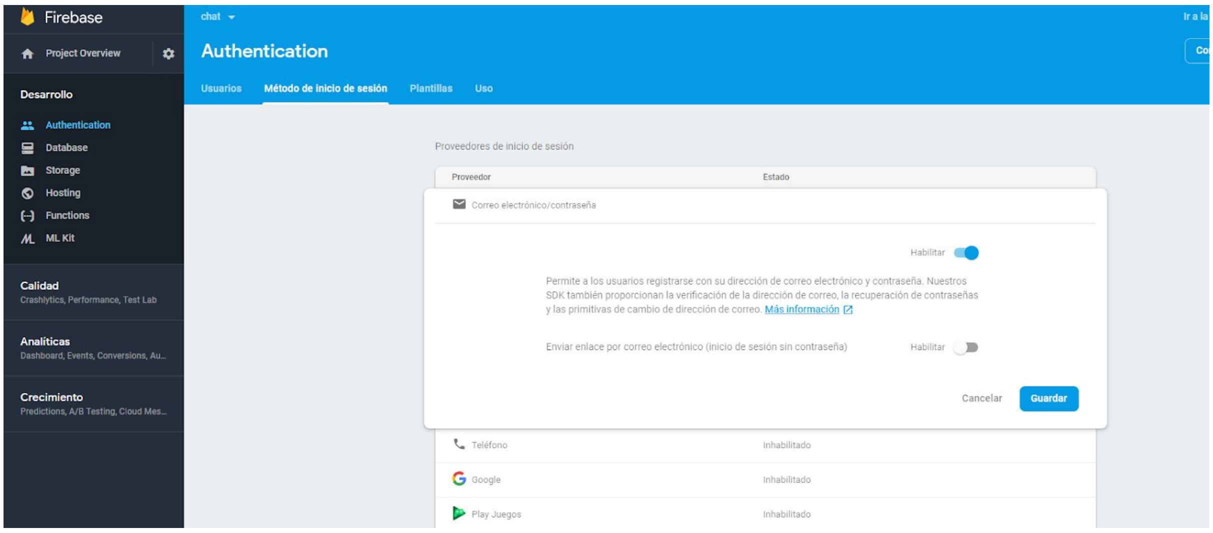
Now we need to set the authentication parameters to do that we need to click on Auth (purple box showed in the screenshot).


Buscar por dirección de correo electrónico, número de teléfono o UID de usuario
Añadir usuario
Identificador
Proveedores
Fecha de creación
Inicio de sesión
UID de usuario
Más información
Consultar la documentación
Configura el método de inicio de sesión
¿Necesitas ayuda para configurar tu aplicación?
iOS

Autentica y administra usuarios de una gran variedad de proveedores sin necesidad de utilizar ningún código del servidor

Screenshot 50: auth firebase

Now we should click in "Configura el método de inicio de sesión", and it will appear the next screenshot, in that we will need to enable the Email and password option, but only the first slider, and click save.



Proveedores de inicio de sesión

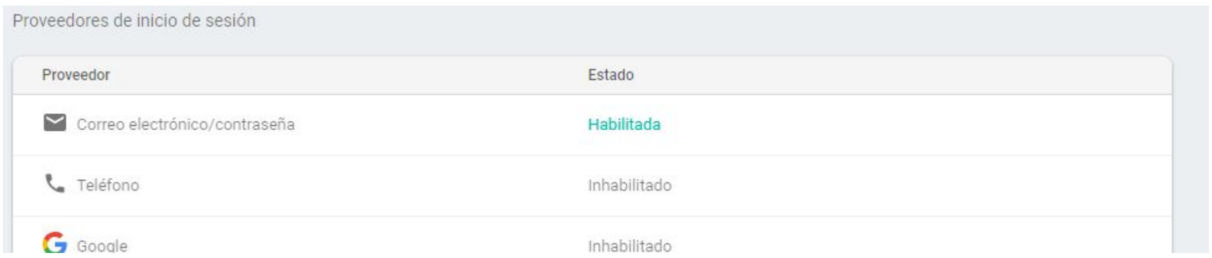
Proveedor	Estado
Correo electrónico/contraseña	
Teléfono	Inhabilitado
Google	Inhabilitado
Play Juegos	Inhabilitado

Habilitar ●

Permite a los usuarios registrarse con su dirección de correo electrónico y contraseña. Nuestros SDK también proporcionan la verificación de la dirección de correo, la recuperación de contraseñas y las primitivas de cambio de dirección de correo. Más información

Enviar enlace por correo electrónico (inicio de sesión sin contraseña) Habilitar ○

Screenshot 51: email and password auth

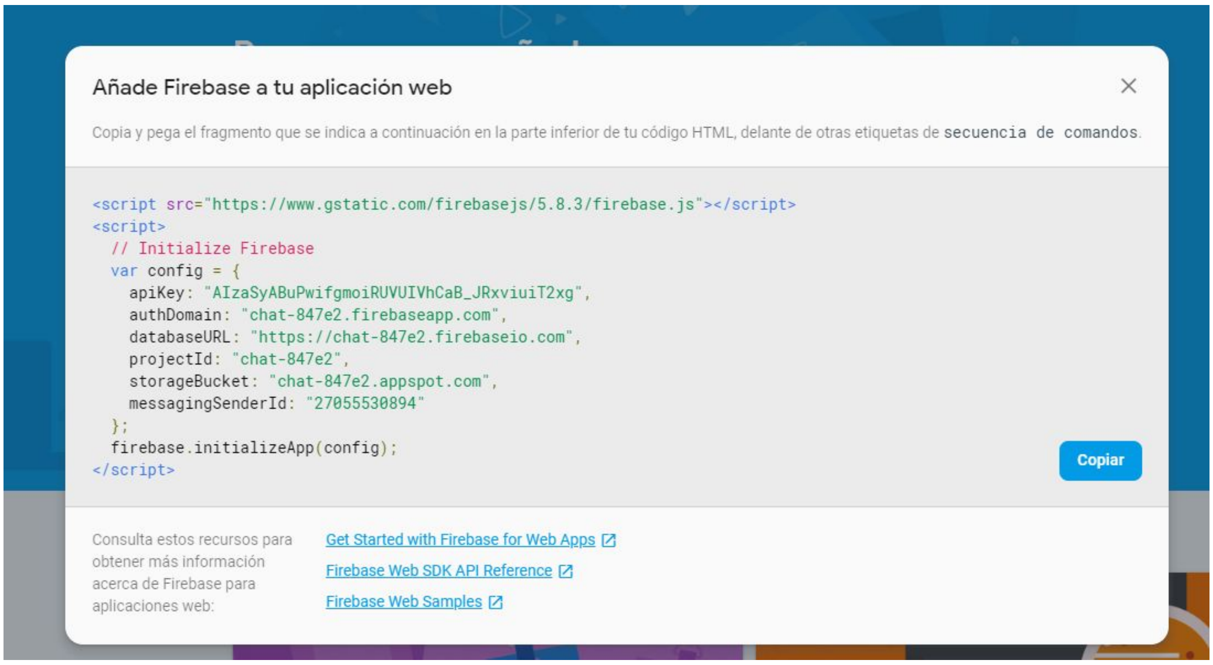


Proveedores de inicio de sesión

Proveedor	Estado
Correo electrónico/contraseña	Habilitada
Teléfono	Inhabilitado
Google	Inhabilitado

Screenshot 52: email and password auth enabled

Now we need to generate the code that will be introduced in our environment ts, to do that we need to go back to the main screen and click on the </> symbol.



Añade Firebase a tu aplicación web

Copia y pega el fragmento que se indica a continuación en la parte inferior de tu código HTML, delante de otras etiquetas de `secuencia de comandos`.

```
<script src="https://www.gstatic.com/firebasejs/5.8.3/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "AIzaSyABuPwifgmoiRUVUIVhCaB_JRxviuiT2xg",
    authDomain: "chat-847e2.firebaseapp.com",
    databaseURL: "https://chat-847e2.firebaseio.com",
    projectId: "chat-847e2",
    storageBucket: "chat-847e2.appspot.com",
    messagingSenderId: "27055530894"
  };
  firebase.initializeApp(config);
</script>
```

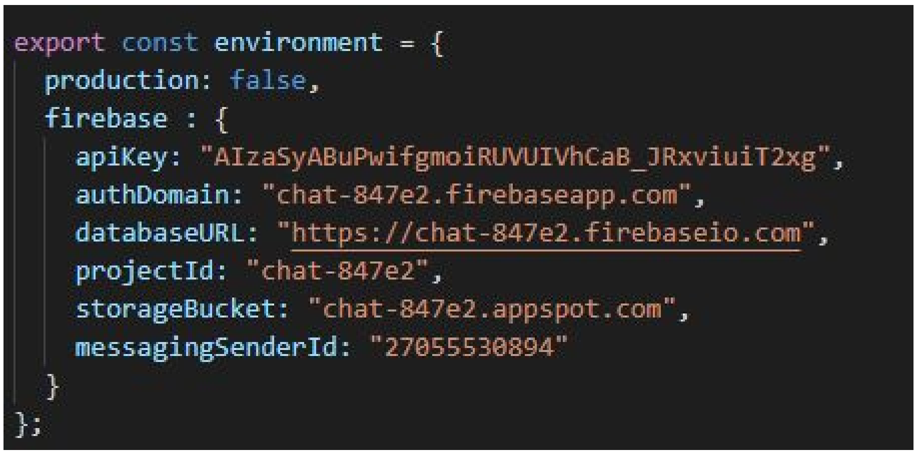
Consulta estos recursos para obtener más información acerca de Firebase para aplicaciones web:

Get Started with Firebase for Web Apps

Firebase Web SDK API Reference

Firebase Web Samples

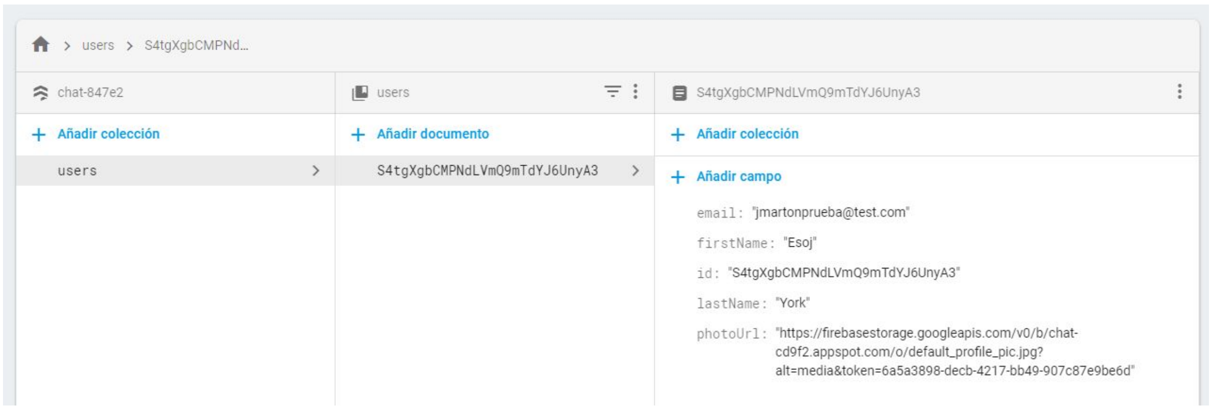
Screenshot 53: add firebase to our chat



```
export const environment = {
  production: false,
  firebase : {
    apiKey: "AIzaSyABuPwifgmoiRUVUIVhCaB_JRxviuiT2xg",
    authDomain: "chat-847e2.firebaseapp.com",
    databaseURL: "https://chat-847e2.firebaseio.com",
    projectId: "chat-847e2",
    storageBucket: "chat-847e2.appspot.com",
    messagingSenderId: "27055530894"
  }
};
```

Screenshot 54: environment typeScript

We need to copy the config variable and post it in environment like its shown on the screenshot.



users > S4tgXgbCMPNd...

chat-847e2 | users | S4tgXgbCMPNdLVmQ9mTdYJ6UnyA3

+ Añadir colección | + Añadir documento | + Añadir colección

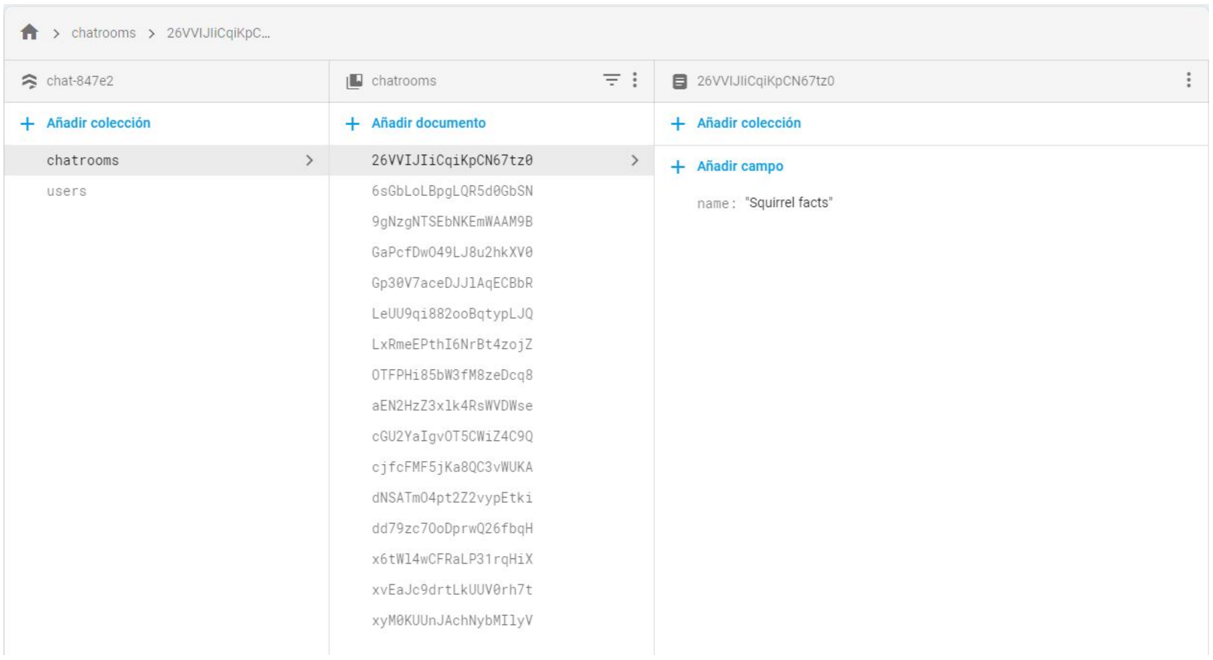
users | S4tgXgbCMPNdLVmQ9mTdYJ6UnyA3

+ Añadir campo

```
email: "jmartonprueba@test.com"
firstName: "Esoj"
id: "S4tgXgbCMPNdLVmQ9mTdYJ6UnyA3"
lastName: "York"
photoUrl: "https://firebasestorage.googleapis.com/v0/b/chat-
          cd9f2.appspot.com/o/default_profile_pic.jpg?
          alt=media&token=6a5a3898-decb-4217-bb49-907c87e9be6d"
```

Screenshot 55: users result

That is what firebase shows us when a user is created, now we will be adding the chatrooms collection, we need to click in add collection and put the name chatrooms, and then we will be adding documents, this is the result:



chatrooms > 26VVIJIiCqiKpC...

chat-847e2 | chatrooms | 26VVIJIiCqiKpCN67tz0

+ Añadir colección | + Añadir documento | + Añadir colección

chatrooms
users

```
26VVIJIiCqiKpCN67tz0
6sGbLoLBpgLQR5d0GbSN
9gNzgNTSEbNKEmWAAM9B
GaPcfDw049LJ8u2hkXV0
Gp30V7aceDJJlAqECBbR
LeUU9qi882ooBqtypLJQ
LxRmeEPthI6NrBt4zojZ
OTFPHi85bW3fM8zeDcq8
aEN2HzZ3xlk4RsWVDWse
cGU2YaIgvOT5CWiZ4C9Q
cjfcFMF5jKa8QC3vWUKA
dNSATmO4pt2Z2vypEtki
dd79zc7OoDprwQ26fbqH
x6tWl4wCFRaLP31rqHiX
xvEaJc9drtLkUUV0rh7t
xyM0KUUnJAchNybMIlyV
```

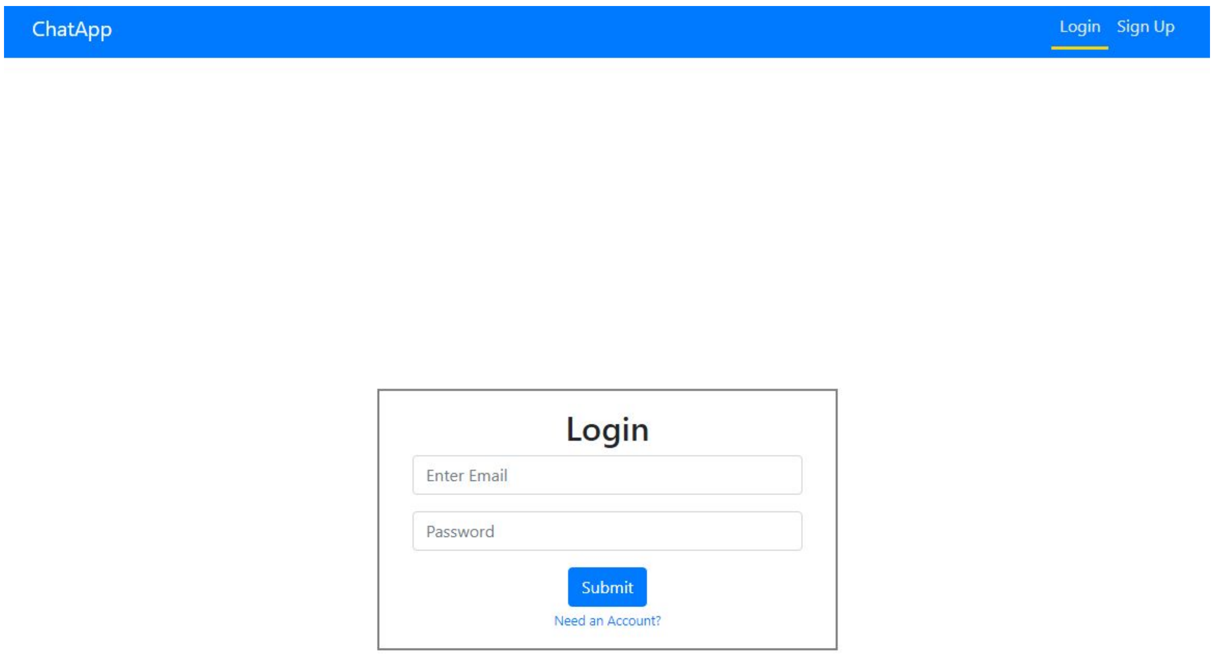
+ Añadir campo

```
name: "Squirrel facts"
```

Screenshot 56: chatrooms result

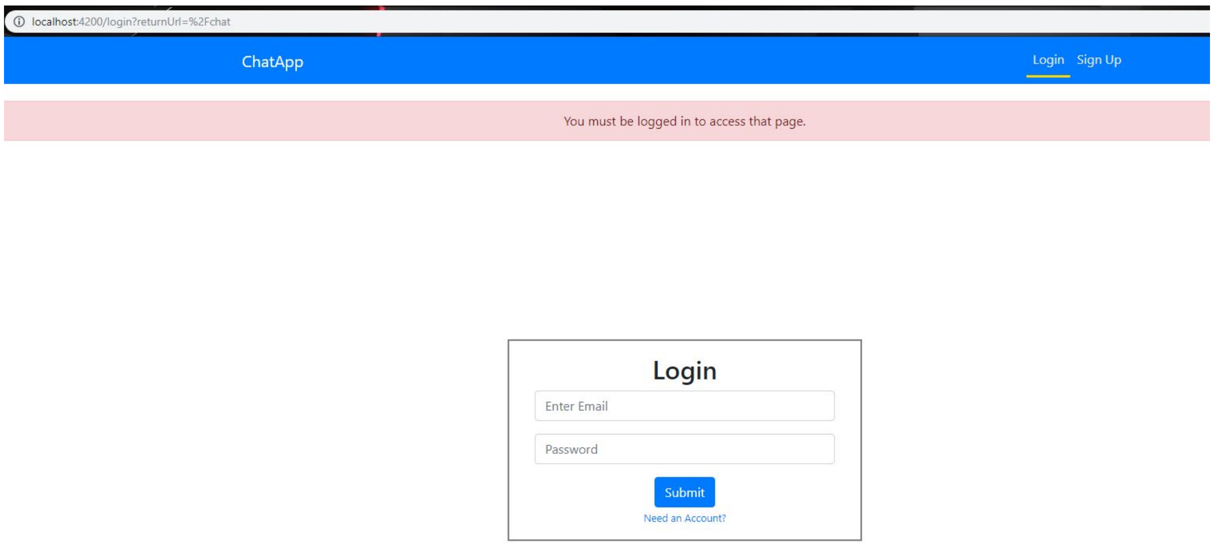
3. User's manual

We open the page we will need to register to have access to the chatroom page, we are located right now in login so we will need to click in the signup tab on the navbar or click in the link below the submit button.



Screenshot 57: login page

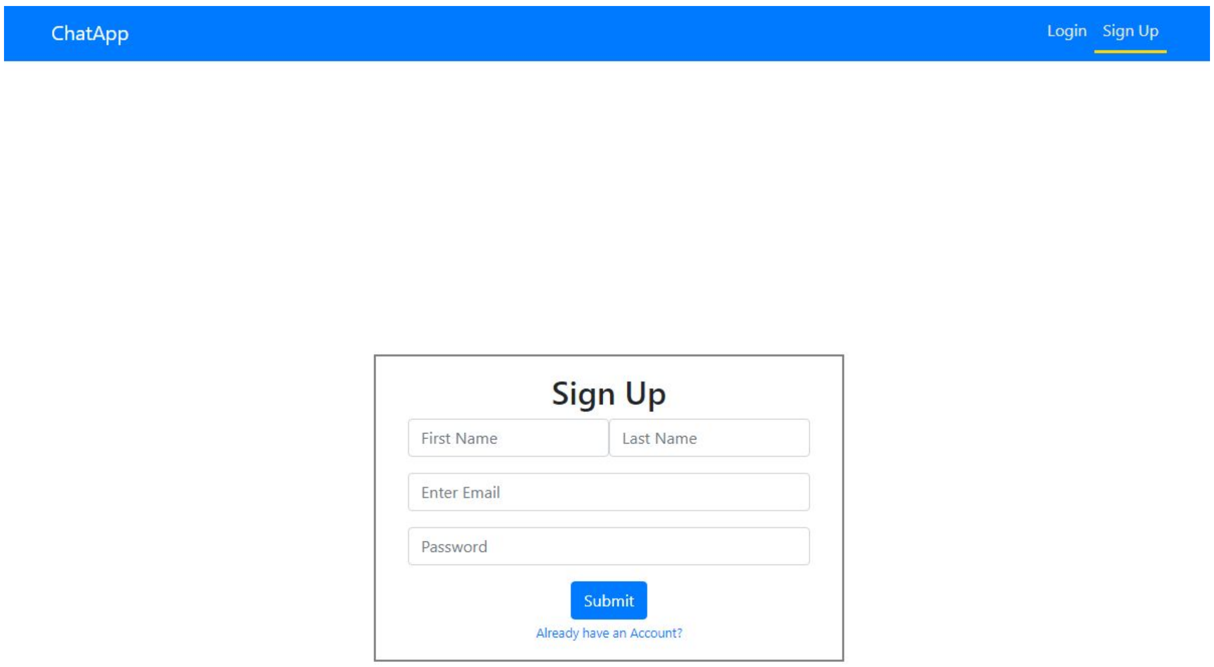
When we click on those 2 things the signup page will pop up, but to prove the functionality of the page if we try to enter the chatroom page by changing the route, this will happen.


localhost:4200/login?returnUrl=%2Fchat

You must be logged in to access that page.

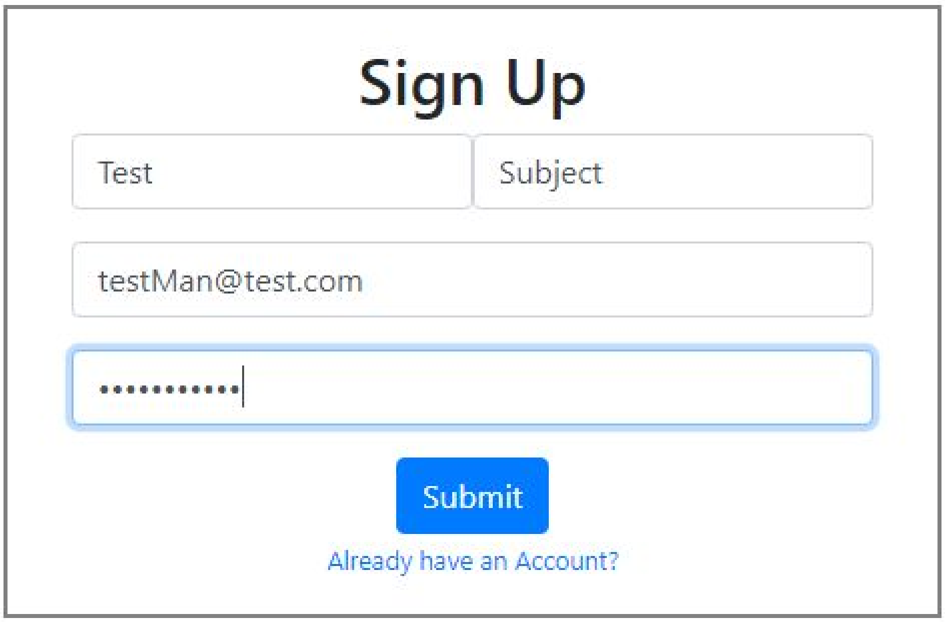
Screenshot 58: Guard page

So here we have the signup page:



Screenshot 59: signup page

Now we will add a test user to see how this works.



Test

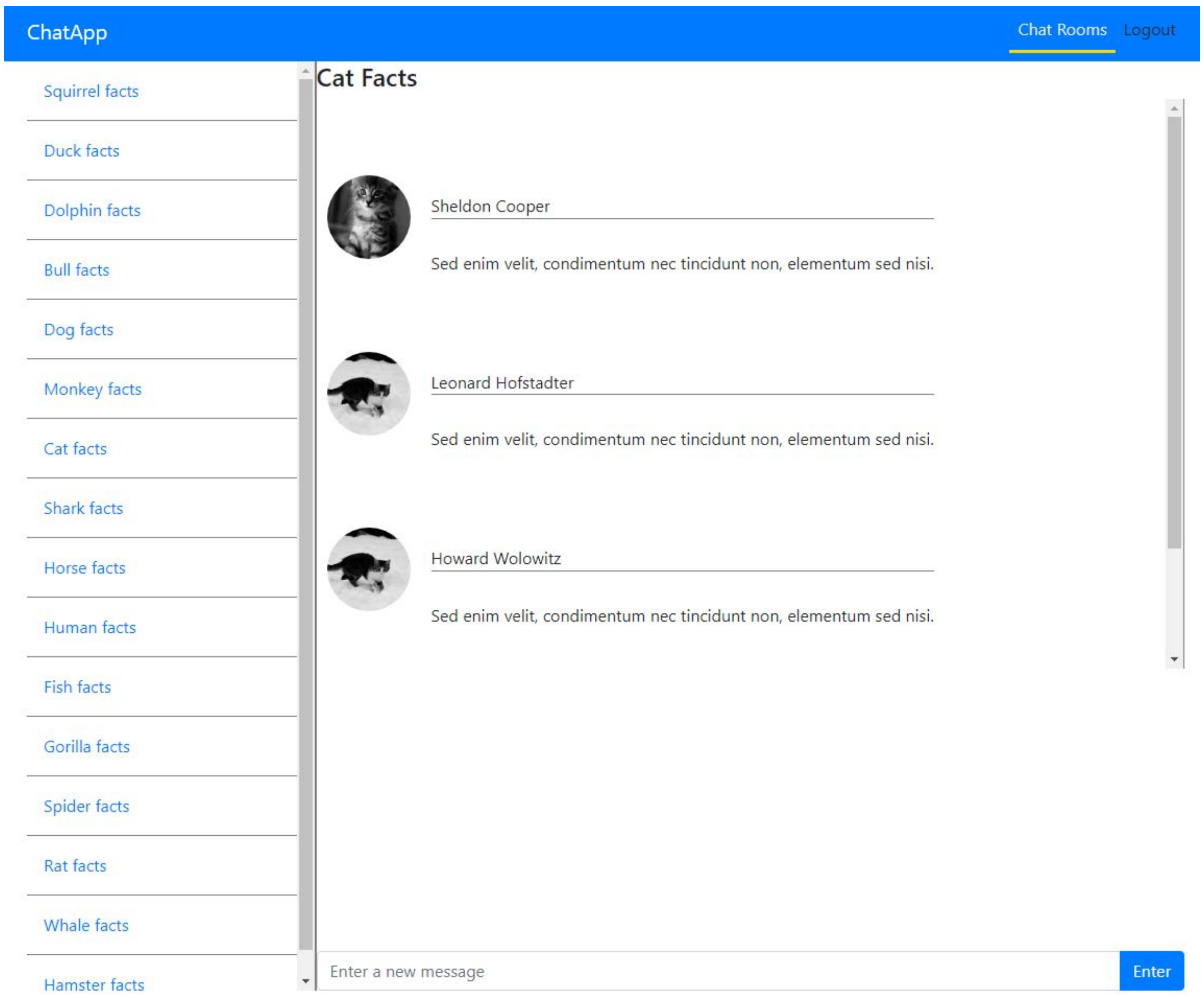
Subject

testMan@test.com

•••••••••••

Screenshot 60: test user

When we click on submit it will pop up the chat window, and our new user will be created in the database as well.



Cat Facts

Sheldon Cooper

Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.

Leonard Hofstadter

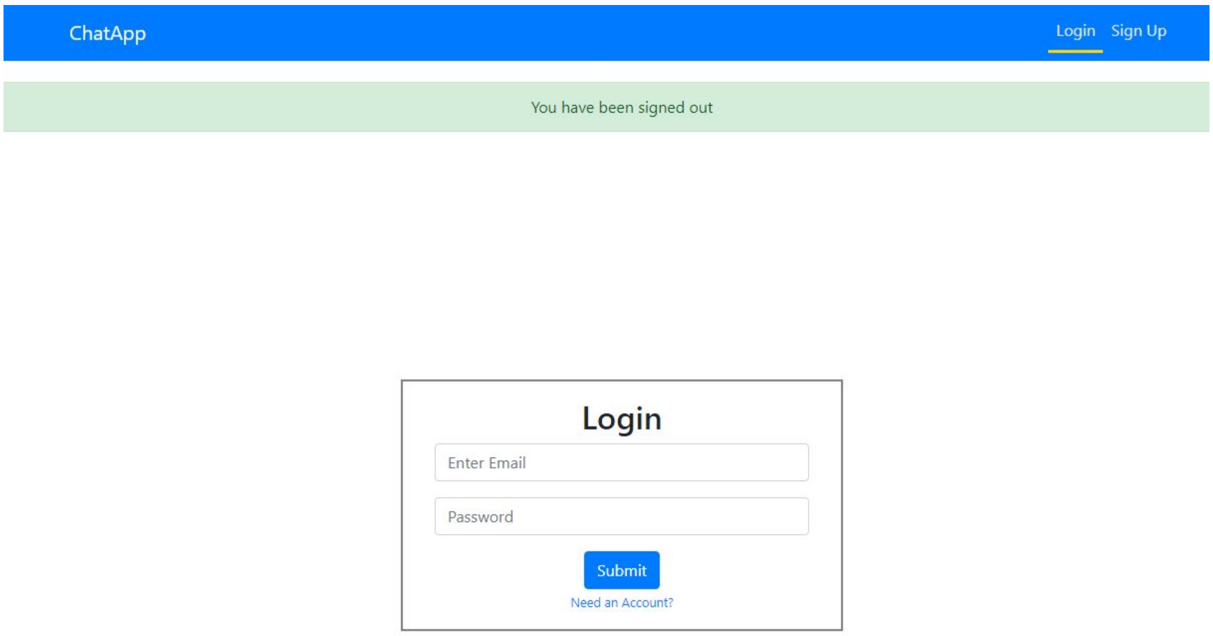
Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.

Howard Wolowitz

Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.

Screenshot 61: Chat app

Now we will click on logout and we will return to the login page

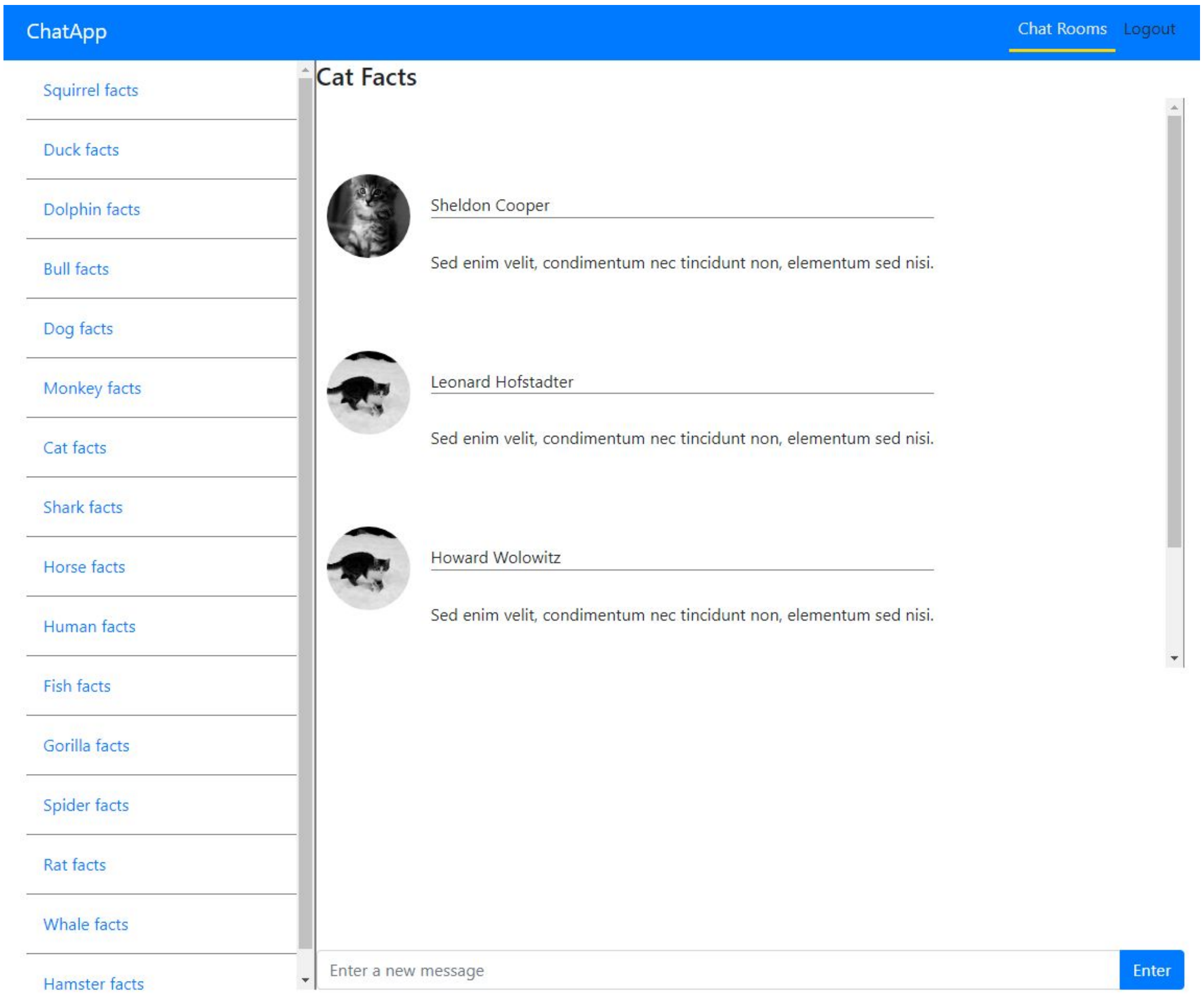


You have been signed out

Screenshot 62: login page sign out

Now we will try to log in with our user

Email: testMan@test.com
Password: thisisatest


ChatApp

Chat Rooms Logout

Squirrel facts

Duck facts

Dolphin facts

Bull facts

Dog facts

Monkey facts

Cat facts

Shark facts

Horse facts

Human facts

Fish facts

Gorilla facts

Spider facts

Rat facts

Whale facts

Hamster facts

Cat Facts

Sheldon Cooper

Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.

Leonard Hofstadter

Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.

Howard Wolowitz

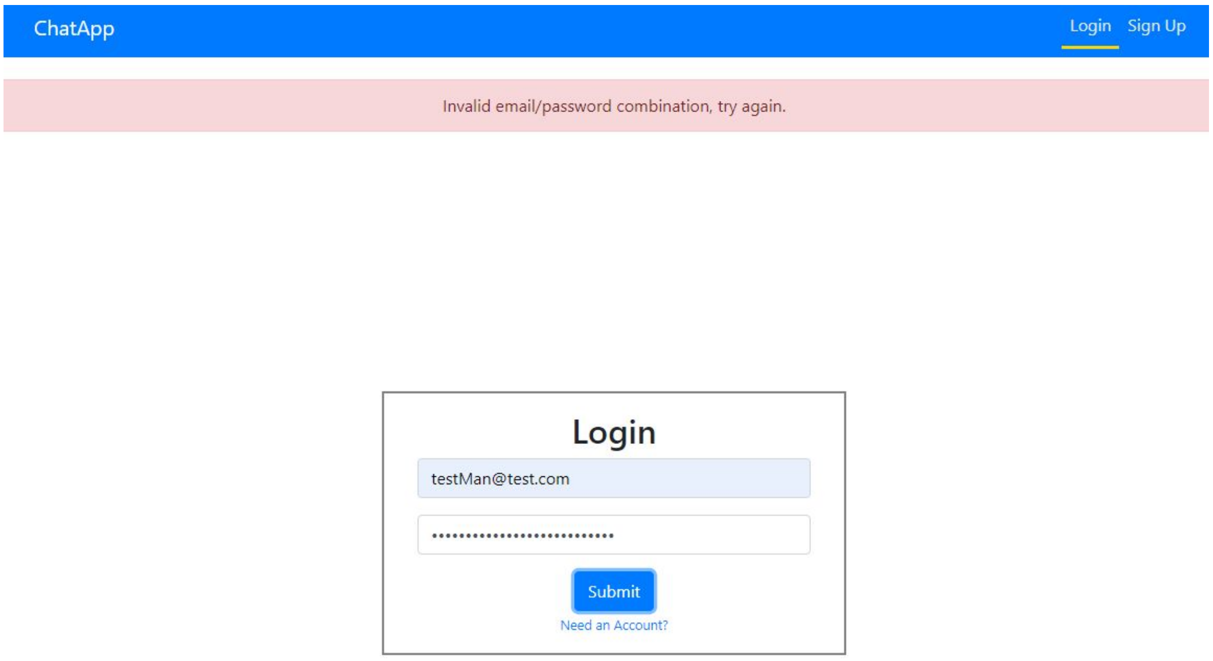
Sed enim velit, condimentum nec tincidunt non, elementum sed nisi.

Enter a new message

Enter

Screenshot 63: chat page part II

If we put a wrong password this will happen:

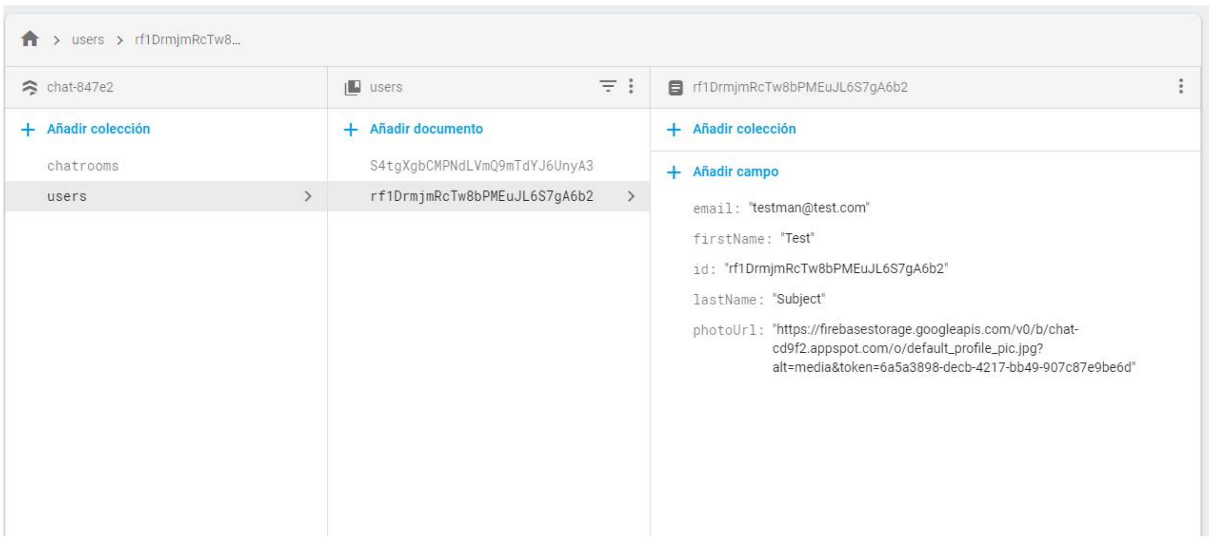


Invalid email/password combination, try again.

testMan@test.com

••••••••••••••••••••••••••••

Screenshot 64: login invalid



chatrooms
users

S4tgXgbCMPNdLVmQ9mTdYJ6UnyA3
rf1DrmjmRcTw8bPMEuJL6S7gA6b2

email: "testman@test.com"
firstName: "Test"
id: "rf1DrmjmRcTw8bPMEuJL6S7gA6b2"
lastName: "Subject"
photoUrl: "https://firebasestorage.googleapis.com/v0/b/chat-cd9f2.appspot.com/o/default_profile_pic.jpg?alt=media&token=6a5a3898-decb-4217-bb49-907c87e9be6d"

Screenshot 65: firebase new user

4. Conclusions

We had a lot of issues adapting to angular and understanding how angular worked, we had issues with the node modules, the error was that it couldn't find @angular/core, when in other components could, we had some issues with the guard, it didn't block the route at first, and most importantly we had issues with the time, the lack of time made that we couldn't fully develop the project and I became ill in a wrong time as well, but we will work on it in a future because it's a really interesting project and it was fun to learn something that is valuable for the future.

Also i really liked using adobe XD it’s a very helpful tool that lets you interact with the relations you choose.

5. References

Angular documentation : https://angular.io/
RxJs: https://blog.angulartraining.com/rxjs-subjects-a-tutorial-4dcce0e9637f
Adobe XD: https://www.youtube.com/watch?v=Yt2troF-Eyc&feature=youtu.be
Observable: http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html
w3schools: https://www.w3schools.com/
firebase: https://firebase.google.com/docs/guides/
stackoverflow:
https://stackoverflow.com/questions/1143796/remove-a-file-from-a-git-repository-without-deleting-it-from-the-local-filesyste