Looking For Primary Head Of School & Teachers - Computers, Subject Lead, Homeroom & Music

Ope

Inventure Academy

ANGULAR / ASP.NET CORE

Secure Angular Site using JWT Authentication with ASP.NET Core Web API

by February 28, 2022

In this tutorial, we will learn how to secure Angular site using JWT Authentication with ASP.NET Core Web API.

The angular site will include a login, signup, tasks and profile screens, the site will be secured using JWT Authentication with Access and Refresh tokens built using ASP.NET Core Web API in .NET 6.

Complete Work Management Suite

Ope

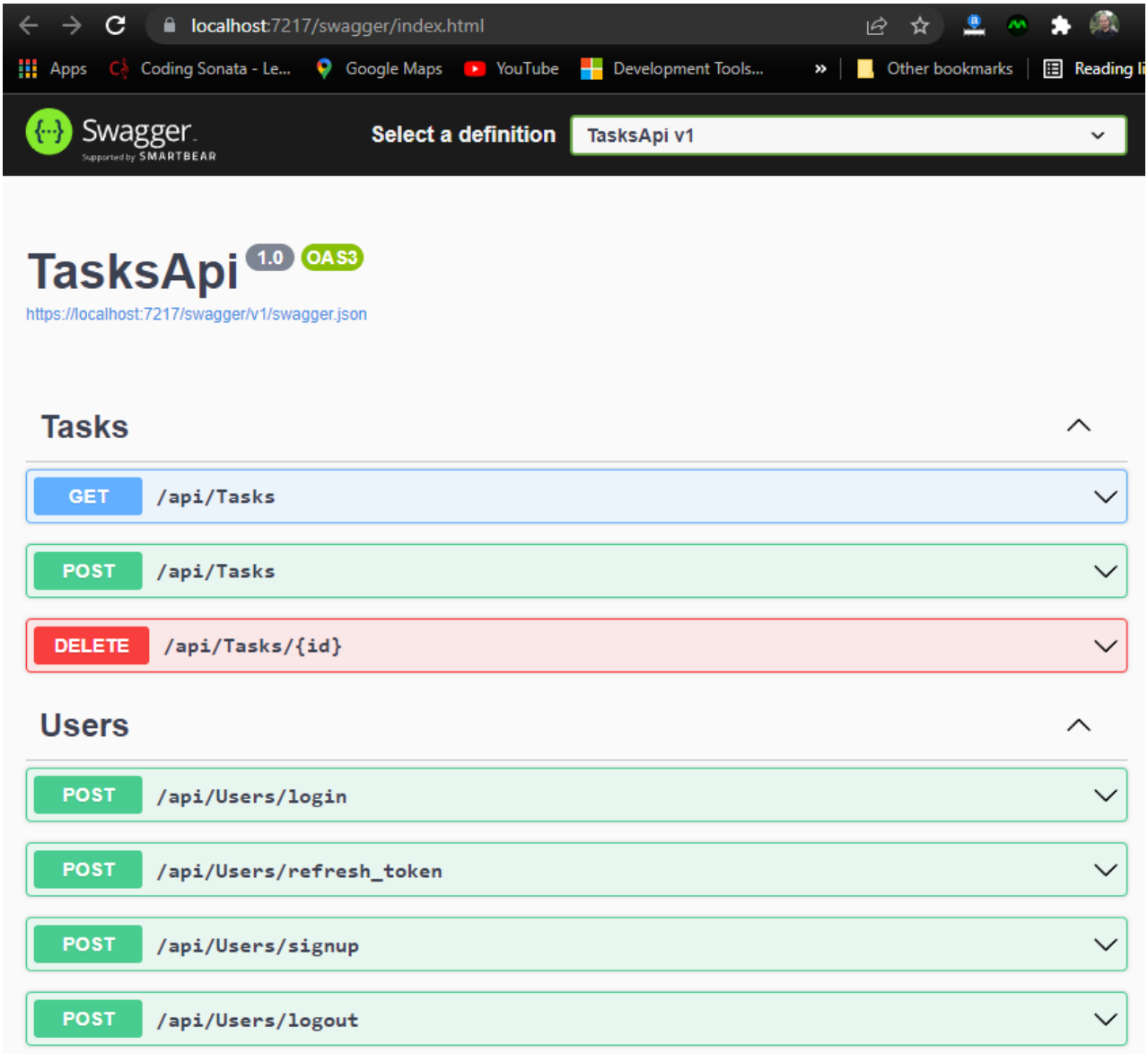
ProHance Workflow

We will use one of my previous tutorials where we already built a Tasks Database using SQL Server Express and connected it with ASP.NET Core Web API through Entity Framework Core. The API includes endpoints with GET, POST, PUT, DELETE http methods.

You can follow my previous tutorial Apply JWT Access Tokens and Refresh Tokens in ASP .NET Core Web API 6 to fully understand how you can secure your APIs with JWT, but for this tutorial we will only need the APIs to be up-and-running and connected to the Tasks Database.

So just complete the database preparation part then clone the final project TasksApi from my GitHub account and continue with this tutorial.

Next, run the TasksAPI project, that you have just cloned, you should be getting the following Swagger page:



Now you have the API part ready, let's start building the front-end part using Angular 2+.

Starting the tutorial to Secure Angular Site using JWT Authentication

Angular App

Angular is a front-end TypeScript framework that enables you to build cutting-edge mobile and desktop web applications in the form of SPA (Single-Page Applications)

As of writing this tutorial, the latest version of Angular is 13. You can check this page to review and check for any updates on Angular.

ML-SQL Server

GitHub/MindsDB: In-Database Machine Learning

MindsDB

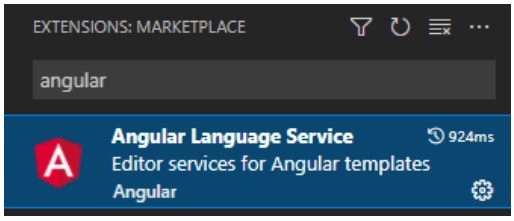
Lear

Before developing our Angular site, we need to setup our development environment with the prerequisites of Angular.

Visual Studio Code

This is the most popular and best IDE for developing front-end web apps, including Angular. It has everything you need to crush your way into building wonderful apps.

Make sure it is installed at your side through this link. Then install the 'Angular Language Service' extension, as you can see below:



This extension will boost your development speed with code completion and other refactoring options.

Node.js

We will need node.js for 2 main reasons:

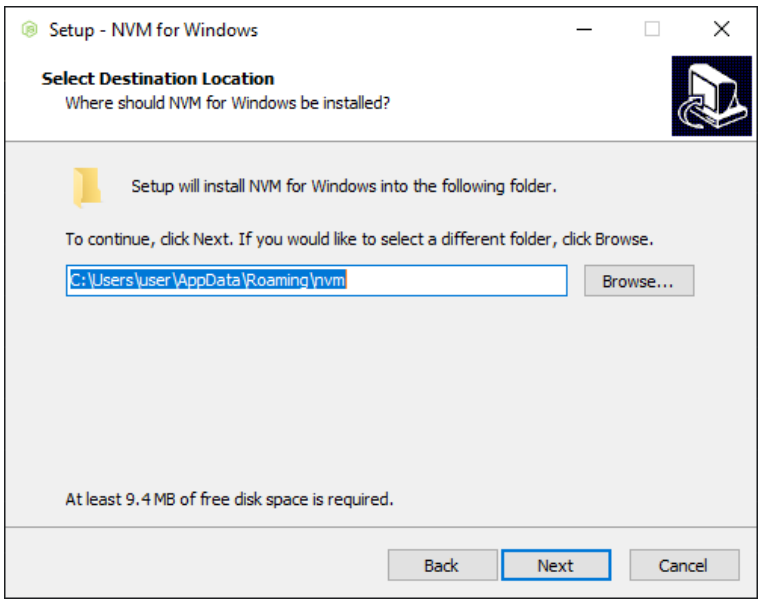
1. To run NPM (Node Package Manager) commands that will allow you to install libraries and packages that are needed for the angular development, one of which is the Angular CLI; This is the command line interface that enables you to run the commands needed for creating different the building blocks and deploying the angular code to localhost server.
2. node.js is a server-side framework which means it can be used to create a localhost to host your angular code on it. At the end you require some hosting server to put your front-end files on it (html, css, js and others).

So let's go and install node.js.

Since we are using windows desktop to do our development, then it is recommended to follow this approach to install node.js on your machine

NVM

nvm is the Node Version Manager that allows you to install node.js in the most correct and proper way. Since we are on windows, it is best to use nvm-windows installer, so let's download and install it.



Once installed, open a cmd or powershell, and type> *nvm ls* , you should get the below response

```
C:\Users\user>nvm list

No installations recognized.
```

Then we will need to see what we should install through the command> *nvm list available*

```
C:\Users\user>nvm list available

|   CURRENT    |     LTS      |  OLD STABLE  | OLD UNSTABLE |
|--------------|--------------|--------------|--------------|
|    17.5.0    |   16.14.0    |   0.12.18    |   0.11.16    |
|    17.4.0    |   16.13.2    |   0.12.17    |   0.11.15    |
|    17.3.1    |   16.13.1    |   0.12.16    |   0.11.14    |
|    17.3.0    |   16.13.0    |   0.12.15    |   0.11.13    |
|    17.2.0    |   14.19.0    |   0.12.14    |   0.11.12    |
|    17.1.0    |   14.18.3    |   0.12.13    |   0.11.11    |
|    17.0.1    |   14.18.2    |   0.12.12    |   0.11.10    |
|    17.0.0    |   14.18.1    |   0.12.11    |    0.11.9    |
|   16.12.0    |   14.18.0    |   0.12.10    |    0.11.8    |
|   16.11.1    |   14.17.6    |    0.12.9    |    0.11.7    |
|   16.11.0    |   14.17.5    |    0.12.8    |    0.11.6    |
|   16.10.0    |   14.17.4    |    0.12.7    |    0.11.5    |
|    16.9.1    |   14.17.3    |    0.12.6    |    0.11.4    |
|    16.9.0    |   14.17.2    |    0.12.5    |    0.11.3    |
|    16.8.0    |   14.17.1    |    0.12.4    |    0.11.2    |
|    16.7.0    |   14.17.0    |    0.12.3    |    0.11.1    |
|    16.6.2    |   14.16.1    |    0.12.2    |    0.11.0    |
|    16.6.1    |   14.16.0    |    0.12.1    |    0.9.12    |
|    16.6.0    |   14.15.5    |    0.12.0    |    0.9.11    |
|    16.5.0    |   14.15.4    |   0.10.48    |    0.9.10    |

This is a partial list. For a complete list, visit https://nodejs.org/en/download/releases
```

It is recommended to install the LTS version, since it is a Long-Term Support version being more stable than the latest release.

Therefore, as of writing this tutorial, the current LTS version is 16.14.0, which is the version that we are going to install now.

Go ahead and type> *nvm install 16.14.0*

```
C:\Users\user>nvm install 16.14.0
Downloading node.js version 16.14.0 (64-bit)...
Extracting...
Complete


Installation complete. If you want to use this version, type

nvm use 16.14.0
```

It will download and install node.js 16.14.0

Then you can type > nvm ls to get the install version, notice that you can have multiple node.js installations and this is what nvm is useful for, it allows you to easily switch between different node.js installations to support you for multiple projects.

```
C:\Users\user>nvm ls

    16.14.0
```

Now the final step is to let nvm use the installation version as your current installation by typing> nvm use 16.14.0

```
C:\Users\user>nvm use 16.14.0
exit status 1: Access is denied.
```

see here we are getting access denied, because this requires admin permissions, so let's close cmd and run it as admin from windows start

```
C:\WINDOWS\system32>nvm use 16.14.0
Now using node v16.14.0 (64-bit)
```

So let's see if node is now working fine, type> node -v

You should get the current version of node:

```
C:\WINDOWS\system32>node -v
v16.14.0
```

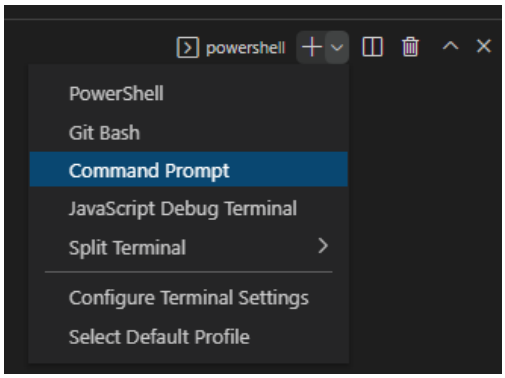
NPM

This is the node package manager, which allows you install and manage the libraries and dependencies of your angular app.

Once you install node.js, npm should be there by default, but let's make sure it is actually there and check its version.

Complete Work Management Suite
Ope
ProHance Workflow

Open Visual Studio Code, then from the Terminal tab, choose New Terminal, from the right side, open the options menu and choose Command Prompt:



Navigate to your designated folder that you will create your angular project on.

Let's make sure the npm is installed, type> npm -v

```
C:\Users\user\source\repos>npm -v
8.3.1
```

Angular CLI

Now let's start by installing the Angular CLI using the npm.

So let's type> *npm install -g @angular/cli*

```
C:\Users\user\source\repos>npm install -g @angular/cli

added 188 packages, and audited 190 packages in 22s

found 0 vulnerabilities
```

Now let's use some angular cli commands to initialize and create our angular app:

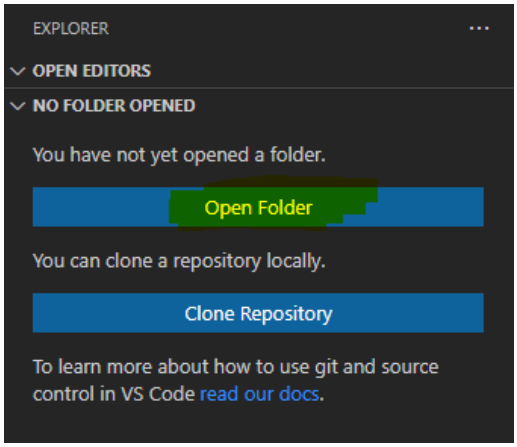
type> *ng new tasks-app*

It will ask you some questions, choose **Yes** and **CSS**

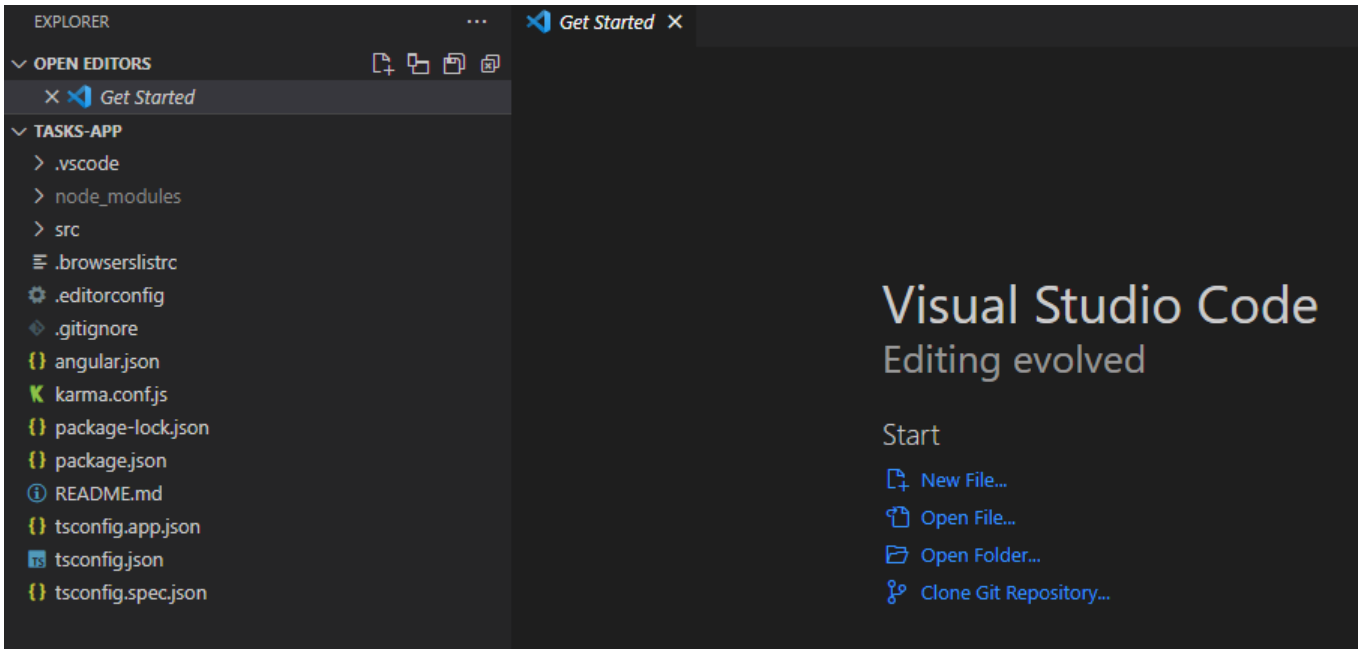
```
C:\Users\user\source\repos>ng new tasks-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

It would take some time to full create a new angular project, since it will be installing many angular-related and dependent packages through npm.

Once the blank app is created, make sure that Visual Studio code is navigating it, by choosing Open Folder and selecting the folder where you've just installed the new tasks-app



You should be getting the below:



That's great, we have our initial empty angular app ready on the IDE, let's use some more Angular CLI commands to generate the boilerplate components and services for us.

For this tutorial, we will need components for login, signup and home which will display the user's tasks.

Also we will create some services that will be used as a middle layer between the components and the backend APIs, moreover, we will need some helpers to perform some useful actions in the code.

App Module

We need to import some libraries to use them within our tasks app. FormsModule will allow us to use the Angular Forms (Template or Reactive Forms) in our template files to build comprehensive forms with validations. And

HttpClientModule is the library that facilitates the network communications with external APIs, which we will use to connect to our backend APIs.

Open App.Module.ts file and add the below import lines:

```
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
```

And in the Modules array, add the 2 modules there:

```
FormsModule,
HttpClientModule
```

Now we have both libraries included in our project.

Requests and Responses

We will have 2 folders that will hold the requests and responses respectively. We will include Angular interfaces with properties structures similar to the ones we have on our API.

Doing such will make the communication between the Angular app and the API more convenient and consistent.

Requests

Now let's generate an interface for LoginRequest under requests folder (the folder will be generated if it is not there):

LoginRequest

From the cmd, input the following command:

ng g interface requests/LoginRequest

```
C:\Users\user\source\repos\tasks-app>ng g interface requests/LoginRequest
CREATE src/app/requests/login-request.ts (34 bytes)
```

```
export interface LoginRequest {
    email: string,
    password: string
}
```

SignupRequest

ng g interface requests/SignupRequest

```
C:\Users\user\source\repos\tasks-app>ng g interface requests/SignupRequest
CREATE src/app/requests/signup-request.ts (35 bytes)
```

```
export interface SignupRequest {
    email: string;
    password: string;
    confirmPassword: string;
    firstName: string;
    lastName: string;
    ts: Date;
}
```

RefreshTokenRequest

ng g interface requests/RefreshTokenRequest

```
C:\Users\user\source\repos\tasks-app>ng g interface requests/RefreshTokenRequest
CREATE src/app/requests/refresh-token-request.ts (41 bytes)
```

```
export interface RefreshTokenRequest {
    userId: number;
    refreshToken: string;
}
```

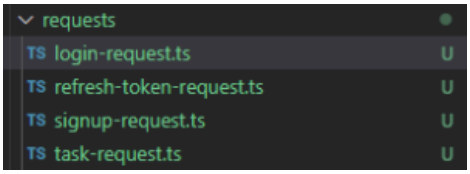
TaskRequest

ng g interface requests/TaskRequest

```
C:\Users\user\source\repos\tasks-app>ng g interface requests/TaskRequest
CREATE src/app/requests/task-request.ts (33 bytes)
```

```
export interface TaskRequest {
    name: string;
    isCompleted: boolean;
    ts: Date;
}
```

At the end, you should see the following folder structure in your VS Code Explorer:



Next, is to create the responses interfaces which will bind to the API responses objects

Responses

TokenResponse

ng g interface responses/TokenResponse

```
C:\Users\user\source\repos\tasks-app>ng g interface responses/TokenResponse
CREATE src/app/responses/token-response.ts (35 bytes)
```

```
export interface TokenResponse {
    accessToken: string;
    refreshToken: string;
}
```

TaskResponse

ng g interface responses/TaskResponse

```
C:\Users\user\source\repos\tasks-app>ng g interface responses/TaskResponse
CREATE src/app/responses/task-response.ts (34 bytes)
```

```
export interface TaskResponse {
    id: number;
    name: string;
    isCompleted: boolean;
    ts: Date;
}
```

ErrorResponse

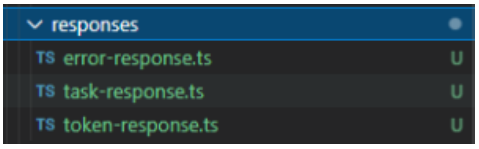
This will hold the data whenever the API returns 422 Unprocessable Entity or 400 Bad Request Http Responses.

ng g interface responses/ErrorResponse

```
C:\Users\user\source\repos\tasks-app>ng g interface responses/ErrorResponse
CREATE src/app/responses/error-response.ts (35 bytes)
```

```
export interface ErrorResponse {
    error:string;
    errorCode: string;
}
```

And this will be the content for the responses folder:



Now before we go to the service part, open the environment.ts file and include the apiUrl in the environment constant, like the below:

```
export const environment = {
  production: false,
  apiUrl: 'https://localhost:7217/api'
};
```

Services

This is layer where you include the calls to the backend APIs along with any other local services to the logic related to storing and retrieving data from session or local storage.

UserService

The user service includes the http calls to our backend users controller which has the endpoints including login, signup, logout, refreshToken and getUserInfo.

So, from the terminal, type: *ng g s services/user*

It will create for you services folder with a user-service.ts file.

Now, let take a look at the user service methods that we will add:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';
import { LoginRequest } from '../requests/login-request';
import { SignupRequest } from '../requests/signup-request';
import { TokenResponse } from '../responses/token-response';
import { UserResponse } from '../responses/user-response';

@Injectable({
  providedIn: 'root'
})
export class UserService {

  constructor(private httpClient: HttpClient) { }

  login(loginRequest: LoginRequest): Observable<TokenResponse> {
    return this.httpClient.post<TokenResponse>(`${environment.apiUrl}/users/login`, loginRequest);
  }

```

```
  signup(SignupRequest: SignupRequest) {
    return this.httpClient.post(`${environment.apiUrl}/users/signup`, SignupRequest, { responseType: 'text'}); // response t
  }

  refreshToken(session: TokenResponse) {
    let refreshTokenRequest: any = {
      UserId: session.userId,
      RefreshToken: session.refreshToken
    };
    return this.httpClient.post<TokenResponse>(`${environment.apiUrl}/users/refresh_token`, refreshTokenRequest);
  }

  logout() {
    return this.httpClient.post(`${environment.apiUrl}/users/signup`, null);
  }

  getUserInfo(): Observable<UserResponse> {
    return this.httpClient.get<UserResponse>(`${environment.apiUrl}/users/info`);
  }

}
```

Notice how simple our methods are? The constructor injects the httpClient which will be used to do the http calls, and each method has this simple httpClient call, with providing the endpoint URL and the request object in case it was a non-get call.

We prefer to keep the services neat and clean without extra business logic, since these usually happen on the component ts file, once we subscribe to the returned Observable.

Moreover, you have to understand that these methods don't do actions (http calls) once they are called, the actions happen once you call the subscribe method on them, only then the http call gets triggered and a response is returned either to the next part (success) or error part (fail), and the third section which is the complete part, it is triggered in both cases, this is where you can put finalizing logic such as stopping a progress bar from loading.

TokenService

The token service will handle the local validation, storage and retrieval of the tokens and key user details. This is a critical service to keep the angular app aware of the authentication related data.

so let's create this service:

ng g s services/token

and here is the code:

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { TokenResponse } from '../responses/token-response';
import { UserService } from './user.service';

@Injectable({
  providedIn: 'root'
})
export class TokenService {

  constructor(private userService: UserService) { }

  saveSession(tokenResponse: TokenResponse) {

    window.localStorage.setItem('AT', tokenResponse.accessToken);
    window.localStorage.setItem('RT', tokenResponse.refreshToken);
    if (tokenResponse.userId) {
      window.localStorage.setItem('ID', tokenResponse.userId.toString());
      window.localStorage.setItem('FN', tokenResponse.firstName);
    }

  }

  getSession(): TokenResponse | null {
    if (window.localStorage.getItem('AT')) {
```

```
      const tokenResponse: TokenResponse = {
        accessToken: window.localStorage.getItem('AT') || '',
        refreshToken: window.localStorage.getItem('RT') || '',
        firstName: window.localStorage.getItem('FN') || '',
        userId: +(window.localStorage.getItem('ID') || 0),
      };

      return tokenResponse;
    }
    return null;
  }

  logout() {
    window.localStorage.clear();
  }

  isLoggedIn(): boolean {
    let session = this.getSession();
    if (!session) {
      return false;
    }

    // check if token is expired
    const jwtToken = JSON.parse(atob(session.accessToken.split('.')[1]));
    const tokenExpired = Date.now() > (jwtToken.exp * 1000);

    return !tokenExpired;

  }

  refreshToken(session: TokenResponse): Observable<TokenResponse> {

    return this.userService.refreshToken(session);
  }

}
```

TaskService

This service will hold the methods to call the tasks related endpoints.

from the terminal, type:

ng g s services/task

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable, Subscription } from 'rxjs';
import { environment } from 'src/environments/environment';
import TaskResponse from '../responses/task-response';

@Injectable({
  providedIn: 'root'
})
export class TaskService {

  constructor(private httpClient: HttpClient) { }

  getTasks(): Observable<TaskResponse[]> {
    return this.httpClient.get<TaskResponse[]>(`${environment.apiUrl}/tasks`);
  }

  saveTask(task: TaskResponse): Observable<TaskResponse> {
    return this.httpClient.post<TaskResponse>(`${environment.apiUrl}/tasks`, task);
  }

  updateTask(task: TaskResponse): Observable<TaskResponse> {
    return this.httpClient.put<TaskResponse>(`${environment.apiUrl}/tasks`, task);
  }

  deleteTask(taskId: number) {
    return this.httpClient.delete<TaskResponse>(`${environment.apiUrl}/tasks/${taskId}`);
  }

}
```

Some notes on the above code:

Even though these methods call endpoints that are authorized (must provide authorization header), we don't pass the headers inside these methods, in the upcoming section we will see how we can apply the authorization header on

each request in a single place, no need to repeat the process inside each method having an http call.

Interceptors

With interceptors, implementing the HttpInterceptor, you can inject some code into the http process pipeline to modify or process the outgoing/incoming http requests/responses.

2 important interceptors that you should add to your application would be the authorization interceptor for requests and the error interceptor for responses.

Authorization Interceptor

Using the terminal, add the below Angular CLI command to let it create the interceptor for you:

ng g interceptor interceptors/auth

Here is the code for the auth.interceptor:

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor,
  HTTP_INTERCEPTORS
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';
import { TokenService } from '../services/token.service';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private tokenService: TokenService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    const requestForApis = request.url.startsWith(environment.apiUrl);
    const isLoggedIn = this.tokenService.isLoggedIn();

    console.log('inside intercept');
    console.log(`request is ${JSON.stringify(request.body)}`);
    if (isLoggedIn && requestForApis) {
      let session = this.tokenService.getSession();
      if (session){
        request = request.clone({ headers: request.headers.set('Authorization', `Bearer ${session.accessToken}`) });
      }

    }
    return next.handle(request);
  }
}
export const AuthInterceptorProvider = { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true };
```

See in the above code, we are checking if the user is logged in and the request is going to our backend APIs, not some other APIs, then we will add the Authorization header, with token type as bearer with the saved JWT token itself.

In case the user is not logged in or the API call is for a different API endpoint, then we just let the request continue its normal flow without any modification on it.

Error Interceptor

Let's create another interceptor, this will intercept the response of the http call, log the responses and handle silently refreshing the access token if the API returned 401 unauthorized response. This interceptor will also propagate the error by a custom error class.

So let's create the interceptor from the terminal:

ng g interceptor interceptors/error

And the code is below:

```
import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor,
  HttpErrorResponse,
  HTTP_INTERCEPTORS
} from '@angular/common/http';
import { catchError, EMPTY, Observable, tap, throwError } from 'rxjs';
import { ErrorResponse } from '../responses/error-response';
import { TokenService } from '../services/token.service';
import { UserService } from '../services/user.service';
import { Router } from '@angular/router';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {

  isRefreshingToken: boolean = false;
  constructor(private tokenService: TokenService, private userService: UserService, private router: Router) { }

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
    return next.handle(request)
      .pipe(
        tap(response => console.log(JSON.stringify(response))),
        catchError((error: HttpErrorResponse) => {

          console.log(JSON.stringify(error));
          let session = this.tokenService.getSession();
          if (error.status === 401 && session != null && !this.tokenService.isLoggedIn() && !this.isRefreshingToken) {
            this.isRefreshingToken = true;
            console.log('Access Token is expired, we need to renew it');
            this.userService.refreshToken(session).subscribe({
              next: data => {
                console.info('Tokens renewed, we will save them into the local storage');
                this.tokenService.saveSession(data);
              },
              error: () => { },
              complete: () => { this.isRefreshingToken = false }
            });
          } else if (error.status === 400 && error.error.errorCode === 'invalid_grant') {
            console.log('the refresh token has expired, the user must login again');
            this.tokenService.logout();
            this.router.navigate(['login']);


          } else {
            let errorResponse: ErrorResponse = error.error;
            console.log(JSON.stringify(errorResponse));

            return throwError(() => errorResponse);
          }

          return EMPTY;
        })
      );
  }
}
export const ErrorInterceptorProvider = { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true };
```

Then we need to include both interceptors within the app module file in the providers array, so let's open app.module.ts

So it should look as the below:

```
providers: [
    AuthInterceptorProvider,
    ErrorInterceptorProvider
  ],
```

Guard

The guard is a special class that can be used to make sure that a secured page is not accessible for any user who is not authenticated with a valid JWT token. This is the entry point to achieve this tutorial's goal which is to Secure Angular Site using JWT Authentication.

So type the below command on the terminal:

ng g guard helpers\auth

```
C:\Users\user\source\repos\tasks-app>ng g guard helpers\auth
? Which interfaces would you like to implement? CanActivate
CREATE src/app/helpers/auth.guard.spec.ts (331 bytes)
CREATE src/app/helpers/auth.guard.ts (457 bytes)
```

and the guard code is here:

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot, UrlTree } from '@angular/router';
import { catchError, EMPTY, map, Observable } from 'rxjs';
import { ErrorResponse } from '../responses/error-response';
import { TokenResponse } from '../responses/token-response';
import { TokenService } from '../services/token.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private router: Router, private tokenService: TokenService) { }
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

    let session = this.tokenService.getSession();
    if (session == null) {
      this.router.navigate(['/login']);
      return false;
    }

    if (!this.tokenService.isLoggedIn()) {

      console.log(`session is expired, let's renew the tokens`);
      // refresh token
      return this.checkSession(session);
    }
    return true;
  }

  checkSession(session: TokenResponse): Observable<boolean> {
    return this.tokenService.refreshToken(session).pipe(
      map(data => {
        console.log(`refreshToken repsonse is ${JSON.stringify(data)}`);
        this.tokenService.saveSession(data);
        return true;
      }),
      catchError((error: ErrorResponse) => {
        console.log(`inside checkSession ${JSON.stringify(error)}`);
        this.router.navigate(['/login']);
        return EMPTY;
      })
    );

  }

}
```

As you can see here, the guard is also checking if the token is expired and doing a silent refresh using the refresh token, in case the silent refresh failed, it means either the refresh token was revoked or got expired and therefore the user has to login again.

Components

These are your UI related building blocks that include both the template and the code behind it.

Tasks (Home)

ng g c tasks

task.component.html

```
<div class="container mt-3">
    <div class="row mt-2 justify-content-center">
        <h2>My Tasks</h2>
    </div>
    <div class="row mt-1 mb-3 justify-content-center align-items-center">
        <div class="col-sm-2"></div>
        <div class="col-sm-8">
            <div class="input-group mb-3">
                <input type="text" class="form-control" [(ngmodel)]="newTaskString" name="newTaskTxt">
                <div class="input-group-append ml-2">
                    <button class="form-control btn btn-success" (click)="addTask()">Add Task</button>
                </div>
            </div>
        </div>
        <div class="col-sm-2"></div>
    </div>
    <div class="row">
        <div class="col-sm-2"></div>
        <div class="col-sm-8">
            <div class="list-group col-sm-10 mt-2" *ngfor="let task of tasks$ | async">
                <a role="button" (click)="updateTask(task)" class="list-group-item list-group-item-action" [ngclass]="{'list
                    <div class="d-flex justify-content-between">
                        <h5 class="mb-1">{{ task.name }}</h5>
                        <small>{{ task.ts | date }}</small>
                    </div>

                </a>
                <button class="btn btn-danger" (click)="removeTask(task)">Remove</button>
            </div>
        </div>

        <div class="col-sm-2"></div>
    </div>
</div>
```

task.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import TaskResponse from '../responses/task-response';
import { TaskService } from '../services/task.service';

@Component({
  selector: 'app-tasks',
  templateUrl: './tasks.component.html',
  styleUrls: ['./tasks.component.css']
})
export class TasksComponent implements OnInit {

  newTaskString: string = '';
  tasks$: Observable<TaskResponse[]> | undefined;
  constructor(private taskService: TaskService) { }

  ngOnInit(): void {
    this.tasks$ = this.taskService.getTasks();
  }

  addTask(): void {
    if (!this.newTaskString){
      return;
    }
    let task: TaskResponse = {
      id: 0,
      isCompleted: false,
      name: this.newTaskString,
      ts: new Date()
    };
    this.taskService.saveTask(task).subscribe(() => this.tasks$ = this.taskService.getTasks());
  }

  updateTask(task: TaskResponse) {
    console.log('inside updatetask');
    console.log(`task id is ${task.id}`);
    task.isCompleted = !task.isCompleted;
    this.taskService.updateTask(task).subscribe(() => {});

  }

  removeTask(task: TaskResponse) {
```

```
    console.log('inside removeTask');
    console.log(`task id is ${task.id}`);
    this.taskService.deleteTask(task.id).subscribe(() => this.tasks$ = this.taskService.getTasks());
  }

}
```

Login

Angular CLI Command: *ng g c login*

login.component.html

```
<div class="container">
  <div class="row m-3 justify-content-center">
    <h2>Login</h2>
  </div>
  <div class="row">
    <div class="col-md-12">
      <div class="card card-container p-3">
        <form *ngif="!isLoggedIn" name="form" (ngsubmit)="ngLoginForm.form.valid &amp;&amp; onSubmit()" #ngloginform="ngForm
          <div class="form-group">
            <label for="email">Email</label>
            <input type="text" class="form-control" name="email" [(ngmodel)]="loginRequest.email" required="" #email="ngMode
            <div class="alert alert-danger" role="alert" *ngif="!email.pristine &amp;&amp; email.errors">
              Email is required!
            </div>
          </div>
          <div class="form-group">
            <label for="password">Password</label>
            <input type="password" class="form-control" name="password" [(ngmodel)]="loginRequest.password" required="" #pas
            <div class="alert alert-danger" role="alert" *ngif="!email.pristine &amp;&amp; password.errors">
              <div *ngif="password.errors['required']">Password is required</div>
            </div>
          </div>
          <div class="form-group">
            <button class="btn btn-primary btn-block">
              Login
            </button>
          </div>
          <div class="form-group">
            <div class="alert alert-danger" role="alert" *ngif="ngLoginForm.submitted &amp;&amp; isLoginFailed">
              Login failed: {{ error.error }}
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { LoginRequest } from '../requests/login-request';
import { ErrorResponse } from '../responses/error-response';
import { TokenService } from '../services/token.service';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  loginRequest: LoginRequest = {
    email: "",
    password: ""
  };

  isLoggedIn = false;
  isLoginFailed = false;
  error: ErrorResponse = { error: '', errorCode: 0 };
  constructor(private userService: UserService, private tokenService: TokenService, private router: Router) { }

  ngOnInit(): void {
    let isLoggedIn = this.tokenService.isLoggedIn();
    console.log(`isLoggedIn: ${isLoggedIn}`);
    if (isLoggedIn) {
      this.isLoggedIn = true;

      this.router.navigate(['tasks']);
```

```
    }
  }

  onSubmit(): void {

    this.userService.login(this.loginRequest).subscribe({
      next: (data => {
        console.debug(`logged in successfully ${data}`);
        this.tokenService.saveSession(data);
        this.isLoggedIn = true;
        this.isLoginFailed = false;
        this.reloadPage();
      }),
      error: ((error: ErrorResponse) => {
        this.error = error;
        this.isLoggedIn = false;
        this.isLoginFailed = true;
      })

    });
  }
  reloadPage(): void {
    window.location.reload();
  }
}
```

Signup

Angular CLI Command: *ng g c signup*

signup.component.html

```
<div class="container mt-3">
  <div class="row m-3 justify-content-center">
    <h2>Sign up</h2>
  </div>
  <div class="row">
    <div class="col-md-12">
      <div class="card card-container p-3">
        <form #ngsignup="ngForm" name="form" (ngsubmit)="onSubmit(ngSignup.value)" *ngif="!isSuccessful" novalidate="">

          <div class="form-group">
            <label for="email">Email</label>
            <input type="text" class="form-control" name="email" [(ngmodel)]="signupRequest.email" email="" #email="ngModel"
            <div class="alert-danger" *ngif="email.errors &amp;&amp; ngSignup.submitted">
              <div *ngif="email.errors['email']">Invalid Email</div>
            </div>
          </div>

          <div class="form-group">
            <label for="password">Password</label>
            <input type="password" class="form-control" name="password" [(ngmodel)]="signupRequest.password" required="" pat
            <div class="alert-danger" *ngif="ngSignup.submitted &amp;&amp; password.errors">
              <div *ngif="password.errors['required']">Password is required</div>
              <div *ngif="password.errors['pattern']"> Enter a strong password. At least one number and one uppercase
                and
                lowercase letter, and at least 8 or more characters.</div>
            </div>
          </div>

          <div class="form-group">
            <label for="confirmPassword">Confirm Password</label>
            <input type="password" class="form-control" name="confirmPassword" [(ngmodel)]="signupRequest.confirmPassword" r
            <div class="alert-danger" *ngif="ngSignup.submitted &amp;&amp; confirmPassword.errors">
              <div *ngif="confirmPassword.errors['required']">Confirm Password is required</div>
              <div *ngif="confirmPassword.errors['pattern']"> Enter a strong password. At least one number and one
                uppercase
                and lowercase letter, and at least 8 or more characters.</div>
            </div>
          </div>

          <div class="form-group">
            <label for="firstName">First Name</label>
            <input type="text" class="form-control" name="firstName" [(ngmodel)]="signupRequest.firstName" required="" #firs
            <div class="alert-danger" *ngif="firstName.errors &amp;&amp; ngSignup.submitted">
              <div *ngif="firstName.errors['required']">First name is required</div>
            </div>
          </div>

          <div class="form-group">
            <label for="firstName">Last Name</label>
            <input type="text" class="form-control" name="lastName" [(ngmodel)]="signupRequest.lastName" required="" #lastna
            <div class="alert-danger" *ngif="lastName.errors &amp;&amp; ngSignup.submitted">
              <div *ngif="lastName.errors['required']">Last name is required</div>
            </div>
```

```
          </div>

          <div class="form-group">
            <input class="btn btn-primary btn-block" type="submit" value="Sign Up">
          </div>
          <div class="alert alert-warning" *ngif="ngSignup.submitted &amp;&amp; isSignUpFailed">
            Signup failed!<br>{{ errorMessage }}
          </div>
        </form>
        <div class="alert alert-success" *ngif="isSuccessful">
          Congrats! You have signed up successfully.
          You can login now to your account.
        </div>
      </div>
    </div>
  </div>
</div>
```

signup.component.ts

```
import { Component, OnInit } from '@angular/core';
import { SignupRequest } from '../requests/signup-request';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent implements OnInit {
  signupRequest: SignupRequest = {
    email: "",
    password: "",
    confirmPassword: "",
    firstName: "",
    lastName: "",
    ts: new Date().toISOString()
  };
  isSuccessful = false;
  isSignUpFailed = false;
  errorMessage = '';
  constructor(private userService: UserService) { }
  ngOnInit(): void {
  }
  onSubmit(signupForm: any): void {
    console.log(JSON.stringify(signupForm));

    this.userService.signup(this.signupRequest).subscribe({
      next: data => {
        console.log(data);
        this.isSuccessful = true;
        this.isSignUpFailed = false;
      },
      error: err => {
        this.errorMessage = err.error.message;
        this.isSignUpFailed = true;
      }
    });
  }

}
```

Profile

Angular CLI Command: *ng g c profile*

profile.component.html

```
<div class="container">
    <div class="row m-3 justify-content-center">
        <h2>My Profile</h2>
    </div>
    <div class="row">
        <div class="col-md-12">
            <div class="card-container">
                <div class="card mt-2 p-2">
                    <h6 class="card-subtitle m-1 text-muted">Email:</h6>
                    <p class="card-text">{{ user.email }}</p>
                </div>
                <div class="card mt-2 p-2">
                    <h6 class="card-subtitle mb-2 text-muted">First Name:</h6>
                    <p class="card-text">{{ user.firstName }}</p>
                </div>
                <div class="card mt-2 p-2">
```

```
                    <h6 class="card-subtitle mb-2 text-muted">Last Name:</h6>
                    <p class="card-text">{{ user.lastName }}</p>

                </div>
            </div>
        </div>
    </div>
</div>
```

profile.component.ts

```
import { Component, OnInit } from '@angular/core';
import { UserResponse } from '../responses/user-response';
import { UserService } from '../services/user.service';

@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.css']
})
export class ProfileComponent implements OnInit {
  user: UserResponse = {
    email: '',
    firstName: '',
    lastName: '',
    creationDate: new Date()
  }
  constructor(private userService: UserService) { }

  ngOnInit(): void {
    this.userService.getUserInfo().subscribe(
      {
        next: (data => {
          this.user = data;
        }),
        error: (() => {
          console.log('failed to get the use info');
        })
      }

    );
  }

}
```

App Component (Main)

This is the main layout for your site, where there is a navigation bar in the header part of the page and below it is the router-outlet which is the directive for the routing library that will toggle the display of the correct component based on the current url.

app.component.html

```
<div>
  <nav class="navbar navbar-expand navbar-dark bg-success">
    <a href="#" class="navbar-brand">{{ title }}</a>
    <ul class="navbar-nav mr-auto" routerlinkactive="active">
      <li class="nav-item">
        <a href="/home" class="nav-link" routerlink="tasks">Home</a>
      </li>
    </ul>
    <ul class="navbar-nav ml-auto" *ngif="!isLoggedIn">
      <li class="nav-item">
        <a href="/signup" class="nav-link" routerlink="signup">Sign up</a>
      </li>
      <li class="nav-item">
        <a href="/login" class="nav-link" routerlink="login">Login</a>
      </li>
    </ul>
    <ul class="navbar-nav ml-auto" *ngif="isLoggedIn">
      <li class="nav-item">
        <a href="/profile" class="nav-link" routerlink="profile">Profile</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" (click)="logout()">Logout</a>
      </li>
    </ul>
  </nav>
  <div class="container">
    <router-outlet></router-outlet>
  </div>
```

```
</div>
```

And here is the code behind of this main component that includes the title and isLoggedIn variables as well as the logout method which will clear the current tokens session from the local storage and navigate to the login screen.

app.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { TokenService } from './services/token.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tasks App';
  isLoggedIn = false;
  constructor(private tokenService: TokenService, private router: Router) { }

  ngOnInit() {
    this.isLoggedIn = this.tokenService.isLoggedIn();
  }

  logout(): void {
    this.tokenService.logout();
    this.isLoggedIn = false;
    this.router.navigate(['login']);
    return;
  }
}
```

App Module

This is the final view of the app module class, with all the needed imports, declarations and providers.

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { SignupComponent } from './signup/signup.component';
import { AuthInterceptorProvider } from './interceptors/auth.interceptor';
import { ErrorInterceptorProvider } from './interceptors/error.interceptor';
import { TasksComponent } from './tasks/tasks.component';
import { ProfileComponent } from './profile/profile.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    SignupComponent,
    TasksComponent,
    ProfileComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
  ],
  providers: [
    AuthInterceptorProvider,
    ErrorInterceptorProvider
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

App Routing

In the app-routing we match the routes with the components, and also we can set some additional options like the canActivate to protect the component from being displayed in case of unauthorized access.

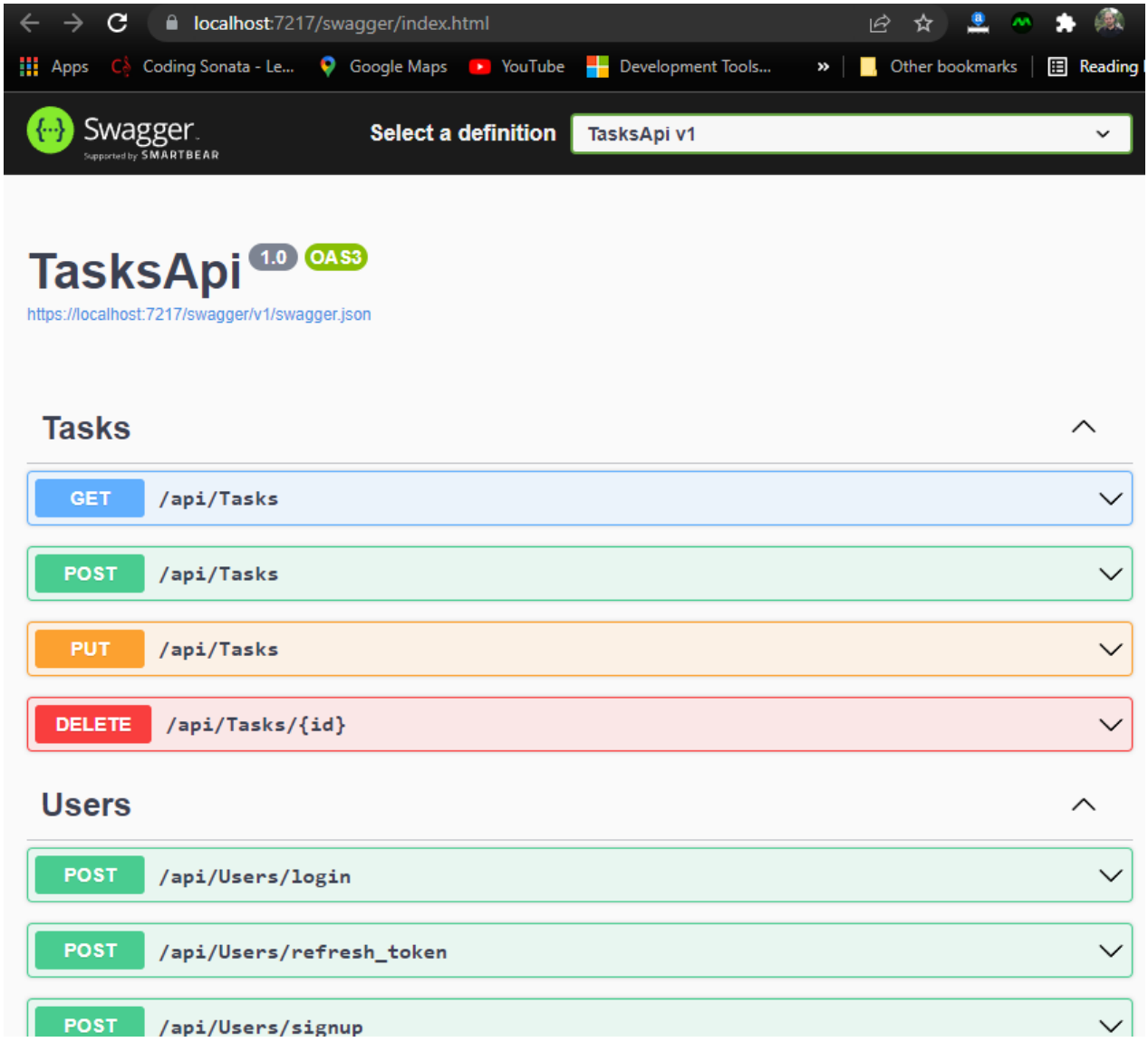
```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AuthGuard } from './helpers/auth.guard';
import { LoginComponent } from './login/login.component';
import { ProfileComponent } from './profile/profile.component';
import { SignupComponent } from './signup/signup.component';
import { TasksComponent } from './tasks/tasks.component';

const routes: Routes = [
  { path: 'tasks', component: TasksComponent, canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent },
  { path: 'profile', component: ProfileComponent, canActivate: [AuthGuard] },
  { path: '', redirectTo: 'tasks', pathMatch: 'full' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Testing our site

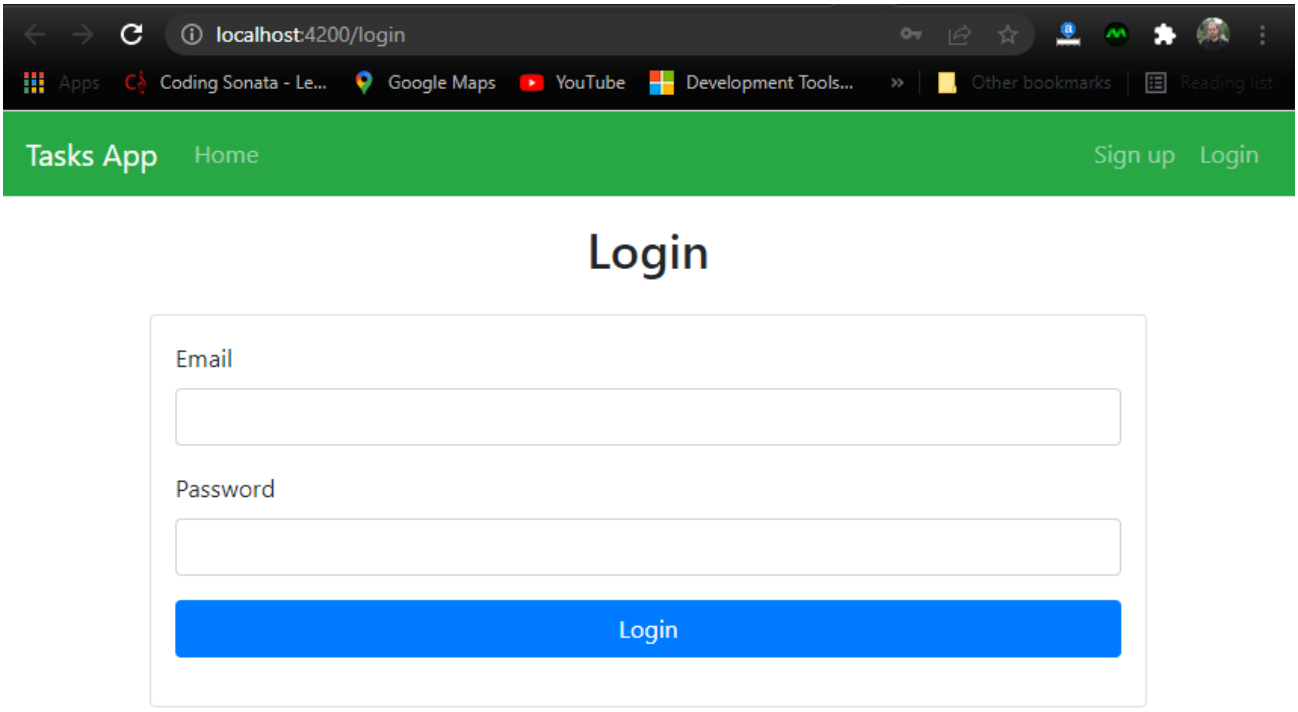
First let's run the API project on localhost, you should get the below swagger screen on your browser:



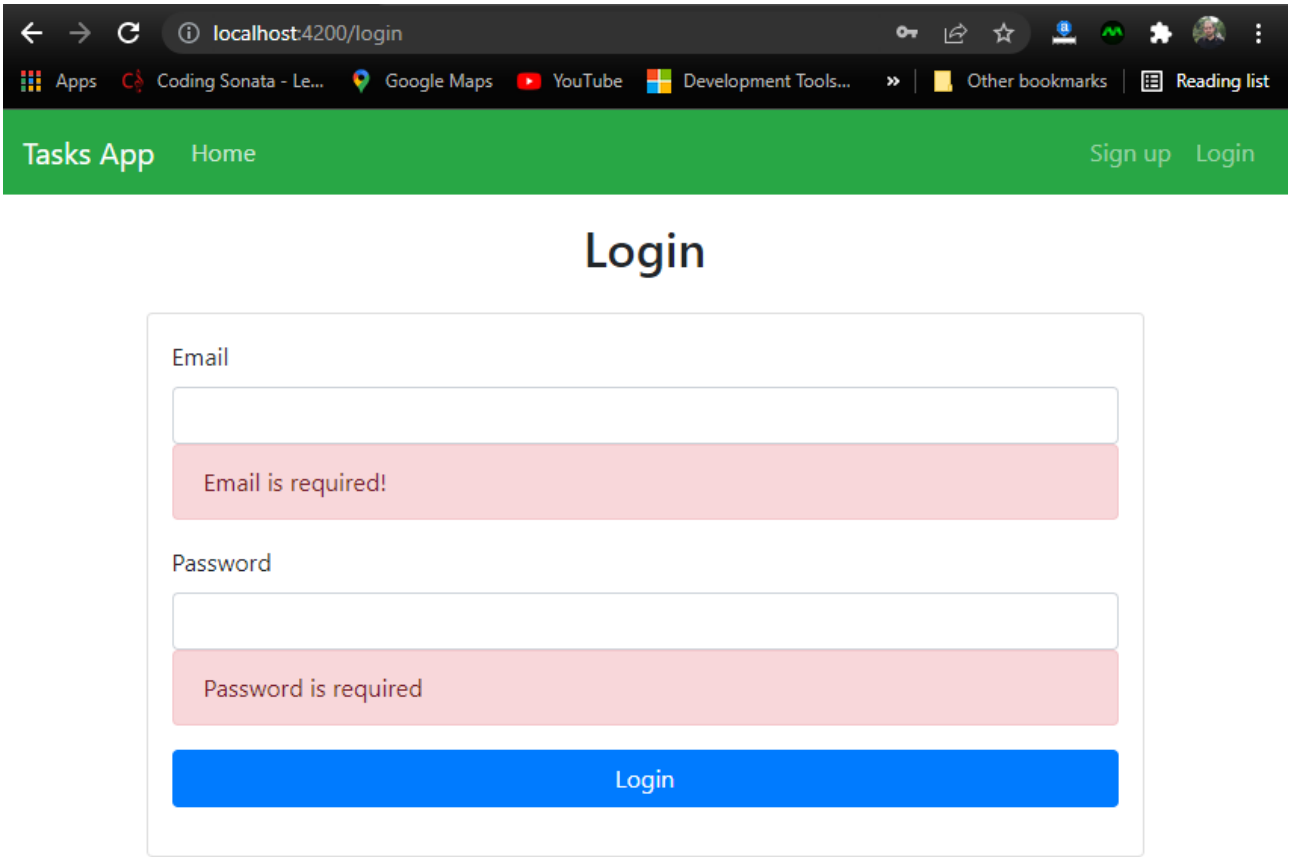
Now let's serve our Angular site on localhost port 4200

In the terminal, type this command: *ng serve –open*

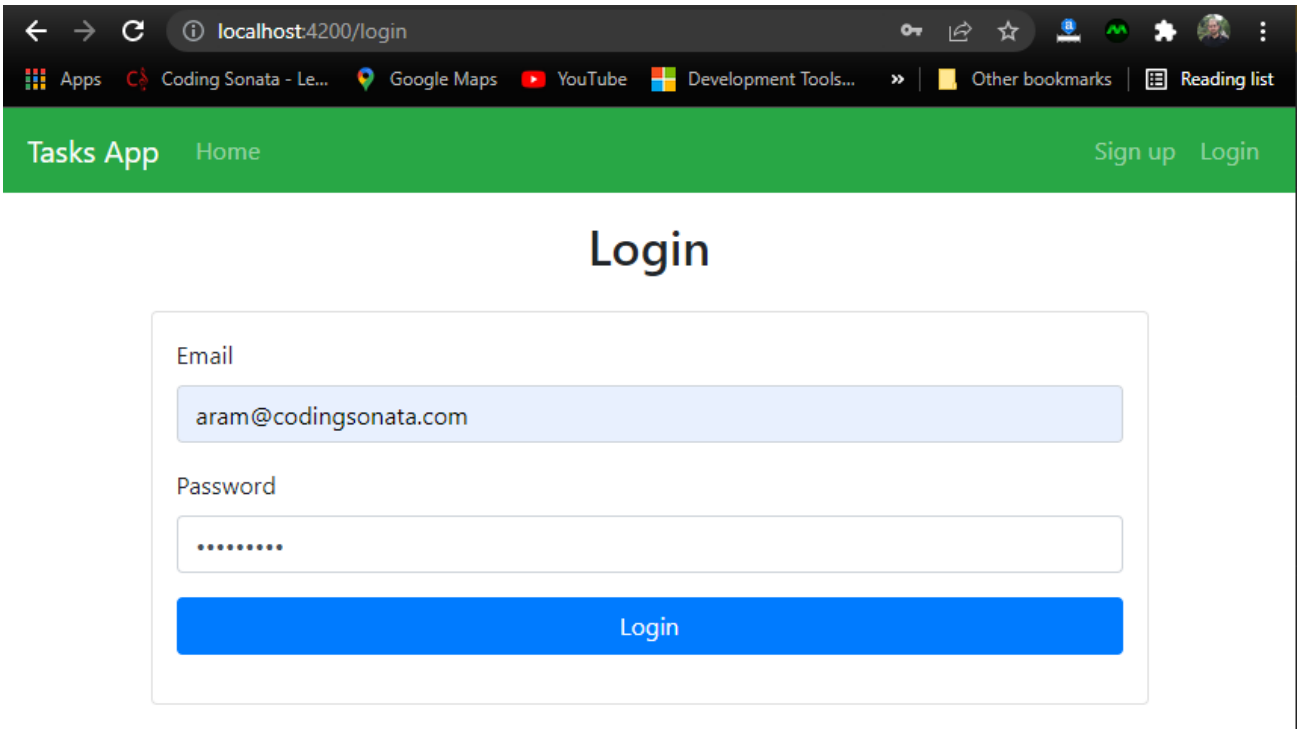
this will build your angular site and serve it under localhost and open the default route on the browser:



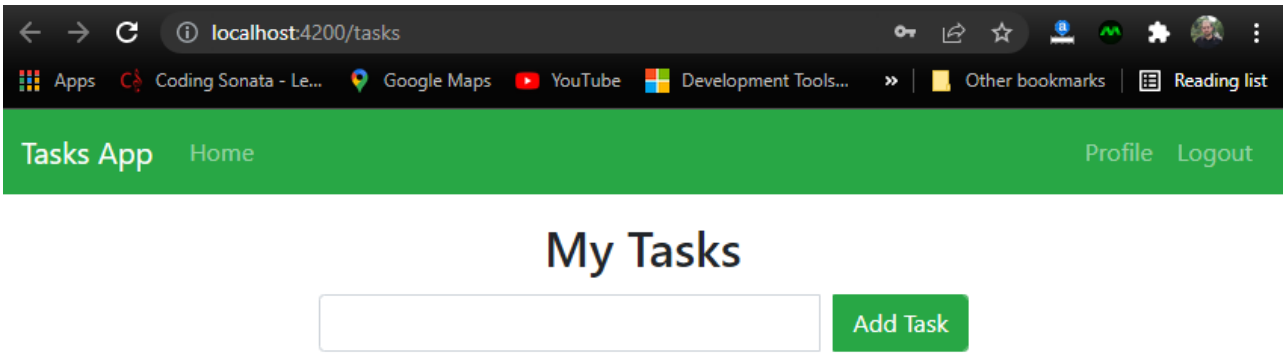
Now let's try to login without entering any value:



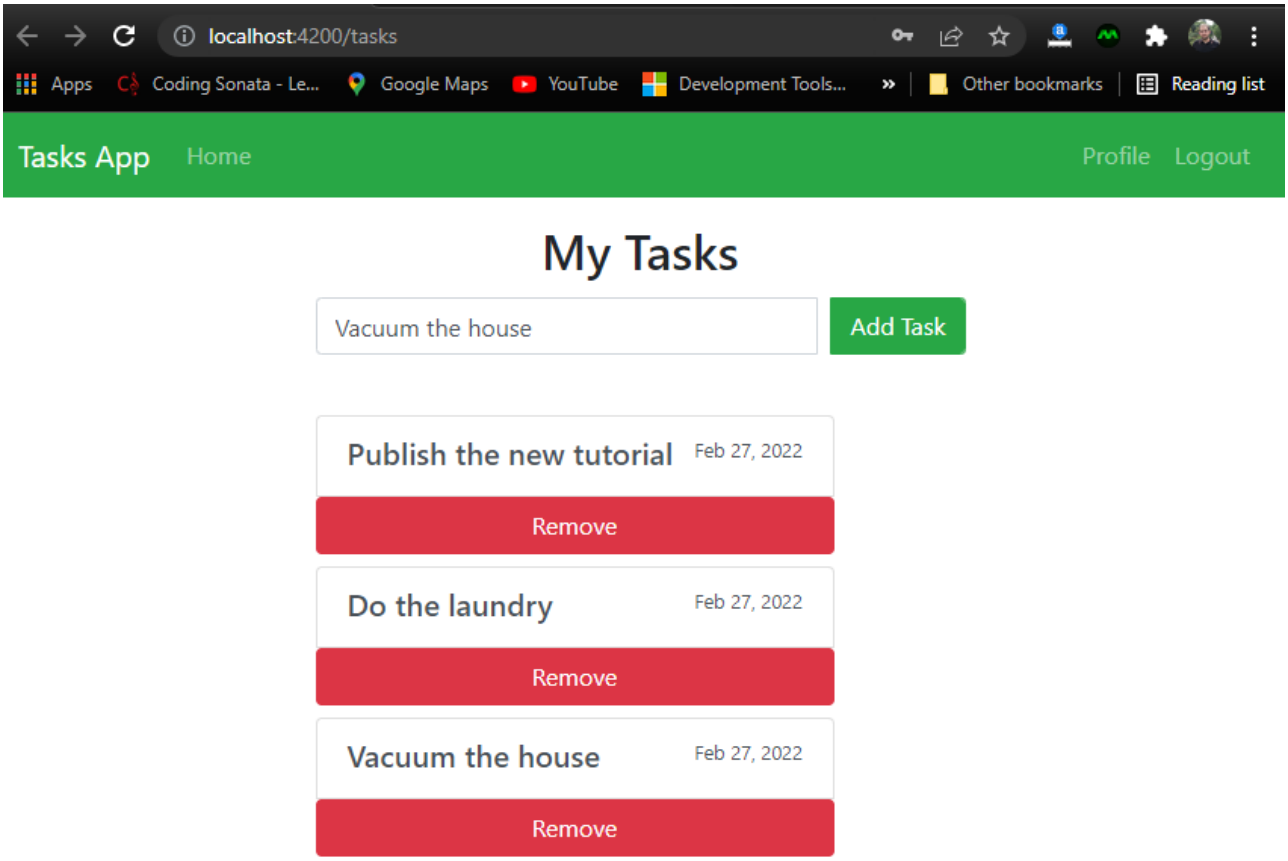
And let's enter some valid credentials:



Now you should be logged in and redirected to the tasks route where you will be able to view, add, update and delete your tasks, also notice how the navigation bar has changed its links instead of sign up and login to profile and logout:

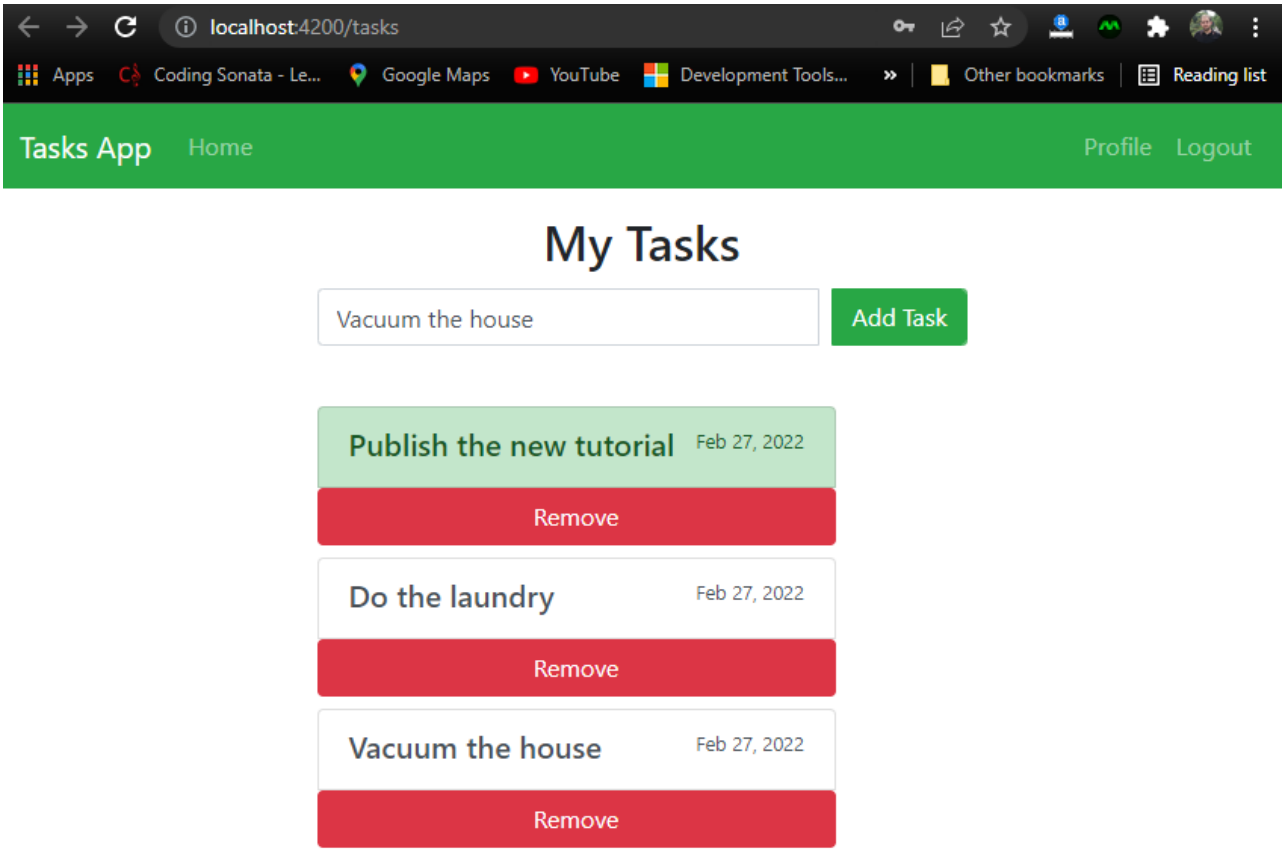


Now let's add some tasks:



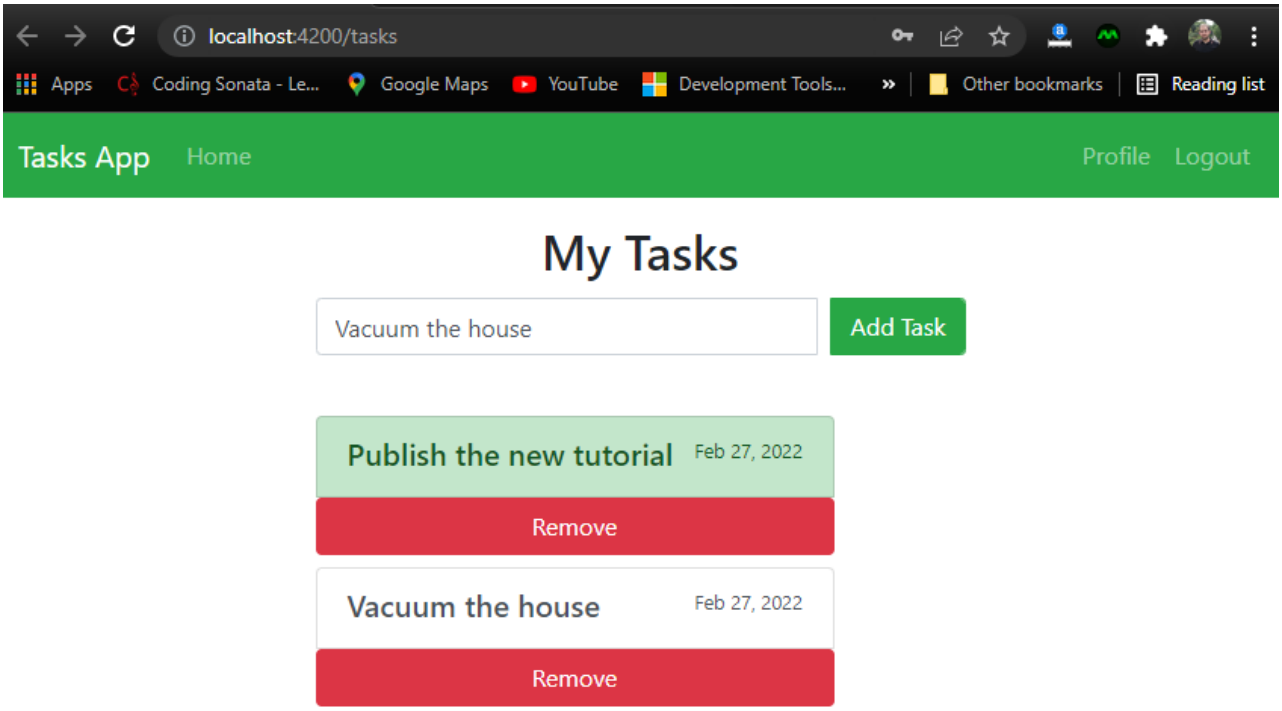
We just enter the task name, and press Add Task. You will see the task added to the list below with a date and remove button.

Let's press on one of the tasks to mark it as completed:

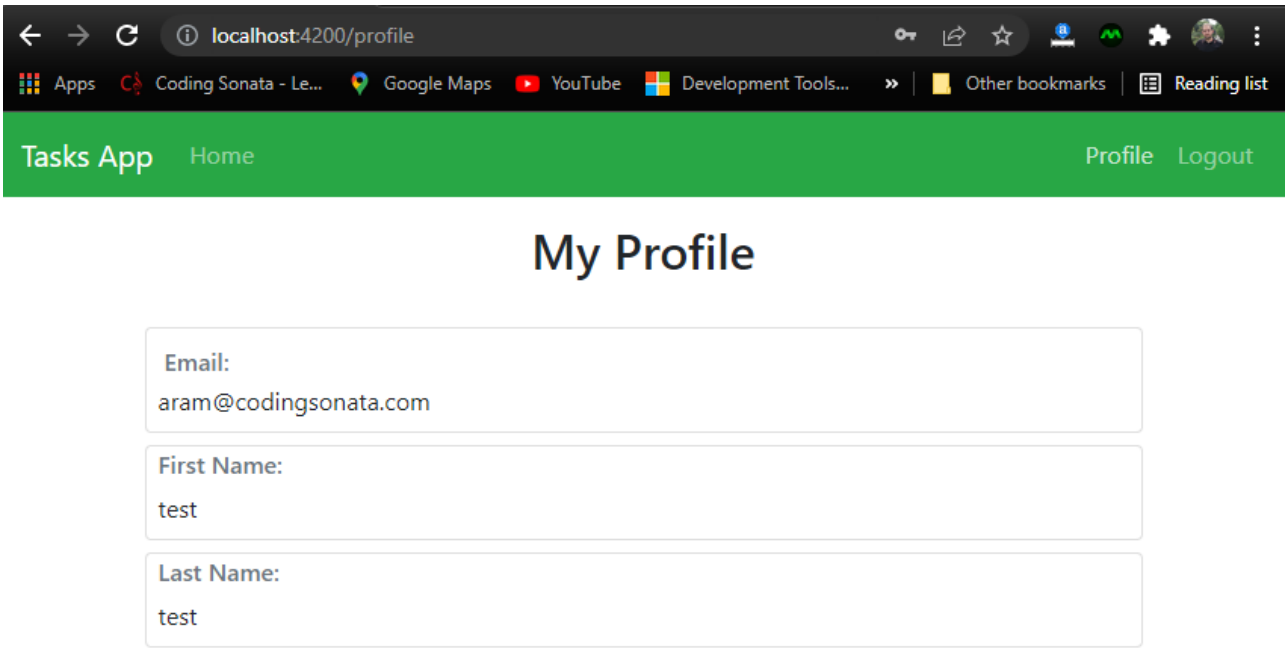


Clicking on the first task, it got marked as completed and therefore its style has changed. Of course, when it comes to CSS and styling you have an open-ended possibilities of what you can achieve to perfect your design.

So let's now remove a task:



That's great. We have a functioning tasks management page here. Let's open Profile to see how the profile will look like:

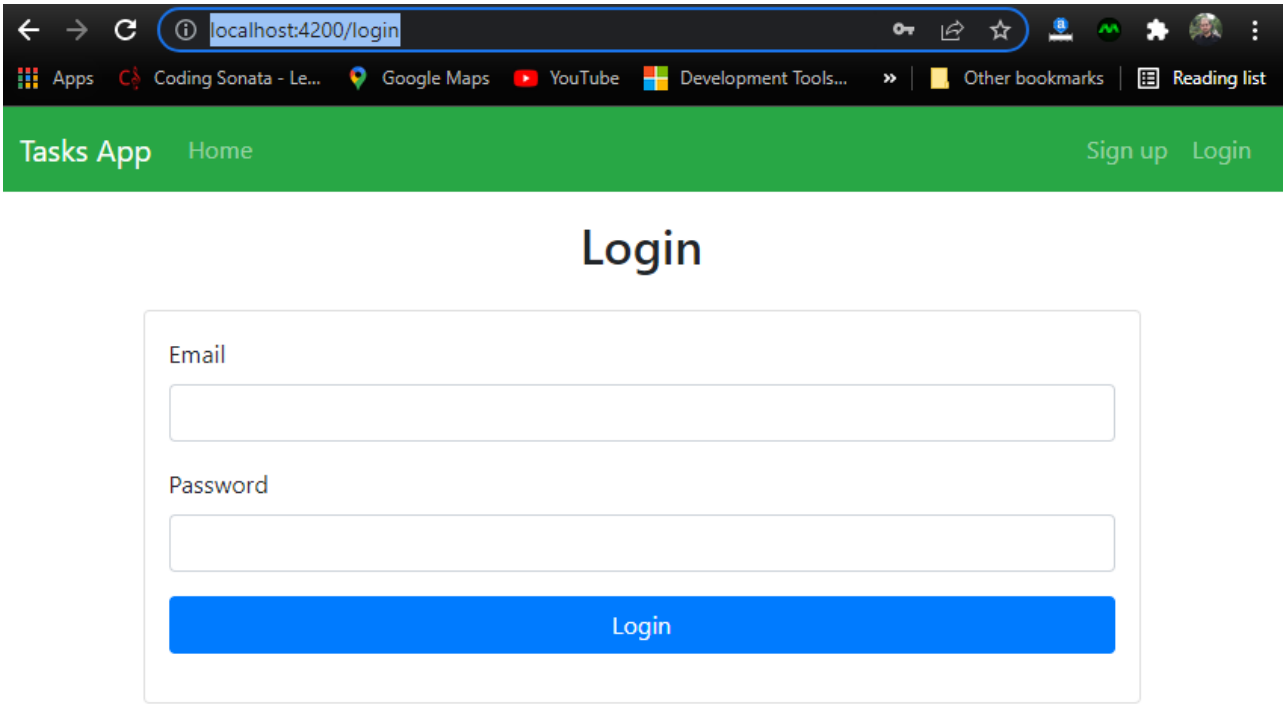


Of course you can continue this part by adding a proper form with fields and validation instead of just plain view of the data and allow the users to update their profile.

Now let's logout from here:

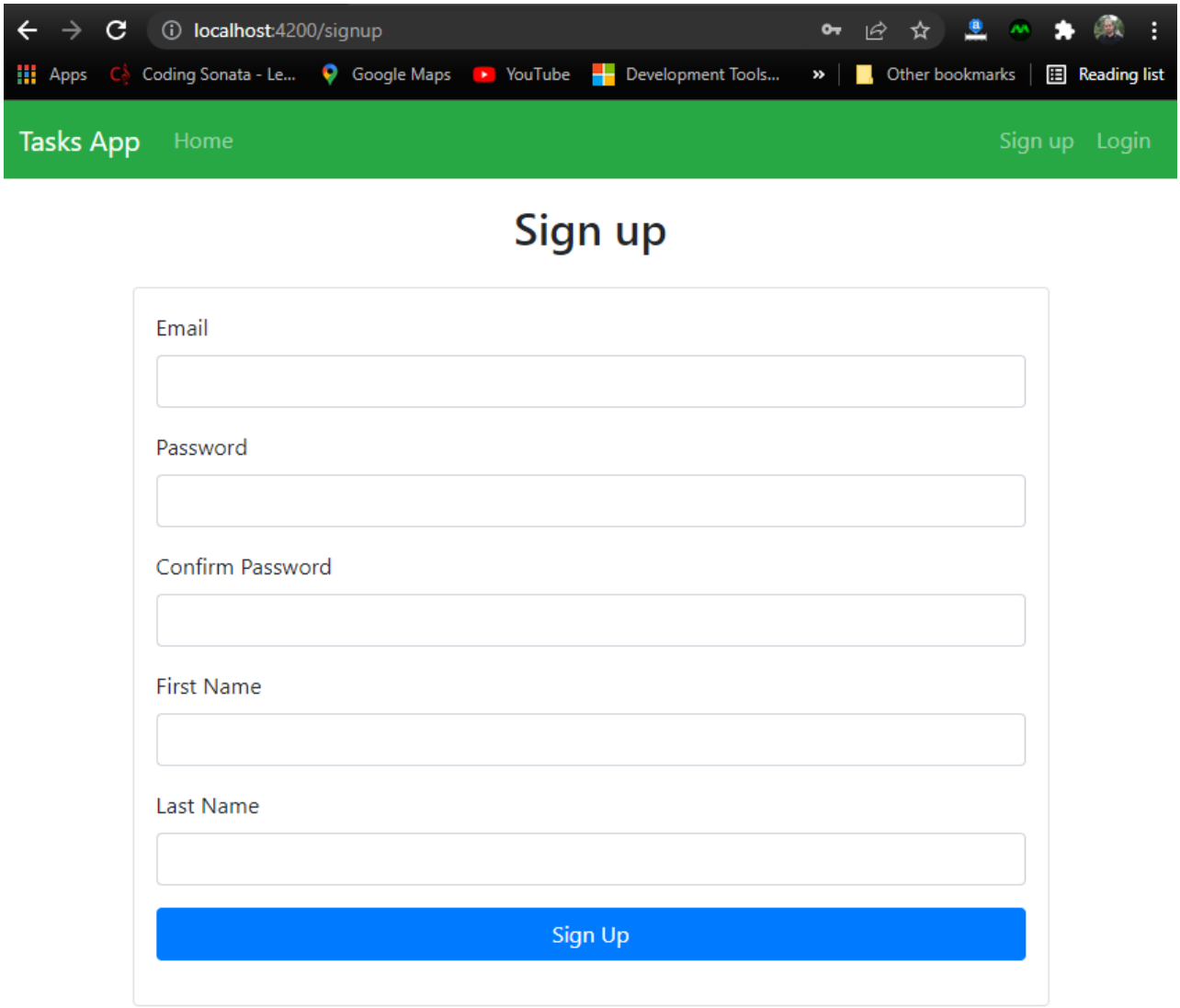
The logout button redirected us to the login screen and cleared the previous session.

Let's try to open the profile page to see if we can access it: http://localhost:4200/profile

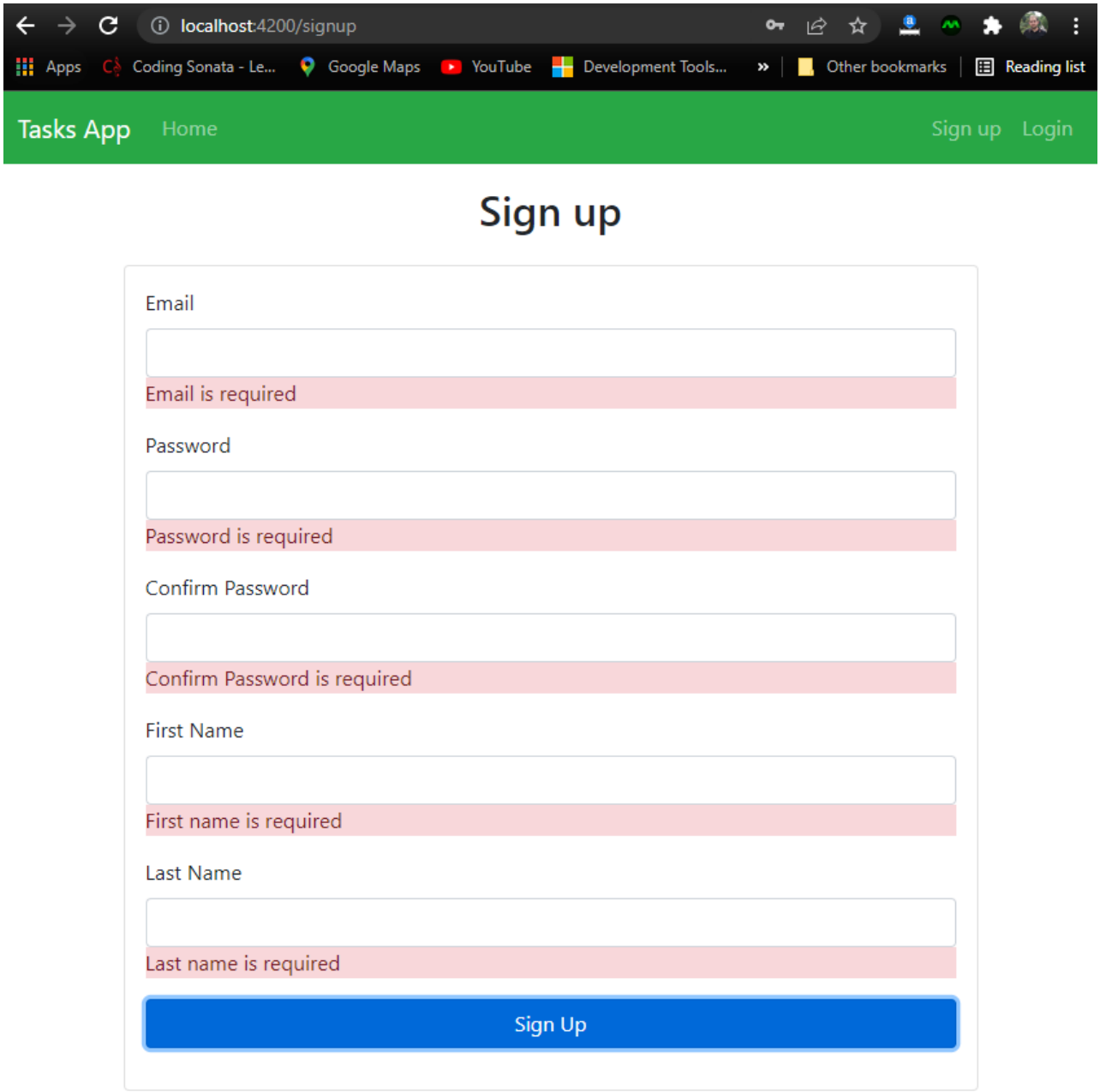


You will be redirected to the login screen again, since it is guarded for only authorized access.

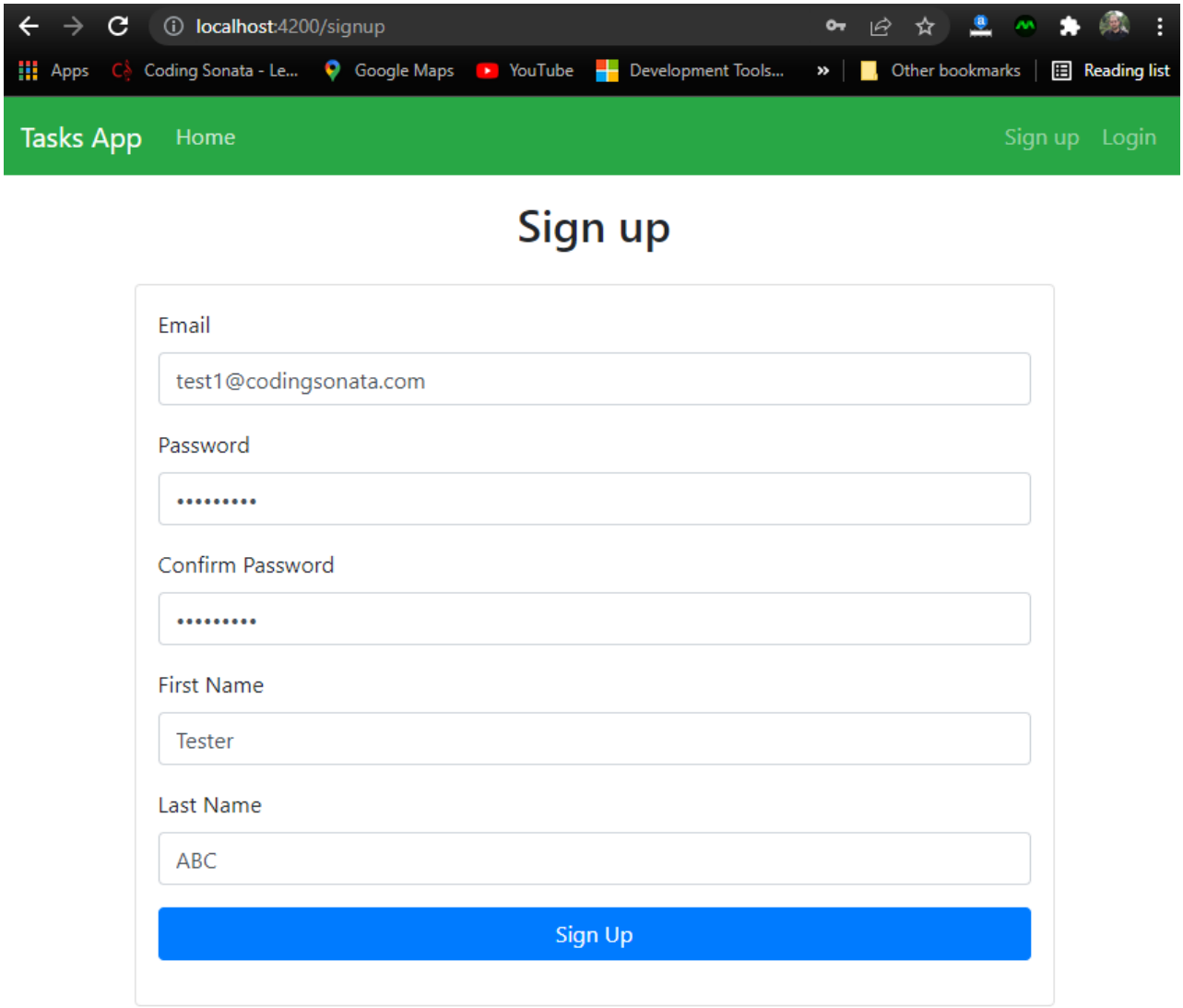
Now let's see the signup screen:



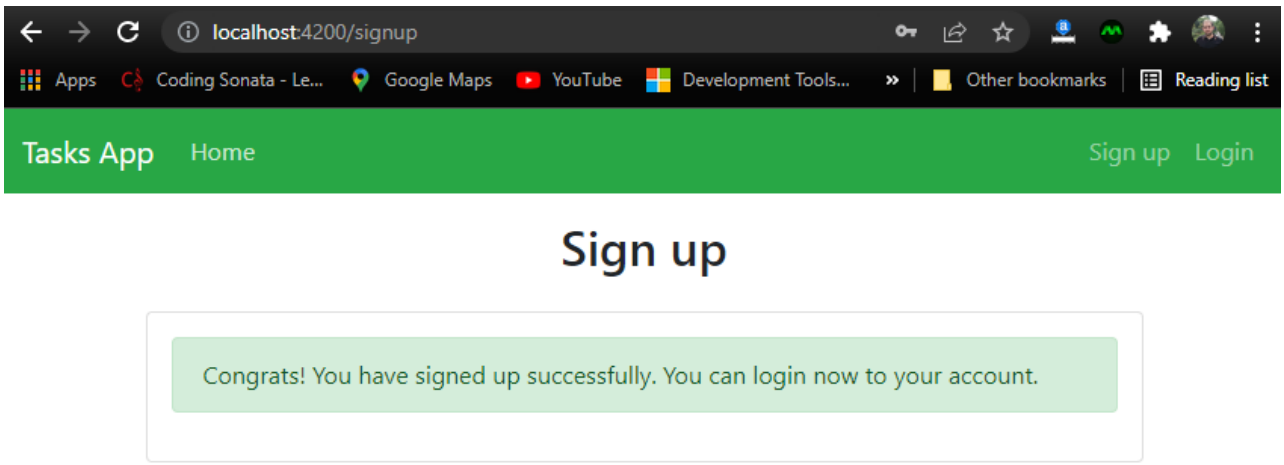
Now let's try to sign up without entering any field:



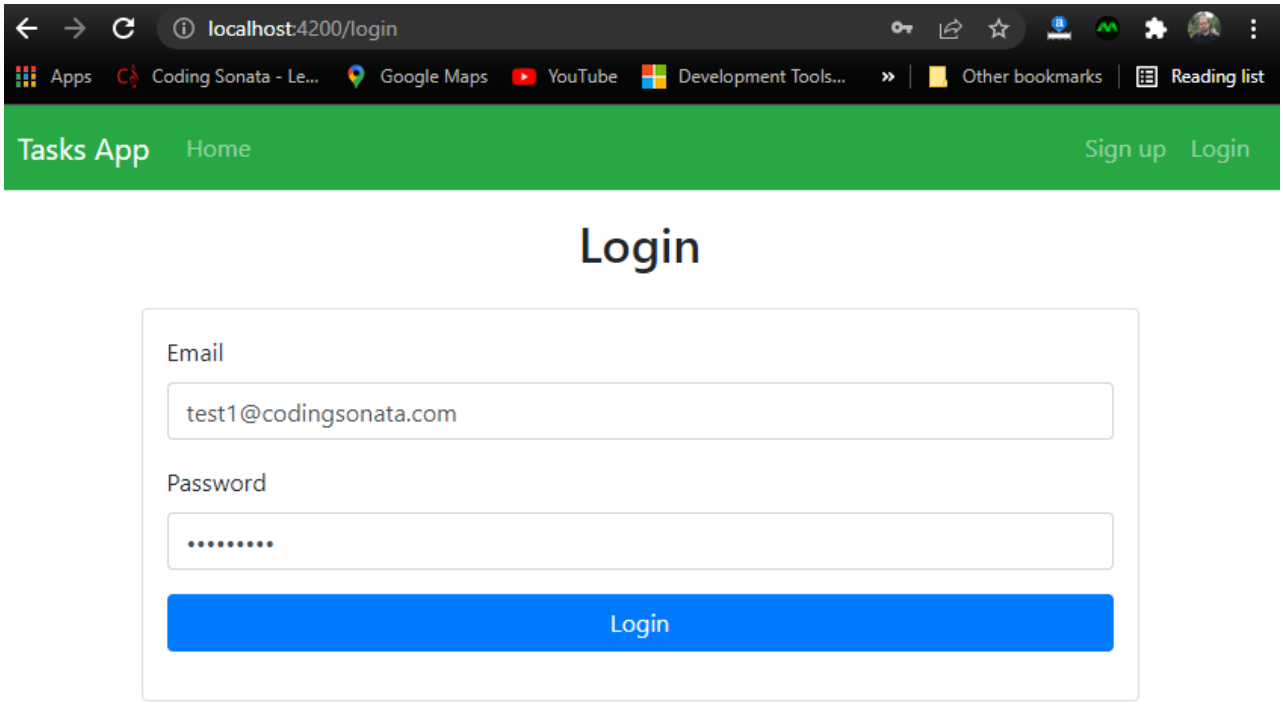
Now let's enter some valid data:



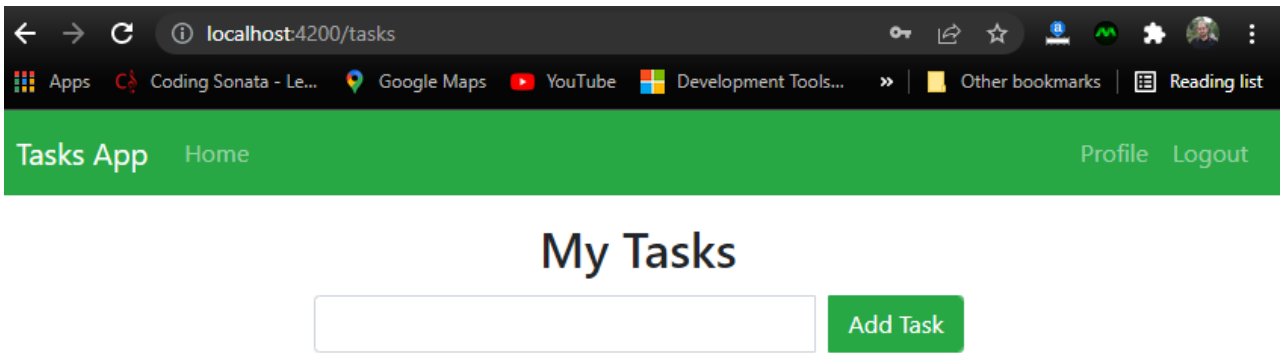
The signup form will disappear and you will get a success message:



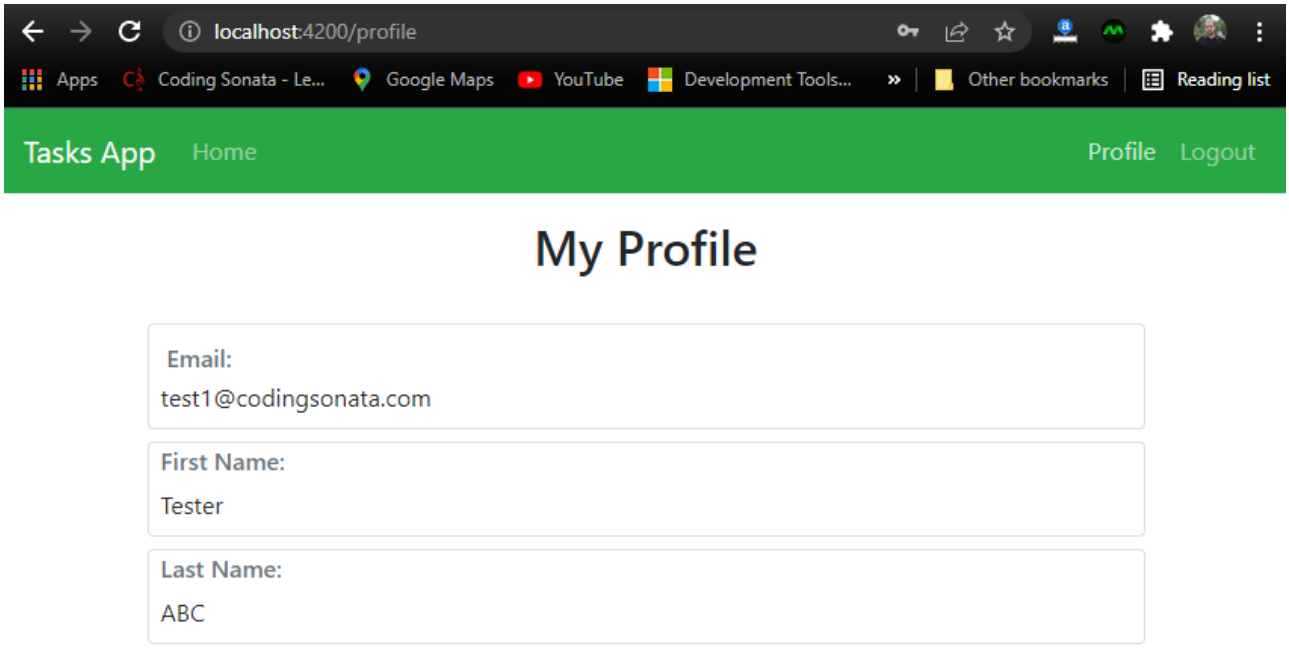
Now let's try to login using the new email and password:



We will be redirected to the tasks screen with no tasks obviously.



And the profile will show the details for the new user:



To test the access token and refresh token, first you need to change the ExpiryDate property on the API project to relatively small numbers like 1 minute for the access token and 2 minutes for the refresh token, and see how the access token will be silently refreshed.

And if the refresh token is expired, you will notice the site will be automatically redirected to login screen.

Summary

In this tutorial we have learned how we can build and secure Angular site using JWT Authentication with Access Tokens and Refresh Tokens. We built components for login, signup, profile, tasks and a function to logout. we added services to connect to the backend APIs previously built with ASP.NET Core Web API which already has the JWT authentication implemented in it. We added a local service to manage storing and retrieving the tokens for the user on the site.

And finally we have run a quick test on the site to verify the behaviour of all the functions and actions.

References

The API project is hosted in GitHub and you can find the tutorial for it here: Apply JWT Access Tokens and Refresh Tokens in ASP.NET Core Web API 6

You can find the Angular project for this tutorial hosted in GitHub as well.

If you want to learn more about Angular, I would highly recommend you take the tour of heroes tutorial, it is the best way you can dig into the Angular world.

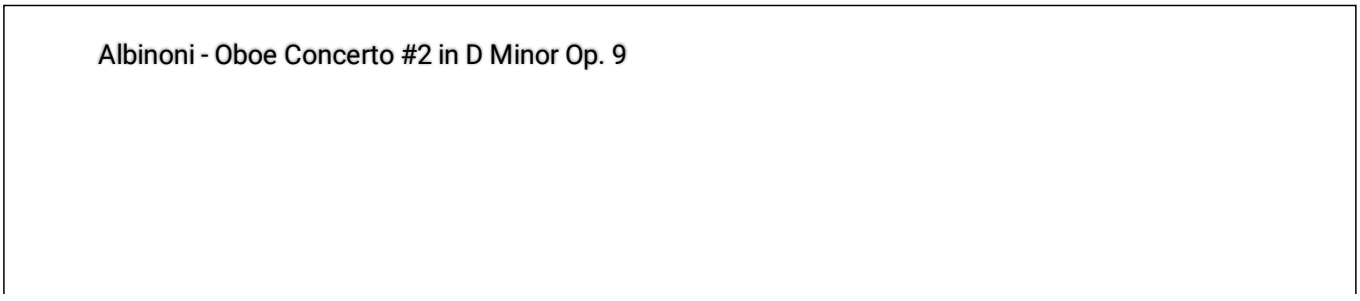
For ASP.NET Core Web API, you can check some of my other tutorials:

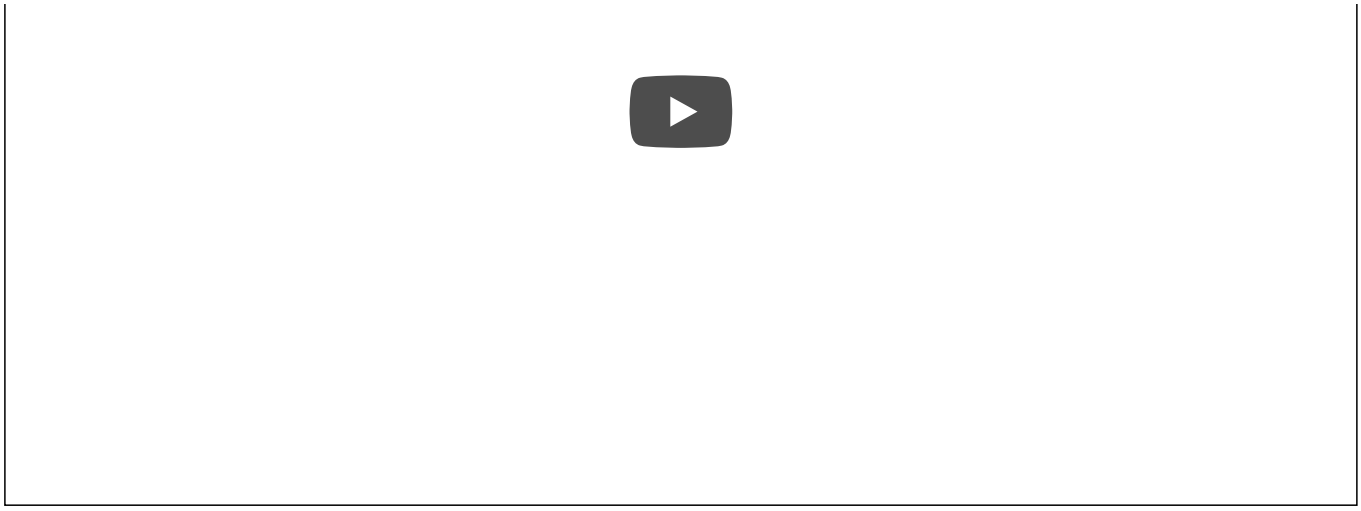
- Localization in ASP.NET Core Web API
- Google reCAPTCHA v3 Server Verification in ASP.NET Core Web API
- Exception Handling and Logging in ASP.NET Core Web API
- Build RESTful APIs Using ASP.NET Core and Entity Framework Core

Bonus

I would dedicate this musical piece for my avid readers and selective music listeners, enjoy and be safe wherever you are!

Albinoni – Oboe Concerto #2 in D Minor Op. 9





PREVIOUS POST
Localization in ASP.NET Core Web API

NEXT POST
Logging with Serilog in ASP.NET Core Web API

View all posts by →

YOU MIGHT ALSO LIKE

Secure ASP.NET Core Web API using API Key Authentication

February 21, 2021

Localization in ASP.NET Core Web API

February 2, 2022

.NET Articles of December – Week #4

December 29, 2020

Leave a Reply

Enter your comment here...

ML-SQL Server

GitHub/MindsDB: In-Database Machine Learning

MindsDB

Lear

SUBSCRIBE WITH EMAIL

Enter your email to receive notifications of new posts.

Email Address

SUBSCRIBE

FOLLOW ME

SUPPORT MY BLOG

Buy me a coffee

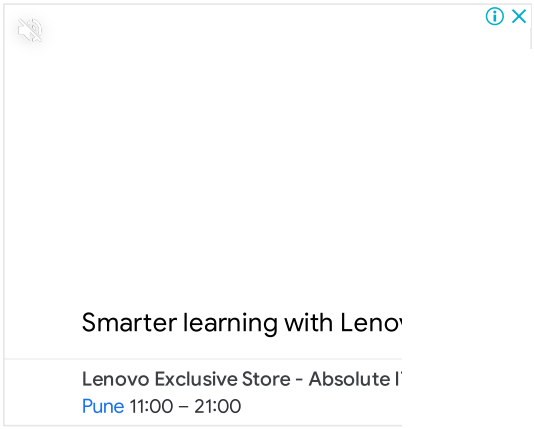
Make learning fun with Len

Lenovo Exclusive Store - Absolute I

Pune 11:00 – 21:00

jooble

Your leading jobs search engine



RECENT POSTS

Logging with Serilog in ASP.NET Core Web API

Secure Angular Site using JWT Authentication with ASP.NET Core Web API

Localization in ASP.NET Core Web API

Google reCAPTCHA v3 Server Verification in ASP.NET Core Web API

Apply JWT Access Tokens and Refresh Tokens in ASP.NET Core Web API 6

CATEGORIES

.NET (16)

.NET Core (17)

Android (7)

Angular (2)

ASP.NET Core (15)

Blogging (4)

C# (9)

Career (3)

javascript (1)

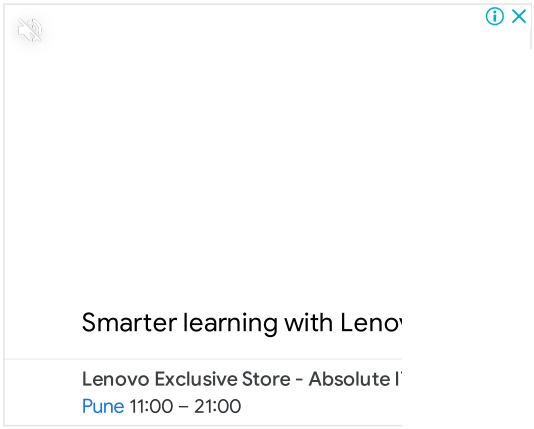
React (1)

self-development (3)

SQL (2)

TAGS

.NET .net 5 .net5 acceptance android approval asp.net core asp.net core web api asp.net web api aspnetcore async asynchronous await blog blogging C# coding debugging deploy development dotnet dotnet5 dotnetcore dump entity framework core exception handling google adsense Google reCAPTCHA v3 iis join localization log4net logging productivity retrofit self-development sql sql-joins task visual studio 2019 vs2022 webapi weekly WordPress.com WordPress.org



Email Address

SUBSCRIBE

Copyright © 2022 Coding Sonata. Powered by WordPress and Bam.