Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення для загального користування з клієнт-серверною архітектурою на основі мови javascript

Виконав(ла): студент(ка) 6 курсу, групи СПм
спеціальності 121
інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Стігайло Р. О (прізвище та ініціали)

Керівник (підпис) Пастух О.А (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2023

АНОТАЦІЯ

Дипломний проект на тему «Розробка прогамного забезпечення для загального користування з клієнт-серверною архітектурою на основоі мови javascript», Стігайла Ростислава Олеговича. Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–62, Тернопіль, 2023. С. – 89, рис. – 21, додат. – 4, бібліогр. – 10.

Мета проекту – розробити систему автоматизованої аторизації у веб-додатку за допомогою JWT. Дана робота включає розробку програмного забезпечення на основі використання баз даних. Для розробки цього програмного продукту та для роботи з СУБД mySQL, було використано мову JavaScript. Технологію.React, Express(Node JS). Проект розроблений з компонентним підходом. При вдалій конкуренції на ринку система буде розвиватись. Система була розроблена для спрощення авторизації та підтримування сесії у веб-додатку. Та й в загальному підвищить зручність та ефективність усіх інтернет-додатків.

Ключові слова: СУБД, mySQL, JavaScript, Express, Node js, React, Redux.

ANNOTATION

Dyploma project on "Development of software for general use with client-server architecture based on the javascript language", Stihailo Rostyslav Olehovich. Ivan Pulyuy Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SP-62 Group, Ternopil, 2023. P. - 56, pic. - 21, bibliogr. - 11.

The purpose of the project is to develop an automated authorization system in a web application using JWT. This work includes the development of software based on the use of databases. JavaScript was used to develop this software product and to work with the mySQL database. Technology.React, Express (Node JS). The project is designed with a component approach. With successful competition in the market, the system will develop. The system was designed to simplify authorization and maintain a session in a web application. And in general will increase the convenience and efficiency of all Internet applications.

Keywords: DBMS, mySQL, JavaScript, .Express, Node js, React, Redux.

ЗМІСТ

ЗМІСТ 6
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ТЕРМІНІВ 7
ВСТУП 8
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10
1.1 Опис предметної області 10
1.2 Постановка задачі розробки 13
1.3 Вибір технології розробки 14
1.4 Опис проектування програмного забезпечення 18
2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ 24
2.1 Побудова схеми бази даних 24
2.2 Побудова зв’язків між сутностями 27
2.2.1 Сутність «Users» 30
2.2.2 Сутність «Token» 32
2.2.3 Сутність «City» 34
3. РОЗРОБКА ВЕБ-ДОДАТКУ 36
3.1 Розробка серверної частини програми 36
3.2 Архітектура серверної частини програми 39
Клієнтська частина зв'язується з серверною за допомогою інтернет протоколу HTTP. 39
3.3 Встановлення та доступ до додатку 42
3.4 Варіанти використання 44
У веб-додатку доступний лише один актор, при запуску додатку ми попадаємо на сторінку Home. 44
3.5 Тестування продукту 47
4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТАЦІЯХ 50
4.1 Охорона праці 50
4.2 Безпека в надзвичайних ситуаціях 53
ВИСНОВОК 59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60
ДОДАТКИ 62
ДОДАТОК А 63
ДОДАТОК В 68
ДОДАТОК Г 78
ДОДАТОК Д 94

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ТЕРМІНІВ

Node.js: Розширена платформа JavaScript, яка дозволяє створювати швидкі та масштабовані мережеві додатки за допомогою серверного JavaScript.

Express.js: Це легка та гнучка веб-фреймворк для Node.js, що дозволяє швидко створювати веб-додатки.

JSON Web Token (JWT): Метод аутентифікації, який дозволяє безпечно передавати інформацію між двома сторонами у вигляді JSON-об'єктів.

REST API: Архітектурний стиль для створення веб-служб, який використовує HTTP-протокол для отримання доступу до даних та операцій з ними.

Client-Server Architecture: Модель взаємодії між клієнтом (користувачем) та сервером, де клієнт відправляє запити, а сервер надає відповіді, обробляє дані та надає ресурси.

Frontend: Частина додатка, яка відповідає за інтерфейс користувача та взаємодію з ним.

Backend: Частина додатка, яка відповідає за обробку запитів користувача, бізнес-логіку та взаємодію з базою даних.

NPM (Node Package Manager): Це пакетний менеджер для JavaScript, який дозволяє встановлювати, оновлювати та керувати пакетами програмного забезпечення Node.js.

WebSocket: Протокол зв'язку, який дозволяє двом або більше пристроям встановлювати двонаправлене з'єднання через мережу для обміну даними в реальному часі.

ORM (Object-Relational Mapping): Техніка, яка дозволяє зв'язувати об'єктно-орієнтовану модель даних з реляційною базою даних.

ВСТУП

У світі стрімко зростаючих технологій та вимог до функціональності програмного забезпечення, розробка систем, що взаємодіють із користувачами через клієнт-серверну архітектуру, стає ключовим аспектом у сфері програмування. Даний дипломний проект орієнтований на створення програмного забезпечення, яке базується на клієнт-серверній архітектурі, з використанням мови програмування JavaScript.

Клієнт-серверна архітектура є фундаментальним концептом у сучасному програмуванні, що дозволяє ефективно розподіляти завдання між клієнтською та серверною частинами програми, що в свою чергу сприяє оптимізації продуктивності, безпеки та масштабованості програмного забезпечення.

Метою даної дипломної роботи є дослідження та реалізація програмного забезпечення, спрямованого на широке загальне користування, з використанням передових технологій та практик розробки програмних продуктів. Для досягнення цієї мети були використані відомі та популярні інструменти, такі як бібліотека React для розробки клієнтської частини, Express для серверної реалізації, а також MySQL та Sequelize для ефективного управління базою даних.

Однією з ключових функцій програми, яка знаходиться у центрі уваги, є механізм авторизації користувачів за допомогою технології JSON Web Token (JWT). Використання JWT для авторизації дозволяє забезпечити безпеку та аутентифікацію у веб-додатку, зокрема при тривалому використанні.

Цей проект створений з метою не лише продемонструвати можливості клієнт-серверної архітектури на базі JavaScript, а й показати практичний аспект застосування відомих та ефективних інструментів для створення програмного забезпечення загального користування.

Ця програма також використовує можливості сторонніх API для розширення своїх функціональних можливостей та отримання додаткових даних, що покращує користувацький досвід. Для цього використано інтерфейси, що забезпечують взаємодію з різноманітними сервісами і джерелами інформації з відповідними протоколами обміну даними.

Програмне забезпечення використовує можливості сторонніх API для отримання оновлених даних про погоду, фінансових показників, новини або інших параметрів, що можуть бути корисними для кінцевого користувача. Використання цих джерел даних значно розширює можливості системи та дозволяє забезпечити користувачів актуальною та вичерпною інформацією.

Додатково, ця система має потенціал для легкої інтеграції з різноманітними платформами та сервісами через відкриті інтерфейси API. Це відкриває нові можливості для майбутнього розвитку та підтримки різноманітних потреб користувачів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сьогодні на просторах інтернету важко знайдеш додаток, що не містить у собі можливість авторизації. Без такого підходу неможливо у наш час, адже на веб просторах є незліченна кількість програм ботів на автоматизованих "юзерів". Авторизація вдало справляється з такими непроханими відвідувачами шляхом блокування певних сервісів для неавторизованих користувачів. Це дає змогу оптимізувати веб-додаток та вберегти сервер від перевантажень.

Саме тому я обрав темою курсового проекту веб-додаток що містить у собі авторизацію та демонструє алгоритм її впровадження у інтерфейс.

Для авторизації потрібна база даних щоб зберігати в ній потрібну інформацію про користувачів, нами була обрана СУБД MySQL.

MySQL — вільна система керування реляційними базами даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. Зазначимо, що у вітчизняній літературі зустрічаються синомінічні абревіатури СУБД та СКБД. У даному підручнику перевага надається – СУБД.

Аутентифікація на основі файлів cookie протягом тривалого часу була методом за замовчуванням для обробки автентифікації користувачів. З діаграми нижче ви можете чітко бачити, як клієнт відправляє облікові дані для входу на сервер, сервер перевіряє дані і створює ідентифікатор сеансу, який зберігається на сервері (із повним станом) і повертається клієнту через set-cookie. На наступний запит ідентифікатор сеансу з файлу cookie перевіряється на сервері та запит обробляється. Після виходу ідентифікатор сеансу буде видалено як із файлу cookie клієнта, так і з сервера.

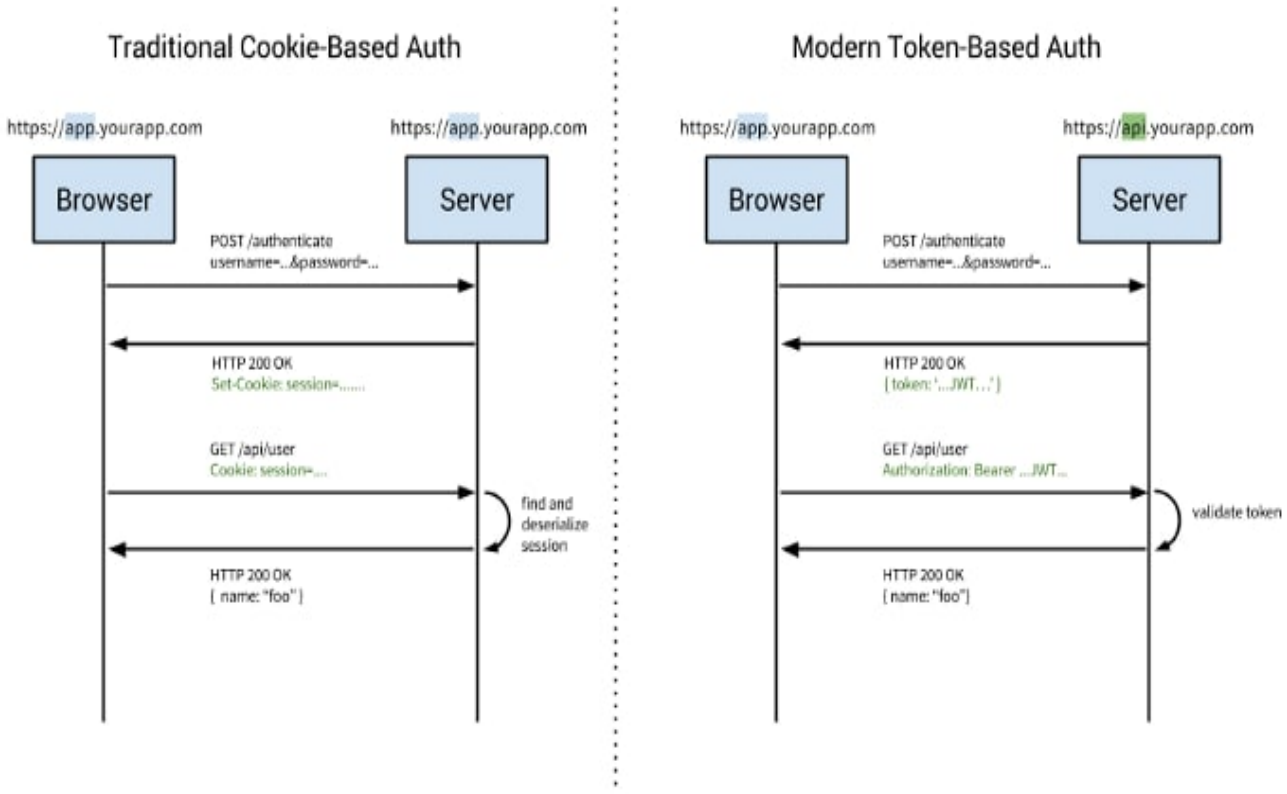

Рисунок 1.1 – Розбіжності у методах автентифікацій.

Аутентифікація на основі файлів cookie протягом тривалого часу була методом за замовчуванням для обробки автентифікації користувачів. З діаграми нижче ви можете чітко бачити, як клієнт відправляє облікові дані для входу на сервер, сервер перевіряє на правильність дані введені користувачем і створює ідентифікатор сеансу, який зберігається на сервері (із повним станом) і повертається клієнту через set-cookie. На наступний запит від користувача на ідентифікатор сеансу з файлу cookie перевіряється на сервері та запит обробляється. Після виходу ідентифікатор сеансу буде видалено як із файлу cookie клієнта, так і з сервера.

З іншого боку, автентифікація на основі Token набирає популярності через зростання кількості односторінкових додатків (SPA) та (RESTful API's) програми. Існують безліч різноманітних способів реалізації аутентифікації на основі маркерів, ми зосередимось на найбільш часто використовуваних у веб-маркерах JSON (JWT). При отриманні облікових даних від клієнта сервер перевіряє їх на вірність і генерує підписаний JWT, що містить інформацію про користувача. Зверніть увагу, маркер ніколи не зберігатиметься на сервері (без стану). За подальшим запитом маркер буде переданий серверу і перевіряється (декодується) на сервері. Токен можна підтримувати на стороні клієнта в локальному сховищі, сховищі сеансів або навіть у файлах cookie. Доступ третьої сторони, якщо нам потрібно виставити наш API поза межами нашого.

Зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування. MySQL надає багатий набір функціональних можливостей, які підтримують безпечне сучасних середовище для зберігання, обслуговування і отримання даних.

Найпоширеніший прийом автентифікації - використання імені користувача та пароля.

Загальний потік під час його реалізації:

- Користувач реєструється за допомогою ідентифікатора, такого як ім'я користувача / електронна пошта / мобільний телефон;
- Додаток зберігає облікові дані користувачів у базі даних;
- Додаток надсилає електронний лист / повідомлення для підтвердження реєстрації;
- Після успішної реєстрації користувач вводить дані для входу;
- Після успішної автентифікації користувачеві надається доступ до певних ресурсів;

- Стан користувача підтримується через Sessions або JWT.

MySQL був розроблений компанією «TcX» для підвищення швидкодії обробки великих об’ємів баз даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним вигідна системам. MySQL з самого початку була дуже схожою на свого попередника mySQL, проте з часом вона все розширювалася і покращувалась зараз MySQL — одна з найпоширеніших на сьогоднішній час систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, інформаційних системах, оскільки має чудову підтримку з боку різноманітних мов програмування таких як JavaScrip, C# і багато інших.

1.2 Постановка задачі розробки

Головною метою розробки цього проекту було вибрано надійний, простий, та швидкий спосіб авторизації користувачів. У курсовому проекті було використано JSON Web Token для підвищення захищеності користувача при довготривалому користувані веб-додатком. Як правило, при використувані JSON-токеном в клієнт-серверних додатка реалізовано наступні схеми: Клієнт проходить автентифікацію в додатку (наприклад, з використаням логіна і пароля). У випадку вдалої автентифікації сервер відправляє клієнту access- і refresh-токени. При подальшому звернені до сервера клієнт використовує access-токен. Сервер провіряє токен на валідність і надає клієнту доступ до ресурсів. У випадку, якщо access-токен стає не валідним, клієнт відправляє refresh-токен у відповідь на який сервер надає два оновлених токени. У випадку, якщо refresh-токен стає не валідним клієнт повинен пройти процес автентифікацій щераз для відновлення звязку з веб-додатком і отримання нового токена. Цей спосіб є дуже ефективний для підвищення безпеки користування веб-додатками і не тільки.

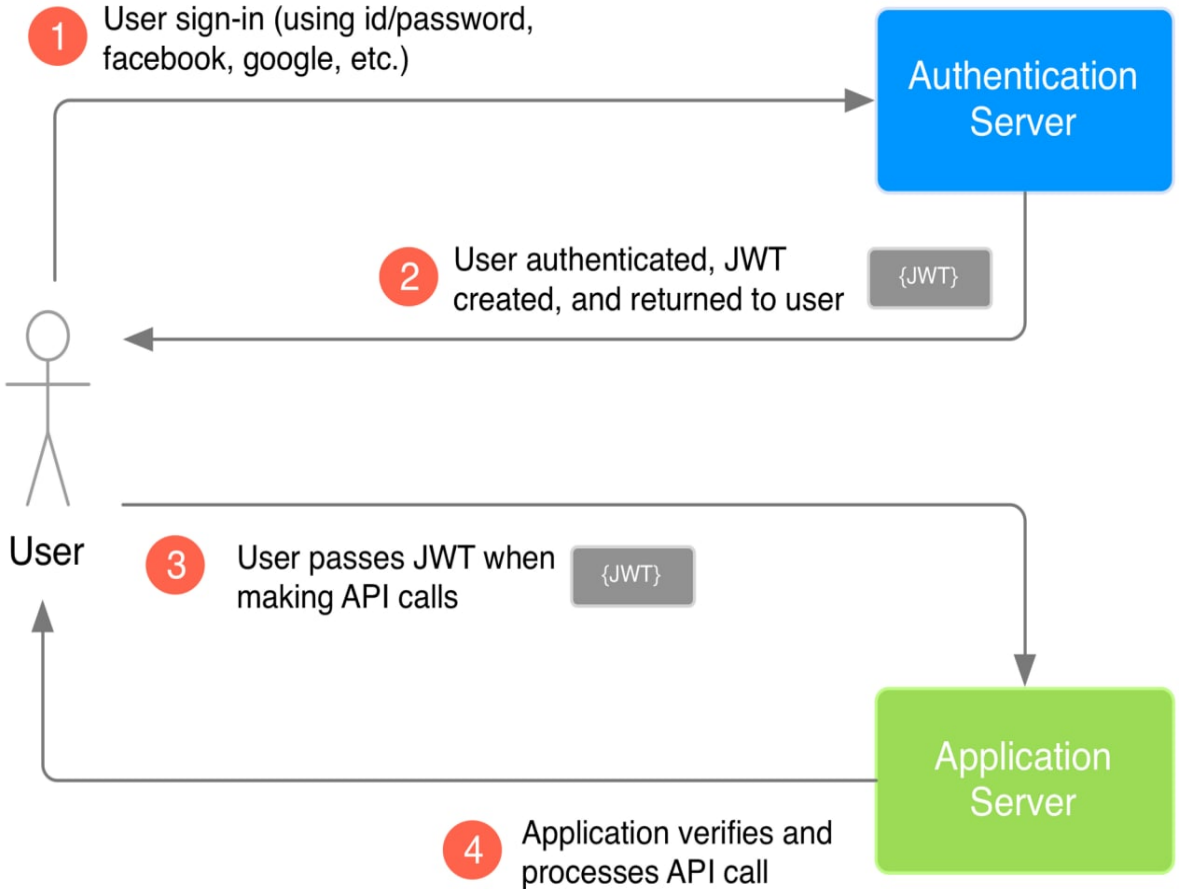

Рисунок 1.2 – Схема авторизації з JWT-токеном

В базі даних повинна бути наступна інформація:

- дані користувача(Логін, Пароль)
- згенеровано при авторизації JWT token та refresh token
- країна та місто користувача

1.3 Вибір технології розробки

Програмне забезпечення буде розрбите на дві частини:

- клієнтська частина (Front-End)– інтерфейс програми з яким буде взаємодіяти користувач. Повинна містити тільки логіку взаємодії з клієнтом, сервером та інформацією, без алгоритмів роботи з даними;
- серверна частина (Back-End) – програмна частина яка містить основні алгоритми та логіку взаємодії з базою даних.

Для клієнтської частини були обрані наступні технології:

- React

  Відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

  React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабуванням. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.
- Redux

  Відкрита JS бібліотека призначена для управління станом програм JavaScript. Найчастіше використовується разом з React або Andular для побудови інфтерфейсів користувача.

Він допомагає писати додатки,які ведуть себе послідовно, працюють в різних середовищах (клієнтських, серверних і нативних), і які легко тестуються. Крім того, вони надають чудові можливості для розробників, такі як редагування коду в сукупності з часовим відладчиком коду. Ви можете використовувати Redux разом з React або з любою другою бібліотекою для View. Redux дуже маленький (2 кб, включає залежність).

- Bootstrap — це безкоштовний набір інструментів з відкритим кодом, призначений для створення веб-сайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків. Bootstrap — це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері. Репозиторій із цим фреймворком є одним із найпопулярніших на GitHub. Основа Bootstrap — модульна сітка з 12 колонок, яка вміє перелаштовуватися в залежності від розміру екрану пристрою відвідувача вашого сайту. Крім того, Bootstrap включає в себе стилі оформлення безлічі елементів сучасних сайтів: кнопок, форм, спливаючих вікон, навігаційних панелей і т.п.

Для серверної частини будуть використовуватись наступні технології: Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Райан Дал (Ryan Dahl). Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати JavaScript-скріпти на сервері та відправляти користувачеві результат їхнього виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js характеризується наступними ключовими властивостями:

- Асинхронна однопотокова модель виконання запитів: Node.js працює у режимі асинхронного виконання, що дозволяє ефективно опрацьовувати багато запитів без блокування.
- Неблокуючий ввід/вивід: Платформа Node.js використовує не блокуючий ввід/вивід, що робить її ефективною в обробці одночасних запитів.
- Система модулів CommonJS: Використання цієї системи дозволяє зручно та ефективно керувати модулями в Node.js.
- Рушій JavaScript Google V8: Node.js використовує потужний рушій Google V8 для виконання високопродуктивних мережевих застосунків, написаних на JavaScript.
- Пакетний менеджер npm: Для управління модулями в Node.js використовується пакетний менеджер npm (Node Package Manager), що забезпечує зручну установку та оновлення різноманітних пакетів.

Ця платформа спроектована для створення масштабованих мережевих додатків, а не лише для роботи з серверними скриптами веб-запитів. Вона використовується як для клієнтських, так і для серверних програм, і використовує розроблений Google рушій V8.

Node.js використовує асинхронну модель запуску коду, що базується на обробці подій у неблокуючому режимі та використанні зворотних викликів (callback). Для мультиплексування з'єднань та інших операцій використовується бібліотека libuv. Це дозволяє Node.js ефективно обробляти багато паралельних запитів, забезпечуючи високу продуктивність системи.

Крім того, варто відзначити фреймворк Express.js, який використовується для розробки серверної частини веб-застосунків на Node.js. Express є відкритим програмним забезпеченням під ліцензією MIT і вважається стандартом для Node.js. Цей фреймворк є мінімалістичним, але має значну кількість плагінів, що забезпечує йому велику функціональність та гнучкість у використанні.

Express.js у своєму розпорядженні має багато службових методів HTTP и проміжних обробників події створювати надійних API можливо швидко і легко. Express являє собою популярний веб-фреймворк, написаний на JavaScript і працює всередині середовища використання node.js. Це модулі проливає світло на деякі ключові моменти перевага цього фреймворка, встановлення середовища розробки і виконання основних задач веб-розробці і розгалуженням.

1.4 Опис проектування програмного забезпечення

REST (Representational State Transfer) – це підхід до архітектури мережевих протоколів, який забезпечує доступ до інформаційних ресурсів. Розроблений Роєм Філдінгом у 2000 році, цей підхід спирається на принципи функціонування Всесвітньої павутини та використання можливостей протоколу HTTP. Побудований на базі HTTP 1.1, REST використовує обмін даними у форматі невеликої кількості стандартних форматів, таких як HTML, XML або JSON.

Ключові принципи REST полягають у підтримці кешування, незалежності від мережевого прошарку та відсутності збереження інформації про стан між запитами та відповідями. Цей підхід сприяє масштабованості системи та її легкій адаптації до нових вимог.

У порівнянні з підходом, що базується на виклику віддалених процедур (RPC), REST відрізняється обмеженістю кількості методів та простотою протоколу. Підхід RPC використовує обмежену кількість мережевих ресурсів з багатьма методами і складним протоколом. У випадку REST велика кількість окремих ресурсів є деякою компенсацією за знижену складність методів та протоколу.

REST AIP містить під собою прості правила:

- Кожен URL являється ресурсом
  При звертанні до методу ресурсу GET повертає описання цього ресурса
- Метод POST добавляє новий ресурс
- Метод PUT змінює ресурс
- Метод DELETE видаляє ресурс

Ці правила являють прості CRUB інтерфейсу для інших додатків, взаємодії з якими відбувається через протокол HTTP.

REST, як і кожен архітектурний стиль відповідає ряду архітектурних обмежень (англ. architectural constraints). Це гібридний стиль який успадковує обмеження з інших архітектурних стилів.[1]

Клієнт-сервер

Перша архітектурна концепція, від якої вона відбирає обмеження, - це клієнт-серверна архітектура. У цій концепції розподіляється функціональність між різними компонентами системи: одні компоненти (сервер) відповідають за зберігання та оновлення даних, тоді як інші (клієнт) відображають ці дані на інтерфейсі користувача та взаємодію з ним. Цей розподілений підхід створює можливість для окремих компонентів розвиватися незалежно один від одного, що сприяє гнучкості та ефективності у розробці та підтримці системи.

Відсутність стану

Наступним обмеженням є відсутність стану у взаємодії між сервером та клієнтом. Це означає, що кожен запит містить усю необхідну інформацію для його обробки і не базується на попередніх даних, які мав сервер від попереднього запиту.

Це обмеження "відсутність стану" не позначається на можливостях архітектури REST будувати тільки станові системи. Відсутність стану вказує на те, що сервер не зберігає інформацію про стан клієнта та послідовність його запитів, оскільки кожен запит сприймається як окремий ізольований від попередніх. Наприклад, коли клієнт відкриває головну сторінку сайту, сервер відповідає на запит, а потім "забуває" про цього клієнта. Цей клієнт може залишити сторінку відкритою протягом довгого часу, а сервер буде реагувати на інші запити, незалежно від цього.

Таким чином, дані про стан сесії (наприклад, ідентифікатор автентифікованого користувача) зберігаються на клієнті і передаються разом з кожним запитом. Це поліпшує масштабованість, оскільки після завершення обробки запиту сервер може вивільнити всі ресурси, задіяні у цій операції, не ризикуючи втратити важливу інформацію. Крім того, спрощується моніторинг і відладка, оскільки для аналізу конкретного запиту досить розглянути саме його. Це також підвищує надійність, оскільки помилка у одному запиті не впливає на інші.

Однак недоліком цього обмеження є зниження продуктивності через необхідність передачі даних сесії з кожним запитом з боку клієнта. Також ускладнюється збереження стану на різних клієнтах через можливі відмінності в реалізації клієнтських середовищ, що відрізняються від серверного середовища, під контролем розробника.

Кешування

Ще одним обмеженням стилю REST є необхідність підтримки кешування у системах, що працюють у цьому форматі. Це означає, що дані, які передаються сервером, мають містити інформацію про те, чи можна їх кешувати, і, у випадку позитивної відповіді, на який термін. Це сприяє підвищенню продуктивності, оскільки уникається надмірних запитів, але також може підірвати надійність системи через можливість застаріння даних в кеші.

Первинна веб-архітектура, створена Тімом Бернерс-Лі, відповідає цим трьом основним обмеженням — клієнт-сервер, безстанівна архітектура з підтримкою кешування. Проте, стиль REST накладає ще додаткові обмеження.

Шари абстракції

Іншим обмеженням, що стосується REST, є розділення на рівні абстракції. Кожен елемент системи розміщується в певному рівні і взаємодіє лише з компонентами в цьому ж рівні або на рівні над ним або під ним. Такий підхід до знання системи одним рівнем дозволяє зменшити складність компонентів.

Останнім архітектурним обмеженням у REST є можливість для клієнтів розширювати свою функціональність через завантаження додаткового коду (code on demand), такого як аплети або скрипти. Це спрощує клієнтів, оскільки вони можуть не включати всі необхідні функції заздалегідь. Проте, це обмеження не є обов'язковим, і його не обов'язково використовувати, якщо воно не принесе переваг для конкретного випадку. Наприклад, дозвіл на завантаження зовнішнього коду може бути небажаним з точки зору безпеки.

Компоненти системи REST взаємодіють, передаючи один одному представлення ресурсів у форматі, який обирається з набору стандартних форматів даних. Вибір формату здійснюється динамічно, з урахуванням вимог клієнта та можливостей сервера. Чи пов'язане представлення зі структурою ресурсу, чи воно результат певних перетворень, залишається позаду інтерфейсу як деталь реалізації.

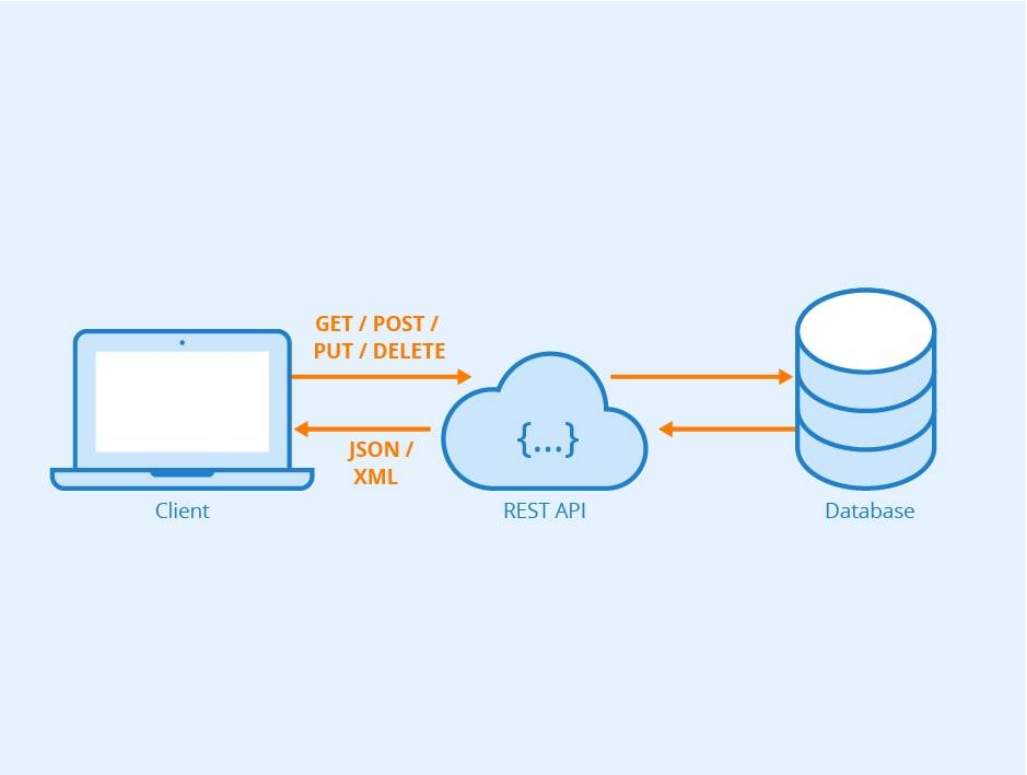

Рисунок 1.3 – Схема роботи REST API (курсового проекту).

CRUD - акронім, що позначає чотири базові функції, які використовуються при роботі з базами даних створення (англ. Create), читання (read), модифікація (update), видалення (delete). Введено Джеймсом Мартіном (англ. James Martin) в 1983 році як стандартну класифікацію функцій по маніпуляції даними.

У SQL цих функцій операціями відповідають оператори Insert (створення записів), Select (читання записів), Update (редагування записів), Delete (видалення записів). У деяких CASE-засобах використовувалися спеціалізовані CRUD-матриці або CRUD-діаграми, в яких для кожної сутності вказувалося, які базові функції з цією сутністю виконує той чи інший процес або та чи інша роль. У системах, що реалізують доступ до бази даних через API в стилі REST, ці функції реалізуються найчастіше (але не обов'язково) через HTTP-методи POST, GET, PUT і DELETE відповідно.

Хоча традиційно оперування в стилі CRUD застосовується до баз даних, такий підхід може бути поширений на будь-які збережені обчислювальні суті (файли, структури в пам'яті, об'єкти). Шаблон проектування ActiveRecord забезпечує відповідність функцій CRUD об'єктно-орієнтованого підходу, і широко використовується в різних фреймворків для доступу до баз даних з об'єктно-орієнтованих мов програмування.

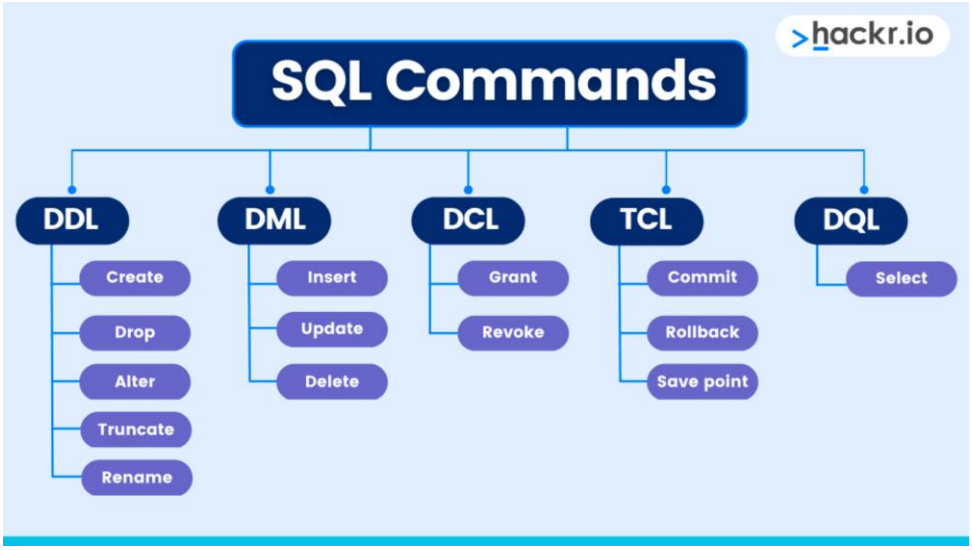

Рисунок 1.4 – Приклади CRUB операції.

2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ

У даному розділі було проаналізовано всі наявні сутності у програмі. Проведено детальний розбір кожної сутності та її призначення, а також, надано графічну візуалізацію таблиць у базі даних MySQL.

2.1 Побудова схеми бази даних

Модель «сутність-зв'язок» (ER-модель) — модель даних, яка дозволяє описувати концептуальні схеми за допомогою[6] узагальнених конструкцій блоків. ER-модель — це мета-модель даних, тобто засіб опису моделей даних. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення, що його реалізує, є модель «сутність-зв'язок». Важливим є той факт, що з моделі «сутність-зв'язок» можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найзагальнішою.

Модель сутність-зв'язок є результатом систематичного процесу, який описує та визначає деяку предметну область. Вона не визначає сам процес, а лише візуалізує його. Дані представлені у вигляді компонентів (сутностей), які пов'язані між собою певними зв'язками, які виражають залежності і вимоги між ними, такі як: одна будівля може бути розділена на нуль або більше квартир, але одна квартира може бути розташована лише в одній будівлі. Сутності можуть мати різні властивості (атрибути), які характеризують їх. Діаграми, створені для представлення цих сутностей, атрибутів і зв’язків графічно, називають сутність-зв'язок діаграмами.

ER-модель зазвичай реалізується в вигляді баз даних. У разі реляційної бази даних, в якій зберігаються дані в таблицях, кожен рядок кожної таблиці являє собою один екземпляр сутності. Деякі поля даних в цих таблицях вказують на індекси в інших таблицях. Такі поля є покажчиками фізичної реалізації зв'язків між сутностями.

Широке розповсюдження реляційних СУБД і їх використання в найрізноманітніших застосуваннях показує, що реляційна модель даних достатня для моделювання різноманітних наочних областей. Проте проектування реляційної бази даних в термінах стосунків на основі того, що короткий розглянуті нами в двох попередніх лекціях механізму нормалізації часто є дуже складним і незручним для проектувальника процесом.

Основними поняттями ER-модели є суть, зв'язок і атрибут.

Зв'язок - це графічно змальовувана асоціація, що встановлюється між двома типами суті. Як і суть, зв'язок - це типове поняття, всі екземпляри обох зв'язуваних типів суті підкоряються встановлюваним правилам скріплення. Тому правильніше говорити про тип зв'язку, що встановлюється між типами суті, і про екземпляри типа зв'язку, що встановлюються між екземплярами типа сущности. У обговорюваному тут варіанті ER-модели ця асоціація завжди є бінарною і може існувати між двома різними типами суті або між типом суті і ним же самим (рекурсивний зв'язок). У будь-якому зв'язку виділяються два кінці (відповідно до існуючої пари зв'язуваної суті), на кожному з яких указуються ім'я кінця зв'язку, ступінь кінця зв'язку (скільки екземплярів даного типа суті має бути присутніми в кожному екземплярі даного типа зв'язку), обов'язковість зв'язку (тобто чи будь-який екземпляр даного типа суті повинен брати участь в деякому екземплярі даного типа зв'язку.

Зв'язок представляється у вигляді ненапрямленої лінії, що сполучає дві суті або ведучу від суті до неї ж самої. При цьому в місці «стиковки» зв'язку з суттю використовуються:

- триточковий вхід в прямокутник суті, якщо для цієї суті в зв'язку можуть (або повинні) використовуватися багато (many) екземплярів суті;
- одноточечний вхід, якщо в зв'язку може (або повинен) брати участь тільки один екземпляр суті.

Атрибутом суті є будь-яка деталь, яка служить для уточнення, ідентифікації, класифікації, числової характеристики або вираження стану суті. Імена атрибутів заносяться в прямокутник, що змальовує суть, під ім'ям суті і змальовуються малими буквами, можливо, з прикладами. Приклад типа суті ЧОЛОВІК з вказаними атрибутами показаний на мал. 9.4. З технічної точки зору атрибути типа суті в ER-модели схожі на атрибути відношення в реляційній моделі даних. І у тому, і в іншому випадках введення іменованих атрибутів вводить деяку типову структуру даних, ім'я якої збігається з ім'ям типа суті в разі ER-модели або з ім'ям змінної відношення в разі реляційної моделі.

Цій типовій структурі повинні слідувати всі екземпляри типа суті або всі кортежі відношення. Але є і важлива відмінність. Нагадаємо, що в реляційній моделі даних атрибут визначається як впорядкована пара <имя_атрибута, имя_домена> (або <имя_атрибута, имя_базового_типа_данных>, якщо поняття домена не підтримується). Заголовок відношення, визначуваний як безліч таких пар, є повним аналогом структурного типа даних в мовах програмування.

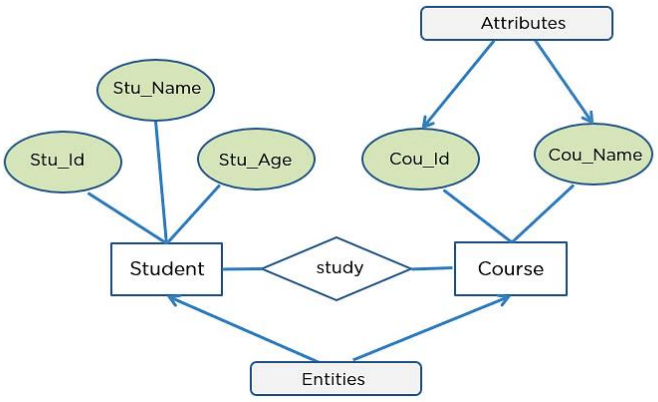

Рисунок 2.1 – Приклад ER- діаграми

Але при проектуванні бази даних мало того, щоб проектувальник переконався в правильному виборі типів суті, що гарантує відмінність екземплярів кожного типа суті. Необхідно повідомити систему автоматизації проектування БД, яким чином розрізнятимуться ці екземпляри, тобто повідомити, як конструюються унікальні ідентифікатори екземплярів кожного типа суті. У ER-моделі у экземпляра типа суті не може бути імені, що призначається користувачем, або зовнішнього унікального ідентифікатора, що призначається системою. Екземпляр типа суті може ідентифікуватися тільки своїми індивідуальними характеристиками, а вони представляються значеннями атрибутів і екземплярами типів зв'язку, що зв'язують даний екземпляр типа суті з екземплярами інших типів суті або цього ж типа суті. Тому унікальним ідентифікатором суті може бути атрибут, комбінація атрибутів, зв'язок, комбінація зв'язків або комбінація зв'язків і атрибутів, що унікально відрізняє будь-який екземпляр суті від інших екземплярів суті того ж типа.

Суть - це реальний або такий, що представляється об'єкт, інформація про яке повинна зберігатися і бути доступной. У діаграмах ER-моделі суть представляється у вигляді прямокутника, що містить ім'я суті. При цьому ім'я суті - це ім'я типа, а не деякого конкретного екземпляра цього типа. Для більшої виразності і кращого розуміння ім'я суті може супроводитися прикладами конкретних екземплярів цього типа.

2.2 Побудова зв’язків між сутностями

Існує три типа звязвків між сутностями у базах даних:

- Один до одного
- Один до багатьох
- Багато до багатьох

Тип зв'язку один - до - одному використовується, коли необхідно відокремити деякий набір відомостей, однозначно пов'язаний з конкретним екземпляром вихідного структурного елементу. Так, наприклад, якщо є необхідність виділити паспортні дані в окремий структурний елемент, щоб забезпечити розмежування прав доступу до відповідних відомостями, то між елементом "Паспортні дані" і "Співробітник" буде встановлено зв'язок один - до - одному.

До зв'язку один - до - одному (1: 1) відносять таку взаємодію структурних елементів, у яких один екземпляр одного елемента може бути пов'язаний лише з одним екземпляром іншого елемента.

Зв’язок один-до-багатьох має місце, коли одному значенню поля першої таблиці може відповідати декілька значень поля другої таблиці, а кожному значенню поля другої таблиці — тільки єдине значення поля першої. Наприклад, СУБД Access може побудувати зв’язок один до-багатьох, якщо ключове поле однієї таблиці пов’язується з не ключовим полем другої таблиці. При цьому перша таблиця вважається головною, а друга — підпорядкованою. Зв’язок один-до-багатьох може бути створений, наприклад, між таблицями Країни та Заповідники бази даних Географія. Поле Код країни є ключовим лише в таблиці Країни: у будь якій країні можуть розташовуватися кілька заповідників.

Наступним типом зв'язку є зв'язок багато - до - багатьох (ЛГ: М), яка відображає багатозначну залежність об'єктів один від одного. Такий зв'язок досить складно обробляти, оскільки виникає безліч аномалій обробки. Наприклад, при видаленні одного примірника будь-якого з двох об'єктів, що беруть участь у встановленні зв'язку, можливо каскадне видалення всіх пов'язаних об'єктів іншого об'єкта. Але наявність такого ж правила в зворотну сторону призведе до циклічного видалення даних, що може привести до їх втрати. Саме тому в базах даних реалізуються захисні механізми обмеження посилальної цілісності, але також на рівні реалізації зв'язків між об'єктами обмежується можливість застосування зв'язку багато - до - багатьох.

Відповідно, з урахуванням цих особливостей її використання, даний тип зв'язку реалізується тільки на рівні концептуального і логічного моделювання бази даних. Це дозволяє розробнику описувати інформаційну структуру предметної області, не захаращуючи її допоміжними елементами, які з'являться в результаті нормалізації логічної моделі бази даних. В результаті, на рівні фізичного представлення бази даних і її моделі зв'язок багато - до - багатьох реалізована не буде і се подання буде відображено множинної зв'язком між атрибутами структурних елементів (сутностей) або, що частіше зустрічається, спеціальним структурним елементом - сутністю-зв'язкою.

До зв'язку багато - до - багатьох (Л ^: М) відносять таку взаємодію структурних елементів, де один примірник одного з елементів може бути пов'язаний з безліччю екземплярів іншого елемента і той же визначається в зворотному напрямку.

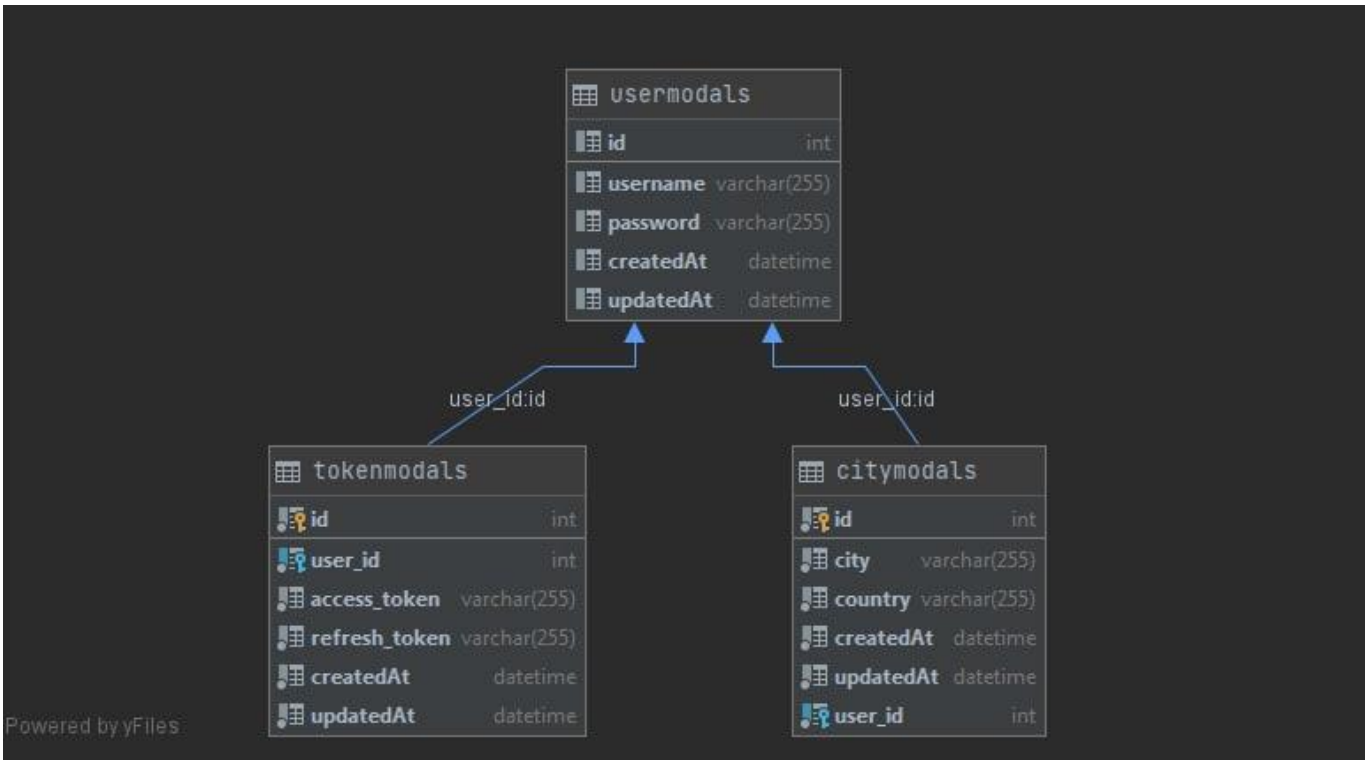

Рисунок 2.2 – ER- діаграма сутностей бази даних.

2.2.1 Сутність «Users»

Дана сутність створена для зберігання у ній основної інформації про користувача, що необхідна при авторизації. Саме по цій табличці буде проводитись вибірка при авторизації користувача.

Таблиця має такі атрибути:

- id - Атрибут, що містить первинний ключ і використовують у всіх зв'язках з даною сутністю. Воно є автоінкрементним.
- Username – Атрибут, що містить електронну пошту користувача і проходить подвійну валідацію, на клієнтській та серверній частинах програми, перед тим як бути записаним у таблицю.
- Password - Атрибут, що містить захешований пароль користувача що також пройшов подвійну валідацію, на клієнтській та серверній частинах програми, перед тим як бути записаним у таблицю.
- CreatedAt - Атрибут, що містить у собі таймштамп, тобто дату та час створення об'єкта.
- UpdatedAt - Атрибут, що містить у собі таймштамп, тобто дату та час оновлення об'єкта.

Таблиця 1 - Сутність «Users»

Атрибути сутності	Обмеження атрибутів					Ключ
	Тип даних	Межі або допустимі значення	Структура (формат)	Умова	Значення за замовчуванням	
ID	int	-	-	Not Null	-	P
Username	varchar	max -255	-	Not Null	-	
Password	varchar	max -255	-	Not Null	-	
CreatedAt	datetime	-	-	Not Null	-	
UpdatedAt	datetime	-	-	Not Null	-	

Рисунок 2.3 – Візуалізація сутності «Usermodals».

2.2.2 Сутність «Token»

Дана сутність створена для зберігання у ній JSON Web Token. JWT генерується у серверній частині програми шляхом алгоритму JWT. Записується у табличку два значення, відповідно сам token та його refresh, що використовується для утримання подальшої сесії у веб-застосунку.

Таблиця має такі атрибути:

- Id - Поле, що містить первинний ключ і використовуєть у всіх зв'язках з даною сутністю. Воно є автоінкрементним.
- Access_token - атрибут, що містить під собою поля, які зберігають Token
- Refresh_token - атрибут, що містить під собою поля, які зберігають refresh –token
- CreatedAt - Поле, що містить у собі таймштамп, тобто дату та час створення об'єкта.
- UpdatedAt - Атрибут, що містить у собі таймштамп, тобто дату та час оновлення об'єкта.
- User_id - Атрибут, що містить під собою поле яке зберігає Foreign key, що зберігає id користувача.

Таблиця 2 - Сутність «Token»

Атрибути сутності	Обмеження атрибутів					Ключ
	Тип даних	Межі або допустимі значення	Структура (формат)	Умова	Значення за замовчуванням	
Id	int	-	-	Auto Incriment	-	P
User_Id	int	-	-	Not Null	-	F
Access_Token	varchar	max -255	-	Not Null	-	
Refresh_Token	varchar	max -255	-	Not Null	-	
CreatedAT	datetime	-	-	Not Null	-	
UpdateAT	datetime	-	-	Not Null	-	

Таблиця 2 - Сутність «Token»

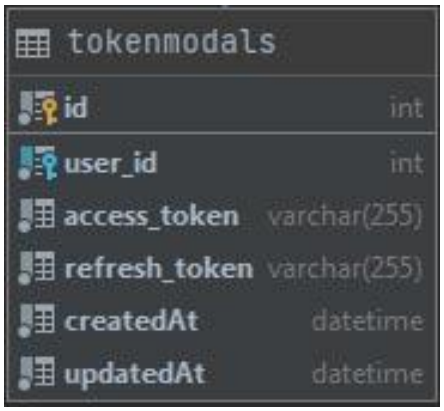

Рисунок 2.4 - Візуалізація сутності «Tokenmodals».

2.2.3 Сутність «City»

Дана сутність створена для зберігання в ній інформації про користувача що не є необхідною для авторизації у веб - застосунку, тому вибірка по ній під час логінації проводитись не буде.

Таблиця має такі атрибути:

- Id - Атрибут, що містить первинний ключ і використовують у всіх зв'язках з даною сутністю. Воно є автоінкрементним.
- User_id - Атрибут, що містить під собою поле яке зберігає Foreign key, що зберігає id користувача
- Country - Атрибут, що містить у собі поле з назвою країни, що користувач ввів у клієнтській частині програми
- City - Атрибут, що містить у собі поле з назвою міста, що користувач ввів у клієнтській частині програми.

Таблиця 3 - Сутність «City»

Атрибути сутності	Обмеження атрибутів					Ключ
	Тип даних	Межі або допустимі значення	Структура (формат)	Умова	Значення за замовчуванням	
Id	int	-	-	Auto Incriment	-	P
City	varchar	Max - 255	-	Not Null	-	
Country	varchar	max -255	-	Not Null	-	
User_Id	int	-	-	Not Null	-	F
CreatedAT	datetime	-	-	Not Null	-	
UpdateAT	datetime	-	-	Not Null	-	

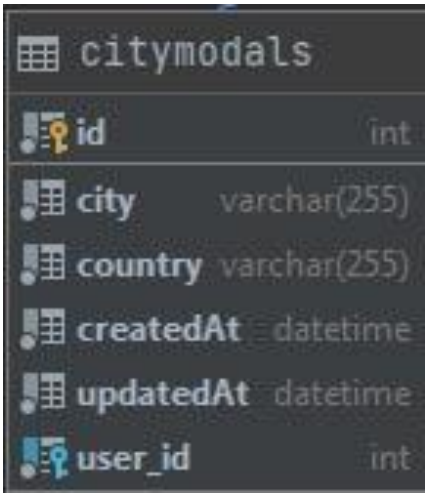

Рисунок 2.4 - Візуалізація сутності «Citymodals».

3. РОЗРОБКА ВЕБ-ДОДАТКУ

3.1 Розробка серверної частини програми

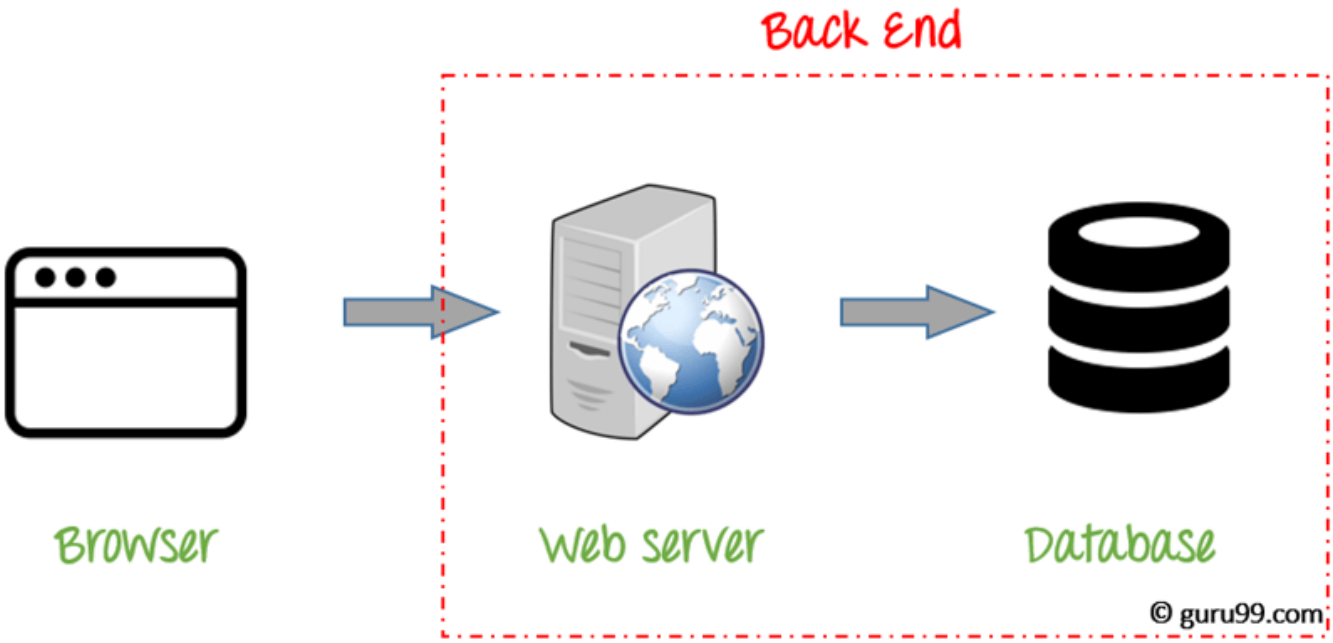

Рисунок 3.1 – Схема роботи BackEnd.

Backend - це та частина проекту, яка відповідає за програмний та апаратний забезпечення, тоді як Frontend представляє клієнтську сторону інтерфейсу проекту, доступного користувачам. Frontend взаємодіє з програмно-апаратною частиною, що прихована від користувача - бекендом. Сам сервер також є частиною бекенду, хоча й належить до апаратної частини.

Це відображено в термінології: front - це все, що бачиться спереду, тобто користувацький інтерфейс, тоді як back - це все, що відбувається позаду куліс, тобто невидиме для кінцевого користувача.

Припустимо, ви оплачуєте покупку в інтернеті: ви вводите дані карти, натискаєте кнопку "оплатити", після чого бачите повідомлення "Ваш платіж прийнято для обробки" – це і є фронтенд.

Одночасно бекенд відповідає за те, як ваші гроші рухаються всередині мережі та обробляються вашим замовленням. Наприклад, коли магазин отримує повідомлення про замовлення та зарахування коштів на рахунок, це вже робота бекенду.

Бекенд-розробники займаються мовами програмування для сервера, такими як Java, Python, PHP, Ruby та інші. Крім того, вони мають уявлення про бази даних, архітектуру, а також апаратну частину сервера - його можливості і характеристики. Зазвичай бекенд-розробники не працюють з елементами, які безпосередньо взаємодіють з користувачем, такими як інтерфейси UI або користувацький досвід UX, або верстkою сторінок. Однак вони мають загальне уявлення про ці аспекти. В основному, їхні завдання пов'язані з точним аналізом та обчисленнями, де вимагається більше логічного мислення, ніж творчого підходу. Важливо, що вони здатні прогнозувати всі можливі наслідки операцій та розуміти причини помилок, що можуть виникнути в процесі взаємодії між клієнтом та сервером.

Поглянемо на процес взаємодії між frontend і backend:

1. Frontend передає інформацію, призначену для користувача, до backend.
2. Ця інформація обробляється.
3. Оброблена інформація повертається назад у цілісній формі, відповідаючи на запит.
4. Ці завдання зазвичай виконують кілька фахівців одночасно, що створює сприятливе взаємодоповнююче робоче середовище.

Існують кілька способів взаємодії frontend і backend:

- HTTP-запит надсилається на сервер, де інформація шукається, вбудовується в шаблон і повертається у вигляді HTML-сторінки.

- Використання AJAX (асинхронний JavaScript та XML). JavaScript, що виконується у браузері, надсилає запит, і відповідь надходить у форматі XML або JSON.

- Односторінкові застосунки, які отримують дані без перезавантаження сторінок. Це можливо завдяки AJAX або фреймворкам Angular і Ember.

- Фреймворки, такі як Ember або бібліотека React, допомагають у використанні програм як у клієнтській, так і у серверній частині. Frontend і backend взаємодіють через AJAX і HTML-код, який обробляється на сервері. Одночасно вони сприяють зручності та ефективності у виконанні завдань обох напрямків розробки.

Функції і обов'язки між frontend і backend розробниками часто розділяються, зважаючи на специфіку завдань кожного напрямку. Однак існують ситуації, коли програмістам потрібно працювати над рішенням проблем як на стороні сервера, так і в клієнтській частині. Це вимагає широкого спектру знань і вмінь, а також уміння ефективно комунікувати між фронтенд і бекенд командами для розв'язання складних завдань.

В деяких випадках зустрічаються універсальні фахівці, які здатні працювати як у frontend, так і у backend. Вони мають глибоке розуміння різних мов програмування, фреймворків та інструментів, що дозволяє їм ефективно переходити від одного напрямку розробки до іншого. Ці спеціалісти, що охоплюють обидва напрямки, можуть виявитися вельми корисними для команд, які шукають універсальних розробників для вирішення широкого спектру завдань.

Такі універсальні програмісти мають можливість працювати над комплексними проектами, здатні адаптуватися до змін та розвивати програмні рішення, які забезпечують оптимальну взаємодію між клієнтською та серверною частинами. Їхні здібності у розумінні різних аспектів розробки дають можливість швидко вирішувати проблеми та забезпечувати стабільну та ефективну роботу програм.

3.2 Архітектура серверної частини програми

Клієнтська частина зв'язується з серверною за допомогою інтернет протоколу HTTP.

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпертекстових документів.HTTP належить до протоколів моделі OSI 7-го прикладного рівня.

Основним призначенням протоколу HTTP є передача веб сторінок (текстових файлів з розміткою HTML), хоча за допомогою нього успішно передаються як інші файли, які пов'язані з веб сторінками (зображення та застосунки), так і не пов'язані з ними (у цьому HTTP конкурує зі складнішим FTP).

HTTP припускає, що клієнтська програма — веб браузер — здатна відображати гіпертекстові веб сторінки та файли інших типів у зручній для користувача формі. Для правильного відображення HTTP дозволяє клієнтові дізнатися мову та кодування символів веб сторінки й/або запитати версію сторінки в потрібних мові/кодуванні, використовуючи позначення зі стандарту MIME.

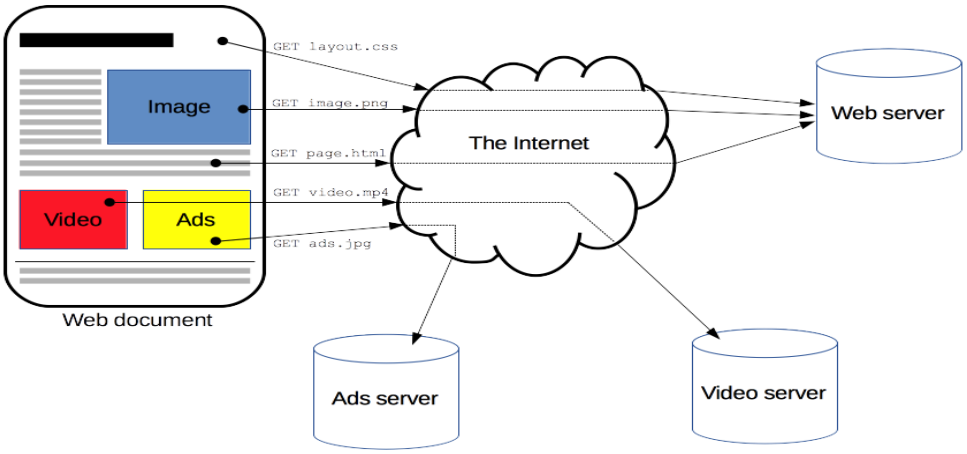

Рисунок 3.2 – Схема зв'язку HTTP.

Після того як запит надійшов з клієнтської частини, перш за все він попадає в Middleware. У проекті реалізовано два вида Middleware.

Функції промежуточної обробки (проміжне програмне забезпечення) - це функції, що мають доступ до об'єкта запрошення (req), об'єкта відповіді (res) та до наступних функцій змінної обробки в циклі додатків "запит-відповідь". Наступна функція змінної обробки, як правило, визначається переміною наступною.

Функції промежуточної обробки можуть виконувати наступні завдання:

- Виконання любого кода.
- Внесення змін у об'єкти запитів та відповідей.
- Завершення цикла “запит-відповідь”.
- Виклик наступної проміжуточної обробки з стека.

Якщо поточна функція промежуточної обробки не завершує цикл “запит-відповідь”, вона повинна викликати next () для передачі управління наступною функцією змінної обробки. У протилежному випадку запрошення зависне.

Middleware при авторизацій:

- check-access-token.js – даний middleware проводить валідацію та перевірку токена який надійшов з клієнтської частини програми
- check- refresh -token.js - даний middleware проводить валідацію та перевірку refresh токена який надійшов з клієнтської частини програми

Middleware при реєстрації:

- check-is-user-exist.js – даний middleware здійснює перевірку на наявність даного usera у базі даних. Якщо такий користувач був знайдений то в реєстрацій буде відмовлено

- check-is-user-valid.js – проводить валідацію таких полів що надійшли з клієнтської частини: username, password, country, city.

Якщо запит успішно пройшов middleware то він попадає у роутер. Маршрутизація визначає, як додаток відповідає на запит клієнта за конкретною адресою (URI). Метод маршруту є виробничим з одного з методів HTTP і приєднується до екземпляра класу express.

Шляхи маршрутів, в поєднанні з методом запиту, визначають конкретні адреси (кінцеві точки), в яких можуть бути створені запити. Шляхи маршрутів можуть являти собою рядки, шаблони рядків або регулярні вирази.

У програмі реалізовано такі роутери API роутери, роутер авторизації і роутер реєстрації. У файлі user.router.js реалізовано маршрутизація у відповідні сервери програми відповідно до URL-запиту. Роутер перенаправляє запит у відповідний сервіс.

```
axios.defaults.headers.post['Content-Type'] ='application/json;charset=utf-8';
axios.defaults.headers.post['Access-Control-Allow-Origin'] = '*';
const response = await axios.post( url: 'http://localhost:5000/api/users/', data)
```

Рисунок 3.3 - Приклад запиту на створення нового користувача.

В сервісі реалізовані функції що взаємодіють з базою даних за допомогою бібліотеки sequelize. При реєстрації використовують такі методи: createUser.

- createUser - створює запис у табличці usermoddels.
- createCity – створює запис у табличці citymoddels.
- getUsers – повертає усі записи з таблиці usermodels.
- getUserByName – проводить вибірку по табличці usermoddels, та шукає співпадіння у атрибуті Username

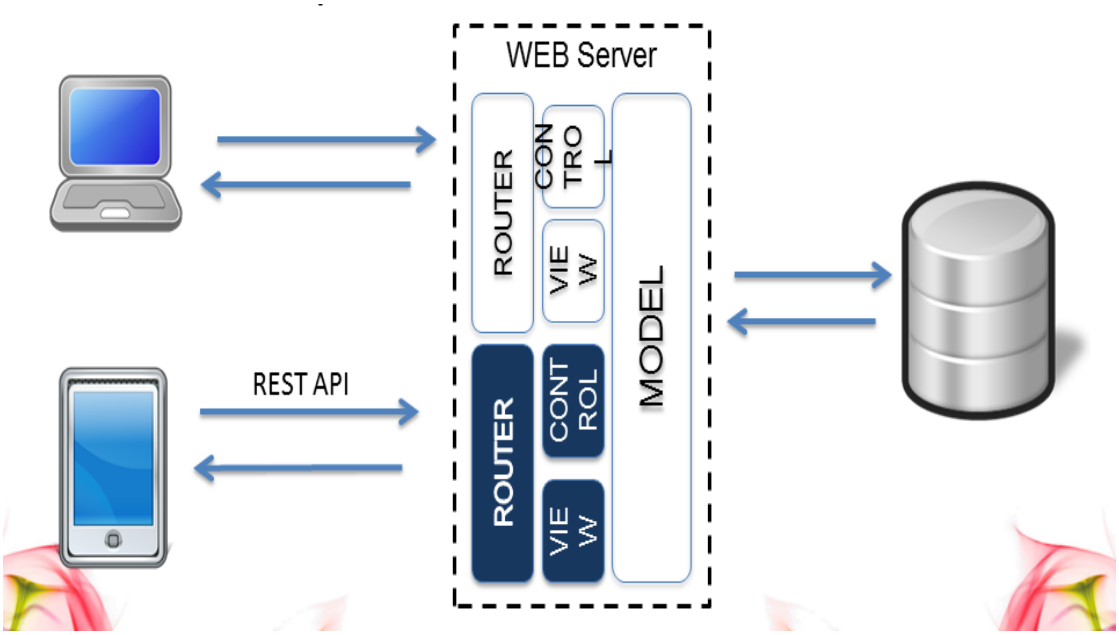

Рисунок 3.4 – Схема серверної частини програми.

3.3 Встановлення та доступ до додатку

Оскільки дипломний проект є веб-додатком для успішного його запуску потрібно мати стальне підключення до мережі Internet. Так, як у клієнтській частині програми використовуються нові методології розробки Front-End необхідно встановити свій клієнтський веб-браузер до останньої наявної версії. Крім JavaScript програма використовує також мову розмітки HTML 5 та мову стилів CSS 3.

HTML — це мова тегів, засобами якої здійснюється розмічання веб-сторінок для мережі Інтернет. Веб-браузери отримують HTML-документи з веб-сервера або з локальної пам'яті й передають документи в мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і спочатку включені сигнали для зовнішнього вигляду документа.

Елементи HTML є будівельними блоками сторінок HTML. За допомогою конструкцій HTML, зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані у візуалізовану сторінку. HTML надає засоби для створення структурованих документів, позначаючи структурну семантику тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML окреслені тегами, написаними з використанням кутових дужок. Теги, такі як і безпосередньо вводять вміст на сторінку. Інші теги, такі як <img /> <input /> <p> оточують і надають інформацію про текст документа і можуть включати інші теги як піделементи. Браузери не показують теги HTML, але використовують їх для інтерпретації вмісту сторінки.

CSS — це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі ж сторінки написані мовами розмітки даних.

CSS є основною технологією всесвітньої павутини, поряд із HTML та JavaScript.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

3.4 Варіанти використання

У веб-додатку доступний лише один актор, при запуску додатку ми попадаємо на сторінку Home.

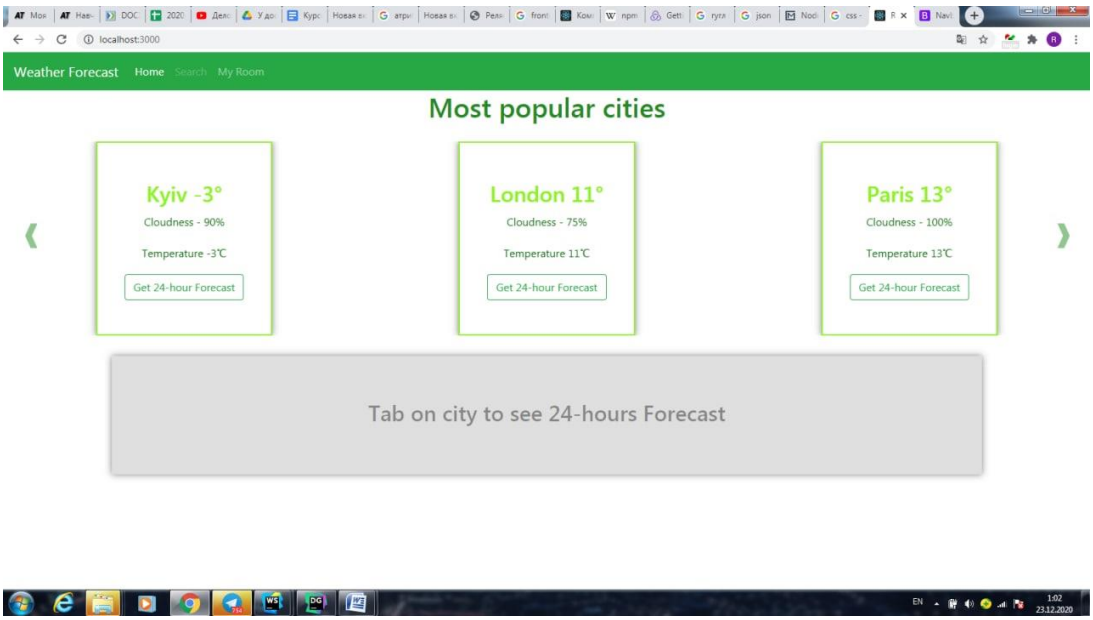

Рисунок 4.1 – Стартова сторінка.

На стартові сторінці неавторизованого користувача, доступно лише функція перегляду погоди у найбільш популярних містах. Доступ до компонента «пошук» їм заборонено.

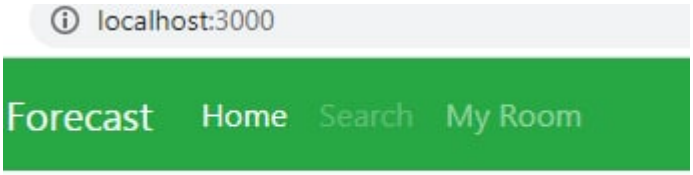

Рисунок 4.2 – Не Активний компонент «Пошуку».

У користувачів є можливість перейти в компонент моя кімната для реєстрації і авторизації на сайті.

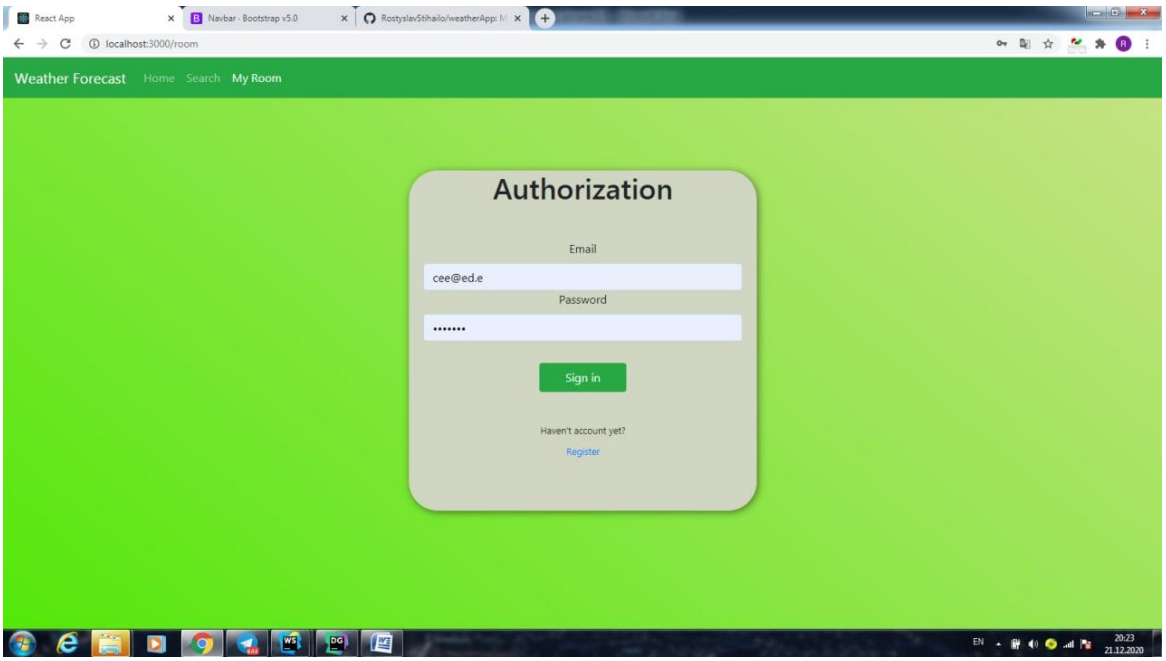

Рисунок 4.3 – Компонент «Моя кімната».

Для не зареєстрованих користувачів є можливість реєстрацій нових користувачів на сайті, шляхом натискання посилання Register.

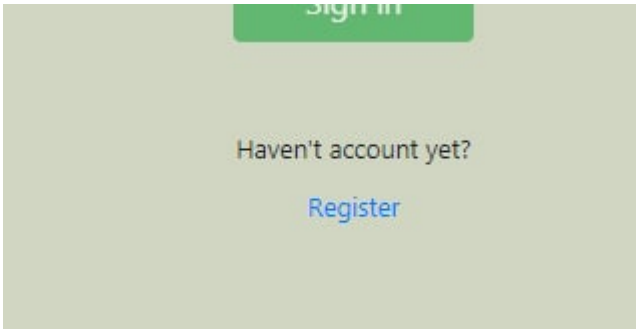

Рисунок 4.4 – Посилання Register.

Після чого користувача попадає на сторінку реєстрацій нових користувачів.

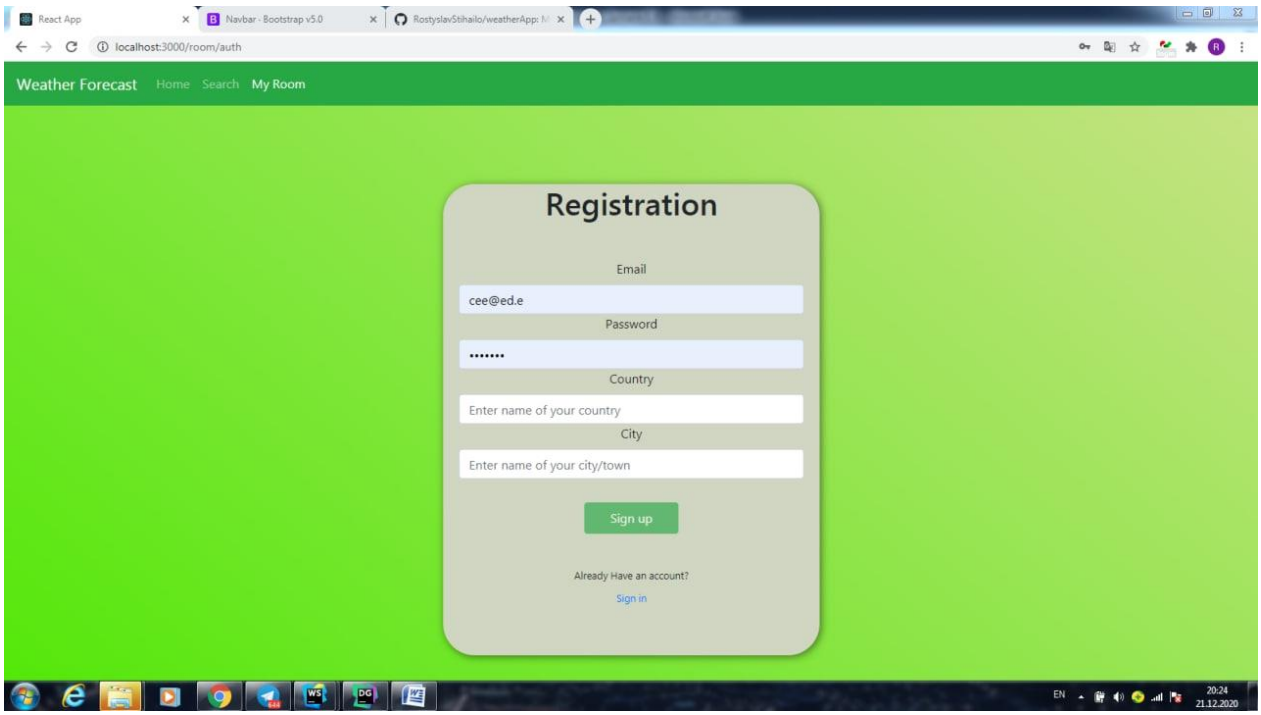

Рисунок 4.5 – Сторінка реєстрацій.

Після реєстрацій користувач отримує можливість зайти на сайт, за своїми даними які він води при реєстрацій. Після успішної авторизації на сайті клієнтська частини отримує від серверної частини згенерований JWT Token.

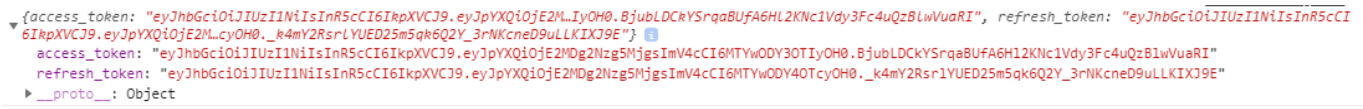

Рисунок 4.6 – Об’єкт сервер при успішні авторизації користувача.

При успішні авторизації створюється запис у базі даних у табличці Tokenmodels. Що містить у собі JWT Token та його Refresh.

	id	user_id	access_token	refresh_token	createdAt	updatedAt
1	1	2	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV…	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV…	2020-12-21 10:54:13	2020-12-21 10:54:13
2	2	4	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV…	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV…	2020-12-21 19:07:11	2020-12-21 19:07:11
3	3	4	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV…	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXV…	2020-12-22 23:15:28	2020-12-22 23:15:28

Рисунок 4.7 – Приклад запису в Базі даних.

Після авторизації компонент пошуку активується і користувач може з ним взаємодіяти.

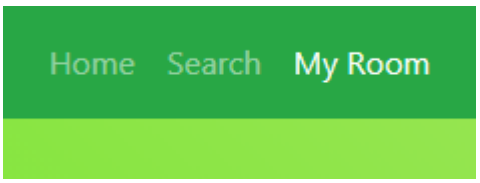

Рисунок 4.8 – Активний компонент пошуку.

3.5 Тестування продукту.

Тестування БД дає можливість виявити переважну більшість критичних місць розроблювальної ІС та перевірити всі механізми, що повинні забезпечити цілісність та конфіденційність даних. Воно проводиться не лише під час розробки системи та схеми БД, але також на етапі супроводу, що пояснюється зміноювимогдосистемизісторонизамовникаабовиявленнямпомилокврроботі тощо. Тестування БД поділяється на види.

Тестування логічної моделі:

- Перевірка моделі на логічну узгодженість і відсутність інформації, що повторюється;

- Пошук можливостей для спрощення логічної моделі. Тестування логічної схеми бази даних:

- Тестування на відповідність нормальним формам (зазвичай третьої);

- Тестування на узгодженість бази;

- Тестування для виявлення даних, що перебувають в надлишку. Тестування фізичної структури бази даних:

- Аналіз і настроювання покриття індексу;

- Аналіз системи зберігання даних (табличні області (Oracle, DB2), масивів даних і груп файлів (MSSQL));

- Аналіз політики безпеки та розробка пропозицій щодо її поліпшення;

- Аналіз та реалізація розподілу бази даних;

- Аналіз та реалізація стратегії реплікації;

Тестування сайту займає ключовий момент в подальшому розвитку інтернет ресурсу і допомагає визначити його якість. Дана послуга дозволяє перевірити на наявність помилок, допущених при розробці. Тестування сайтів виконується для нових ресурсів і тих, які вже давно працюють, але не показують хороших результатів, ресурсів на етапі модернізації.

Комплекс робіт складається з декількох етапів для визначення правильності виведення інформації, структури сайту, зручності використання, швидкості роботи, уразливості ресурсу.

Підхід до перевірки сайту на наявність помилок і відповідності технічного завдання безпосередньо залежить від виду тестування, проте сам процес і етапи робіт мають чітку стратегію виконання. Що перевіряється на веб сайті? Фахівці з тестування сайтів виконують перевірку відповідно до поставлених завдань клієнта.

Коли необхідно виконати комплексне тестування ресурсу, дотримуються наступні етапи?

- перевірка функціональності
- перевірка зручності використання сайту
- тестування швидкості роботи і продуктивності
- тестування на вразливість, безпеку сайту
- тестування правильності структури html коду
- тестування логічної структури побудови

- кожен з цих видів тестування має свій алгоритм перевірки
- Тестування функціоналу сайта

Перевірка функціоналу ресурсу ставитися до основних видів тестування веб сайту. Тестувальник перевіряють правильність реалізації функціональної частини. Багато замовників навіть не здогадується, що сайт може нормально не працювати на різних браузерах. Кросбраузерність важливий момент, який не беруть розробники, а замовник в подальшому втрачає клієнтів.

Тестування виконується на всіх популярних браузерах Chrome, Firefox, Opera, Сафарі, Microsoft Edge.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТАЦІЯХ

4.1 Охорона праці

Зазвичай процес розробки веб-додатку є тяжким та довготривалим та виснажливим, так як приходиться працювати з багатьма технологіями та великим обсягом даних, через це існує ймовірність припуститися помилки. Усім добре відомо, що одним із головних методів забезпечення ефективної роботи з комп'ютером є забезпечення належних умов праці, так як при недостатньому освітлені, просторі чи неякісному дисплеї, людина відчуватиме фізичний та з часом психологічний дискомфорт, який обов'язково відобразиться на результатах роботи. Для того щоб запобігти цьому, потрібно дотримуватись ряду наказів та стандартів.

В наказі № 207 від 14.02.2018 НПАОП 0.00-7.15-18 [15], якраз описується частина вимог, яких потрібно дотримуватись при роботі з екранними пристроями.

Відповідно до третьої частини наказу [15], робоче місце працівника має відповідати наступним вимогам:

1. Робоче місце має мати достатні розміри, щоб працівник мав простір для зміни робочого положення та рухів.
2. Усе випромінювання від екранних пристроїв має бути знижене до гранично допустимого рівня з погляду безпеки та охорони здоров'я працівників.
3. Усі елементи робочого місця та їх розташування мають відповідати ергономічним, антропологічним, психофізіологічним вимогам, а також характеру виконуваних робіт.
4. Освітлення робочого місця має створювати відповідний контраст між екраном і навколишнім середовищем та відповідати вимогам ДСанПІН 3.3.2.007- 98.
5. Мікроклімат виробничих приміщень має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [16].

6. Робочий стіл чи поверхня повинні бути достатнього розміру та мати поверхню з низькою відбивною здатністю, бути гнучкою під час розміщення екрана, клавіатури, документів чи устаткування.
7. Робоче крісло має бути стійким і дозволяти працівнику легко рухатися та займати зручне положення. Сидіння має регулюватися по висоті, спинка сидіння – по висоті, та з можливістю нахилу. Для зручності слід передбачати підніжку для тих, кому це необхідно.

Також потрібно пам’ятати про безпеку, відповідно до четвертої частини наказу [15], потрібно дотримуватись наступних мінімальних вимог безпеки:

1. Перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.
2. Після закінчення роботи пристрої слід відключати від живлення.
3. При виникнення аварійної ситуації необхідно в той же час відключити пристрій від електричної мережі.
4. Не допускається:

– Ремонтувати чи виконувати технічне обслуговування, і налагодження екранних пристроїв на робочому місці працівника під час роботи з екранними пристроями;

– Вимикати захисні пристрої чи проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;

– Працювати з несправними екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, мигання та інші несправності.

5. Під час виконання робіт з комп’ютером, пов’язаних з нервово-емоційним напруженням мають дотримуватися оптимальні умови мікроклімату відповідно до вимог ДСН 3.3.6.042-99 [16].

Для забезпечення безпеки відповідно до п’ятої частини наказу «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [15], усі екранні пристрої повинні відповідати наступним мінімальним вимогам безпеки:

Відповідно до третьої частини наказу [15], робоче місце працівника інтернетмагазину, який працює над контентним наповненням сайту має відповідати наступним вимогам:

1. Екранні пристрої не мають бути джерелом ризику для працівників.
2. Усе випромінювання має бути зведене до мінімального рівня з погляду безпеки і охорони здоров’я працівників.
3. Символи на дисплеї мають бути чіткими, відповідного розміру. Між символами і рядками символів повинна бути правильна відстань.
4. Зображення на дисплеї має бути стабільним, без миготінь або інших видів несправності.
5. Яскравість та контрастність символів має легко регулюватися, а також швидко адаптуватися до навколишніх умов.
6. Під час вибору монітора, слід надавати перевагу тим пристроям, які мають можливості повороту та нахилу екрану.
7. При потребі монітор може бути закріпленим на окремому столі чи підставці.
8. При виборі монітора надавайте перевагу дисплеям з матовим покриттям, щоб мінімізувати відблискування або відбивання світла.
9. При виборі клавіатури, слід надавати перевагу тій, яка відкидається і є автономною, щоб працівник міг вибрати зручну робочу позу й уникнути втоми рук.
10. Поверхня клавіатури має бути матовою, щоб уникнути віддзеркалювання.
11. Устаткування, яке входить до робочої станції, не повинно виділяти надлишкового тепла.
12. Під час розробки, вибору, замовлення та модифікації програмного забезпечення, а також під час розробки завдань, що передбачають використання устаткування з екранними пристроями, роботодавець має керуватися таким програмним забезпеченням, яке відповідає розв’язуваним завданням і є простим у використанні, а де необхідно - адаптованим до рівня знань і досвіду працівника.

Отже, для безпечної та ефективної роботи контент-менеджера інтернетмагазину забезпечено належні умови праці, починаючи від робочого місця та його оснащення, та закінчуючи мікрокліматом робочого середовища, відповідно до вимог чинного законодавства.

4.2 Безпека в надзвичайних ситуаціях

Ефективність роботи людини з комп’ютером значною мірою визначається функціональним станом людини. Психофізіологічні та емоційні перенапруження, втома людини-оператора можуть призвести в комп’ютеризованих системах керування до помилок і як наслідок – до значних економічних втрат.

Згідно зі статистичними даними від 40 до 75% аварій літаків зумовлено людським фактором [17]. Відмови комп’ютеризованої системи керування рухом залізничного транспорту, на гірничо-збагачувальних комбінатах з вини операторів становлять понад 50% їх загальної кількості, причому значна їх частина спричинена невідповідністю функціонального стану оператора складності виконуваної роботи.

Трудова діяльність користувачів комп’ютерів відбувається у певному виробничому середовищі, яке впливає на їх функціональний стан. Найбільш значимі – фізичні фактори виробничого середовища, до яких належать електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ціла низка світлотехнічних показників.

Трудовий процес суттєво впливає на психофізіологічні можливості користувачів комп’ютерів, оскільки їх діяльність характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес користувачів комп’ютерів відзначається значними інформаційними навантаженнями.

Професійні якості та виробничий досвід, які визначають внутрішні засоби діяльності, обумовлюють надійну та безпомилкову діяльність користувачів комп’ютерів, дозволяють знаходити безпечні методи розв’язання виробничих завдань навіть у нестандартних ситуаціях.

Зовнішні засоби діяльності, які в основному визначаються ергономічними показниками щодо організації робочого місця, формою та параметрами його елементів, просторового розташування основного і допоміжного устаткування, можуть суттєво знизити фізичні та психофізіологічні навантаження, що діють на користувачів комп’ютерів.

Оскільки робота користувачів комп’ютерів найчастіше проходить за активної взаємодії з іншими людьми, то виникають питання раціоналізації міжособистісних стосунків. Цей комплекс питань порушує як психологічні, так і соціально- психологічні аспекти трудових взаємовідносин, які також є факторами "ризику", що відчутно впливають на функціональний стан користувачів комп’ютерів.

Визначення та вивчення факторів, що впливають на функціональний стан користувачів комп'ютерів дозволить виділити основні причини виникнення станів напруженості, стомлення, стресу і здійснити відповідні профілактичні заходи.

Отже, до основних факторів, що впливають на функціональний стан користувачів комп’ютера належать:

1. середовище – характеризується такими шкідливими факторами:

1.2 фізичні: електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ряд світлотехнічних показників;

1.3 хімічні: пил, шкідливі хімічні речовини, які виділяються при роботі принтера і копіювальної техніки;

1.4 біологічні: підвищений вміст в повітрі патогенних мікроорганізмів, особливо у приміщенні з великою кількістю працюючих, при недостатній вентиляції, особливо у період епідемій; психофізіологічні: напруження зору та уваги, інтелектуальні та емоційні навантаження, тривалі статичні навантаження і монотонність праці.

2. трудовий процес - характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес користувачів комп'ютерів відзначається значними інформаційними навантаженнями;
3. внутрішні засоби діяльності – це професійні риси та виробничий досвід, які обумовлюють надійну та безпомилкову діяльність користувачів комп'ютерів, дозволяють знаходити безпечні методи розв'язання виробничих завдань навіть у нестандартних ситуаціях;
4. зовнішні засоби діяльності - визначаються ергономічними показниками щодо організації робочого місця, форми та параметрів його елементів, просторового розташування основного і допоміжного устаткування, які можуть суттєво знизити фізичні та психофізіологічні навантаження, що діють на користувачів комп'ютерів;
5. соціально-психологічні фактори трудових взаємовідносин.

У професійних операторів частіше зустрічаються порушення органів зору, опорно-рухового апарату, центральної нервової, серцево-судинної, імунної та статевої систем, захворювання шкіри. Зафіксована значна кількість скарг операторського персоналу на загальне недомагання, передчасне стомлювання, головний біль, порушення функцій органів зору, які здійснювали несприятливий психофізіологічний вплив на самопочуття та працездатність операторів.

Сучасна професія користувача ВДТ належить до розумової праці, яка характеризується: високою напруженістю зорових функцій; одноманітною позою; великою кількістю стереотипних висококоординованих рухів, що виконуються лише м'язами кистей рук на фоні малої загальної рухової активності; значним нервовоемоційним компонентом, особливо в умовах дефіциту часу; роботою з великими масивами інформації, що викликає активізацію уваги та інших вищих психічних функцій. Крім того, при роботі з дисплеями на електронно-променевих трубках виникає вплив на користувача цілої низки факторів фізичної природи — електростатичні поля, радіочастотне та рентгенівське випромінювання тощо.

Діяльність професіоналів можна поділити на три групи:

1. Діяльність, яка пов'язана з виконанням нескладних багаторазово повторюваних операцій, що не вимагають великого розумового напруження. Наприклад, робота операторів комп'ютерного набору, працівників довідкових служб.
2. Діяльність, яка пов'язана із здійсненням логічних операцій, що постійно повторюються. Це робота інженера-економіста, інженера-проектувальника, оператора автоматизованого виробництва.
3. Діяльність, коли в процесі роботи необхідно приймати рішення за відсутності заздалегідь відомого алгоритму. Наприклад, робота інженера програміста, диспетчерів руху залізничного транспорту, аеропортів тощо.

У користувачів, які інтенсивно використовують комп'ютер в умовах значних розумових напружень досить часто (40—70%) виникають психологічні та поведінкові порушення (нервозність, роздратування, тривога, нерішучість, замкнутість тощо). Серед користувачів ВДТ в США і Європі значного поширення набуло специфічне захворювання, яке отримало назву синдром комп'ютерного стресу (СКС). СКС супроводжується головним болем, запаленням очей, алергією, роздратованістю, млявістю і депресією. Інформаційне перевантаження користувачів ВДТ супроводжується низкою специфічних захворювань, які називають інформаційними. Першим симптомом їх є головний біль. Дослідження, проведені в США, Німеччині, Швейцарії та інших країнах, показали, що робота з обслуговування ВДТ супроводжується підвищеним напруженням зору, інтенсивністю і монотонністю праці, збільшенням статичних навантажень, нервово-психічним напруженням, впливом різного виду випромінювань та ін. Внаслідок цього серед операторів ВДТ, як зазначають фахівці Всесвітньої організації охорони здоров'я, частіше, ніж в інших групах працюючих, трапляються такі професійні захворювання, як передчасна стомлюваність, погіршення зору, м'язові і головні болі, психічні й нервові розлади, хвороби серцево-судинної системи, онкологічні захворювання та ін. Вважається, що стан організму операторів ВДТ визначається комплексним впливом факторів трудового процесу і середовища, значення яких є неоднаковим. На операторів з малим стажем роботи на ВДТ домінуючий вплив чинять фактори середовища, а на операторів зі стажем понад 5 років - фактори трудового процесу. Комп'ютерний зоровий синдром (КЗС) - комплекс порушень здоров'я, який може виникати у користувачів персональних комп'ютерів (ПК) [18]. У користувачів ПК дуже поширені кон'юнктивіти і блефарити, патогенетично пов'язані з КЗС. Синдром розвивається при умові, що робоче місце організовано неправильно - у користувача незручне крісло, відсутні пюпітри для паперів, підставки для ніг та кистей рук, не встановлена висота і нахил монітора відносно очей, відстань від очей до екрана. За таких умов тіло людини при роботі займає вимушене положення: спина статично напружена, шия витягнута, плечі жорстко фіксовані.

Напружені м'язи погіршують кровотік у сонних артеріях, а недостатнє кровозабезпечення головного мозку веде до очманіння, появи головного болю. На фоні шийного остеохондрозу з'являється відчуття випирання очних яблук, туману в очах, мушок та райдужних кіл у полі зору. Розвитку КЗС сприяє поганий мікроклімат приміщення, значна загальна іонізація та мікробне забруднення, а також куріння.

Національною радою з наукових досліджень США для стану зорового дискомфорту був уведений термін "астенопія", який означає "будь-які суб'єктивні зорові симптоми чи емоційний дискомфорт, що є результатом зорової діяльності". Симптоми астенопії були класифіковані на "очні" (біль, печія та різь в очах, почервоніння повік та очних яблук, ломота у надбрівній частині тощо) та "зорові" (пелена перед очима, мерехтіння, швидка втома під час зорової роботи та ін.).

Таким чином, на користувача комп'ютера впливає комплекс факторів. Урахування ступеня та якості впливу цих факторів на функціональний стан дозволяють розробити заходи та засоби щодо забезпечення безпеки, підвищення працездатності та збереження здоров'я користувачів комп'ютерів.

ВИСНОВОК

Під час виконання дипломного проекту було створено веб-додаток, спрямований на надання функціоналу авторизації з використанням бази даних. Зазначеній програмі було проведено всебічне тестування, алгоритм авторизації було реалізовано в зрозумілій формі. Головною метою створення додатку було надання інформації про погодні умови авторизованим користувачам.

Клієнтську та серверну частини програми реалізовано з використанням можливостей об’єктно-орієнтованої мови програмування JavaScript. В якості бази даних було обрано СУБД MySQL, основні переваги якої включають швидкість роботи, захищеність та простоту використання. Для взаємодії з СУБД використовувалась бібліотека Sequelize, що забезпечила зручність і ефективність в маніпулюванні даними.

Проект був розроблений у середовищі програмування WebStorm, яке було використано для написання як серверної, так і клієнтської частин програми. Для взаємодії з базою даних був використаний редактор коду DataGrip від компанії Jetbrains.

Програма демонструє чітку структуру, розбиту на класи, кожен з яких виконує відповідні завдання у клієнтській та серверній частинах програми. У розробці застосовувався компонентний підхід до програмування, що сприяло більшій організації та підтримці програми.

Дипломний проект відповідає всім необхідним умовам та був реалізований відповідно до методичних вказівок. Пояснювальна записка була відформатована та відповідає усім потрібним умовам для успішного виконання дипломного проекту.

У цілому, розробка даного програмного забезпечення підтверджує його високий рівень функціональності, безпеки та відповідність вимогам, що ставиться до сучасного програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React [Електронний ресурс]//Facebook Open Source – 2020. - Режим доступу до ресурсу: https://uk.reactjs.org.
2. React [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/React.
3. Node.js [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js.
4. MySQL [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MySQL.
5. Методичний вказівник [Електронний ресурс] // Бази даних: – 2020. Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=118596.
6. Node.JS [Електронний ресурс]//OpenJs Foundation – 2020. Режим доступу до ресурсу: https://nodejs.org/uk/about.
7. JSON [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JSON.
8. Куваєв Я. Г. Нормалізація відношень при проектуванні БД [Електронний ресурс] // Я. Г. Куваєв. – 2020. Режим доступу до ресурсу: https://rdb.dp.ua/uk/chapter_03.
9. Express.js [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Express.js.
10. Redux [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Redux.
11. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» // В.С. Стручок – Тернопіль: ФОП Паляниця В. А., –156 с. Режим доступу до ресурсу: https://elartu.tntu.edu.ua/handle/lib/39196.

12. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» // автор-укладач В.С. Стручок– Тернопіль: ФОП Паляниця В. А., – 156 с. Режим доступу до ресурсу: http://elartu.tntu.edu.ua/handle/lib/39424

ДОДАТКИ

ДОДАТОК А
ТЕЗИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

МАТЕРІАЛИ XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ

2023

УДК **004.4**

Олег Пастух докт. техн. наук, проф., Ростислав Стігайло

Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАГАЛЬНОГО КОРИСТУВАННЯ З КЛІЄНТ-СЕРВЕРНОЮ АРХІТЕКТУРОЮ НА ОСНОВОІ МОВИ JAVASCRIPT

Oleh Pastukh Dr., Prof., Rostislav Stigailo

DEVELOPMENT OF SOFTWARE FOR GENERAL USE WITH CLIENT-SERVER ARCHITECTURE BASED ON THE JAVASCRIPT LANGUAGE

Мета проекту – розробити систему автоматизованої аторизації у веб-додатку за допомогою JWT. Дана робота включає розробку програмного забезпечення на основі використання баз даних. Для розробки цього програмного продукту та для роботи з СУБД mySQL, було використано мову JavaScript. Технологію.React, Express (Node JS). Проект розроблений з компонентним підходом. При вдалій конкуренції на ринку система буде розвиватись. Система була розроблена для спрощення авторизації та підтримування сесії у веб-додатку. Та й в загальному підвищить зручність та ефективність усіх інтернет- додатків [1, 2, 3].

Висновок у цьому проекті було розроблено систему автоматизованої аторизації у веб-додатку за допомогою JSON Web Token (JWT), яка дозволяє перевіряти власника даних JSON, що містять набір клеймів. Для реалізації цієї системи було використано мову програмування JavaScript, базу даних MySQL, технології React та Express, а також компонентний підхід до розробки. Система мала наступні функціональні можливості: реєстрація, авторизація, видача, перевірка, оновлення та вихід користувачів за допомогою JWT. Система мала також наступні переваги: безпека, простота та швидкість. Проект був успішно реалізований та протестований за допомогою різних інструментів, таких як Postman, Jest, Mocha, Chai тощо. Проект демонстрував правильну роботу системи автоматизованої аторизації у веб-додатку за допомогою JWT, а також відповідність всім поставленим вимогам.

Література

1. Node.JS [Електронний ресурс]//OpenJs Foundation – 2020. Режим доступу до ресурсу: https://nodejs.org/uk/about.

2. JSON [Електронний ресурс] // Wikipedia. – 2020. - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JSON.

3. Куваєв Я. Г. Нормалізація відношень при проектуванні БД [Електронний ресурс] // Я. Г. Куваєв. – 2020. Режим доступу до ресурсу: https://rdb.dp.ua/uk/chapter_03.

ДОДАТОК Б
Лістинг коду

```
create table tokenmodals
(
    id           int auto_increment
        primary key,
    user_id      int          not null,
    access_token  varchar(255) not null,
    refresh_token varchar(255) not null,
    createdAt     datetime     not null,
    updatedAt     datetime     not null,
    constraint tokenmodals_ibfk_1
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_10
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_2
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_3
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_4
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_5
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_6
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_7
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_8
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint tokenmodals_ibfk_9
        foreign key (user_id) references usermodals (id)
            on update cascade
);

create index user_id
```

```
    on tokenmodals (user_id);

create table citymodals
(
    id        int auto_increment
        primary key,
    city      varchar(255) not null,
    country   varchar(255) not null,
    createdAt datetime     not null,
    updatedAt datetime     not null,
    user_id   int          not null,
    constraint CityModals_user_id_foreign_idx
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint citymodals_ibfk_1
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint citymodals_ibfk_2
        foreign key (user_id) references usermodals (id)
            on update cascade,
    constraint citymodals_ibfk_3
        foreign key (user_id) references usermodals (id)
            on update cascade
);

create index user_id
    on citymodals (user_id);
```

```
create table usermodals
(
    id        int auto_increment
        primary key,
    username  varchar(255) not null,
    password  varchar(255) not null,
    createdAt datetime     not null,
    updatedAt datetime     not null
);
```

ДОДАТОК В

Лістинг коду серверної частини

package.json:

```
{
  "name": "untitled",
  "version": "1.0.0",
  "description": "",
  "main": "src/app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node src/app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.0.0",
    "express": "^4.17.1",
    "joi": "^14.3.1",
    "jsonwebtoken": "^8.5.1",
    "mysql2": "^2.2.5",
    "sequelize": "^6.3.5"
  }
}
```

app,js:

```
const express = require('express');



const {apiRouter} = require("./routes");

const {sequelize} = require('./dataBase');



const app = express();
```

```
app.use(express.json());

app.use(express.urlencoded());


app.use('/api', apiRouter)

app.use('*', (err, req, res, next) => {

    res

        .status(err.status || 400)

        .json({

            message: err.message,

            code: err.customCode

        });

});


sequelize.sync({alter: true})

    .then(() => app.listen(5000, err => err && console.log(err) ||
console.log('Listen 5000 ...')))

    .catch(console.log);


process.on('unhandledRejection', reason => {

  console.log(reason);

  process.exit(0);

});
```

src/controllers/auth/auth.controller.js:

```
const {statusesCode} = require("../../configs");

const {authService} = require("../../services");

const {jwtTokinazer} = require("../../helpers");

const {ErrorHandler} = require("../../errors");

const {checkHash} = require("../../helpers");


module.exports = {

    loginUser: async (req, res, next) => {

        const {password} = req.body;

        const user = req.user;


        const status = await checkHash(user.password, password);


        if(!status) {

            return next(new ErrorHandler('Bad request', 4001, 400));

        }


        const tokens = jwtTokinazer();

        await authService.createTokens({...tokens, user_id: user.id})


        res.json(tokens)
```

```
    },

    logoutUser: async (req, res, next) => {
        const access_token = req.get('Authorization');

        await authService.deleteByParams({access_token});

        res.sendStatus(statusesCode.OK);
    },

    refreshToken: async (req, res, next) => {
        const refresh_token = req.get('Authorization');

        await authService.deleteByParams({refresh_token});

        const tokens = jwtTokinazer();
        await authService.createTokens({...tokens, user_id: req.userId});

        res.json(tokens)
    }
}
```

src/services/auth/auth.service.js:

```
const TokenModal = require("../../dataBase/models/Token");


module.exports = {
    createTokens:  (tokens) => {
        return TokenModal.create(tokens);
    },


    deleteByParams: async (params) => {
        return  TokenModal.destroy({
            where: params
        });
    },


    getTokensByParams: (params) => {
        return TokenModal.findOne({
            where: params
        });
    }
}
```

src/dataBase/models/Token.js:

```
const {Model, DataTypes} = require('sequelize');
```

```
const {sequelize} = require('../index');


const User = require('./User')


class TokenModal extends Model {
}


TokenModal.init({
    id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    user_id: {
        type: DataTypes.INTEGER,
        allowNull: false
    },
    access_token: {
        type: DataTypes.STRING,
        allowNull: false
    },
    refresh_token: {
```

```
        type: DataTypes.STRING,

        allowNull: false

    }

}, {sequelize})


TokenModal.belongsTo(User, {foreignKey: 'user_id'});


module.exports = TokenModal;
```

src/routes/auth/auth.router.js:

```
const {Router} = require('express')


const authController = require("../../controllers/auth");

const {usersMiddlewares, authMiddlewares} = require("../../middlewares");


const authRouter = Router();


authRouter.post('/',usersMiddlewares.findUserByUsername,
authController.loginUser)

authRouter.post('/refresh',authMiddlewares.checkRefreshToken,
authController.refreshToken)

authRouter.post('/logout', authController.logoutUser)


module.exports = authRouter;
```

src/middlewares/auth/check-access-token.js:

```
const jwt = require('jsonwebtoken');


const {authService} = require("../../services");
const {jwtSecrets} = require("../../configs");
const {ErrorHandler} = require("../../errors");


module.exports = async (req, res, next) => {
    const token = req.get('Authorization');


    if(!token) {
        return next( new ErrorHandler('Not token', 4002, 400))
    }


    jwt.verify(token, jwtSecrets.JWT_SECRET, err => {
        if(err) {
            return next( new ErrorHandler('Not token', 4011, 401))
        }
    });


    const isTokenExist = await authService.getTokensByParams({access_token:
token});
```

```
    if(!isTokenExist) {
        return next( new ErrorHandler('Not token', 4011, 401));
    }

    req.userId = isTokenExist.user_id;

    next();
}
```

src/middlewares/users/find-user-by-username.js:

```
const {statusesCode} = require("../../configs");
const {statusError, ErrorHandler} = require("../../errors");
const {usersService} = require("../../services");

module.exports = async (req, res, next) => {
    const {username} = req.body;

    const user = await usersService.getUserByParams({username})

    if(!user) {
        return next(new ErrorHandler(
            statusError.USER_NOT_FOUND.message,
```

```
            statusError.USER_NOT_FOUND.code,
            statusesCode.NOT_FOUND
        ))
    }

    req.user = user;
    next();
}
```

ДОДАТОК Г

Лістинг клієнтської частини коду

package.json:

```
{
  "name": "untitled",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "axios": "^0.21.0",
    "bootstrap": "^4.5.3",
    "is_js": "^0.9.0",
    "node-sass": "^5.0.0",
    "react": "^17.0.1",
    "react-bootstrap": "^1.4.0",
    "react-dom": "^17.0.1",
    "react-redux": "^7.2.2",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.1",
    "redux": "^4.0.5",
```

```
    "redux-thunk": "^2.3.0",
    "web-vitals": "^0.2.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
```

```
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

src/index.js:

```
import 'bootstrap/dist/css/bootstrap.css';
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import {BrowserRouter} from "react-router-dom";
import {createStore, applyMiddleware} from "redux";
import {Provider} from 'react-redux';
import rootReduser from "./redux/redusers/rootReduser";
import reduxThunk from 'redux-thunk'


const store = createStore(rootReduser, applyMiddleware(
    reduxThunk
))
```

```
ReactDOM.render(

      <Provider store={store}>
            <BrowserRouter>
                <App />
            </BrowserRouter>
      </Provider>
  ,
  document.getElementById('root')
);

reportWebVitals();
```

src/App.js:

```
import React from 'react'
import './App.css';
import Layout from "./components/Layout";

class App extends React.Component {
```

```
  render() {
    return (
      <React.Fragment>
        <Layout/>
      </React.Fragment>
    )
  }
}

export default App;
```

src/components/Layout.js:

```
import React from 'react';
import {Navbar} from "./Navbar";
import {Route, Switch} from 'react-router-dom'
import Home from "./Home";
import Search from "./Search";
import Room from "./Room";
import Reg from "./registr";



export default function Layout () {
    return(
```

```
      <React.Fragment>
          <Navbar/>
          <Switch>
              <Route path='/' exact component={Home}/>
              <Route path='/search' component={Search}/>
              <Route path='/room/auth' exact component={Reg}/>
              <Route path='/room' exact component={Room}/>
          </Switch>
      </React.Fragment>
    )
}
```

src/components/Navbar.js:

```
import React from 'react';
import {NavLink} from "react-router-dom";

export const Navbar = () => (
    <nav className='navbar navbar-dark bg-success navbar-expand-lg'>
        <div className='navbar-brand'>
            Weather Forecast
        </div>
        <ul className='navbar-nav'>
```

```
            <li className='nav-item'>
                <NavLink to='/' exact className='nav-link'>Home</NavLink>
            </li>
            <li className='nav-item'>
                <NavLink to='/search' className='nav-link'>Search</NavLink>
            </li>
            <li className='nav-item'>
                <NavLink to='/room' className='nav-link'>My Room</NavLink>
            </li>
        </ul>
    </nav>
)
```

src/redux/redusers/rootReduser.js:

```
import {combineReducers} from "redux";
import homeReduser from "./home";
import AuthReduser from "./AuthReduser";

export default combineReducers({
    home: homeReduser,
    auth:AuthReduser
})
```

src/redux/redusers/AuthReduser.js:

```
const initialState = {

    Jwt: null,

    isLoggedin: false

}



export default function AuthReduser (state = initialState, actions){

    switch (actions.type){

        default: return state

    }

}
```

src/redux/redusers/home.js:

```
import {FETCH_ERROR, FETCH_FORECAST, FORECAST_ERROR, FORECAST_START, ITEMS_FETCHED} from "../actoins/actionType";



const initialState = {

    townData: null,

    loading: true,

    loading2:null,

    forecast: null,

    days: ['Sunday' , 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'],

    month: ['January',
```

```
        'February',
        'March',
        'April',
        'May',
        'June',
        'July',
        'August',
        'September',
        'October',
        'November',
        'December']
}

export default function homeReduser(state = initialState, action){

    switch (action.type){
        case ITEMS_FETCHED: return {
            ...state, loading: false, townData: action.data
        }
        case FETCH_ERROR: return{
            ...state, loading: true
        }
```

```
        case FORECAST_START: return {
            ...state, loading2: true
        }
        case FETCH_FORECAST: return{
            ...state, loading2: false, forecast: action.data
        }
        case FORECAST_ERROR: return{
            ...state, loading2: true
        }
        default: return state
    }

}
```

src/redux/actoins/actionType.js:

```
export const ITEMS_FETCHED = 'ITEMS_FETCHED';
export const FETCH_ERROR = 'FETCH_ERROR';

export const FETCH_FORECAST = 'FETCH_FORECAST';
export const FORECAST_ERROR = 'FORECAST_ERROR';
export const FORECAST_START = 'FORECAST_START';
```

```
export const ON_EMAIL_CHANGE = 'ON_EMAIL_CHANGE'

export const ON_PASSWORD_CHANGE = 'ON_EMAIL_CHANGE'
```

src/redux/actoins/ActionsAuth.js:

```
import axios from 'axios'



export function auth(data){

return async dispatch => {

    try{

        console.log(data);

        axios.defaults.headers.post['Content-Type']
='application/json;charset=utf-8';

        axios.defaults.headers.post['Access-Control-Allow-Origin'] = '*';

      const response = await axios.post('http://localhost:5000/api/users/',
data)

        console.log(response.data)

       /* fetch('http://localhost:5000/api/users/', {

            method:'POST',

            body: JSON.stringify(data),

            mode: 'no-cors', // no-cors, *cors, same-origin

            cache: 'no-cache', // *default, no-cache, reload, force-cache,
only-if-cached

            credentials: 'same-origin', // include, *same-origin, omit

            headers: {
```

```
                'Content-Type': 'application/json'
            },
        }).then(res => res.json())
            .then(response => {
                console.log(response)})*/
    }
    catch (e){
        console.log(e)
    }
}
}

export function login(data){
    return async dispatch => {
        try{
            console.log(data)
            const response = await
axios.post('http://localhost:5000/api/auth/', data)
            console.log(response.data)
        }
        catch(e) {
            console.log(e)
        }
```

```
    }
}
```

src/redux/actoins/actions.js:

```
import axios from 'axios'

import {FETCH_ERROR, FETCH_FORECAST, ITEMS_FETCHED, FORECAST_ERROR, FORECAST_START} from "./actionType";



export function fetchWeather(){

   return async dispatch => {

       try {

       const data = []

     const kyiv = await axios.get('https://api.openweathermap.org/data/2.5/weather?q=Kyiv&appid=3d59a8198a9dea4c10226f1783e27c35')

       data.push(kyiv.data)

       const london = await axios.get('https://api.openweathermap.org/data/2.5/weather?q=London&appid=3d59a8198a9dea4c10226f1783e27c35')

       data.push(london.data)

       const paris = await axios.get('https://api.openweathermap.org/data/2.5/weather?q=Paris&appid=3d59a8198a9dea4c10226f1783e27c35')

       data.push(paris.data)

       const moscow = await axios.get('https://api.openweathermap.org/data/2.5/weather?q=Moscow&appid=3d59a8198a9dea4c10226f1783e27c35')
```

```
            data.push(moscow.data)

            const berlin = await
axios.get('https://api.openweathermap.org/data/2.5/weather?q=Berlin&appid=3
d59a8198a9dea4c10226f1783e27c35')

            data.push(berlin.data)

            const boston = await
axios.get('https://api.openweathermap.org/data/2.5/weather?q=Boston&appid=3
d59a8198a9dea4c10226f1783e27c35')

            data.push(boston.data)

        dispatch(itemsFetched(data))}

        catch (e){

            dispatch(fetchError(e))

        }

    }

}



export function fetchForecast(name){

    return async dispatch => {

        try {

            //const days = 3

            //console.log(name)

            dispatch(forecastStart())

            const forecast = await
axios.get(`https://api.openweathermap.org/data/2.5/forecast?q=${name}&appid
=3d59a8198a9dea4c10226f1783e27c35`)
```

```
            dispatch(forecastFetched(forecast))
        }
        catch (e){
            dispatch(forecastError(e))
        }
    }
}

export function forecastFetched(data){
    return{
        type: FETCH_FORECAST,
        data
    }

}

export function itemsFetched(data){
    return{
        type: ITEMS_FETCHED,
        data
    }
}
```

```
export function forecastStart(){
    return{
        type: FORECAST_START
    }
}

export function forecastError(e){
    return {
        type: FORECAST_ERROR,
        error: e
    }
}

export function fetchError(e){
    return {
        type: FETCH_ERROR,
        error: e
    }

}
```

ДОДАТОК Д

Диск з матеріалами магістерської