# Events Talent Week

## Summary

This is the final report about my Web Frameworks project. The project covers all the labs done during the semester.

I added pieces of code from the project files to prove the development process is complete for the whole project. There are some comments in the code.

I attached several screenshots throughout the document in order to prove the result of the work.

And I added the github link to my repository.

## Table of contents

Introduction

Software Requirements

Procedure

Repository link

API

Angular component

Security

Project working

# 

# Introduction

This project is about creating a web application using the MVC (Model View Controller) architecture.

The project starts with a home page, where two links come up. One to register a new member in the system, and a second one to log in the system with an existing username and the corresponding password.

After registering, Users return the home page, and are able to log in.

Users will access the login page, and after entering a valid username and password, they will move to the home page. Home page contains two different links: to ‘logout’ and access to the ‘List of events’. If users enter an invalid username and password, the web page will keep asking for username and password.

IN the ‘List of events’, users will be able to see up to 10 events stored in the MongoDB database.

When users click ‘logout’, they will return to the home page with the original links: ‘register’ and ‘login’.

# Software Requirements

I use MongoDB, Express, Angular and Node JS (MEAN).

First of all, I install Node JS. After that, I installed the Node modules, Express, MongoDB and all the packages I needed with “npm install” command.

I created my MongoDB Atlas account to host my non SQL database.

# Procedure

On the root folder I created a folder called “app\_server” where I moved the views and “routes” folders. I also created two new folders, “models” and “controller”. All of this, to respect the MVC architecture.

I changed the new location for the “routes” folder in the “app.js” file.

const indexRouter = require('./app\_server/routes/index');

I created the web pages needed for the project. First, I made the pages in HTML, and then I used the “pughtml.com” tool to convert them into PUG.

// PUG code for the log in web page

To generate PUG code, first I made the HTML code for the page, and then I used the “pughtml.com” web tool to convert it.

head

title

| #{title}

h1 Tralee Talent Event

br

h2 Log in

br

p

| Log into the system to access the list of events to be attended

br

table

tr

td E-mail:

td

input(type='text', title='email')

tr

td Password:

td

input(type='password', title='password')

br

input(type='submit', value='Log in')

| &quot;

// End of code

I created a controller file, where I specify the pages to be open and the URL corresponding to each of them.

// log in page controller

const login = function(req, res){

res.render('login', {

title: 'Log In'

});

};

// end of code

I added on the “index.js” file the controllers for each page

router.get('/events', ctrlPage.events);

In the previous step to connect to the database, I created an array with 10 records of events manually to prove that the communication between the controller and the pages is correct.

// code for the array

title: 'Events' ,

event: [{

name: 'Ghaelach Football',

hour: '12:30',

day: '01/DIC/2022',

location: 'Austin Stacks Park',

town: 'Tralee'

}

// end of code

To make the PUG file generate the 10 records, I use a ‘for each’.

// code of the for each

each e in event

tr

td #{e.name}

td #{e.hour}

td #{e.day}

td #{e.location}

td #{e.town}

// end of code

I created my MongoDB atlas account to host my non SQL database on the cloud. I installed the MongoDB shell on my laptop, and I am able to connect to the cloud with my credentials.

mongosh "mongodb+srv://secaiv:A1kalina@cluster0.hwsu78d.mongodb.net/events?retryWrites=true&w=majority"

I created my database and I added a couple of records.

A screenshot of a computer

Description automatically generated with medium confidence

In the “models” folder I created a “db.js” file to connect to the database

A picture containing application

Description automatically generated

I created a simple JSON schema for the API.

// code for the schema

const eventSchema = new mongoose.Schema({

name: String,

Hour: String,

day: String,

location: String,

town: String

});

// end of code

The project is users have to register to have access to the list of events that they can enjoy in the “Events Talent Week”.

There is a registration page, where users will have to enter the required details in order to have access to the system.

* Forename
* Last name
* Password
* Address
* Phone
* E-mail
* Date of Birth

Once users are registered, they will have the option to log into the system.

When they are in, users will be able to see all the list of events available for the festival along with their details.

* Name
* Hour
* Date
* Location
* Town

# Repository link

Github link:

<https://github.com/Segade/wf_project/> (new link)

# API

For the API, I create a new folder called “app\_api” in the root project folder. Within, I created some new folders called ‘models’, ‘controllers’ and ‘roots’.

I had to change the ‘app.js’ file to get the project worked with the section. Therefore, I specify the path to the new index file in the ‘app\_api’.

// code in the app.js

const apiRoutes = require('./app\_api/routes/index');

app.use('/api', apiRoutes);

In the ‘roots’ folder, I added an ‘index.js’ file to handle the different URLs and the operations to execute (get or post).

// index file

const express = require('express');

const router = express.Router();

// import the controller functions for events and members

const ctrlRegistration = require('../controllers/registration');

const ctrlMember = require('../controllers/members');

// Events

Router

// if the URL has a parameter for the event

.route('/events/:eventid')

// using the get method, to retrieve one event using the ID

.get(ctrlRegistration.eventsRead);

// if the URL doesn’t have a parameter for the events

.route('/events')

// using the get method to retrieve all the events

.get(ctrlRegistration.eventsList);

Router

// if the URL doesn’t have a parameter for the members

.route('/members')

// using the get method to retrieve all the members

.get(ctrlMember.membersList)

// using the post method to insert a new member in the database

.post(ctrlMember.memberCreate);

Router

// if the URL has a parameter for the members

.route('/members/:memberid')

// using the get method to retrieve a member using the parameter

.get(ctrlMember.memberRead);

module.exports = router;

In the ‘models’ folder, I create the files for the schemas of the events and members, with one file for each.

// events schema

const mongoose = require('mongoose');

const eventSchema = new mongoose.Schema({

name: {

type:String,

required: true

},

Hour: {

type: String,

required: true

},

day: {

type: String,

required: true

},

location: {

type: String,

required: true

},

town: {

type :String,

required: true

}

});

mongoose.model('Event', eventSchema);

// member schema

const mongoose = require('mongoose');

const memberSchema = new mongoose.Schema({

name: String,

surname: String,

address: String,

phone: String,

email: String,

password: String,

dob: {

type: Date,

'default': Date.now

}

});

mongoose.model('Member', memberSchema);

In the ‘controllers’ folder, I created one js file for events called ‘ registration.js’ and a js file for members called ‘members.js’. These files will handle the different requests for the getters and the post for each schema.

// registration file (for events)

const mongoose = require('mongoose');

// the below variable is to link the schema with the controller

const event = mongoose.model('Event');

// function to retrieve all the events

const eventsList = function(req, res) {

console.log('in eventsList');

event

.find() // command to retrieve the events from the database

.exec((err, ev) => { // it stores the result in the ‘eve’ variable

res

.status(200)

.json(ev);// success, I get the JSON object

});

};

///// function to retrieve one event by object ID

const eventsRead = function (req, res) {

if (req.params && req.params.eventid) { // no parameter

event

.findById(req.params.eventid) // command to retrieve the specific event

.exec((err, ev) => { // it stores the result in the ‘eve’ variable

if (!ev) {// if there is no result

res

.status(404)

.json({

"message": "eventID not found"

});

return;

} else if (err) {

res

.status(404)

.json(err);

return;

}

res // success, there is a result

.status(200)

.json(ev);// it returns the JSON object

});

} else {

res

.status(404)

.json({

"message": "No eventid in request"

});

}

};

module.exports = {

eventsList,

eventsRead

} ;

// Members

const mongoose = require('mongoose');

// variable to link the member schema with the controller

const Member = mongoose.model('Member');

// post method to add a member to the database

const memberCreate = function(req, res) {

Member.create({ // command to insert

// the parameters to be used from the body

name: req.body.name,

surname: req.body.surname,

address: req.body.address,

phone: req.body.phone,

email: req.body.email,

password: req.body.password,

}, (err, Member) => {

if (err) { // problems with the insert

res

.status(400)

.json(err);

} else {

Res // success

.status(201)

.json(Member);

}

});

};

// function to list all the members

const membersList = function(req, res) {

console.log('in membersList');

Member

.find() // command to retrieve all the members from the database

.exec((err, mem) => { // it stores the result in the ‘mem’ variable

res // success

.status(200)

.json(mem); // it returns the JSON with the members

});

};

// function to retrieve one member by ID

const memberRead = function (req, res) {

if (req.params && req.params.memberid) {// no parameters

Member

.findById(req.params.memberid)// command to retrieve the specific member

.exec((err, mem) => {// it stores the result in the ‘mem’ variable

if (!mem) {// no result

res

.status(404)

.json({

"message": "memberID not found"

});

return;

} else if (err) {// if there is some kind of error

res

.status(404)

.json(err);

return;

}

res

.status(200)

.json(mem);

});

} else {

res

.status(404)

.json({

"message": "No memberid in request"

});

}

};

module.exports = {

memberCreate,

memberRead,

membersList

};

## Testing the API

### Getters of list of records screenshots

Events:

Graphical user interface, text, application, email

Description automatically generated

Members:

Graphical user interface, text, application, email

Description automatically generated

Getters of one record screenshot

Events:

Graphical user interface, text, application

Description automatically generated

Members:

Graphical user interface, text, application, email

Description automatically generated

### Post

Members:

Graphical user interface, text, application, email

Description automatically generated

# Angular component

First of all the Angular CLI must be installed with command:

npm install -g @angular/cli

I created the folder for the Angular project. For my project I called it "ng\_project". In this folder, I created a new Angular project with the command:

ng new events

The new Angular project works properly. Therefore, after all the Lab1 testings I started to develop my own project.

I create a new Angular component for my own project with the command:

ng generate component events

In the app-module.ts file, I set my "app-events" component as the base for the Angular project.

// code

@NgModule({

declarations: [

EventsComponent

],

I change the selector in the "index" file to the "app-events" tag.

// code

<app-events></app-events>

I paste in my events-component.html file, the HTML code to start the process.

// code

<table> <tr>

<th> Name </th> <th> Hour </th> <th> Day </th> <th> Location </th> <th> Town </th>

</tr> <tr>

<td> </td> <td> </td> <td> </td> <td> </td> <td> </td>

</tr> </table>

At this point, the table comes up when I run the Angular project.

In the "events-component.ts" file, I create a new class with the schema of the data I am going to display in my Angular project.

// code

export class EventsList{

name!: string;

hour!: string;

day!: string;

location!: string;

town!: string;

}

First, I declare a single variable with only one record, which is displayed in the browser. After this, I convert the variable in an array with several records, which are displayed too. (I do not write the whole process, because it is not relevant for the final project)

At this stage, the following step is to connect the Angular project to the database and bind the EventsList class with the "events" MongoDB collection through my API. For this purpose, I create a service with the command:

ng generate service events-data

The "events-data" service is created in the "app" folder.

I configure the service properly for working with the project. I can do "http" request and return a Promise.

// code

import { Injectable } from '@angular/core';

import { HttpHeaders, HttpClient } from '@angular/common/http';

import { EventsList } from './events/events.component';

@Injectable({

providedIn: 'root'

})

export class EventsDataService {

constructor(private http : HttpClient) { }

private apiBaseUrl = 'http://localhost:3000/api/';

public getEvents(): Promise<EventsList[]> {

const url: string = `${this.apiBaseUrl}/events `;

return this.http

.get(url)

.toPromise()

.then(response => response as EventsList[])

.catch(this.handleError);

}

private handleError(error: any): Promise<any> {

console.error('Something has gone wrong', error);

return Promise.reject(error.message || error);

} // end get events

} // end of file

At this stage, the service is connected to the API properly, and gets the list of events from the database. I just need to bind the service with the "events" component to display the data from the database in the browser. For this, I have to import, inject and provide the service in the component.

// code

import { Component, OnInit } from '@angular/core';

import { EventsDataService } from '../events-data.service';

export class EventsList{

name!: string;

hour!: string;

day!: string;

location!: string;

town!: string;

}

@Component({

selector: 'app-events',

templateUrl: './events.component.html',

styleUrls: ['./events.component.css'],

providers: [EventsDataService] // provides the service

})

export class EventsComponent implements OnInit {

// injection of the service

constructor(private eventsDataService : EventsDataService) { }

eventsList: EventsList[] = []; // arrays that stores the data

private getEvents () : void {

this.eventsDataService

.getEvents()

.then(foundEvents => {

this.eventsList = foundEvents;// the data from the database is stored in the eventsList array

}); // end then function

} // end get events

ngOnInit() {

this.getEvents(); // the service is called after it is available

}

}

The "app-module.ts" file looks like this:

// code

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpHeaders, HttpClientModule } from '@angular/common/http';

import { EventsComponent } from './events/events.component';

@NgModule({

declarations: [

EventsComponent

],

imports: [

BrowserModule,

HttpClientModule

],

providers: [],

bootstrap: [EventsComponent]

})

export class AppModule { }

The Angular project is up and running properly. Now it is time to build the Angular project to integrate it in the Express application with the command.

ng build events

In the Express root folder, I create a new folder called "app\_public" with a subfolder called "build". Here I copy the polyfills, main and runtime files from the "dist" folder.

To be able to get the Express working with Angular, I open the Angular port in the Express application.

For this purpose, I add the following code to the "app.js" file

// code

app.use('/api', function(req, res, next) {

res.header('Access-Control-Allow-Origin', 'http://localhost:4200');

res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');

next();

});

app.use('/', index);

app.use('/api', apiRoutes);

I

I add a new root for a static resource for the "app\_public" in the "app.js".

app.use(express.static(path.join(\_\_dirname, 'app\_public')));

I add in the "/app\_server/views/layout.pug" to add new scripts.

block scripts

Finally, I add all the files to the "events.pug" file and I call the Angular component in the file.

// code

doctype html

head

title= title

h1 Tralee Talent Event

br

h2 Events Registration

br

p

| Select and register on the event you want to attend

br

app-events Loading

block scripts

script(src='/build/runtime.23a3ddd7ef4ebdb2.js')

script(src='/build/polyfills.d9946f31cba37647.js')

script(src='/build/main.18c7585cda16fdf9.js')

# The API in the frontend

For my project, the API is integrated into my Angular component. The component uses the API method that retrieves all the events from the database and gets the JSONs in order to display them in the data page.

In the ‘events-data.service.ts’ file I call the proper API method.

// code in the events-data.service.ts

private apiBaseUrl = 'http://localhost:3000/api/';

const url: string = `${this.apiBaseUrl}/events `;

.get(url)

.toPromise()

.then(response => response as EventsList[])// the JSONs are stored in the EventsList

In the ‘events.components.ts file I declare the get method to retrieve the JSONs and pass it to the ‘app-module.ts’

// code events.components.ts

export class EventsComponent implements OnInit {

constructor(private eventsDataService : EventsDataService) { }

eventsList: EventsList[] = [];

private getEvents () : void {

this.eventsDataService

.getEvents()

.then(foundEvents => {

this.eventsList = foundEvents;

Finally, in the ‘events.componenets.html’ file I loop the eventsList array with all the events with the ‘ngfor’ statement

// code for events.componenets.html

<tr \*ngFor="let eachEvent of eventsList">

<td>{{eachEvent.name}} </td>

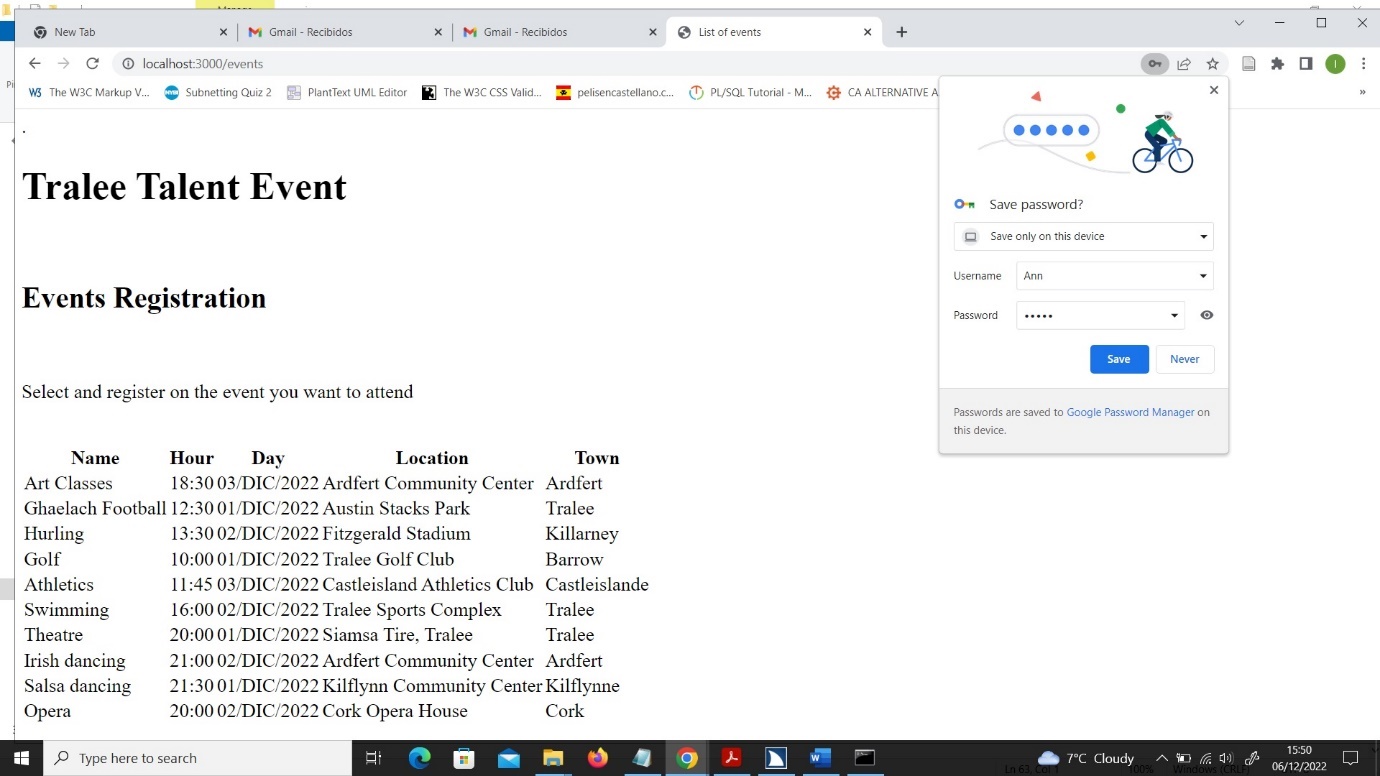
<td>{{eachEvent.hour}} </td>

<td>{{eachEvent.day}} </td>

<td>{{eachEvent.location}} </td>

<td>{{eachEvent.town}} </td>

</tr>



# Security

To do the passport part, I commenced installing all the dependencies that are in the passport lab document.

I use the same example used in class for my own project.

In the 'app.js' file I add the required lines of code to get the passport worked.

// code added in the app.js

var bodyParser = require('body-parser');

var mongoose = require('mongoose');

var passport = require('passport');

var LocalStrategy = require('passport-local').Strategy;

var flash = require('connect-flash');

var Account = require('./app\_server/models/account');

passport.use(new LocalStrategy(Account.authenticate()));

passport.serializeUser(Account.serializeUser());

passport.deserializeUser(Account.deserializeUser());

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: false }));

app.use(require('express-session')({

secret: 'keyboard cat',

resave: false,

saveUninitialized: false

}));

app.use(passport.initialize());

app.use(flash());

app.use(passport.session());

In the 'index.js' file in the 'app\_server' folder I added the post and get methods to register a new member and to check the database for the 'login' page.

// code in the index.js

// using the post method after pressing 'submit' in the 'login' page

// it access the database through the authenticator, since the password is encrypted.

router.post('/login', passport.authenticate('local', { failureRedirect: '/login', failureFlash: true }), (req, res, next) => {

req.session.save((err) => {

if (err) {

return next(err); // login failed

}

res.redirect('/');// success, it redirects to the home page

});

});

// using the post method to register a new member

router.post('/registration', (req, res, next) => {

// it creates anew member in the accounts collections

Account.register(new Account({ username : req.body.username }), req.body.password, (err, account) => {

if (err) {

return res.render('register', { error : err.message });

}

// the authenticator method creates new session

passport.authenticate('local')(req, res, () => {

req.session.save((err) => {

if (err) {

return next(err);

}

res.redirect('/');// it redirects to the home page

});

});

});

});

//logout

router.get('/logout', (req, res, next) => {

req.session.destroy();

res.redirect('/');

req.session.save((err) => {

if (err) {

return next(err);

}

res.redirect('/');

});

});

In my controller file, I have to set the 'user' variable to be able to keep it across the web site.

// code in my registration.js controller

// in the get method of the home page

user : req.user

//in the get method of the login page to inicialite the variable

user : req.user, error : req.flash('error'),

\*\*\*SSL

To add the SSL functionality to my project, first I made the corresponding modifications to the 'app.js' in order to allow 'https' in my project.

// code added to app.js

var fs = require('fs');

var http = require('http');

var https = require('https');

var privateKey = fs.readFileSync('./sslcert/key.pem', 'utf8');

var certificate = fs.readFileSync('./sslcert/cert.pem', 'utf8');

var credentials = {key: privateKey, cert: certificate};

const app = express();

var httpServer = http.createServer(app);

var httpsServer = https.createServer(credentials, app);

httpServer.listen(8000);

httpsServer.listen(443);

After this, I installed the OpenSSL application on my laptop.

I run the OpenSSL command line and execute the following commands to generate the certificate.

openssl genrsa -out key.pem

openssl req -new -key key.pem -out csr.pem

openssl x509 -req -days 9999 -in csr.pem -signkey key.pem -out cert.pem

During this process, I configure my certificate answering the question the application asks me.

Finally, I create a new folder in the project root folder called 'sslcert', and paste the corresponding files for the certificate.

csr.pem

key.pem

cert.pem

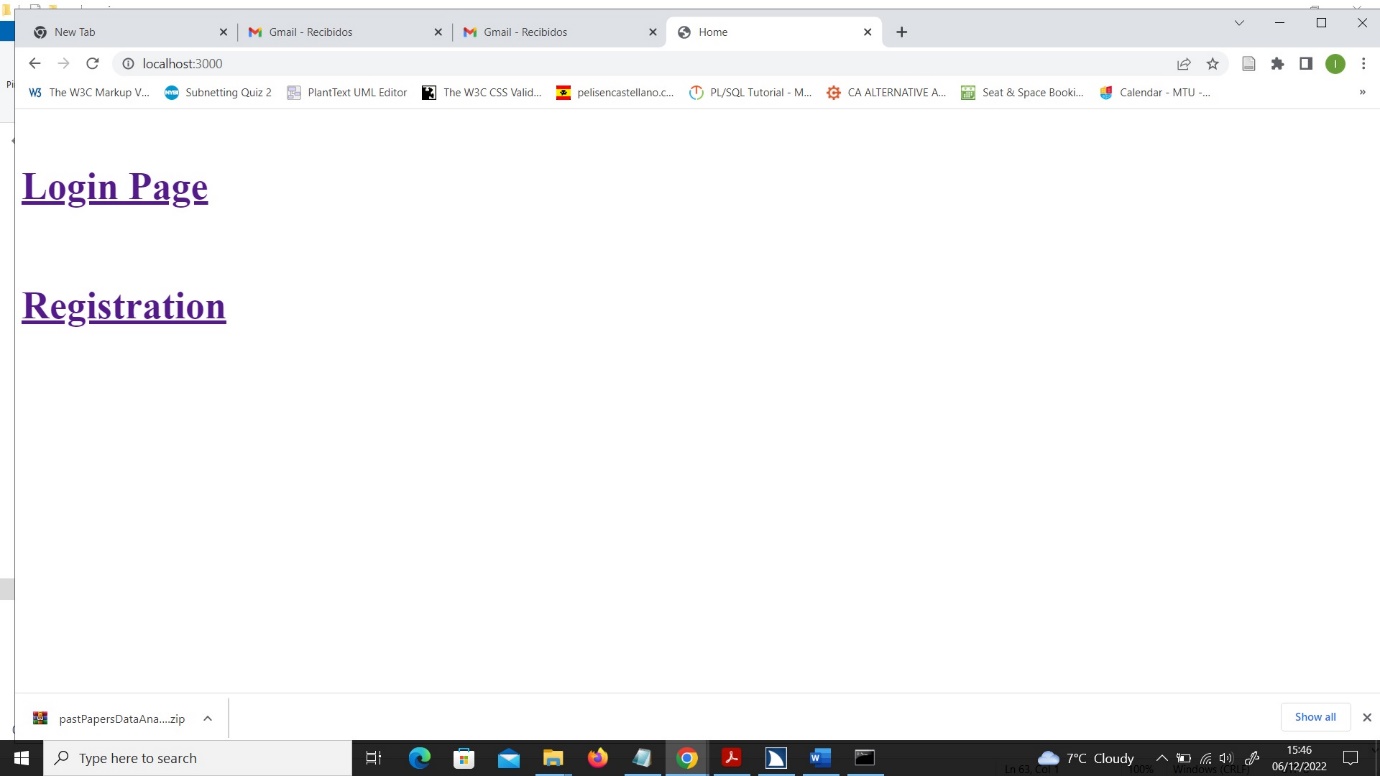
Now the project is ready to work with 'http'. I try to use the URL

https://localhost:443

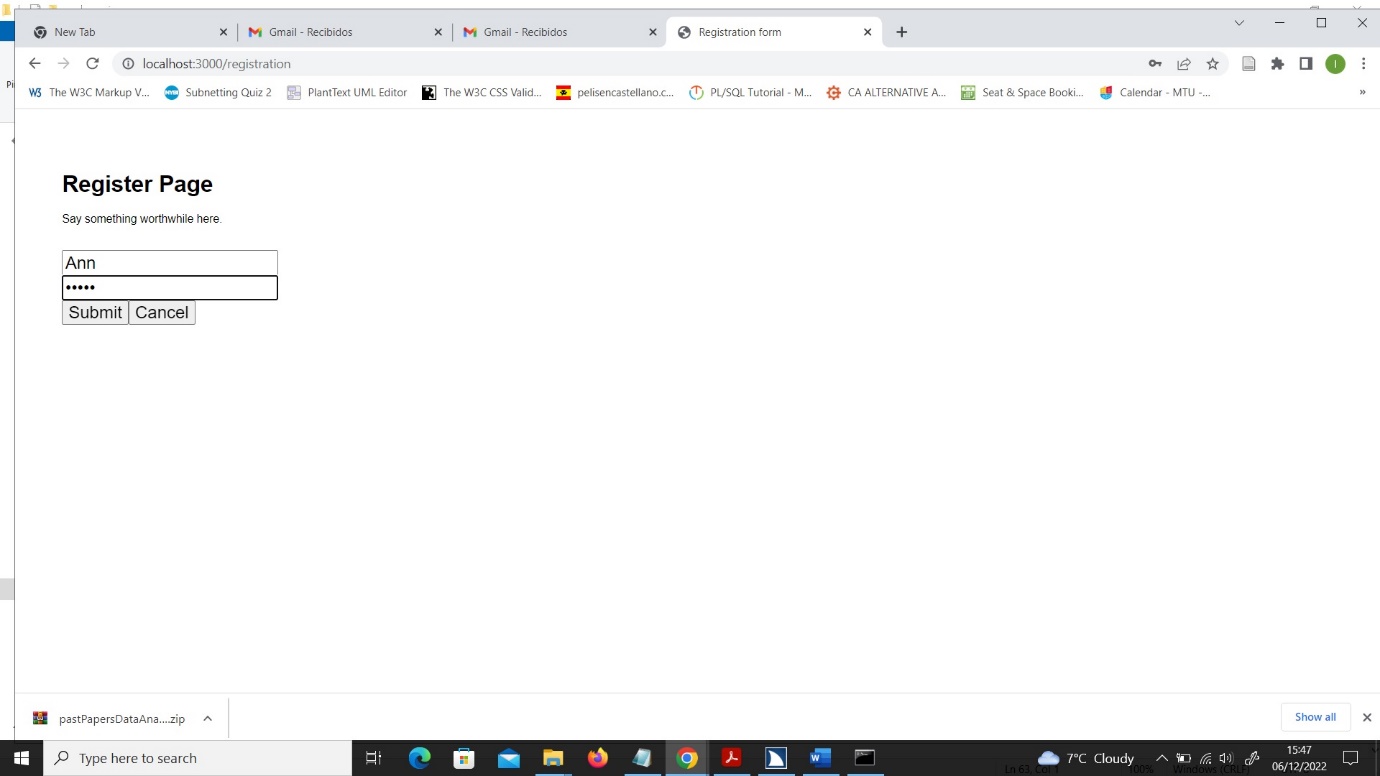
And after allowing the access to the web page I can work with my project using 'https'

# Project working

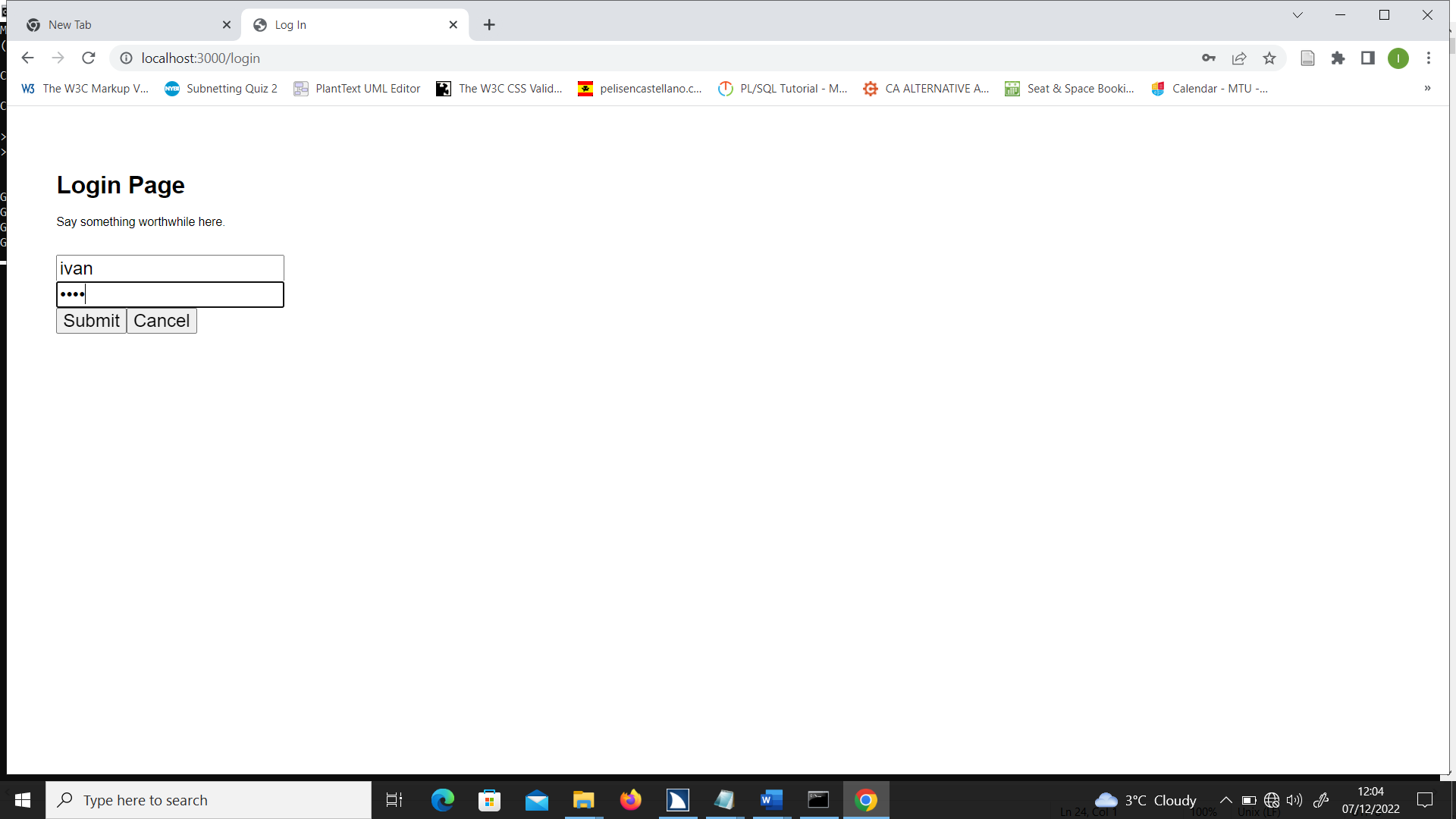
Home page:



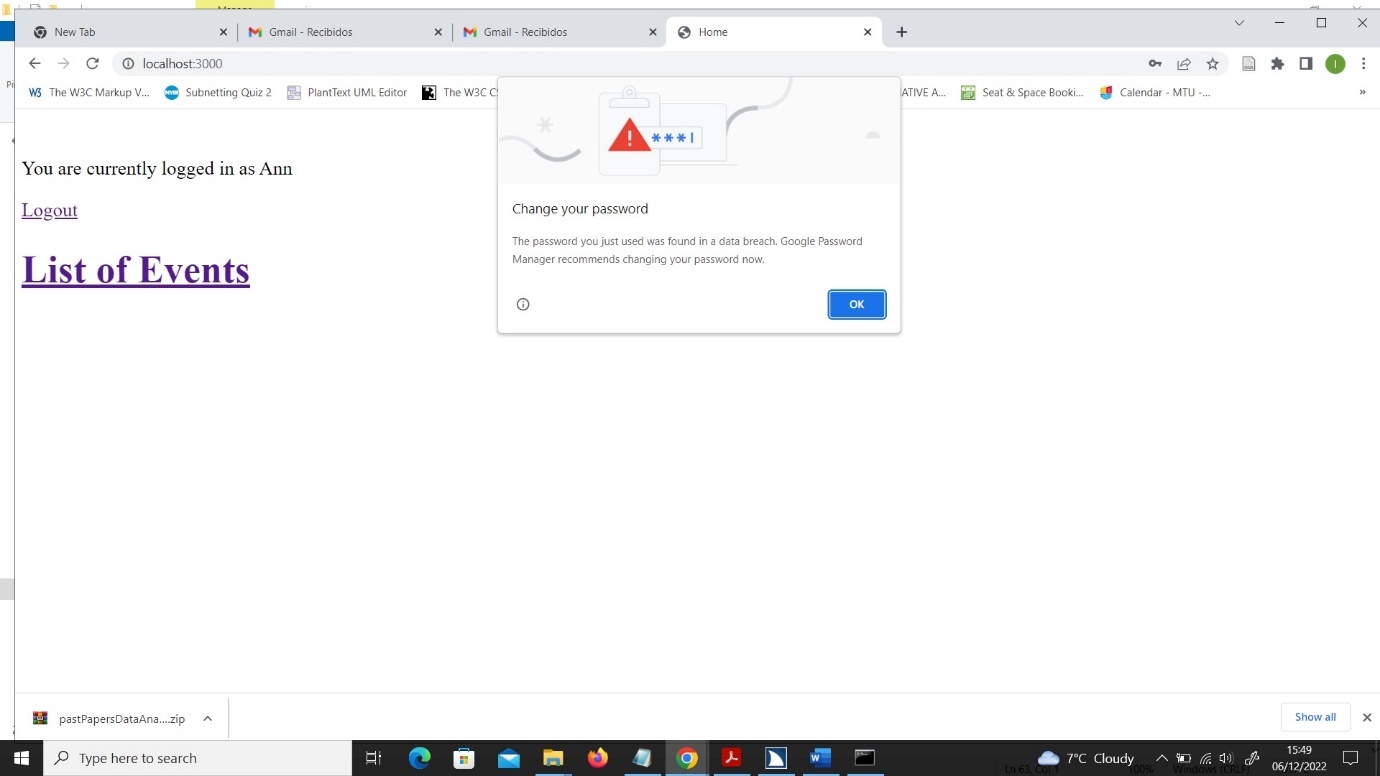
Registration page:



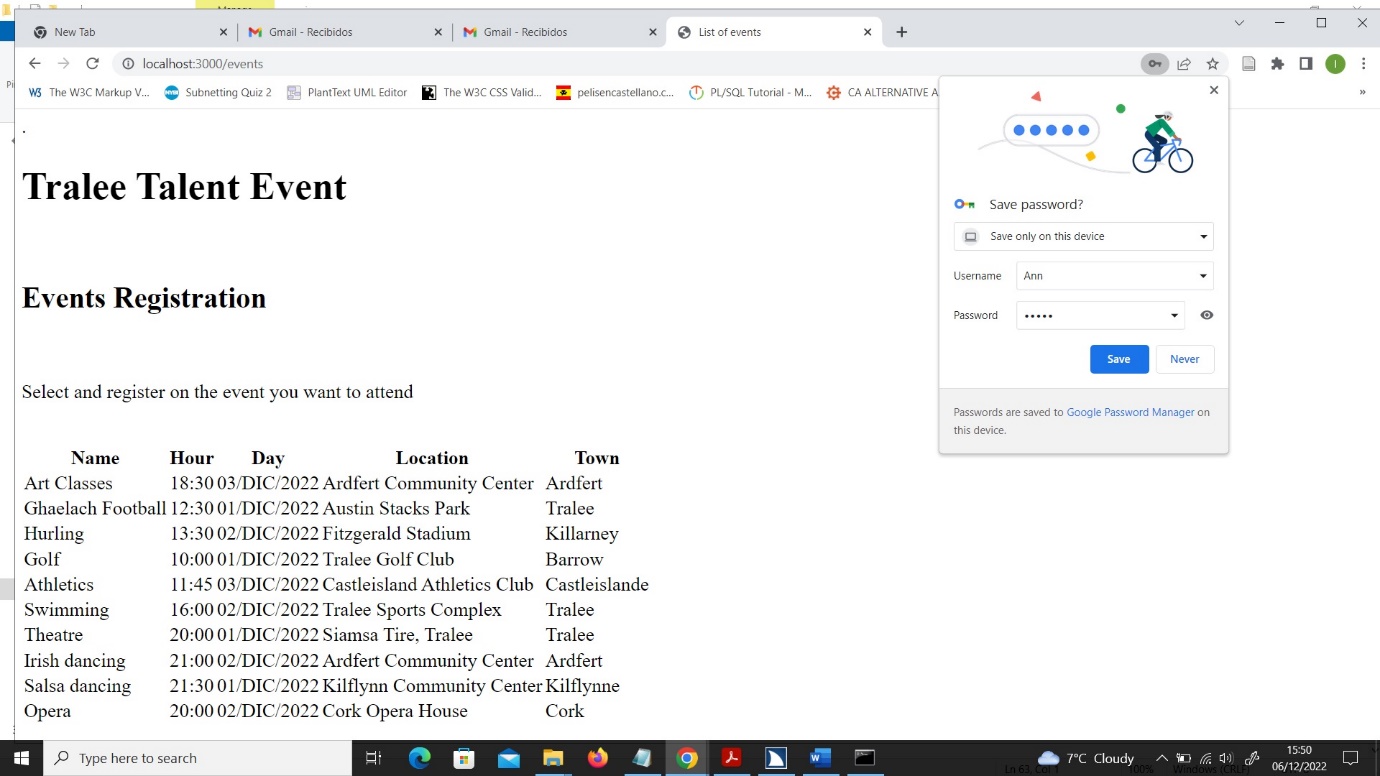
Login page:



Menu page:



Events page:



SSL example page:

