МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Секція інформаційно-комунікаційні технології

ВИПУСКНА РОБОТА

на тему:

«Веб-сайт інтернет-провайдера «Median» мовою JavaScript з використанням фреймворку Express. Серверна частина»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Шутилєва О. В.**

Студент гр. ІНдн-72с **Кухаренко Ю.В.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
«_____» _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу групи ІНдн-72с спеціальності «122 – Комп'ютерні науки» дистанційної форми навчання Кухаренко Юрія Валентиновича.

Тема: «Веб-сайт інтернет-провайдера «Median» мовою JavaScript з використанням фреймворку Express. Серверна частина».

Затверджена наказом по СумДУ
№_____від_____2021 р.

Зміст пояснювальної записки: 1) літературний огляд за тематикою роботи; 2) постановка задачі; 3) вибір методу рішення задачі; 4) практична реалізація веб-сайту інтернет-провайдера.

Дата видачі завдання «_____»_____2021 р.
Керівник випускної роботи _____ Шутилєва О.В.
Завдання прийняв до виконання _____ Кухаренко Ю.В.

РЕФЕРАТ

Записка: 51 стор., 10 рис., 12 додатків, 16 джерел.

Об'єкт дослідження – сфера інтернет послуг провайдера.

Мета роботи – розробити серверну частину веб-сайту інтернет провайдера "Median".

Методи дослідження – експертний аналіз інструментів для розробки форми реєстрації/авторизації веб-сайту інтернет провайдера "Median".

Результат – розроблена серверна частина форми реєстрації/авторизації веб-сайту інтернет провайдера "Median" за допомогою JavaScript (JS), HTML, CSS, Node.js, npm, express, nodemon, mongoose, база даних розміщена в MongoDB.

NODE.JS, EXPRESS, MONGODB,
MONGOOSE

ЗМІСТ

ВСТУП

Виникнення інтернету почалося з розвитку комп’ютерів в 1950 роках та з появою наукових та прикладних концепцій глобальних обчислювальних мереж практично одночасно в різних країнах таких, як Великобританія та Франція.

Приблизно до 1980 року комп’ютерні мережі були доступні спеціалізованим агентствам, а після 1980 року персональні комп’ютери почали поширюватися в приватному порядку. Викликало величезний попит на мережі. У приватних осіб виявлявся інтерес до особистого спілкування та отримання файлів для своїх ПК.

Зараз інтернет доступний практично для кожної людини, якій цікаві новини, ознайомлення з потрібною інформацією та інше. Наприклад, організація чи компанія, захоче залишити інформацію в інтернеті про себе то це можна зробити за допомогою веб-сайту.

Інтерфейс веб-сайту інтернет-провайдера “Median” простий та інтуїтивно зрозумілий та зручний. На цьому веб-сайті буде розміщена форма для реєстрації та авторизації на платформі Node.js.

Якщо розглянути веб-сайт з точки зору системного адміністратора то проблем не виникає так, як проект вимагає до себе мінімум зусиль по експлуатації та адмініструванння.

Метою випускної роботи є розробка серверної частини веб-сайту інтернет-провайдера “Median”.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Типи архітектурних систем

На цей час використовуються дві моделі організації обчислень та обробки інформації:

- модель розподілених обчислень,
- модель централізованих обчислень.

У 1950 році виникла модель централізованих обчислень. Таку модель можна представити, як аплікацію, що працює на одному або на декількох комп'ютерах, які не з'єднанні між собою.

У 1980 році виникла модель розподілених обчислень, цю модель також можна представити, як аплікацію, яка працює на декількох з'єднаних між собою комп'ютерах [2].

Різниця між моделлю розподілених обчислень та моделлю централізованих обчислень, полягає в тому, що в моделі централізованих обчислень один пристрій обробки та один пристрій збереження даних, хоча пристроїв вводу-виводу (терміналів) може мати декілька. Модель розподілених обчислень використовується мінімум двома пристроями обробки та збереження даних [2].

Архітектура клієнт-сервер, вважається найбільш поширеною версією моделі розподілених обчислень [2].

Клієнт-серверна архітектура складається з двох компонентів [3]:

- клієнт – комп'ютер, що розташований на стороні користувача, відправляє запит на сервер для надання інформації або виконання певних дій. Клієнт після отримання, відправляє запит на сервер, там він обробляється, потім готовий результат повертається клієнтові. Одночасно сервер може обслуговувати кілька клієнтів. Якщо одночасно надійшло більше одного запиту то вони встановлюються в чергу та виконуються сервером послідовно. Інколи запити можуть мати пріоритети. Запит з високим пріоритетом буде

виконуватися першим.

- сервер – потужний комп’ютер, призначений для вирішення певних завдань з виконання програмних кодів, виконання сервісних функцій за запитом клієнтів, надання користувачам доступу до певних ресурсів, зберігання інформації бази даних (рис.1).

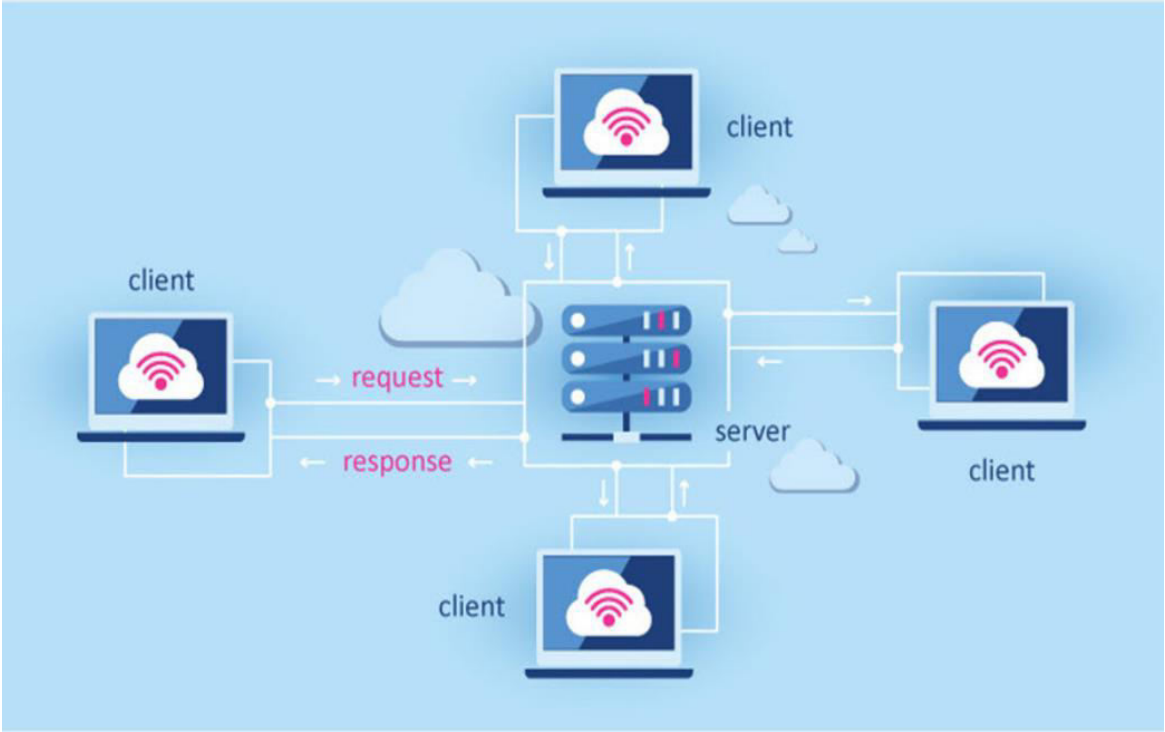

Рисунок 1 – Клієнт-серверна архітектура

Прикладом клієнт-серверної взаємодії є сервіс WWW. На багатьох веб-серверах, на яких розміщується інформація, являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML та каскадних стилів CSS. Значна частина веб-ресурсів на сучасному етапі динамічна, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача [4].

Архітектура «клієнт-сервер» полягає в поділі мережевого додатку на кілька компонентів, кожен з яких реалізує специфічний набір сервісів. Виконуючи серверні або клієнтські функції компоненти такого додатку можуть

виконуватися на різних комп'ютерах. Це дозволить підвищити надійність та безпеку мережевих додатків та мережі в цілому [4].

Модель клієнт-серверної взаємодії розподіляє обов'язки між клієнтом та сервером. Три рівні операцій [5]:

- рівень представлення даних, який по суті являє собою інтерфейс користувача та відповідає за представлення даних користувачу та введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура (рис.2) передбачає взаємодію двох програмних модулів – клієнтського та серверного. У залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта – управління даними, що зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, у якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача здійснюється на стороні клієнта. Товстими клієнтами називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

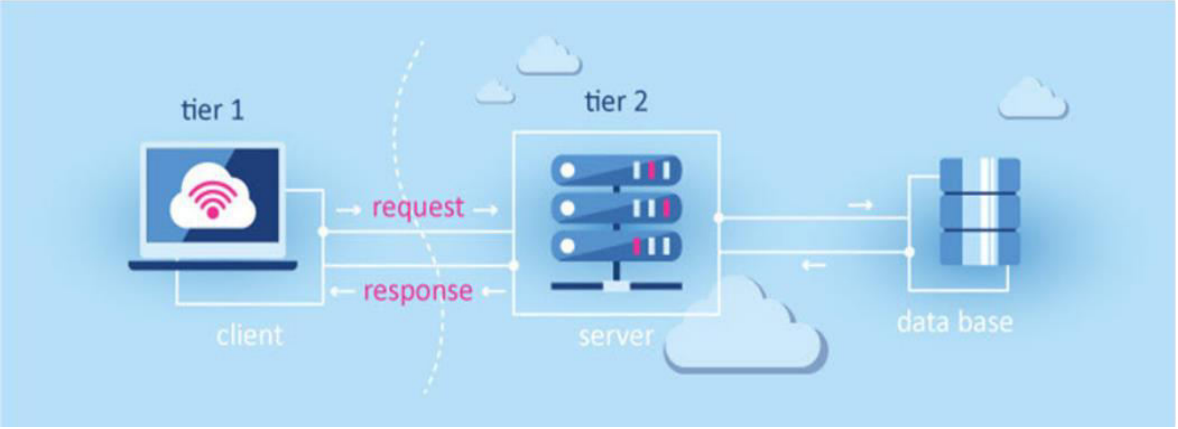

Рисунок 2 – Дворівнева клієнт-серверна архітектура

1.2 Постановка задачі

Метою випускної роботи є розробка серверної частини веб-сайту інтернет-провайдера “Median”. Сайт буде розроблено на програмній платформі Node.js з використанням фреймворку Express на мові JavaScript.

Розроблений веб-сайт повинен мати основні вимоги:

- адаптивність;
- можливість реєстрації/авторизації;
- простий та зрозумілий дизайн;
- можливість замовлення послуг без авторизації;
- протестована форма авторизації/реєстрації в програмі Postman.

Для реалізації мети необхідно виконати наступні завдання:

- створення серверної частини, з можливістю взаємодії з базою даних;
- вибрати методи рішення;
- розробка веб-сайту.

Користувач повинен мати змогу пройти реєстрацію/авторизацію без довгого очікування підтвердження реєстрації адміністратором.

2. ВИБІР МЕТОДУ РІШЕННЯ ЗАДАЧІ

Серверна частина веб-сайту інтернет-провайдера буде розроблено з використанням мови JavaScript в Visual Studio Code.

Серверну частину буде реалізовано за допомогою фреймворку Express.js. на платформі Node.js.

2.1 Вибір Node.js

Серверна частина буде реалізована за допомогою фреймворку Node.js, що дозволить написати сервер на JavaScript. Даний фреймворк має наступні переваги:

- добре працює з великим потоком трафіку;
- дає можливість одночасної обробки великої кількості запитів з низьким часом відгуку;
- використовує рушій V8.

Механізми запитів побудовані на подіях, що унеможливлює заблокувати працюючий рушій – це величезна перевага для операцій введення-виведення, підключення до баз даних, читання дисків, забезпечується висока продуктивність, тому що використовується модель event-driven non-blocking IO, в якій є тільки один потік, який обслуговує клієнтські запити.

У Node прийнята модель, що принципово відрізняється від загальної моделі платформи для побудови серверів додатків, де масштабованість досягається за допомогою багатопотоковості. Архітектура Node знижує споживання пам’яті та підвищує пропускну здатність, спрощує модель програмування.

Зараз платформа Node швидко розвивається. Дуже багато розробників вважає її привабливою альтернативою традиційному підходу до розробки веб-застосувань – на базі Python, Apache, PHP.

У основі Node лежить автономна віртуальна машина JavaScript з

додатками, що робить її придатною для програмування загального призначення на розробку серверів.

Платформу Node ніколи не можна, порівнювати ні з мовами програмування: PHP/Python/Ruby/Java, які використовуються для розробки веб-додатків, ні з контейнерами, що реалізують протокол HTTP (Apache/Tomcat/Glassfish). На теперішній час багато хто вважає, що потенційно, вона може замінити традиційні стеки веб-додатків, веб-сайтів.

Непридатна Node для розробки персональних веб-додатків, веб-сайтів з графічним інтерфейсом користувача. Node не має вбудованого еквіваленту Swing. Немає бібліотеки ГІК (графічний інтерфейс користувача), що підключається для Node, впровадити Node в браузер теж не можна. Якби для Node існувала бібліотека ГІК, то на цій платформі можна було будувати персональні веб-додатки. Не так давно з’явилося декілька проектів із створення інтерфейсу між Node та GTK.

Крім вбудованої можливості виконання коду на JavaScript, модулі, що входять в дистрибутив, надають такі можливості:

- утиліти командного рядка (які буде включено до скриптів оболонки);
- інструменти для написання інтерактивних консольних програм (цикл читання-виконання-друку);
- функції управління процесами для моніторингу дочірніх процесів;
- бінарний об'єкт даних Buffer;
- механізм роз'ємів TCP і UDP з повним набором викликів відповіді;
- пошук у DNS;
- інструменти для побудови серверів і клієнтів протоколів http і HTTPS;
- побудовані на базі бібліотеки TCP-роз'ємів;
- доступ до файлової системи;
- вбудована підтримка офлайн-тестування [7].

Ще Node.js має велику кількістю модулів різного призначення, що дозволяє використовувати готові рішення, але при цьому дає свободу при розробці. У проекті буде використано Node.js модуль:

Express – гнучкий веб-фреймворк для веб-сйтів Node.js, що надає великий набір функцій для веб-сайтів [8]. Express є базовим пакетом, для створення http-серверів, що дозволяє швидко і зручно створити сервер, містить величезну кількість утиліт, що спрощують роботу зі створення веб-сайту.

Nodemon – модуль, який використовується для відстеження змін файлів у каталозі, у якому було запущено вузол. Якщо вони змінюються, програма Node автоматично перезапускає програму Node;

MongoDB – це система управління базами даних з відкритим вихідним кодом (DATABASE), яка не вимагає опису макета таблиці. Написана мовою C.

Mongoose – бібліотека JavaScript, часто використовується в Node.js з базою даних MongoDB. Дозволяє ідентифікувати об'єкти за допомогою строго-типізованої діаграми, яка відповідає документу MongoDB. Mongoose забезпечує величезний набір функціональних можливостей для створення і роботи зі схемами. В MongoDB може зберігатися вісім типів:

- String;
- Number;
- Date;
- Buffer;
- Boolean;
- Mixed;
- Object;
- Array.

Для кожного типу даних можна:

- встановити значення за промовчанням;
- встановити налаштовану функцію перевірки даних;
- вказують на те, що поле потрібно заповнити;
- встановити get-функцію (геттер), яка дозволяє маніпулювати даними до його повернення як об'єкт;
- встановити set-feature, який дозволяє маніпулювати даними перед його

зберіганням в базі даних;

- визначити індекси для швидших даних.

Деякі типи даних також можуть бути налаштовані для збереження та отримання даних з бази даних. Наприклад, для типу даних String можна вказати наступні додаткові параметри [9]:

- перетворення даних на нижній регістр;
- перетворення даних у верхній регістр;
- визначення формального виразу, який дозволяє обмежити параметри даних, дозволені для збереження під час процесу перевірки даних;
- визначення списку, що дозволяє встановити список припустимих рядків.

Пакетний менеджер npm – система управління пакетами, що встановлюється разом з Node.js. Вона дозволяє завантажувати модулі (або пакети) Node.js для розширення функціональності застосування [5].

За допомогою npm можна встановити пакет локально або глобально. У локальному режимі пакети розміщуються в каталозі node_modules батьківського каталогу. Каталог належить поточному користувачу. Глобальні пакети встановлюються в «префіксі»/lib/node_modules/catalog, власником якого є root (префіксом у цьому випадку зазвичай є каталог /usr/ або /usr/local). Якщо використовувати sudo для встановлення пакета то таке може призвести до помилок керування живленням у вирішенні сторонніх залежностей, створюючи проблему безпеки.

Для використання пакетів npm проект повинен містити файл з ім'ям package.json. У цьому пакеті знайдено метадані, пов'язані з проектами.

Метадані відображають декілька аспектів проекту в такому порядку:

- назва проекту;
- оригінальна версія;
- опис;
- точка входу;
- тестові команди;

- Git репозиторій;
- ключові слова;
- ліцензія;
- обмеження [6].

Головною метою, вважається автоматизоване управління залежностями та пакетами. Це означає, що можна вказати всі залежності проекту у файлі package.json, тоді в будь-який час, можна почати роботу з проектом, запускаючи npm install отримаємо всі встановлені залежності.

Можна вручну завантажити бібліотеки, скопіювати їх у правильні каталоги. Проте, якщо проект зростає, це швидко стає трудомістким і брудним. Це також ускладнює співпрацю та обмін з проектом. Для розробника JavaScript на стороні сервера, npm являється незамінним інструментом.

2.2. Вибір бази даних

Бази даних є невід'ємною частиною більшості сучасних веб-сайтів. У даний момент існують різноманітні бази даних за структурою та іншими характеристикам. Вибір бази повинен забезпечувати швидку, коректну роботу програми.

У якості бази даних для розробки веб-сайту була обрана система управління базами даних MongoDB. Дана система має ряд переваг перед звичайними SQL базами даних, перш за все завдяки тому, що MongoDB є NoSQL базою даних, що означає, що вона не вимагає опису схеми таблиць. У NoSQL відсутні такі обмеження властиві реляційної моделі як недостатня продуктивність, забезпечені більш легкі засоби доступу до даних та їх зберігання. Основна відмінність від MySQL полягає в тому, що в MongoDB запити пишуться на BSON (бінарний JSON), а не на мові SQL, що означає, що робота з цією системою може здійснюватися через JavaScript вираження. Крім цього, MongoDB може працювати практично на всіх системах – Linux, Windows, OSX. MongoDB активно розвивається: постійно з'являються

оновлення.

MongoDB добре підходить у випадках, коли таблиці можна представити у вигляді колекції об'єктів, як у випадку розробки, де необхідно зберігати виключно список користувачів, їх проектів.

Так само не менш важливим аргументом є факт, що для Node.js існує модуль mongodb, що дозволяє досить швидко додавати базу даних в проект, а так само і модуль mongoose, що дозволяє зручно і без особливих складнощів управляти базою даних в проекті [10].

2.3. Вибір середовища розробки Visual Studio Code

Visual Studio Code – редактор вихідного коду, розроблений Microsoft для Windows, Linux та macOS. Позиціонується як «легкий» редактор коду для кросплатформеної веб розробки. Включає в себе відладчик, інструменти для роботи з Git, підсвічування синтаксису, IntelliSense та засоби для ре факторингу. Має широкі можливості для кастомізації: призначені для користувача теми, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, готові збірки розповсюджуються під ліцензією [11].

Переваги використання редактора Visual Studio Code [12]:

- самостійно підставляє деякі поширені фрагменти коду;
- пам'ятає назви змінних і підказує їх, щоб не було помилок;
- вміє завантажувати код на Git;
- допомагає налагоджувати код.
- підтримує плагіни, які збільшують швидкість роботи розробника.

VS код готовий до роботи після встановлення, за замовчуванням в ньому встановлено більшість веб-плагінів. Після завантаження можна відразу почати програмування в новому файлі [13].

2.4. Вибір GitHub

GitHub – це платформа розробки, яку розробники використовують для зберігання проектів, назва hub, пов'язана з ім'ям. Веб-хостинг GitHub, заснований на Git, являється службою контролю версій для забезпечення контролю доступу до таких функцій: відстеження помилок, запити функцій, управління завданнями та інше. GitHub використовується для збереження коду завдяки вбудованим інструментам для його перевірки та покращення.

GitHub – доступний, безкоштовний сервіс з відкритим вихідним кодом, який забезпечує віддалений доступ до репозиторію Git, забезпечує розміщення коду та допомагає керувати життєвим циклом розробки програмного забезпечення. Має такі функції: спільне використання коду декількома людьми, відстеження помилок.[14]

GitHub має ряд таких переваг GitHub:[15]

- безкоштовне обслуговування;
- дуже швидкий пошук в структурі репозиторію;
- пропонує практичні інструменти для співпраці гарну інтеграцію з Git.
- легко інтегрується з іншими сторонніми сервісами.
- також працює з TFS, HG і SVN.

2.5. Обґрунтування вибору програмної реалізації.

Технології, що будуть використані на сервері, відповідають принципам відкритості вихідних кодів, актуальності в наш час та можливістю виконання на будь-якій операційній системі. Тому було обрано Express.js, який дозволяє швидко та якісно розробити великі та малі серверні системи. Завдяки цього фреймворку можна використовувати безмежну кількість різноманітних пакетів, які підтримуються Node.js. Express.js дуже швидко встановлюється на будь-якій машині. Для управлінням пакетами було використано npm – від розробників Node.js, для зручності управління пакетами та бібліотеками, які будуть

використані у веб-сайті.

Мову програмування обрав JavaScript, що надає різноманітний “синтаксичний цукор”, дозволить скоротити об’єм шаблонного коду. Базою даних було обрано MongoDB через відкритий вихідний код. Обрану базу даних можна розгорнути на будь-якій операційній системі. Сукупність обраних технологій дає змогу побудувати надійний та вдалий програмний продукт, що захищено від патентних позовів, так як всі використані технології покриті ліцензіями, що дають доступ до вихідного коду.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-САЙТУ ІНТЕРНЕТ-ПРОВАЙДЕРА

3.1. Установка необхідних фреймворків та модулів

У цьому розділі буде розглянуто послідовність розробки веб-сайту. Для зручності розробка веб-сайту буде здійснено на локальному сервері.Перш за все, необхідно встановити Node.js. Для цього потрібно скачати дистрибутив для операційної системи з офіційного сайту. [16] Після скачування запустити установку відповідно до інструкції. Разом з Node.js буде встановлений стандартний пакетний менеджер npm, за допомогою якого можна швидко встановити Node модулі.

Потім можна приступити до встановлення модулів. Першим модулем буде express, що дозволить створити http сервер. Для цього потрібно написати команду в терміналі:

npm i express npm mongoose nodemon

Для зручності розробки потрібно встановити модуль nodemon, що дозволить не перезапускати розроблений проект при кожній зміні. Цей модуль стежить за файлами в каталозі проекту і перезапускає програму Node, якщо відбуваються якісь зміни. Для роботи з цим модулем необхідно встановити його глобально (npm дозволяє встановлювати модуль глобально, тобто цей модуль буде доступний з будь-якого місця).

Після цього таким методом будуть встановлені необхідні модулі express, mongoose, nodemon. Вищевказані модулі за замовчуванням будуть встановлюватися в папку з іменем node_modules, а саме там Node буде їх знаходити при використанні в проекті.

3.2 Створення локального сервера

Повернуся в файл pascage.json для створення скрипту, що дозволить не перезавантажувати сторінку при кожній зміні.

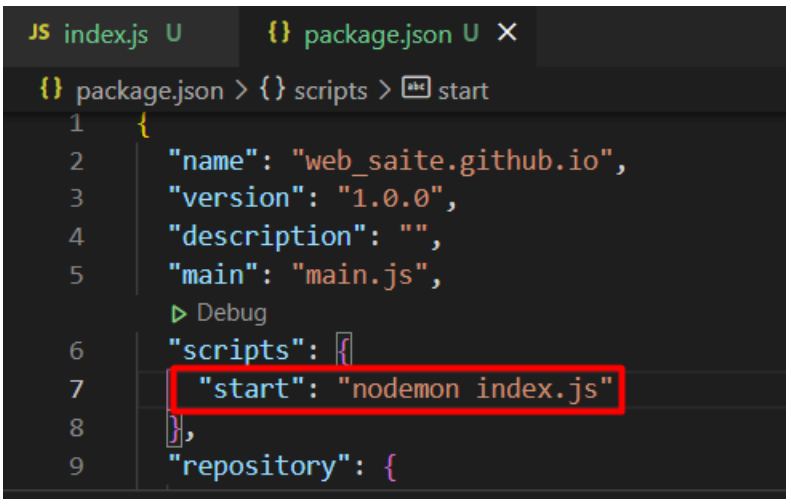

Рисунок 3 – Скрипт nodemon

Дії, описані в попередньому розділі, були підготовкою до початку розробки. Тепер потрібно створити сам сервер. Для цього потрібно створити файл index.js, який буде запущений для подальшого ввімкнення сервера. У цьому файлі потрібно підключити модулі. Для цього зроблено необхідною функцією, вказавши необхідні модулі в якості аргументу:

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

Потім було створено сервер.

```
const PORT = process.env.PORT || 5000
const app = express()
const start = () => {
try {
app.listen(PORT, () => console.log('server
started on port ${PORT}'))
} catch (e){
console.log(e)
}
}
start ()
```

У цьому випадку буде створено локальний сервер, який прослуховуватиме локальний порт 5000.

3.3 Робота з Mongodb

Як було описано раніше, MongoDB є базою NoSQL, що не потребує опису макета таблиці, а дані зберігаються як документи, що представляють JSON-подібні структури, які об'єднуються в колекції. Для початку потрібно визначити, які дані зберігати. Оскільки потрібна лише база даних для зберігання списку користувачів та їхніх тарифів, знадобиться одна колекція користувачів.

Копія колекції користувачів повинна мати такі атрибути:

- ім'я користувача,
- пароль,
- Адреса електронної пошти (e-mail),
- Статус підтвердження (прийняти)
- тариф.
- У свою чергу масив "тариф" об'єктів містить об'єкти з наступними атрибутами:
- назва тарифу,
- дата створення
- на рисунку 4 зображена структура бази даних.

Рисунок 4 – База даних

MongoDB повинен бути встановлений, щоб почати використовувати його. Було додано проект та користувача в MongoDB, вказавши IP адресу користувача який має доступ до бази. Запущено базу даних, за допомогою MongoDB, який знаходиться на диску: G:

Щоб використовувати базу даних у проекті, потрібно з'єднати модулі mongodb і mongosed. База даних запускається, коли отримуються певні поштові запити, тому ці модулі будуть потрібні в маршрутизаторах.js. У файлі authRouter.js визначено маршрути по яким будуть відправлені запити.

authRouter.js

middlewaree > authRouter.js > ...

```
1   const Router = require('express')
2   const router = new Router()
3   const controller = require('./authController')
4   const { check } = require("express-validator")
5   const authMiddleware = require('./middlewaree/authMiddleware')
6   const roleMiddleware = require('./middlewaree/roleMiddleware')
7
8   router.post('/registration', [
9       check('username', "Имя пользователя не может быть пустым").notEmpty(),
10      check('password', "Пароль должен быть больше 4 и меньше 10 символов").isLength({ min: 4, max: 10 })
11  ], controller.registration)
12  router.post('/login', controller.login)
13  router.get('/users', roleMiddleware(["ADMIN"]), controller.getUsers)
14
15  module.exports = router
16
```

Рисунок 5 – AuthRouter.js

Для запиту на реєстрацію, створено новий файл. Для перевірки чи працюють запити в рамках запиту /реєстрації функції обробки пост-обробки в файлі AuthController.js потрібно визвати функцію json, яка поверне одне з нижченаведених повідомлень.

```
const User = require('./models/User')
const Role = require('./models/Role')
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { validationResult } = require('express-validator')
const {secret} = require("./config")

const generateAccessToken = (id, roles) => {
  const payload = {
    id,
    roles
  }
  return jwt.sign(payload, secret, {expiresIn: "24h"} )
}

class authController {
  async registration(req, res) {
    try {
      const errors = validationResult(req)
      if (!errors.isEmpty()) {
        return res.status(400).json({message: "Ошибка при регистрации", errors})
      }
      const {username, password} = req.body;
      const candidate = await User.findOne({username})
      if (candidate) {
```

```
            return res.status(400).json({message: "Пользователь с таким именем уже существует"})
            }
            const hashPassword = bcrypt.hashSync(password, 7);
            const userRole = await Role.findOne({value: "USER"})
            const user = new User({username, password: hashPassword, roles: [userRole.value]})
            await user.save()
            return res.json({message: "Пользователь успешно зарегистрирован"})
          } catch (e) {
            console.log(e)
            res.status(400).json({message: 'Registration error'})
          }
        }

        async login(req, res) {
          try {
            const {username, password} = req.body
            const user = await User.findOne({username})
            if (!user) {
              return res.status(400).json({message: `Пользователь ${username} не найден`})
            }
            const validPassword = bcrypt.compareSync(password, user.password)
            if (!validPassword) {
              return res.status(400).json({message: `Введен неверный пароль`})
            }
            const token = generateAccessToken(user._id, user.roles)
            return res.json({token})
          } catch (e) {
```

```
            console.log(e)
            res.status(400).json({message: 'Login error'})
          }
        }

        async getUsers(req, res) {
          try {
            const users = await User.find()
            res.json(users)
          } catch (e) {
            console.log(e)
          }
        }
      }

      module.exports = new authController()
```

Наступним кроком буде тестування запиту в програмі Post. Тестування пройшло успішно, на запит повернулась відповідь зі статусом 200 OK.

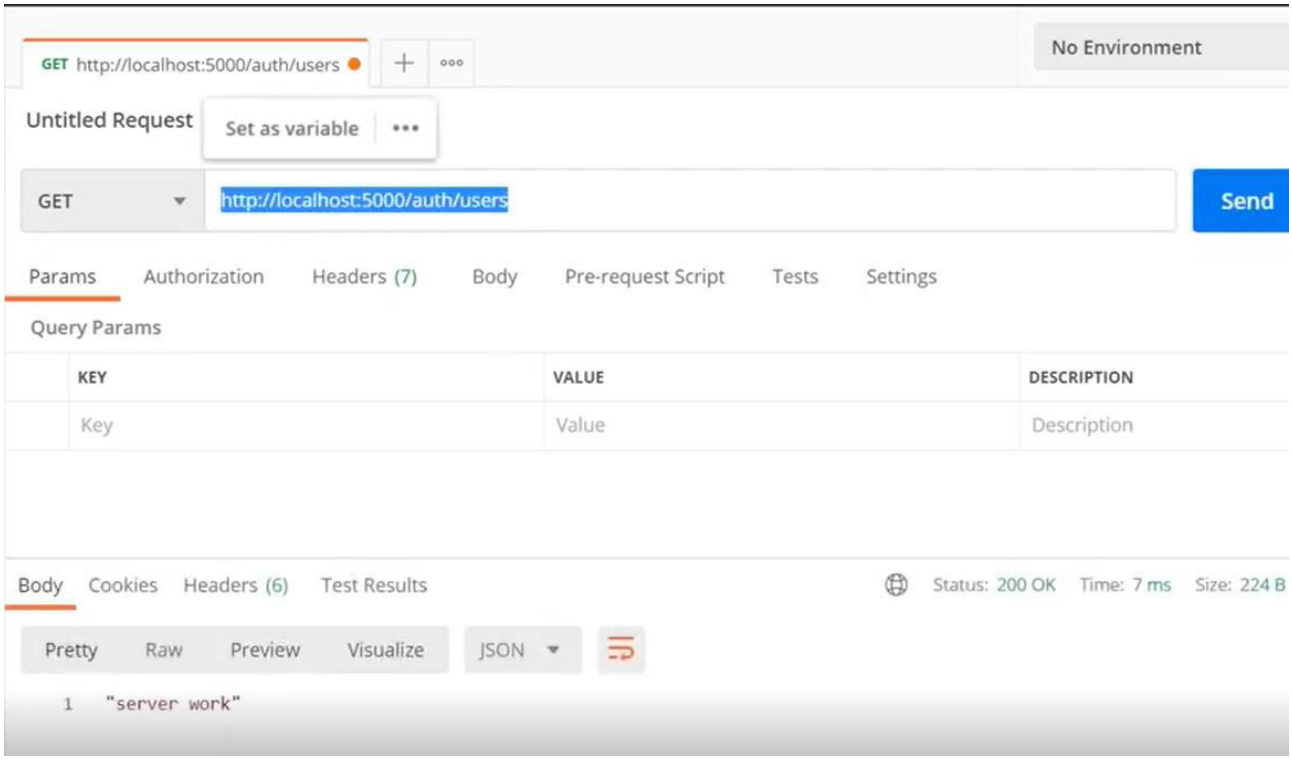

Рисунок 6 – Тестування запиту методом Get

Спочатку був пошук користувача за іменем await User.findOne, якщо такий користувач знайдений то повернеться повідомлення: *return res.status(400).json({message: "Пользователь с таким именем уже существует"*

3.4 Верстання сторінки реєстрація/авторизація

Перед тим, як почати верстку сторінок, потрібно зробити повноцінні макети. В рамках цього проекту в програмі figma створив макет сторінки реєстрація/авторизація. На рисунку 7 зображено макет сторінки.

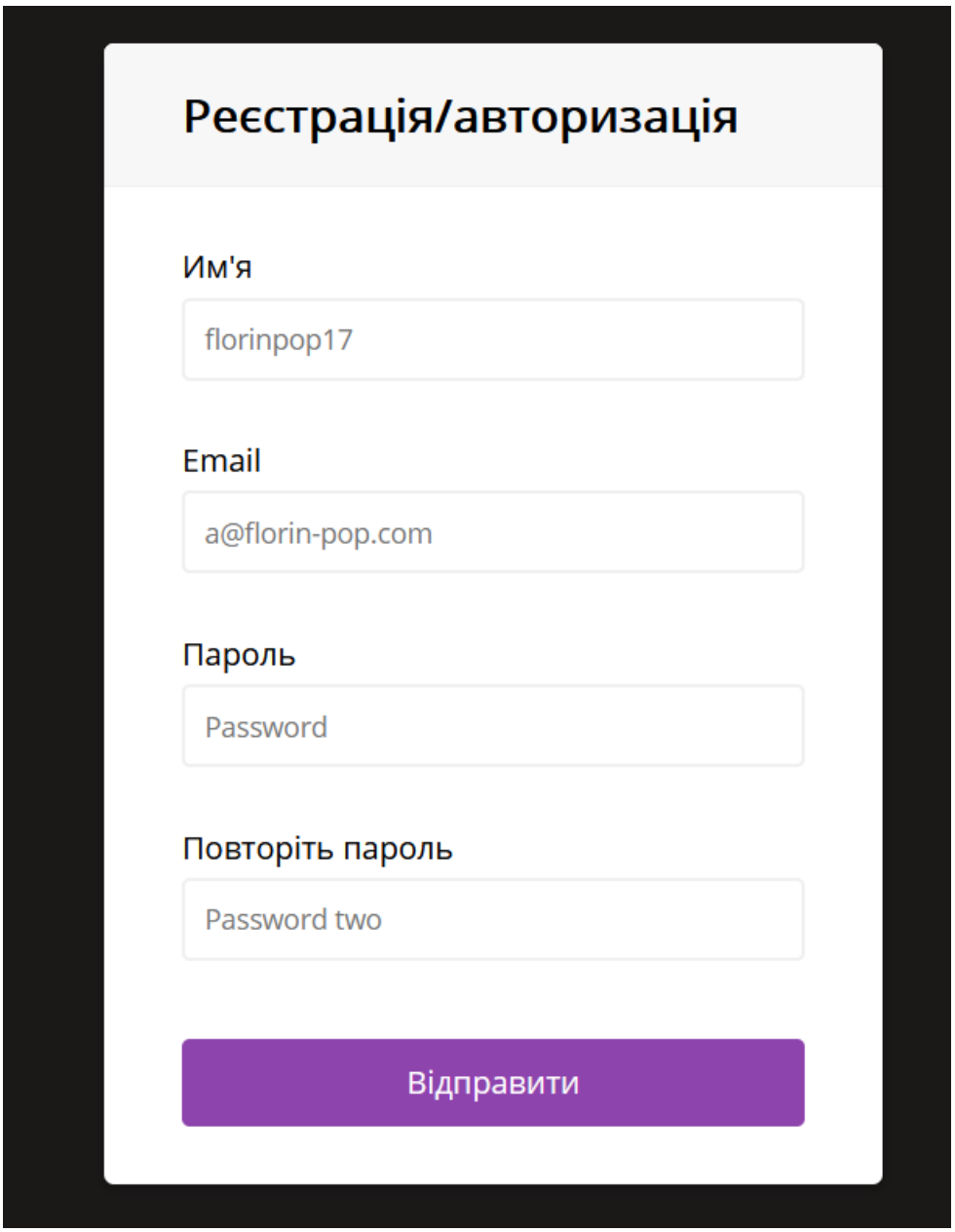

Рисунок 7 – Макет сторінки авторизації/реєстрації

Сторінка форми реєстрації створено за допомогою стандартних мов розмітки сторінок HTML і CSS, інтерактивності добавлено за допомогою мови JavaScript. Стилі також були описані стандартною мовою CSS. Якщо будуть невірно заповнені поля вищевказаної форми то користувач побачить повідомлення, що потрібно правильно записати.

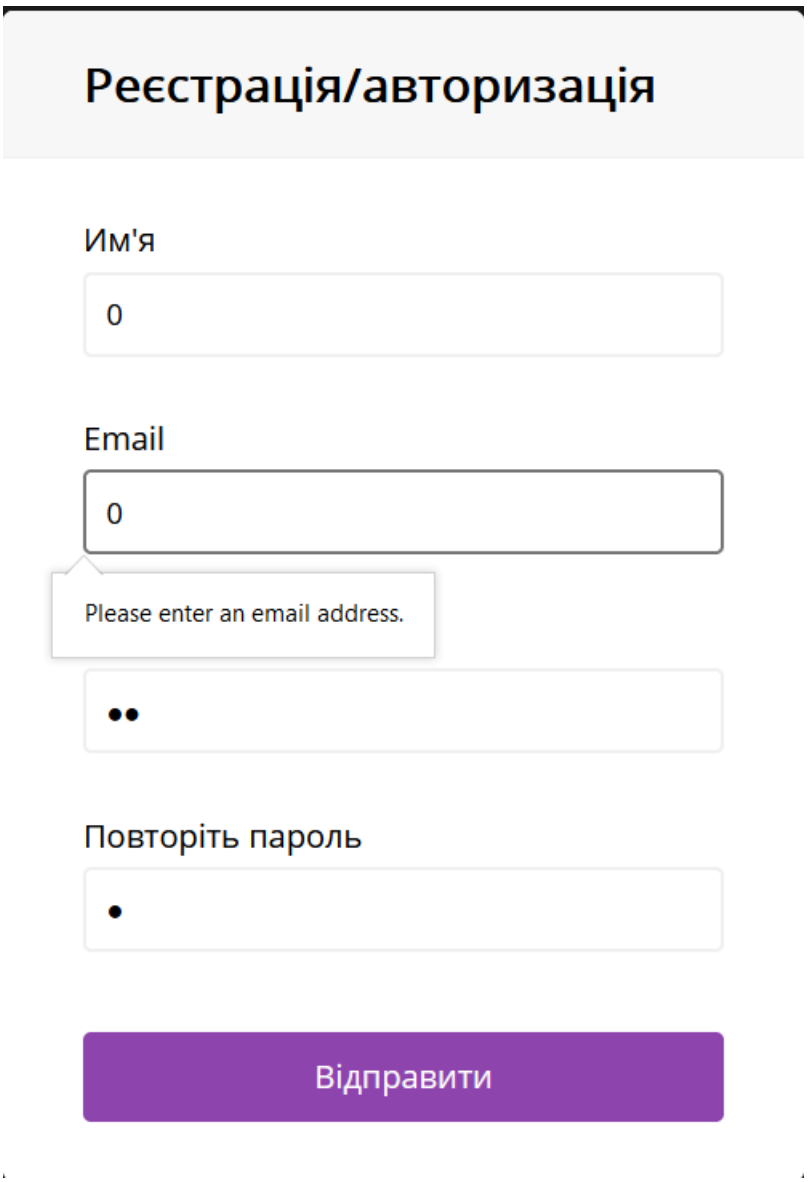

Рисунок 8 – Макет сторінки авторизації/реєстрації

У результаті верстання цієї сторінки, були отримані такі сторінки:

- registration.html,
- main.css,
- main.js.

3.5 Структура проекту

У завершений проект входить:

- серверні та налаштовуванні сценарії,

- файли розмітки та стилі сторінок сайту,
- використані модулі та бібліотеки,
- файли бази даних.

У результаті розробки був отриманий проект з наступною структурою:

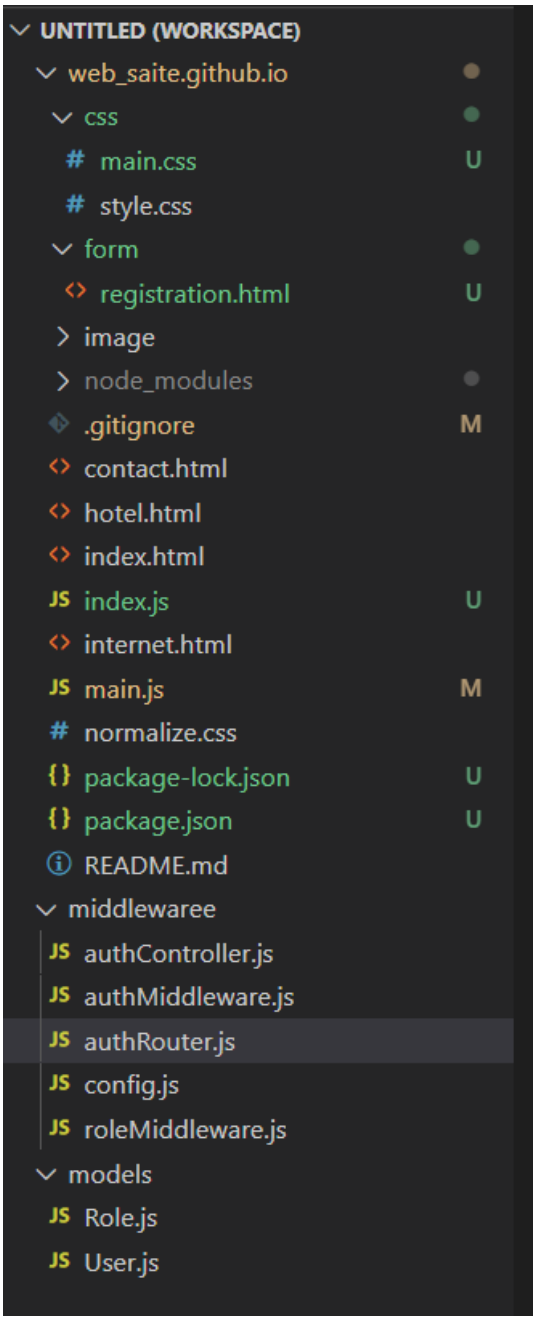

Рисунок 9 – Структура файлів.

Main.css and style.css – містить файли стилів,

registration.html – містить файл з розміткою сторінки,

img – містить зображення,

node_modules - містить модулі, що використовуються в проекті,

.gitignore – містить приховані файли,

contact.html, hotel.html, index.html – містить html код сторінки,

main.js – містить файл на мові js,

normalize.css – містить настройки для сторінки,

package-lock.json – містить всі пакети,

package.json – містить файл конфігурації,

authController.js –служить для зберігання функцій взаємодії з користувачем,

authMiddleware.js – служить для запису функцій для авторизованих користувачів,

authRouter.js – служить для визначення маршрутів,

config.js – служить для збереження секретного ключа,

Rote.js – для збереження даних про користувача, адміністратора,

User.js – для збереження даних про користувача.

3.6 Запуск та тестування сервера в програмі Postman.

У Postman введено пароль та логін від імені адміністратора, відправлено запит. Зроблено копіювання токіну, відправлено на сервер. В файлі authRouter.js додана роль адміністратор після тестування отримано повідомлення “Пользователь успешно зарегистрирован” в Postman.

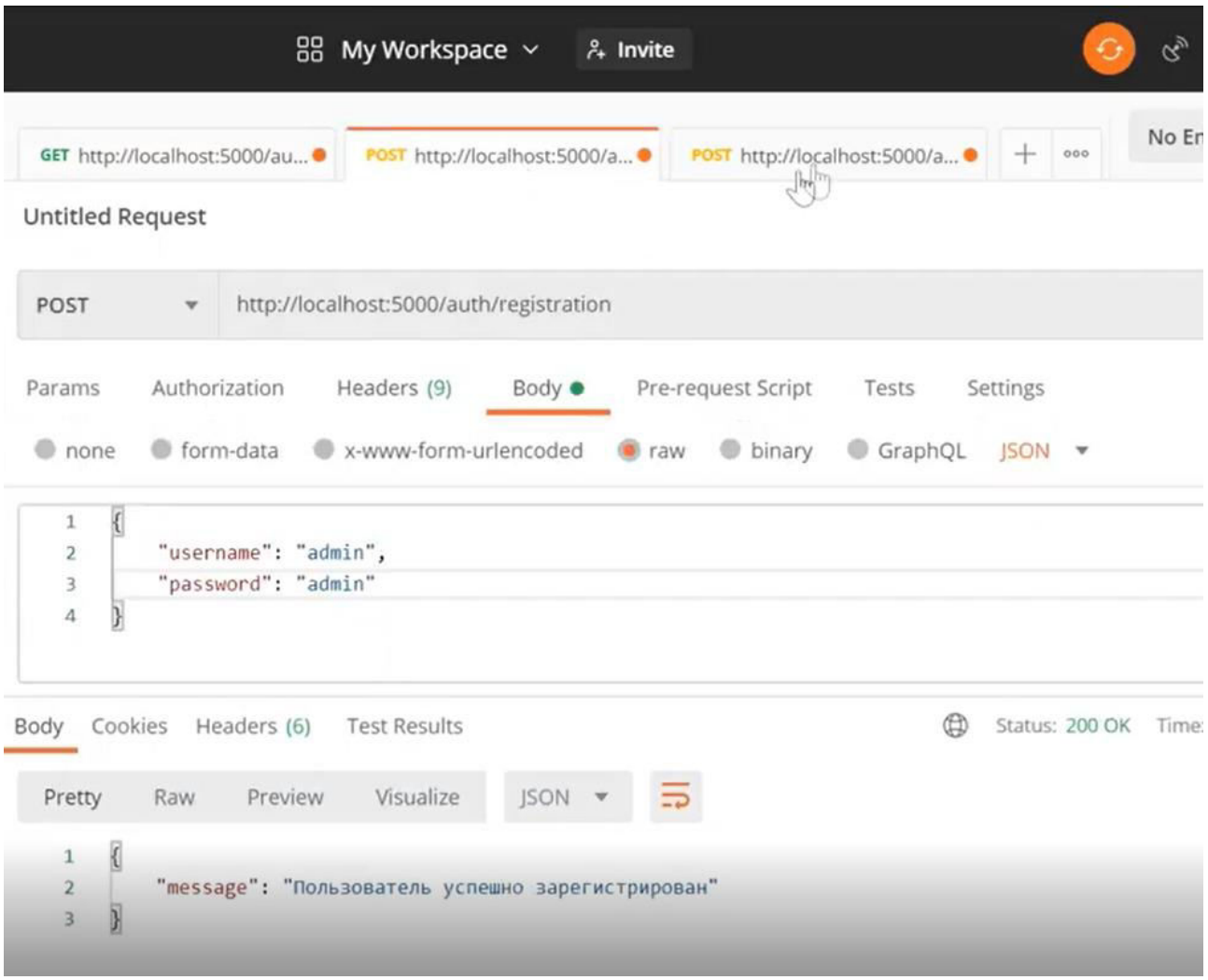

Рисунок 10 – Повідомлення про успішну реєстрацію

Перевірка тестування пройшла успішно, отримано працюючу форму реєстрації/авторизації.

ВИСНОВКИ

У результаті цієї роботи була досягнута головна мета: веб-сайт, що дозволяє користувачеві зареєструватися без довгих очікувань на відповідь адміністратора. Для досягнення цієї мети реалізовані всі необхідні етапи проектування та розробки веб-сайту:

- проведено аналіз предметний аналіз веб-сайту, визначено основні вимоги до функціональності для забезпечення його конкурентоспроможності.
- дизайн веб-сайту, який ідентифікує загальні методи системи, такі як: взаємодія системи з користувачем, алгоритми обміну даними між сервером і клієнтською частинами веб-сайту, а також графічне представлення майбутнього сайту,
- інструменти та методи розробки були обрані, щоб зробити веб-сайт найбільш ефективним,
- розроблено веб-сайт та проведено функціональне тестування на основі дизайну.

Отриманий веб-сайт відповідає всім вимогам.

Подальший розвиток функціоналу даного веб-сайту є дуже перспективним та підвищить конкурентну перевагу.

СПИСОК ЛІТЕРАТУРИ

1. Умберто Е., Як написати дипломну роботу. Пер. за ред. О. Глотова. — Тернопіль: Мандрівець, 2007. — 224 с.

2. Інформаційні технології в інфраструктурі ринку Лекція № 8 [Електронний ресурс] – режим доступу: lect8.pdf (mariroz.com)

3. Клієнт-серверна архітектура [Електронний ресурс] – режим доступу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

4. Клієнт-серверна архітектура та ролі серверів [Електронний ресурс] – режим доступу: Клієнт-серверна архітектура та ролі серверів. | by Ivan Zmerzlyi | Medium

5.Взаємодія клієнт-сервер [Електронний ресурс] – режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229

4.Саймон Холс. Стек Mean Mongo, Express, Angular, Node –С: Пітер, 2017 – 496 с.

6. Що таке npm. [Електронний ресурс] – режим доступу: https://www.hostinger.ru/rukovodstva/chto-takoe-npm

7.Node.js. Розробка серверних веб-додатків в JavaScript: Пер. с англ. Слинкіна А. А. – М.: ДМК Пресс, 2012. – 144 с.

8. Офіційний сайт модуля Express [Електронний ресурс] – режим доступу: https://expressjs.com/ru/

9. Введення в мангуст для MongoDB і Node.js – [Електронний ресурс] – режим доступу: https://code.tutsplus.com/ru/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527

10. Що таке MongoDB - [Електронний ресурс] – режим доступу: https://ylianova.ru/raznoe-2/mongo-chto-eto-kogda-vybrat-bazu-dannyx-mongodb-gajd-dlya-novichkov.html#i-34

11.Стаття про Visual Studio Code в енциклопедії «Вікіпедії» [Електронний

ресурс] – режим доступу:

https://ru.wikipedia.org/wiki/Visual_Studio_Code

12. Стаття Що таке Visual Code [Електронний ресурс] – режим доступу: https://thecode.media/visual-studio-code/

13. Visual Studio Code [Електронний ресурс] - режим доступу:

https://thecode.media/visual-studio-code/

14.Стаття Gitlab проти GitHub [Електронний ресурс] – режим доступу:

https://senior.ua/articles/gitlab-protiv-github

15.Стаття GitHub vs GitLab: переваги та недоліки цих платформ [Електронний ресурс] – режим доступу:

https://blog.desdelinux.net/ru/github-против-gitlab/#Ventajas-2

16. Офіційний сайт Node [Електронний ресурс] – режим доступу:

https://nodejs.org/en/download/

ДОДАТОК 1

Registration.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="../css/main.css">
  <link rel="stylesheet" href="../main.js">
  <title>Реєстрація</title>
</head>

<body>
  <div class="container">
    <div class="header">
      <h2>Реєстрація/авторизація</h2>
    </div>
    <form id="form" class="form">
      <div class="form-control">
        <label for="username">Им'я</label>
        <input type="text" placeholder="florinpop17" id="username" />
        <i class="fas fa-check-circle"></i>
        <i class="fas fa-exclamation-circle"></i>
        <small>Error message</small>
```

```
    </div>
    <div class="form-control">
      <label for="username">Email</label>
      <input type="email" placeholder="a@florin-pop.com" id="email" />
      <i class="fas fa-check-circle"></i>
      <i class="fas fa-exclamation-circle"></i>
      <small>Error message</small>
    </div>
    <div class="form-control">
      <label for="username">Пароль</label>
      <input type="password" placeholder="Password" id="password" />
      <i class="fas fa-check-circle"></i>
      <i class="fas fa-exclamation-circle"></i>
      <small>Error message</small>
    </div>
    <div class="form-control">
      <label for="username">Повторіть пароль</label>
      <input type="password" placeholder="Password two" id="password2" />
      <i class="fas fa-check-circle"></i>
      <i class="fas fa-exclamation-circle"></i>
      <small>Error message</small>
    </div>
    <button>Відправити</button>
  </form>
 </div>
</body>
</html>
```

Main.css

```
@import url('https://fonts.googleapis.com/css?family=Muli&display=swap');
@import
url('https://fonts.googleapis.com/css?family=Open+Sans:400,500&display=swap');

* {
box-sizing: border-box;
}

body {
background-color: #1B1818;
font-family: 'Open Sans', sans-serif;
display: flex;
align-items: center;
justify-content: center;
min-height: 100vh;
margin: 0;
}

.container {
background-color: #fff;
border-radius: 5px;
box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);
overflow: hidden;
width: 400px;
max-width: 100%;
}

.header {
```

```
border-bottom: 1px solid #f0f0f0;
background-color: #f7f7f7;
padding: 20px 40px;
}

.header h2 {
margin: 0;
}

.form {
padding: 30px 40px;
}

.form-control {
margin-bottom: 10px;
padding-bottom: 20px;
position: relative;
}

.form-control label {
display: inline-block;
margin-bottom: 5px;
}

.form-control input {
border: 2px solid #f0f0f0;
border-radius: 4px;
display: block;
```

```
font-family: inherit;
font-size: 14px;
padding: 10px;
width: 100%;
}

.form-control input:focus {
outline: 0;
border-color: #777;
}

.form-control.success input {
border-color: #2ecc71;
}

.form-control.error input {
border-color: #e74c3c;
}

.form-control i {
visibility: hidden;
position: absolute;
top: 40px;
right: 10px;
}

.form-control.success i.fa-check-circle {
color: #2ecc71;
```

```
visibility: visible;
}

.form-control.error i.fa-exclamation-circle {
color: #e74c3c;
visibility: visible;
}

.form-control small {
color: #e74c3c;
position: absolute;
bottom: 0;
left: 0;
visibility: hidden;
}

.form-control.error small {
visibility: visible;
}

.form button {
background-color: #8e44ad;
border: 2px solid #8e44ad;
border-radius: 4px;
color: #fff;
display: block;
font-family: inherit;
font-size: 16px;
```

```
padding: 10px;
margin-top: 20px;
width: 100%;
}
```

.gitignore

```
/node_modules
/.idea
```

Index.js

```
const e = require('express')
const mongoose = require('mongoose')
const PORT = process.env.PORT || 5000

const app = express()

const start = async () => {
  try {
    await
mongoose.connect(`mongodb+srv://qwerty:qwerty123@cluster0.b6pb9.mongodb.net
/auth_roles?retryWrites=true&w=majority`)
    app.listen(PORT, () => console.log(`server started on port ${PORT}`))
  } catch (e) {
    console.log(e)
  }
}
```

start()

main.js

```
function myFunction() {
  let x = document.getElementById("myTopnav");
  if (x.className === "topnav") {
    x.className += " responsive";
  } else {
    x.className = "topnav";
  }
}

onst form = document.getElementById('form');
const username = document.getElementById('username');
const email = document.getElementById('email');
const password = document.getElementById('password');
const password2 = document.getElementById('password2');

form.addEventListener('submit', e => {
  e.preventDefault();

  checkInputs();
});

function checkInputs() {
  // trim to remove the whitespaces
  const usernameValue = username.value.trim();
```

```
const emailValue = email.value.trim();
const passwordValue = password.value.trim();
const password2Value = password2.value.trim();

if (usernameValue === '') {
  setErrorFor(username, 'Username cannot be blank');
} else {
  setSuccessFor(username);
}

if (emailValue === '') {
  setErrorFor(email, 'Email cannot be blank');
} else if (!isEmail(emailValue)) {
  setErrorFor(email, 'Not a valid email');
} else {
  setSuccessFor(email);
}

if (passwordValue === '') {
  setErrorFor(password, 'Password cannot be blank');
} else {
  setSuccessFor(password);
}

if (password2Value === '') {
  setErrorFor(password2, 'Password2 cannot be blank');
} else if (passwordValue !== password2Value) {
  setErrorFor(password2, 'Passwords does not match');
```

```
  } else {
    setSuccessFor(password2);
  }
}

function setErrorFor(input, message) {
  const formControl = input.parentElement;
  const small = formControl.querySelector('small');
  formControl.className = 'form-control error';
  small.innerText = message;
}

function setSuccessFor(input) {
  const formControl = input.parentElement;
  formControl.className = 'form-control success';
}

function isEmail(email) {
  return /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/.test(email);
}
```

package.json

```
{
 "name": "web_saite.github.io",
 "version": "1.0.0",
```

```
"description": "",
"main": "main.js",
"scripts": {
 "start": "nodemon index.js"
},
"repository": {
 "type": "git",
 "url": "git+https://github.com/Tatiana-hryhorenko/web_saite.github.io.git"
},
"keywords": [],
"author": "YuriyKukharenko",
"license": "ISC",
"bugs": {
 "url": "https://github.com/Tatiana-hryhorenko/web_saite.github.io/issues"
},
"homepage": "https://github.com/Tatiana-hryhorenko/web_saite.github.io#readme",
"dependencies": {
 "express": "^4.17.1",
 "mongoose": "^5.12.12",
 "nodemon": "^2.0.7",
 "npm": "^7.15.1"
}
}
```

authController.js

```
const User = require('./models/User')
```

```
const Role = require('./models/Role')
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { validationResult } = require('express-validator')
const {secret} = require("./config")

const generateAccessToken = (id, roles) => {
  const payload = {
    id,
    roles
  }
  return jwt.sign(payload, secret, {expiresIn: "24h"} )
}

class authController {
  async registration(req, res) {
    try {
      const errors = validationResult(req)
      if (!errors.isEmpty()) {
        return res.status(400).json({message: "Ошибка при регистрации", errors})
      }
      const {username, password} = req.body;
      const candidate = await User.findOne({username})
      if (candidate) {
        return res.status(400).json({message: "Пользователь с таким именем уже существует"})
      }
```

```
            const hashPassword = bcrypt.hashSync(password, 7);
            const userRole = await Role.findOne({value: "USER"})
            const user = new User({username, password: hashPassword, roles: [userRole.value]})
            await user.save()
            return res.json({message: "Пользователь успешно зарегистрирован"})
          } catch (e) {
            console.log(e)
            res.status(400).json({message: 'Registration error'})
          }
        }

        async login(req, res) {
          try {
            const {username, password} = req.body
            const user = await User.findOne({username})
            if (!user) {
              return res.status(400).json({message: `Пользователь ${username} не найден`})
            }
            const validPassword = bcrypt.compareSync(password, user.password)
            if (!validPassword) {
              return res.status(400).json({message: `Введен неверный пароль`})
            }
            const token = generateAccessToken(user._id, user.roles)
            return res.json({token})
          } catch (e) {
            console.log(e)
```

```
      res.status(400).json({message: 'Login error'})
    }
  }

  async getUsers(req, res) {
    try {
      const users = await User.find()
      res.json(users)
    } catch (e) {
      console.log(e)
    }
  }
}

module.exports = new authController()
```

authMiddleware.js

```
const jwt = require('jsonwebtoken')
const { secret } = require('../config')

module.exports = function (req, res, next) {
  if (req.method === "OPTIONS") {
    next()
  }

  try {
    const token = req.headers.authorization.split(' ')[1]
```

```
    if (!token) {
      return res.status(403).json({ message: "Пользователь не авторизован" })
    }
    const decodedData = jwt.verify(token, secret)
    req.user = decodedData
    next()
  } catch (e) {
    console.log(e)
    return res.status(403).json({ message: "Пользователь не авторизован" })
  }
};
```

authRouter.js

```
const Router = require('express')
const router = new Router()
const controller = require('./authController')
const { check } = require("express-validator")
const authMiddleware = require('./middlewaree/authMiddleware')
const roleMiddleware = require('./middlewaree/roleMiddleware')

router.post('/registration', [
  check('username', "Имя пользователя не может быть пустым").notEmpty(),
  check('password', "Пароль должен быть больше 4 и меньше 10 символов").isLength({ min: 4, max: 10 })
], controller.registration)
router.post('/login', controller.login)
router.get('/users', roleMiddleware(["ADMIN"]), controller.getUsers)
```

```
module.exports = router
```

roleMiddleware.js

```
const jwt = require('jsonwebtoken')
const { secret } = require('../config')


module.exports = function (roles) {
  return function (req, res, next) {
    if (req.method === "OPTIONS") {
      next()
    }

    try {
      const token = req.headers.authorization.split(' ')[1]
      if (!token) {
        return res.status(403).json({ message: "Пользователь не авторизован" })
      }
      const { roles: userRoles } = jwt.verify(token, secret)
      let hasRole = false
      userRoles.forEach(role => {
        if (roles.includes(role)) {
          hasRole = true
        }
      })
      if (!hasRole) {
        return res.status(403).json({ message: "У вас нет доступа" })
```

```
      }
      next();
    } catch (e) {
      console.log(e)
      return res.status(403).json({ message: "Пользователь не авторизован" })
    }
  }
};
```

Role.js

```
const { Schema, model } = require('mongoose')


const Role = new Schema({
  value: { type: String, unique: true, default: "USER" },
})

module.exports = model('Role', Role)
```

User.js

```
const { Schema, model } = require('mongoose')


const User = new Schema({
  username: { type: String, unique: true, required: true },
  password: { type: String, required: true },
```

```
  roles: [{ type: String, ref: 'Role' }]
})

module.exports = model('User', User)
```