Client-Server Web Application

Изисквания към задачата:

* валидация на данните (мейл, имена и парола)
* записване на данните в релационна база данни (MySQL, PostgreSQL и т.н.).
* login/logout
* страница за промяна на имена и парола на потребителя
* пълно покритие с unit тестове (100% покритие на функциите в кода)
* кепча - да се напише като код, без използване на външен сървиз (CAPTCHA)

**Quickstart guide**

За да можете да стартирате апликацията нужно е да имате база данни която е активна в момент на стартиране или да имате инсталиран някой 3-rd party софтуер които ще симулира база данни както е например **XAMPP/MAMPP**, и да създадете базата данни която искате да ползвате със полетата: ID (PK), Username, Userpass, Usermail.

След това е нужно да се изменят данните които ще са необходими да се сървъра свърже със базата. Този конфигурационнен файл се намира във .**env**. Влезните в него и измените съществуващите данни с вашите.

На края стартирайте един терминал и напишете команда:

**npm start** които ще извърши командата **"start": "nodemon app.js"**

**Документация за приложението:**

За да направиме една клиент-сервър връзка трябва да имаме фронтенд и бекенд на нащто приложение. Аз ще реализирам този приложение със JavaScript Node JS MySQL (XAMPP) и други депенденцита която ще са ни необходими за този приложение като express, nodemon, mysql и т.н..

Стартираме VS Code и създаваме нов проект и със помощ на npm (Node JS) инициялизираме всички неохдими неща за да можеме да стартираме един сървър когато ползваме Express

За да свържеме базата данни нужно е да имаме локална инстанцич на нащта база от която ще четеме данните. За това ще ползваме XAMPP които след инсталирането можем да стартирам MySQL и Apache. Чрез код във нащия app.js свързваме базата със нащта локална инстанция.

За да можеме да създадеме база нужно е да отидеме на <http://localhost/phpmyadmin> което е админ на нащта локална база и чрез която можем да вмъкваме данни и да създаваме нови таблици. Имаме една таблица която се вика userinfo със някой данни в нея и чрез db.query заявки можем да ползваме наща база данни.

Чрез използване на Express създаваме директория за нашия уеб-сайт и я свързваме със файловете Views и Public които ще садържат всички наши HTML, CSS и JS неща за фротенда които ще потребителите ползват. Нужно е било да се създаде връзка за фронтенда със бекенда за да те могат да комуникират помежду си. След когато вързката е установена можем да създаваме нови пътеви (routes) и изгледи.

Във нашия главния HTML (HBS) файл имаме Navbar които садържа линкове към Home, Register, Login и Account страниците които сме създали във Views файла и сме със помощ на Express добавили пътовете (routes) чрез които ще се показват на фронтенда. То ест това са всичките страници които искаме да се приказват и чрез кои път ще минават на пъта от бекенда до фронтенда да нащия потребител може да ги види.

Във формите за регистраница и влизане имаме полета за име, е-мейл, парола , повторяща парола и RECAPTCHA ( както е казано във изисквания към тази апликация).

За да дадем функционалоността на наща форма трябва да зададем кои Action искаме да се изпълни и Method-а по които да се изпълни (името на акция е релативната директория на файла които садържа методите да се изпъват а метода представлява какъв тип заявка искаме да подадеме между фронтенда на бекенда ( в нашия случай е POST/GET) понеже искаме да испратим данните от формата да бъдат валидирани от страната на бекенда които е свързан с базата данни.

За регистриране е необходимо да се подаде име, е-мейл, парола два пъти и кепча да се испълни. Всичко това се предава на бекенда във Auth.js където имаме логиката за влизане и автентикация на данните и ако всичко е на ред преминаваме към потребителския профил където ще съществуват опции да се смени парола и мейл.

За логин функцията е само необходимо да се провери дали потребитела вече съществува и ако да, да влезне и да му бъде зададен токен за сесия, така ще можеме да следиме дали някой е логнат или не.

След когато успешно потребитела влезе в профила си ще може да посети страницата Account. Тя е само активна ако потребитела вече съществува, ако не сървъра връща логин страницата където може да си въведе данните.

Дали някой потребител е активен може да се види с помощ на JWT (Json Web Tokens) които са токени за достъп на данните и се проверява дали потребитела има активен токен и ако да може да влиза ако не трябва още веднъж да се логне за да получи нов токен. В страницата Account потребитела може да си види данните и също така има възможност да си смени паролата и името (иска се от условието).

За кепча създаваме функция която ще генерира 6 произволни симболи които ще бъдат подадени на фронтенда когато искаме да се регистрираме или влезнем в аканута си. Прави се проверка дали кепчата е изпълнена и ако всичките полета са на ред, потребителат може да продължи с влизането.

Тестовете можем да направим със Jest което е една библиотека със вече създадени функции за тестване на функции. Когато правиме тестовете просто проверяваме дали нещо и какво е това нещо което се предава на нащи функции. След когато направим всички случаи които когат да се случват когато потребителат предаде данните със формата и всички възможни грешки които ще бъдат предадени на фронтенда за потребитела да има няква идеа какво се случва.

За стартиране на тестовете в терминала въведите: **npx jest**

Със това приключват всичките изисквания от условието за тази апликация.

**Описание на садържанието на файловете:**

**Package.json node\_modules package-lock.json**

Това са автоматично генериране файлове които са просто депенденцита за да може Node да работи и всички библиотеки които сме инсталирали. Всичките могат да се виждат във package.json някой от тези са: **cookie-parser jsonwebtoken** – те са за запомняне на потребители които са влезнали в нащия уебсайт, **dotenv** – конфигурационнен файл за базата данни и настройки за токените, **nodemon** – полезна библиотека която стартира отново сървъра когато има сменения (Node JS версия на Live Server във VS Code) , **express** представява нашия сървър и всичко за него, как се приказва, какво съвръзваме и всички комуникации от бекенда до фронтенда, **hbs** – хандлебарс е просто начин за приказ на данните на фронтенда (вместо HTML) , **mysql** – базата данни която ни позволява да успоставиме връзка с нея и даваме заявки за потребителите.

**App.js** – Това е нащия главния файл където са намира самия сървър които се стартира, в него имаме също и всичките настройки които ще са ни необходими да се свържем с базата данни и със потребитела на фронтенда и какво да му прикажеме.

**Controllers:**

Auth.js – Тук се намират всичките функции които ползваме на фронтенда, примерно за: влизане, регистрация, излизане, проверка дали потребителя е логнат или не и методи за сменяване на парола и името на потребитела.

**Views:**

Тук се намират всичките нащи страници които потребитела може да види, а това са: home, register, login и account. В всяка от тези страници имаме един nav-bar които ни ползва да отиваме от една до друга страница. Ако се опитаме да влезнем в нашия профил без да сме се логнали ще получиме опцията да влезнем в сайта.

**Public:**

Тук се намира CSS-a свързан за нащи Views.

**Routes :**

Auth.js – във този файл се намират всички методи които могат да бъдат извикани от фронтенда. Когато се това случи трябва да знаеме къде сочат тези функции, за това имаме роутес. Т.е пътеве които съеднияват извикания метод от фронтенда със отговарящият метод от контролерите.

Myroutes.js – във този файл се намират всичките страници които можеме да видиме от фронтенда. Тук имаме методи които взимат страницата од бекенда и я предават на потребитела на фронтенда със помощ на командата render която връща като отговор страницата която искаме да видим и още някой неща които не можем да получиме от фронтенда но са ни нужни да прикажем (В нащия случай – кепча).

**Tests:**

Unit.test.js – чрез използване на Jest които ни помага да направим тестове за нащи методи тук имаме всички възможни ситуации които могат да се случват и как би се програмата справила с тези грешки които са възникнали и просто да се види дали метода работи това за каквото да е бил предвиден да работи.