МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ КОНТРОЛЮ ШВИДКОСТІ ДОРОЖНЬОГО РУХУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____Марат ЖИВОТОВСЬКИЙ

«11» червня 2025 р.

Керівник д-р техн. наук, доцент _____ Ірина КАЛІНІНА

«11» червня 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Животовського Марата Олексійовича
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для контролю швидкості дорожнього руху».

Керівник роботи: Калініна Ірина Олександрівна, професор кафедри інтелектуальних інформаційних систем, д-р техн. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «11» червня 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: експертні оцінки методів контролю та аналізу швидкості дорожнього руху; наявні технології та архітектурні підходи для розробки систем моніторингу дорожнього трафіку (наприклад, комп’ютерний зір, датчики швидкості, IoT-рішення); вимоги до обробки даних у реальному часі та візуалізації інформації про швидкість руху; нормативні вимоги щодо обмеження швидкості та відповідні стандарти безпеки.

Очікуваний результат: вебзастосунок на мікросервісній архітектурі для моніторингу та контролю швидкості дорожнього руху, аналіз швидкості руху транспорту в реальному часі; адаптивний інтерфейс для різних користувачів (адміністратори, поліція, водії).

4. Перелік питань, що підлягають розробці:

– аналіз наявних застосунків для контролю швидкості дорожнього руху;

– архітектурні підходи, паттерни проектування та технології, що використовуються для створення вебзастосунку контролю швидкості дорожнього руху.

– розробка вебзастосунку.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Ірина КАЛІНІНА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Марат ЖИВОТОВСЬКИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «20» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок для контролю швидкості дорожнього руху

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КР	18.10.2024	18.10.2024	
2	Отримання завдання на виконання КР	20.12.2024	20.12.2024	
3	Складання календарного плану роботи на весь період виконання КР	30.01.2025	30.01.2025	
4	Отримання завдання на переддипломну практику	21.04.2025	22.04.2025	
5	Проходження переддипломної практики, збір та аналіз матеріалів до КР	24.04.2025	30.04.2025	
6	Розробка звіту з переддипломної практики	01.05.2025	02.05.2025	
7	Виконання КР: аналіз існуючих систем для контролю дорожнього руху, огляд існуючих технологій, розробка ПЗ	05.05.2025	11.06.2025	
8	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	
9	Доробка та остаточне оформлення КР	17.05.2025	29.05.2025	
10	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	
11	Подання КР рецензенту	02.06.2025	02.06.2025	
11	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	11.06.2025	
12	Захист КРБперед екзаменаційною комісією (ЕК)	18.06.2025	18.06.2025	

Керівник роботи _____ Ірина КАЛІНІНА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Марат ЖИВОТОВСЬКИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 402 ЧНУ ім. Петра Могили

Животовського Марата Олексійовича

на тему: **«ВЕБЗАСТОСУНОК ДЛЯ КОНТРОЛЮ ШВИДКОСТІ ДОРОЖНЬОГО РУХУ»**

Дана кваліфікаційна робота присвячена аналізу та розробці вебзастосунку для контролю швидкості дорожнього руху. Метою роботи є створення ефективної та надійної системи, яка дозволить водіями та відповідним органам убезпечити та стабілізувати стан дорожнього руху, мінімізуючи кількість дорожньо-транспортних пригод (ДТП) шляхом контролю швидкості дорожнього руху.

У роботі розглянуто основні вимоги до систем такого типу, проведено аналіз існуючих рішень на ринку та визначено їх відмінності. На основі цього аналізу було розроблено власний проект системи, який уявляє собою вебзастосунок, що використовує мікросервісну архітектуру, взаємодію з іншими сервісами та можливості інтеграції з іншими системами.

Розроблена система складається з декількох частин, серед яких:

– сервіс контролю швидкості водія;
– сервіс сповіщення про перевищення швидкості;
– сервіс аутентифікації;
– вебзастосунок для взаємодії зі створеною системою.

Особливу увагу приділено забезпеченню високої швидкості обробки даних та зручності використання системи як для адміністратора, так і для кінцевого користувача.

Запропонована система дозволяє значно покращити процес обробки отриманих даних, забезпечуючи високу швидкість, точність і безпеку.

Об’єкт роботи – процес розробки вебзастосунку для контролю швидкості дорожнього руху.

Предмет роботи – методи і технології моніторингу, обробки даних прогнозування швидкості транспортних засобів.

Метою даної роботи є розробка програмних засобів, методів та алгоритмів для створення вебзастосунку для контролю швидкості дорожнього руху.

Кваліфікаційна робота бакалавра містить 103 сторінки, 67 рисунків, 3 таблиці, 30 використаних джерел та 12 додатків.

Ключові слова: система, вебзастосунок, контроль швидкості дорожнього руху, автоматизація, мікросервіс.

ABSTRACT

of a bachelor's degree work
of a student of group 402 at Petro Mohyla Black Sea National University

Zhyvotovskyi Marat

on topic: **«WEB APPLICATION FOR CONTROLLING ROAD TRAFFIC SPEED»**

This qualification work is devoted to the analysis and development of a web application for controlling road traffic speed. The purpose of the work is to create an effective and reliable system that will allow drivers and relevant authorities to secure and stabilize the state of road traffic, minimizing the number of road accidents (Accidents) by controlling road traffic speed.

The work considers the main requirements for systems of this type, analyzes existing solutions on the market and identifies their differences. Based on this analysis, an own system project was developed, which is a web application that uses a microservice architecture, interaction with other services and the possibility of integration with other systems.

The developed system consists of several parts, including:

– driver speed control service;

– speeding notification service;

– authentication service;

– web application for interaction with the created system.

Special attention is paid to ensuring high data processing speed and ease of use of the system for both the administrator and the end user.

The proposed system allows to significantly improve the process of processing the received data, ensuring high speed, accuracy and security.

The object of the study is the process of developing a web application for controlling road traffic speed.

The subject of the work is methods and technologies for monitoring, data processing and predicting the speed of vehicles.

The purpose of this work is to develop software tools, methods and algorithms for creating a web application for controlling road traffic speed.

The bachelor's qualification work contains 103 pages, 67 figures, 3 tables, 30 sources used, and 12 appendices.

Keywords: system, web application, road traffic speed control, automation, microservice.

ЗМІСТ

ВСТУП

Обрана тема є актуальною в сучасному світі. Безпека дорожнього руху є пріоритетним завданням для транспортних засобів і суспільства загалом. Одним із головним факторів, що впливають на кількість дорожньо-транспортних пригод (ДТП), є перевищення швидкості. Вебзастосунок для контролю швидкості дорожнього руху розробляється з метою поліпшення безпеки на дорогах шляхом автоматизації моніторингу дотримання швидкісних режимів на різних ділянках доріг.

У сучасному середовищі існує проблема, коли швидкість на дорозі контролюється частковою мірою. Цю проблему можна пояснити забезпеченістю інфраструктури в залежності від розмірів міста та фінансування правоохоронних органів. У зв'язку з цим, системи контролю швидкості руху відрізняються у містах та селищах (за їх наявності). Ефективна система відстеження швидкості зможе вирішити цю проблему повною мірою.

Основними завданнями цієї кваліфікаційної роботи бакалавра є:

– аналіз існуючих рішень у сфері контролю швидкості дорожнього руху;
– створення вимог для вебзастосунку;
– вибір технологій, методів та архітектурних рішень для вебзастосунку;
– розробка програмної реалізації системи.

Об’єкт роботи – процес розробки вебзастосунку для контролю швидкості дорожнього руху.

Предмет роботи – методи і технології моніторингу, обробки даних та прогнозування швидкості транспортних засобів.

Метою даної роботи є розробка програмних засобів, методів та алгоритмів для створення вебзастосунку для контролю швидкості дорожнього руху.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ У СФЕРІ КОНТРОЛЮ ШВИДКОСТІ ДОРОЖНЬОГО РУХУ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери програмного забезпечення систем контролю швидкості дорожнього руху

Система контролю швидкості дорожнього руху призначена для автоматизації процесу відстеження, контролю руху водіїв на дорогах загального користування. Вона має забезпечити ефективне функціонування попередження дорожньо-транспортних пригод внаслідок перевищення ліміту швидкості та інших ймовірних причин.

Елементами описаної системи є найпростіші частини цілої системи, які можна об’єднати в підсистеми. Підсистеми можна описати як умовно неподільні частини, які не можуть повноцінно функціонувати окремо за межами системи та визначаються від поставлених задач. Вони можуть виконувати такі дії, як збереження, передача отриманої інформації, її відображення тощо. Розбиття найпростіших елементів в підсистеми забезпечить структуровану організацію, їх надійність, структурну цілісність та швидкість взаємодії.

1.2 Система аналізу швидкості та поточного стану дорожнього руху

Системою аналізу швидкості та поточного стану дорожнього руху є система, яка обробляє вхідні дані та надсилає вихідні дані, базуючись на аналізі. ЇЇ можна розділити на наступні підсистеми (етапи):

- реєстрація та аутентифікація користувача;
- збір даних про стан дорожнього руху;
- їх попередня обробка для подальшого аналізу;
- безпосередній аналіз;
- їх використання (візуалізація);
- навчання моделей штучного інтелекту для ефективнішої обробки даних (опціонально).

Користувач реєструється в застосунку та аутентифікується для запису їх даних. Реєстрація відбувається через вже існуючі системи (наприклад BankID або Дія) [1, 2]. Одною з вимог для збереження отриманих даних є гарантування їх конфіденційності та захищеності під час несанкціонованого доступу, тому під час проєктування подібної системи необхідно впевнитись у захисті даних, наприклад використанням хешування чутливих даних (логін, пароль, біометричні дані тощо).

В подальшому при використанні застосунку записуються вхідні дані про швидкість водія, а також навантаженість трафіку з використанням GPS-даних, різноманітних датчиків на дорогах, метеостанцій та сервісів для навігації (Google Maps, Waze тощо) [3, 4]. В подальшому ці дані оброблюються (фільтруються від шуму та помилок, синхронізуються та агрегуються) та будуть використовуватись для навчання моделей штучного інтелекту для оптимізації роботи системи.

Після повної обробки даних виконується їх повний аналіз, після чого відбувається візуалізація оброблених даних для користувача (швидкість водія, стан дорожнього руху).

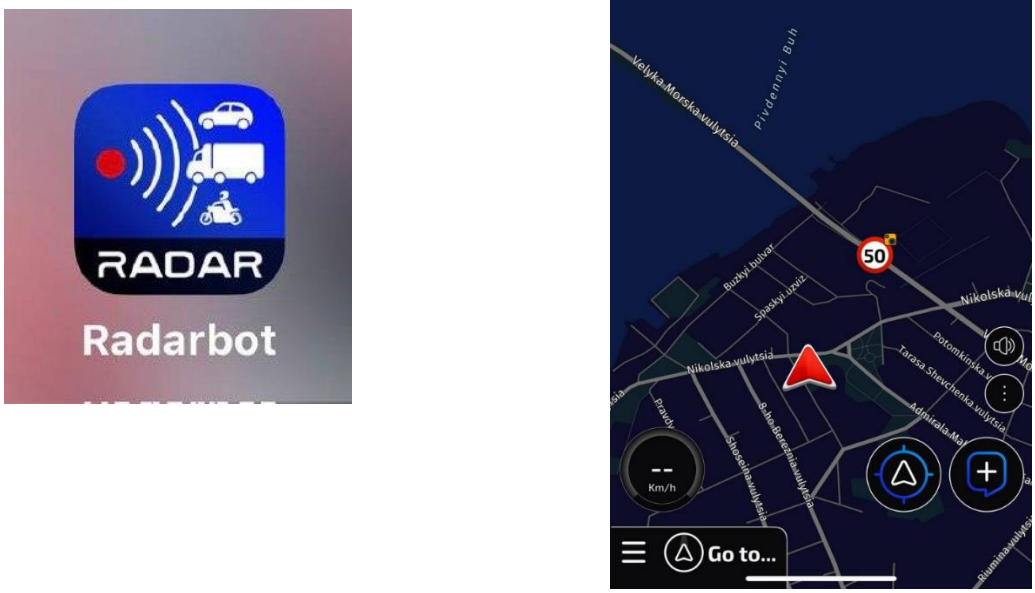
1.3 Огляд та аналіз наявних рішень систем контролю швидкості дорожнього руху

1.3.1 Приклади аналогів систем в світі

Станом на 2025 рік, в світі представлено безліч інтелектуальних рішень для контролю швидкості дорожнього руху. Нижче зазначено декілька актуальних застосунків.

1. «Radarbot: Speed Cameras | GPS» – інтелектуальна система контролю швидкості руху від компанії Iteration Mobile S.L., яка дозволяє користувачеві бачити актуальну інформацію щодо стану дорожнього руху, локації знаків регулювання руху, його поточної швидкості, геолокації зон з обмеженою швидкістю, локацією радарів. Цей застосунок пропонує інтуїтивно зрозумілий інтерфейс з доступними функціями та сповіщеннями. Він є безкоштовним, однак

для використання повного передбаченого функціоналу застосунку необхідно придбати платне доповнення. Також, застосунок підтримує візуалізацію на пристроях Apple Watch, що є одночасно і перевагою, і недоліком, оскільки застосунок розроблено лише під системи IOS та екосистему Apple, що є перевагою для користувачів IOS та недоліком для користувачів інших операційних систем (Android), тому що вони не зможуть користуватись застосунком взагалі [5].



а) б)

Рисунок 1.1 – Логотип застосунку Radarbot (а) та інтерфейс застосунку (б)

2. «Speedometer 55 GPS Speed & HUD» – система, яка дозволяє відстежувати швидкість водія та встановлювати ліміти швидкості. В порівнянні з попереднім застосунком, цей має сильно спрощений функціонал, який обмежується виміром швидкості, відстеженням поточної геолокації користувача, його координат (довготи та широти)та відстеженням даних про поїздки (часу, відстані та витраченого палива). Також застосунок має корисну функцію логів для чорного ящика, що дозволить у випадку автомобільної катастрофи

відстежити шлях подій автомобіля – учасника. Він має такий самий недолік, як і перший застосунок – обмеження для використання лише на ОС IOS [6].

а)

б)

Рисунок 1.2 – Логотип застосунку Speedometer (а) та інтерфейс застосунку (б)

3. «Speed Control App» – інтелектуальна система розробника MiVSell Technology Company SA, яка відстежує швидкість водія та показує діючі обмеження на дорозі, базуючись на геолокації користувача. Застосунок має інтуїтивно зрозумілий та простий інтерфейс без зайвих функцій, який дозволяє ефективно контролювати власну швидкість при русі в потоці або на трасі. Його

перевагою є те, що він створений на ОС Android, що убезпечить більшу кількість користувачів в зв’язку з поширеністю смартфонів на даній ОС [7].

а) б)

Рисунок 1.3 – Логотип застосунку Speed Control App (а) та інтерфейс застосунку (б)

1.3.2 Порівняння систем контролю швидкості руху

Найбільш доцільним способом порівняння вищеописаних застосунків є структуризація аналізу інтелектуальних рішень у вигляді таблиці 1.1.

Для об’єктивного порівняння треба використати наступні критерії:

– характеристики системи;
– платформа, на якій розроблена система;
– функціонал;
– переваги;
– недоліки;
– можливість масштабування в довгостроковій перспективі.

Таблиця 1.1 – Порівняльна характеристика систем контролю дорожнього руху

Характеристика /назва застосунку	**Radarbot: Speed Cameras | GPS**	**Speedometer 55 GPS Speed & HUD**	**Speed Control App**
Характеристики системи	Інтелектуальна система контролю швидкості руху, яка дозволяє користувачеві бачити актуальну інформацію щодо стану дорожнього руху, локації знаків регулювання руху, його поточної швидкості, геолокації зон з обмеженою швидкістю, локацією радарів.	Система, яка дозволяє відстежувати швидкість водія та встановлювати ліміти швидкості.	Інтелектуальна система, яка відстежує швидкість водія та показує діючі обмеження на дорозі, базуючись на геолокації користувача.
Платформа	IOS	IOS	Android
Функціонал	Попередження водіїв про стаціонарні та мобільні камери контролю швидкості, радари й інші дорожні загрози в режимі реального часу; забезпечення навігації з GPS та голосові підказки, допомагаючи дотримуватись швидкісного режиму й уникати штрафів.	Відображає точну швидкість руху за допомогою GPS та підтримує режим проекції на лобове скло (HUD) для зручного перегляду під час водіння; дозволяє встановлювати сповіщення про перевищення швидкості та вести історію поїздок.	Контроль швидкості транспортного засобу з можливістю встановлення обмежень та отримання сповіщень про їх перевищення; надає статистику поїздок, допомагаючи аналізувати стиль водіння та покращувати безпеку на дорозі.

Кінець таблиці 1.1

Переваги	Широкий функціонал застосунку	Простота реалізації, зручний HUD	Простота реалізації
Недоліки	Обмеженість однією вузьконаправленою ОС (IOS), необхідність заплатити гроші за повний функціонал застосунку	Обмеженість однією вузьконаправленою ОС (IOS), необхідність заплатити гроші за повний функціонал застосунку, недостатній функціонал	Необхідність заплатити гроші за повний функціонал застосунку, недостатній функціонал
Можливість масштабування	Низький рівень	Високий рівень	Високий рівень

1.4 Використання архітектури для побудови системи контролю швидкості руху

Для реалізації програмного забезпечення необхідно окреслити архітектуру, яка буде застосовуватись при «побудові» застосунку. Для проєктування подібної системи доцільно буде використати або монолітну архітектуру, або мікросервісну архітектуру [8].

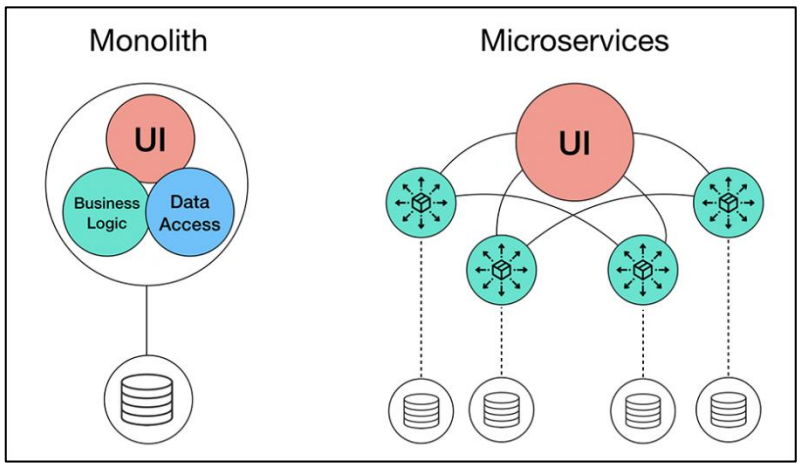


Рисунок 1.4 – Зіставлення відмінностей між монолітною та мікросервісною архітектурами

1.4.1 Мікросервісна архітектура

Мікросервісна архітектура –підхід розробки програмного забезпечення, який полягає у створенні системи з сукупності невеликих сервісів з окремо визначеним певним функціоналом. Одну функціональну одиницю прийнято вважати мікросервісом. Завдяки великій кількості таких одиниць будується одна цілісна система, яка вирізняється децентралізованістю. Більш того, така система може бути написана з можливістю вибору специфічних мов програмування та баз даних залежно від функціонального призначення сервісу. Зв’язок таких сервісів між собою відбувається стандартними протоколами (HTTP, MQTT, gRPC).

Мікросервісна архітектура також вирізняється своєю гнучкістю та надійністю. Виходячи з положення, що система, основана на мікросервісній архітектурі, складається з різних мікросервісів, можна зробити висновок, що при помилці в одному з мікросервісів не буде ламатись вся система, а лише один певний елемент, що спрощує можливість хотфіксів. Також можна виокремити той факт, що мікросервіс можна легко оновлювати та масштабувати у разі необхідності, що спрощує роботу розробникам та пришвидшує процес.

Основними недоліками мікросервісної архітектури можна назвати складність її коректної реалізації та супроводу, оскільки необхідний досвід роботи з мікросервісами та окремий розробник, який буде створювати їх (DevOps). Окрім цього, вона має вищі вимоги до інфраструктури у вигляді сервісів, контейнерзації та моніторингу.

Кажучи про її застосування, необхідно зазначити, що використання подібної архітектури є доцільним в великих системах, які розраховані на широкий функціонал та будуть супроводжуватись відповідними фахівцями з достатніми hard skills.

1.4.2 Монолітна архітектура

Монолітна архітектура – архітектура програмного забезпечення, в якому

застосунок побудований як одне ціле. Прикладом використання монолітної архітектури можна навести певний застосунок, який умовно можна поділити на три частини: UI (User Interface) на стороні клієнта (складається з HTML-компоненту та JS (JavaScript), які запущені у браузері на пристрої користувача; базу даних; програмний застосунок на стороні сервера. Програма на стороні сервера обробляє запити HTTP, надіслані користувачем, виконує логіку домену та буде заповнювати HTML-«інтерфейс» для надсилання користувачеві в браузер.

Зазвичай монолітну архітектуру використовують в системах, які не потребують великого функціоналу, оскільки вся обробка запитів може бути виконана в одному процесі. Це дозволяє використовувати повний функціонал мов програмування при розробці для розділення програм на структури, функції, класи, простори імен, інтерфейси тощо [9, 10].

Основними перевагами такої архітектури є її простота реалізації та тестування. Для подібної системи не потрібен вузьконаправлений досвід розробки та навчені фахівці, достатньо базового розуміння написання структури коду. Також, розробка такої системи провокує менше витрат на інфраструктуру, ніж зазвичай.

Однак, монолітна архітектура має зеркальні недоліки в порівнянні з мікросервісною. Вона менш гнучна при масштабуванні, оскільки написана як одне ціле та не має можливості зміни конкретних функцій та компонентів без впливу на всю систему. Також, така система не підтримує інтеграції з іншими системами.

Монолітна архітектура застосовується у випадку, якщо система невелика або середня за розміром; функціонал чітко визначений і змінюється нечасто; є потреба в простоті розгортання та підтримки.

1.4.3 Порівняння монолітної та мікросервісної архітектур

Порівняння монолітної та мікросервісної архітектур зазначено в таблиці

1.2.

Таблиця 1.2 – Порівняльна характеристика наведених архітектур написання застосунку

Характеристика / вид архітектури	**Монолітна архітектура**	**Мікросервісна архітектура**
Простота розробки на старті	+	-
Масштабованість	-	+
Гнучкість у виборі технологій	-	+
Простота розгортання	+	-
Незалежність компонентів	-	+
Стійкість до помилок	-	+
Швидкість змін і оновлень	-	+
Легкість тестування	+	-
Складність інфраструктури	+	-
Можливість окремої вигрузки (деплою) частин системи	-	+
Дороговизна підтримки у великій системі	-	+
Продуктивність локальної розробки	+	-
Наявність мережевих накладних витрат	+	-

Підбиваючи підсумки порівняння, можна сказати, що монолітна

архітектура більше підходить для розробки малих застосунків, які не потребують створення великої кількості функцій, в той час як мікросервісна архітектура розрахована на масштабні проєкти з великою кількістю функцій.

1.5 Постановка задачі

Для розробки вебзастосунку контролю швидкості дорожнього руху необхідно використати визначені технології та інструменти.

1.5.1 Планування технологій для реалізації застосунку

З існуючих мов програмування для реалізації подібної системи (програми) буде доцільно використати такі, як JavaScript та Python. Ці мови можна використовувати разом для написання більш ефективної та оптимальної системи, оскільки кожна окрема мова програмування має свої плюси та мінуси. Щоб вони могли взаємодіяти коректно, потрібно забезпечити обмін повідомленнями між програмами, які написані різними мовами програмування.

Крім того, необхідно визначити фреймворки та бібліотеки для більш високотехнологічної реалізації ПО. В контексті розробки з використанням мови Python існують такі фреймворки, як Flask або Django, в той час як для JavaScript існують React, Vue та Angular.

Проєктування архітектури грає важливу роль у створенні ефективного застосунку. З основних архітектур, які використовуються для подібних задач, можна назвати монолітну та мікросервісну архітектури. В даному випадку краще буде застосувати мікросервісну архітектуру, оскільки цей вебзастосунок розрахований на можливість масштабування у майбутньому. Окрім того, мікросервісну архітектуру буде простіше реалізовувати в межах окреслених мов програмування та фреймворків.

Кажучи про обмін даними між застосунками, слід зазначити про використання API (методи взаємодії між різними компонентами системи) для передачі даних.

Вибір програмного забезпечення для керування базами даних (простіше кажучи, СКБД) є подібною задачею до вибору архітектури, оскільки визначною рисою реляційних баз даних є швидкість доступу і запису даних взамін на складність масштабування в довгостроковій перспективі. До них відносять такі БД, як SQLite, MySQL, PostgreSQL, MariaDB. Для вирішення проблеми масштабування існують нереляційні СКБД, такі як MongoDB, Redis, Amazon DynamoDB та інші [11].

Окрім того, до вибору СКБД необхідно розробити структуру таблиць для збереження даних, яка буде оптимізованою і дозволить уникнути надлишковість накопичуваних даних.

Вибір інтегрованого середовища розробки (IDE) є особистим вибором кожного користувача, в залежності від поставлених задач розробки. Найкращим вибором для розробки подібної системи є Visual Studio Code з можливістю ставити розширення для мов програмування та бібліотек, а також PyCharm спеціально для розробки на Python. Для розробки вебзастосунків мовою JavaScript краще підійде або Visual Studio Code, або специфіковані IDE (наприклад Webstorm для HTML/CSS/JS).

1.5.2 Вимоги до програмної реалізації

Спираючись на вищезазначені технології, можна окреслити загальну структуру застосунку.

Система буде створена з використанням мікросервісної архітектури. Цей вибір обумовлений перспективами масштабування застосунку в майбутньому, а також обраними технологіями розробки застосунку.

Для розробки програми буде використано декілька мов програмування, а саме JavaScript з Python, а разом з ними і фреймворки React та Aiohttp. Вони забезпечать оптимальну розробку застосунку в зв’язку зі специфікою кожної окремої мови програмування та їх використання для певних задач.

В системі буде створено одну базу даних. Ця база даних буде містить в собі

дані про користувачів, які пройшли реєстрацію, а також про їх сесії в ньому. В якості платформи для бази даних обрано нереляційну БД MongoDB для перспективи масштабування у окреслених межах.

Висновки до розділу 1

В цьому розділі розглянуто предметну сферу, а саме проаналізовано системи контролю швидкості дорожнього руху, виконано порівняння існуючих інтелектуальних рішень (систем), визначено їх сильні та слабкі сторони та окреслено загальне розуміння структури майбутнього застосунку. В ході дослідження було досягнуто висновку, що для поточного проєкту краще буде використати монолітну архітектуру, яка найкраще підходить під його задачі.

Існуючі рішення використовуються по всьому світу, тому буде доцільним створити окрему систему, розраховану на регіон або країну, яка буде націлена на звичайного водія будь-якої категорії. У подальшому така система може бути розширена та просунута на глобальний ринок.

Основними вимогами до програми було окреслено використання мікросервісної архітектури, використання найсучасніших мов програмування та фреймворків та сучасної бази даних для створення ультимативного продукту, який буде просто супроводжувати у довгостроковій перспективі і розробляти оновлення з новим функціоналом.

На основі проведеного аналізу існуючих рішень в сфері контролю швидкості дорожнього руху можна дійти висновку, що для створення оптимального та ефективного вебзастосунку по контролю швидкості дорожнього руху необхідно скомпонувати весь найкращий існуючий функціонал.

2 АРХІТЕКТУРНІ ПІДХОДИ, ТЕХНОЛОГІЇ, ПАТЕРНИ, ЩО ВИКОРИСТОВУЮТЬСЯ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ПО КОНТРОЛЮ ШВИДКОСТІ ДОРОЖНЬОГО РУХУ

2.1 JavaScript (JS)

JavaScript – це мультипарадигмова мова програмування, яка надає можливість застосування різних підходів до програмування, зокрема ООП (об’єктно-орієнтований), функціональний та імперативний стилі. Ця мова програмування широко використовується у ролі мови сценаріїв в браузерах для створення інтерактивності веб-сторінки [12].

Архітектурні особливості наведено на рис. 2.1.

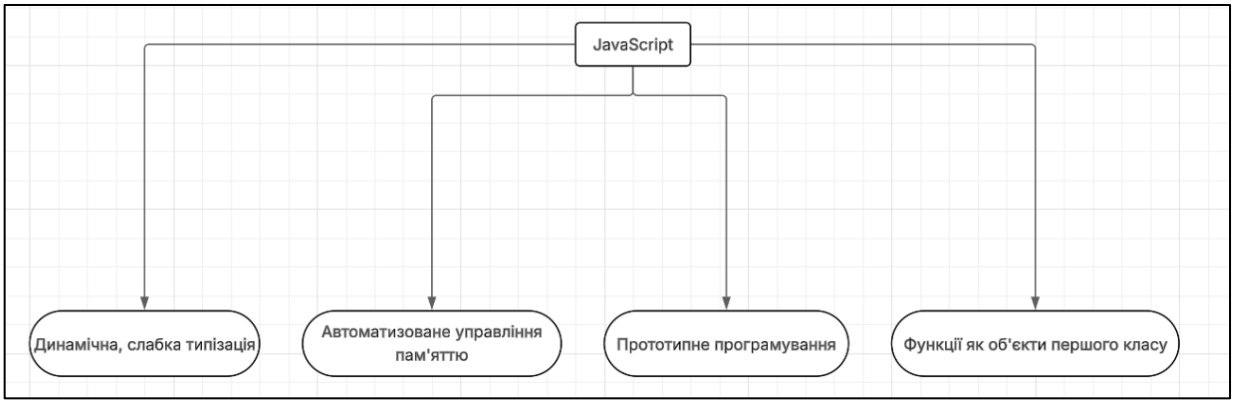


Рисунок 2.1 – Основні особливості JavaScript

Основною ідеєю при розробці JavaScript було створити мову, схожу на Java, яка буде використовуватись як мова скриптів для веб-розробки. Її визначальною рисою є те, що ця мова не належить жодній існуючій компанії, що робить її унікальною в порівнянні з іншими існуючими мовами програмування.

Для створення застосунку з використанням мови JavaScript необхідно завантажити будь який IDE, який підтримує цю мову (наприклад Visual Studio Code, Webstorm). Такий застосунок має стандартну структуру – frontend, backend та БД. JavaScript використовується як фундамент складової frontend разом з HTML/CSS. Backend-складову можна розділити на інструменти – Node.js

(умовно JavaScript для серверу) та фреймворки за необхідності. В якості БД можна обрати будь-яку, зазвичай для з'єднання з Node.js використовують MongoDB.

Основними недоліками мови можна назвати відсутність стандартних бібліотек, інтерфейсів до веб-серверів і БД та системи управління пакетами, яка відстежувала б залежності та автоматично встановлювала їх.

Станом на 2025 рік, останньою версією є ECMAScript 2024. Версії попередніх років (ECMAScript, 2023, 2022) також підтримуються.

2.1.1 Node.js

Node.js – кросплатформе середовище виконання Javascript, яке можна застосувати майже для будь-якого проєкту в залежності від його типу. Він використовує рушій V8 JavaScript, який є основою Google Chrome поза браузером, що сприяє його високій ефективності при роботі та розробці [13].

Середовище Node.js працює як один процес без створення нового потоку під кожний запит. Цей процес складається з набору асинхронних примітивів вводу-виводу в стандартній бібліотеці, що спрямовано на уникнення блокування виконання JavaScript-коду.

Під час того, як Node.js виконує операції типу вводу-виводу, він не блокує існуючий потік, запобігаючи зайвому навантаженню процесора шляхом уникнення циклів, а відновлює його роботу після отримання відповіді. Саме ця риса дозволяє Node.js опрацьовувати велику кількість одномоментних підключень до одного сервера, зводячи нанівець складність керування паралельними потоками.

Як вже було зазначено вище, Node.js – це середовище виконання JavaScript, що спрощує задачу для розробників, оскільки їм не потрібно вивчати нову мову задля написання серверної частини коду.

2.1.2 React.js

React.js – кросплатформний фреймворк для JavaScript з відкритим вихідним кодом для розробки користувацьких інтерфейсів (UI). Також він може використовуватись для створення односторінкових і мобільних застосунків, як інші фреймворки такого типу (Vue, Angular) [14].

Головна мета React – забезпечити користувачеві приємний UI, а розробнику – максимально спростити процес розробки шляхом її високої швидкості, простоти та можливості масштабування.

Для створення проєкту в розробника має бути встановлений Node.js. Після цього в терміналі IDE прописуються наступні команди (рис. 2.1).

```
npm create vite@latest my-react-app --template react
cd my-react-app
npm install
npm run dev
```

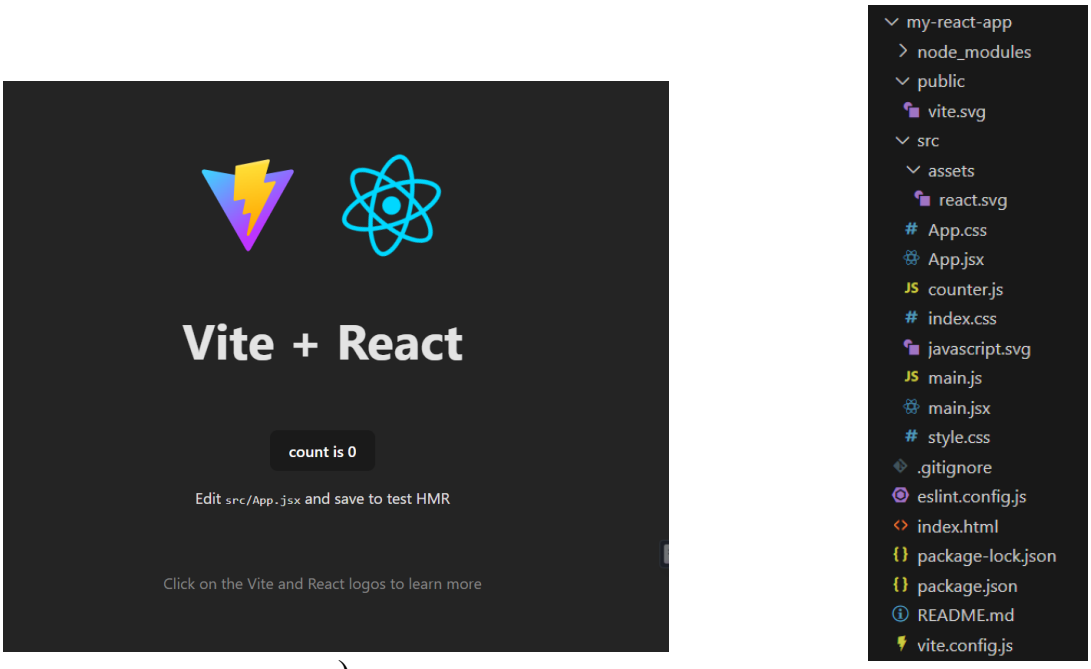
Рисунок 2.1 – Фрагмент коду в терміналі для створення проєкту на React

Процес створення проєкту зазначено на рис. 2.2.

```
C:\Users\User\Desktop\my-react-app>npm create vite@latest my-react-app --template react

◇  Select a framework:
│  React

◇  Select a variant:
│  JavaScript

◇  Scaffolding project in C:\Users\User\Desktop\my-react-app\my-react-app...

└  Done. Now run:

  cd my-react-app
  npm install
  npm run dev


C:\Users\User\Desktop\my-react-app>npm install
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'vite@6.3.3',
npm WARN EBADENGINE   required: { node: '^18.0.0 || ^20.0.0 || >=22.0.0' },
npm WARN EBADENGINE   current: { node: 'v21.7.3', npm: '10.5.2' }
npm WARN EBADENGINE }

added 139 packages, and audited 153 packages in 10s

32 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\User\Desktop\my-react-app>npm run dev
```

Рисунок 2.2 – Процес створення проєкту

Результат створення та структура проєкту зазначена на рис. 2.3.



а) б)

Рисунок 2.3 – Результат створення проєкту (а) та структура проєкту (б)

Основними особливостями React можна зазначити такі:

а) односпрямований потік даних. Властивості (props) передаються від батьківських компонентів до дочірніх. Дочірні компоненти отримують ці властивості як незмінні значення, тобто вони не можуть безпосередньо змінювати їх або ініціювати зміни через зворотні виклики (callback-функції);

б) віртуальний DOM. React створює копію (кеш) DOM в пам'яті. Це дозволяє йому ефективно порівнювати поточний стан інтерфейсу з попереднім і оновлювати тільки ті частини реального DOM браузера, які дійсно змінилися. Розробник може працювати так, ніби сторінка оновлюється повністю, але React сам визначає, які компоненти потребують перемальовування, без зайвого оновлення інших;

в) JSX. Це розширення JavaScript, яке дозволяє писати UI-структуру за допомогою синтаксису, схожого на HTML. JSX використовується для створення компонентів, але їх також можна писати і звичайним JavaScript;

г) методи життєвого циклу. Це спеціальні функції, які дозволяють розробнику запускати код на різних етапах "життя" компонента. До них належать:

1) shouldComponentUpdate: дозволяє уникнути непотрібного перемальовування компонента, якщо він не змінився (повертаючи false);

2) componentDidMount: викликається після першого відображення компонента і часто використовується для отримання даних з віддалених джерел через API;

3) componentWillUnmount: викликається перед видаленням компонента з DOM, наприклад, для відписки від подій;

4) render: обов'язковий метод у кожному компоненті, який викликається при зміні даних для оновлення інтерфейсу;

д) React Hooks. Дозволяють використовувати стан та інші можливості React у функціональних компонентах без необхідності писати класи. Створення власних хуків дозволяє виносити логіку компонента в багаторазові функції.

Схематично вони зображені на рис. 2.4.

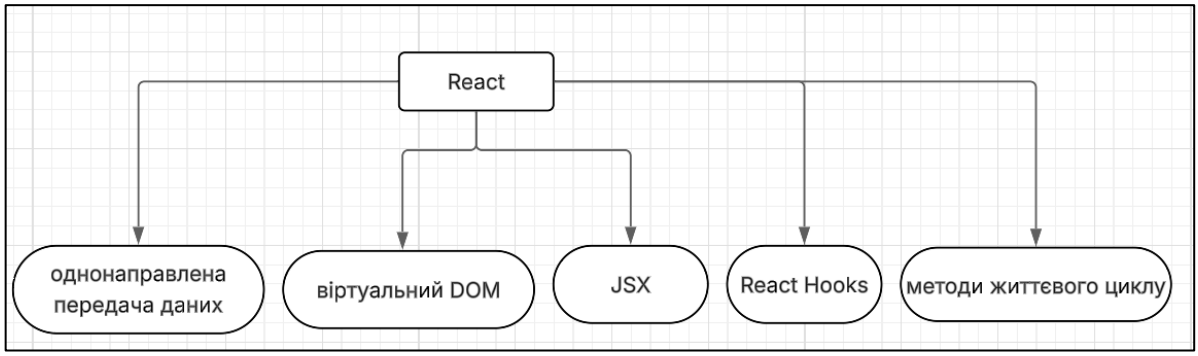


Рисунок 2.4 – Особливості React

2.2 Python

Python — це високорівнева мова програмування. Вона є інтерпретованою та об'єктно-орієнтованою, а також має сувору динамічну типізацію. [15]. Python

надає засоби роботи з даними високого рівня, поєднуючи їх із динамічною семантикою та зв’язуванням під час виконання, що сприяє швидкій розробці. Крім того, мова підтримує модулі та пакети, що полегшує модульність і повторне використання коду. Головною перевагою можна зазначити велику кількість стандартних бібліотек, які спрощують розробку майже для всіх задач. Python чудово підходить для завдань, що вимагають роботи з комплексними числами та цілими числами будь-якого розміру, оскільки має для цього необхідні інструменти та забезпечує вищу швидкість порівняно з іншими мовами програмування.

Щодо типів даних, Python використовує динамічну типізацію, а це означає, що тип змінної визначається безпосередньо під час виконання програми. Серед основних типів варто відзначити підтримку цілих чисел необмеженої величини та комплексних чисел. Крім того, Python має вбудовану бібліотеку для зручної роботи з рядками, зокрема тими, що закодовані в Unicode.

Серед колекцій Python підтримує такі структури, як списки (аналоги масивів), словники, множини та кортежі.

Узагальнена інформація про типи і структури даних зазначена на рис. 2.5.

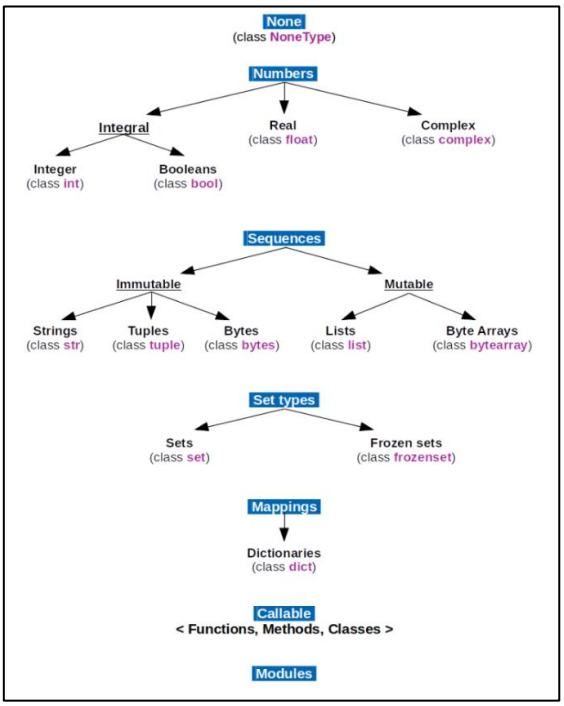


Рисунок 2.5 – Інформація про типи і структури даних в Python

Останньою версією Python на момент 2025 року вважається версія Python 3.12.10. Слід зазначити, що Python не підтримує сумісність версій 3.x з версіями 2.x, що примушує розробників адаптуватись під новітні версії.

2.3 Aiohttp

Aiohttp — це асинхронна бібліотека для Python, яка дає змогу створювати HTTP-клієнти та сервери. Вона дозволяє розробляти веб-додатки з використанням асинхронного програмування [16].

Простіше кажучи, це технологія, що значно прискорює обмін даними між застосунком та його користувачами завдяки асинхронним процесам. Такий підхід збільшує швидкість роботи веб-сервісів у кілька разів. Низька продуктивність таких сервісів часто пов'язана не зі складними розрахунками, а з очікуванням операцій вводу-виводу.

Основними причинами вибору використання саме цієї технології є такі:

– універсальність. Це фактично єдине повноцінне рішення для одночасної реалізації HTTP-сервера та клієнта в Python;

– підтримка вебсокетів. Aiohttp надає вбудовану підтримку серверних та клієнтських вебсокетів, уникаючи так званого "callback hell" (складних вкладених функцій зворотного виклику);

– проміжне ПЗ. Вона включає проміжне програмне забезпечення (middleware) для веб-сервера з підтримкою сигналів та маршрутизації, що легко інтегрується;

– висока паралельність. Це просте та зручне рішення для моделювання високого рівня паралелізму в Python-проєктах, особливо там, де немає багатопоточності.

2.4 MongoDB

MongoDB — це документоорієнтована нереляційна система керування базами даних (СКБД) з відкритим вихідним кодом, яка не вимагає попереднього

опису схеми таблиць [17]. Вона займає проміжне положення між швидкими та масштабованими системами, що працюють з даними у форматі "ключ/значення", та функціональними реляційними СКБД зі зручним формуванням запитів. Простими словами, MongoDB — це своєрідний гібрид нереляційних і реляційних СКБД, що поєднує в собі найкращі риси обох типів.

Ключова особливість MongoDB полягає у зберіганні даних у JSON-подібному форматі (BSON). Ця модель документів спрощує кодування, управління та прискорює роботу за рахунок внутрішнього групування пов'язаних даних.

З основних можливостей MongoDB можна назвати наступні:

– документоорієнтоване сховище. Зберігає дані у гнучких документах, що відповідають об'єктам у коді;

– гнучка мова запитів. Дозволяє легко формувати запити для отримання та маніпулювання даними;

– динамічні запити. Можливість створювати запити "на льоту", адаптуючись до потреб застосунку;

– повна підтримка індексів. Забезпечує швидкий доступ до даних за допомогою індексування;

– профілювання запитів. Допомагає оптимізувати продуктивність, аналізуючи виконання запитів;

– швидке оновлення даних "на місці". Дозволяє ефективно змінювати дані без повного перезапису документів;

– ефективне зберігання бінарних даних. Оптимізована для зберігання великих обсягів бінарних файлів, таких як фото та відео;

– аудит операцій. Надає можливість відстежувати зміни даних у базі;

– відмовостійкість та масштабованість. Підтримує асинхронну реплікацію, набори реплік та шардінг для забезпечення високої доступності та розподілу навантаження;

– підтримка MapReduce. Дозволяє виконувати складні агрегації та аналіз даних за допомогою парадигми MapReduce.можливість роботи відповідно до парадигми MapReduce.

Узагальнені властивості наведено на рис. 2.7.

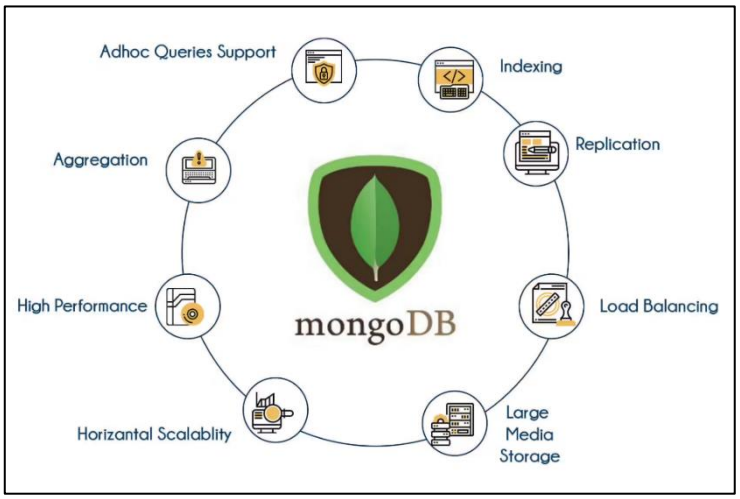


Рисунок 2.7 – Можливості MongoDB

Висновки до розділу 2

В цьому розділі розглянуто архітектурні підходи, паттерни проєктування та технології, які будуть використані в розробці вебзастосунку по контролю швидкості дорожнього руху, а саме технології для створення мікросервісної архітектури на JavaScript та механізми збереження даних. Більш детально розглянуто JavaScript як основну складову розробки frontend-складової, Node.js для забезпечення backend-складової, React.js як фреймворк для створення лаконічного користувацького інтерфейсу (UI), мову Python з фреймворком aiohttp для взаємодії з системою.

Технології, які обрано для реалізації, схематично зображено на рис. 2.8.

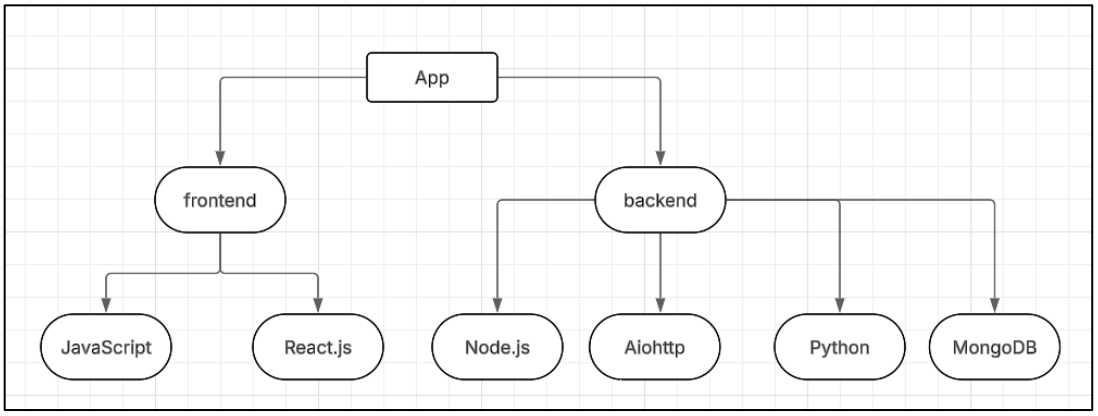


Рисунок 2.8 – Технології для розробки вебзастосунку

3 РЕАЛІЗАЦІЯ СИСТЕМИ КОНТРОЛЮ ДОРОЖНЬОГО РУХУ ТА СУПРОВІДНОГО ВЕБЗАСТОСУНКУ

3.1 Макет системи та дизайн вебзастосунку

Відповідно до поставлених задач та вищеописаних матеріалів, необхідно описати макет системи для найбільш точної реалізації.

Говорячи про макет системи, нижче описано алгоритм її роботи.

Користувач заходить на сайт застосунку, де йому необхідно пройти аутентифікацію.

У випадку, якщо користувач не зареєстрований, він може легко пройти реєстрацію. Після реєстрації користувачеві необхідно буде підтвердити свій акаунт шляхом переходу за посиланням з електронної пошти. Під час реєстрації дані користувача будуть вноситись в базу даних і кожному користувачеві буде призначатись унікальний токен. При аутентифікації користувача його дані вносяться, шифруються у токен та порівнюються з тим токеном, який був внесений у базу даних при реєстрації. У випадку, якщо токен не співпадає, тоді користувач отримує помилку про неправильні дані для входу. У випадку співпадіння токену користувач успішно проходить аутентифікацію.

Передбачено два варіанти користування застосунком – з аутентифікацією та без аутентифікації. Без аутентифікації користувач втрачає можливість синхронізації з ботом у Telegram для повідомлень про перевищення заданого ліміту швидкості.

Після успішного проходження реєстрації та аутентифікації користувач задає ліміт швидкості для себе на пристрої. У випадку перевищення швидкості і якщо користувач не зареєстрований, йому надсилається звуковий сигнал про перевищення. Якщо користувач зареєстрований, йому надсилаються повідомлення через бот у Telegram про перевищення швидкості та рекомендації щодо утримання оптимальної швидкості в межах допустимого без порушень правил дорожнього руху.

У застосунку користувач може обрати необхідну йому систему виміру швидкості (км/год, м/с, мілі/год). Вимір швидкості буде здійснюватись шляхом відстеження зміни позиції пристрою у просторі за допомогою геолокації.

Для спрощеного розуміння вищезазначений алгоритм наведено у вигляді блок-схеми, створеної за допомогою Lucidchart [18] на рис. 3.1-3.2.

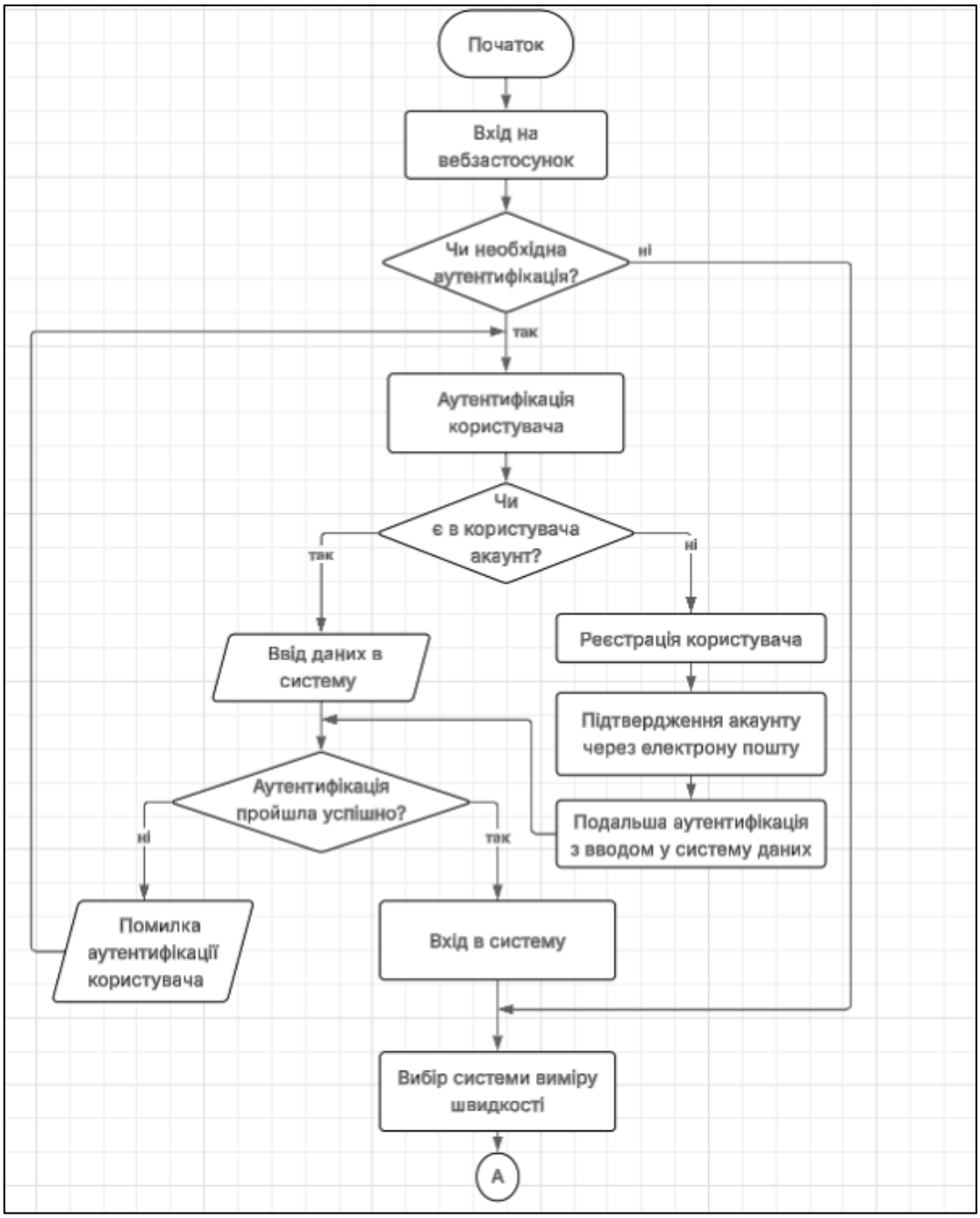


Рисунок 3.1 – Початок блок-схеми з алгоритмом реєстрації

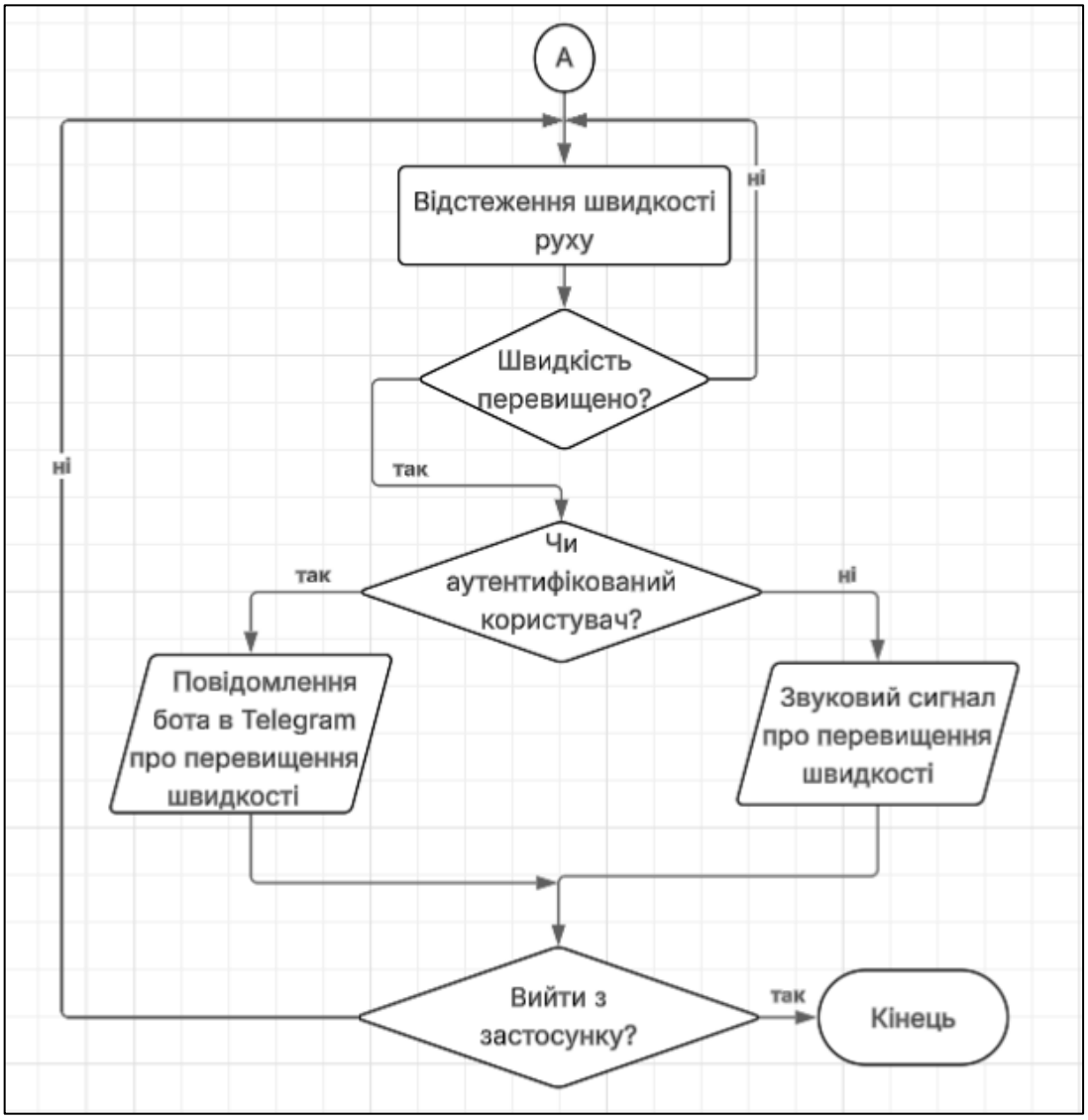


Рисунок 3.2 – Кінець блок-схеми (сегмент з алгоритмом роботи застосунку)

Для подальшого розуміння написання UI-компоненту було створено мокап (дизайн) застосунку з використання платформи Canva [19]. Головна сторінка, сторінка аутентифікації, реєстрації та підтверджувального письма зображені на рис. 3.3-3.6.

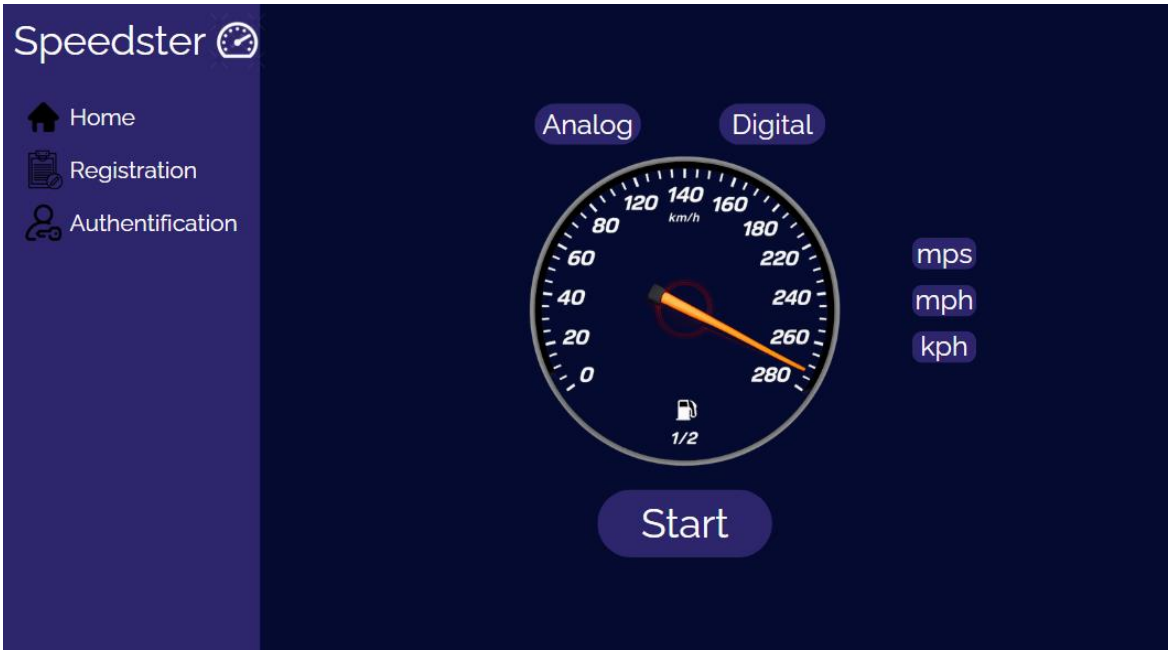


Рисунок 3.3 – Дизайн головної сторінки вебзастосунку на ПК

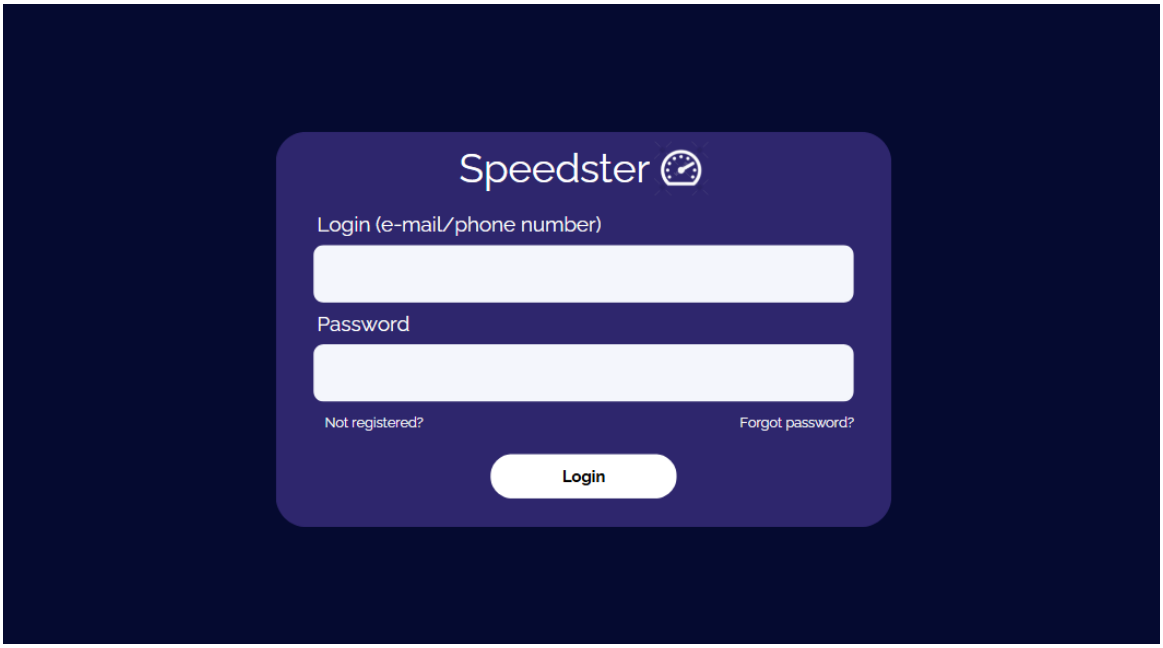


Рисунок 3.4 – Сторінка аутентифікації вебзастосунку на ПК

Рисунок 3.5 – Сторінка реєстрації у вебзастосунку на ПК

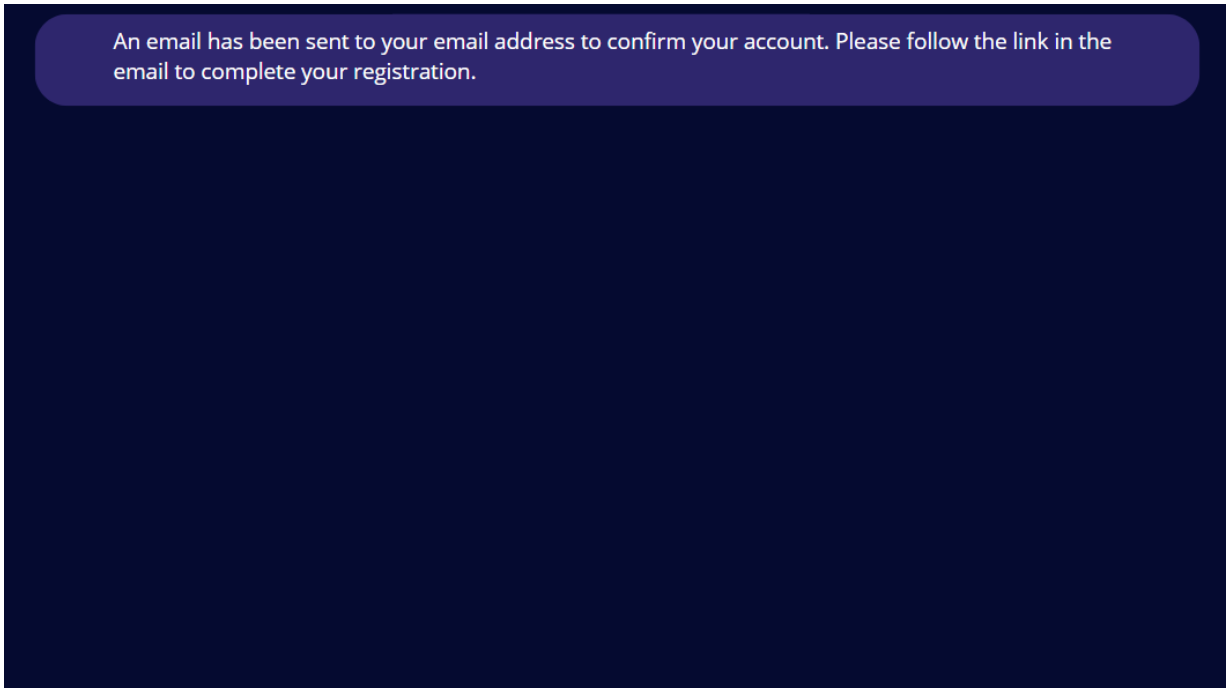


Рисунок 3.6 – Сторінка підтвердження акаунту вебзастосунку на ПК

Слід зазначити, що для повної коректної реалізації дизайну необхідно було б розробити окремий мокап для мобільних пристроїв. Ця проблема вирішується створенням адаптивності для інших пристроїв, окрім ПК під час безпосередньої розробки UI в середовищі розробки проєкту.

3.2 Розробка UI-компонентів застосунку

Для розробки UI-компонентів застосунку використовується фреймворк React. Принцип роботи застосунку, створеного на подібному фреймворку, є односторінковим, тобто завантаження необхідного коду виконується на одну сторінку. Наслідком цього є пришвидшений час повторного завантаження одних й тих самих елементів, що оптимізує роботу застосунку загалом [20].

Для реалізації розробленого раніше мокапу створено проєкт за допомогою терміналу середовища Visual Studio Code. Для подальшої розробки треба виконати рефакторинг коду та структури проєкту.

Основою такого застосунку є два файли: App.js та index.js. App.js – це функція, в якій вказуються всі необхідні елементи, які потім необхідно буде завантажити в процесі переміщення по застосунку. В файлі index.js безпосередньо вказується структура застосунку. Як висновок, цей файл відповідає за рендер попередньо вказаних компонентів у App.js. Лістинг коду App.js та index.js зазначено на рис. 3.7-3.8.

```
import { Routes, Route } from 'react-router-dom';
import MainPage from './components/MainPage/MainPage';
import LoginPage from './components/LoginPage/LoginPage';
import RegisterPage from './components/RegisterPage/RegisterPage';

function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<MainPage />} />
        <Route path="/register" element={<RegisterPage />} />
        <Route path="/login" element={<LoginPage />} />
      </Routes>
    </div>
  );
}

export default App;
```

Рисунок 3.7 – Фрагмент коду App.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom'
import App from './App';
import { HeadProvider } from 'react-head';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <HeadProvider>
    <App />
    </HeadProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Рисунок 3.8 – Фрагмент коду index.js

Побудова односторінкових застосунків заснована на принципі створення сторінок як окремих компонентів. Наприклад, була поставлена задача створити дві сторінки – головну сторінку та сторінку авторизації користувача. Для реалізації цієї задачі створюється два окремих компоненти – компонент головної сторінки та сторінка авторизації. Кожний компонент можна розбити на більш дрібні для того, щоб у випадку дублювання блоків елементів на різних сторінках можна було використовувати один і той самий компонент, що в свою чергу збільшує швидкість розробки подібних застосунків.

На базі мокапу створено декілька компонентів-сторінок. Приклад їх вигляду зазначено на рис. 3.9-3.11.

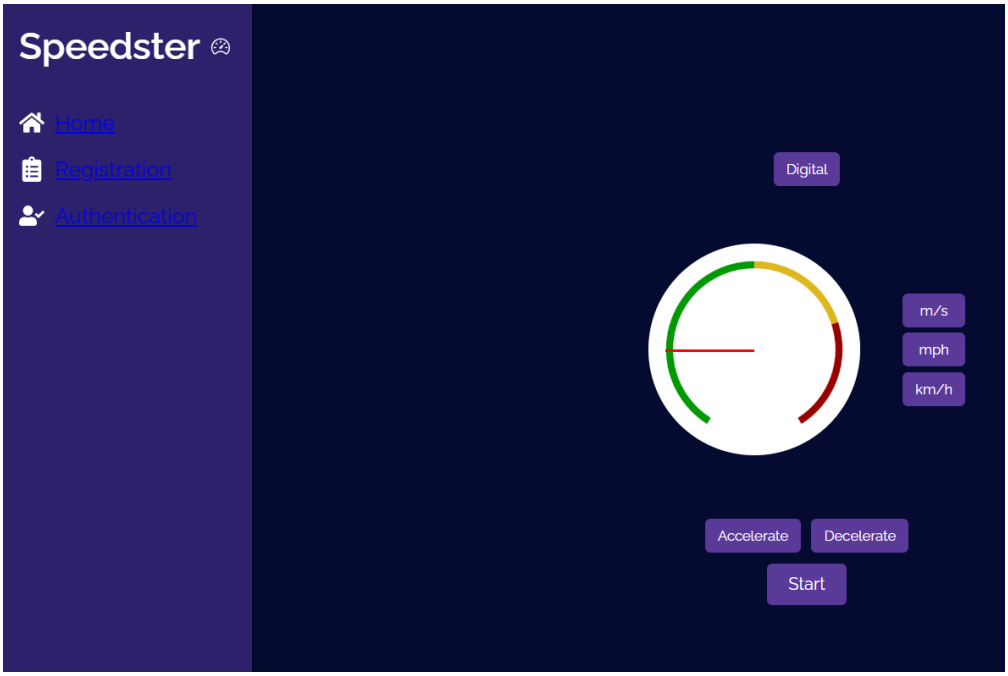


Рисунок 3.9 – Розроблений дизайн головної сторінки застосунку

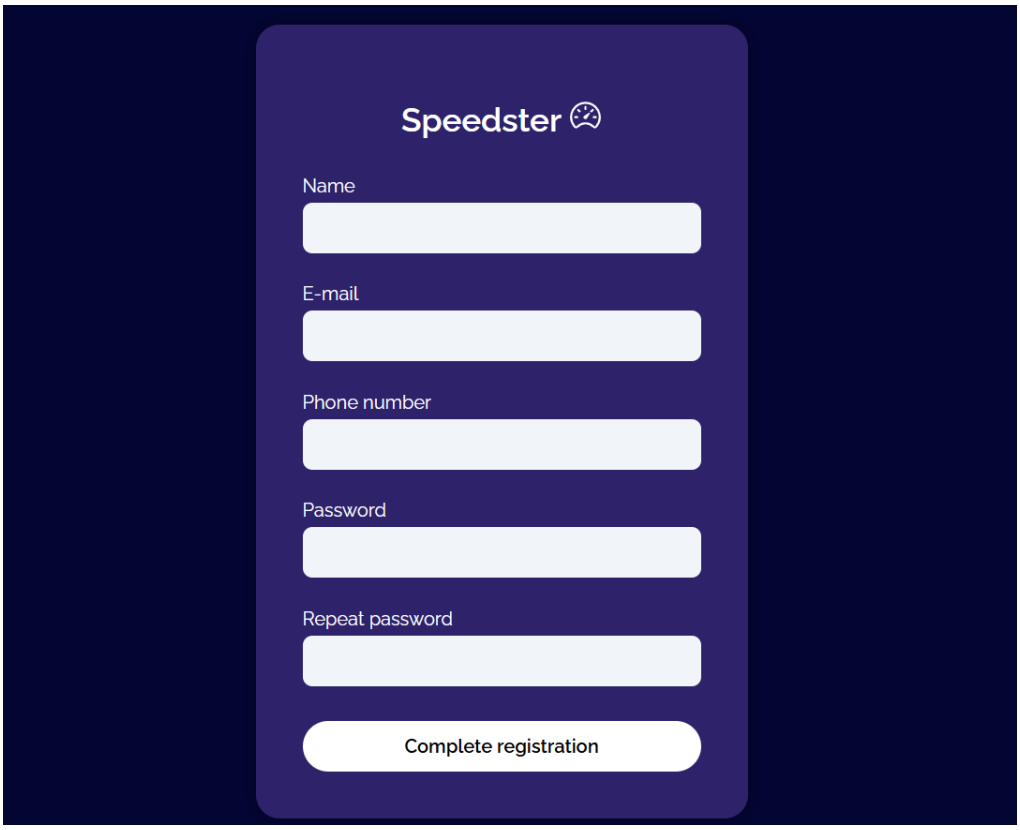


Рисунок 3.10 – Розроблений дизайн сторінки реєстрації

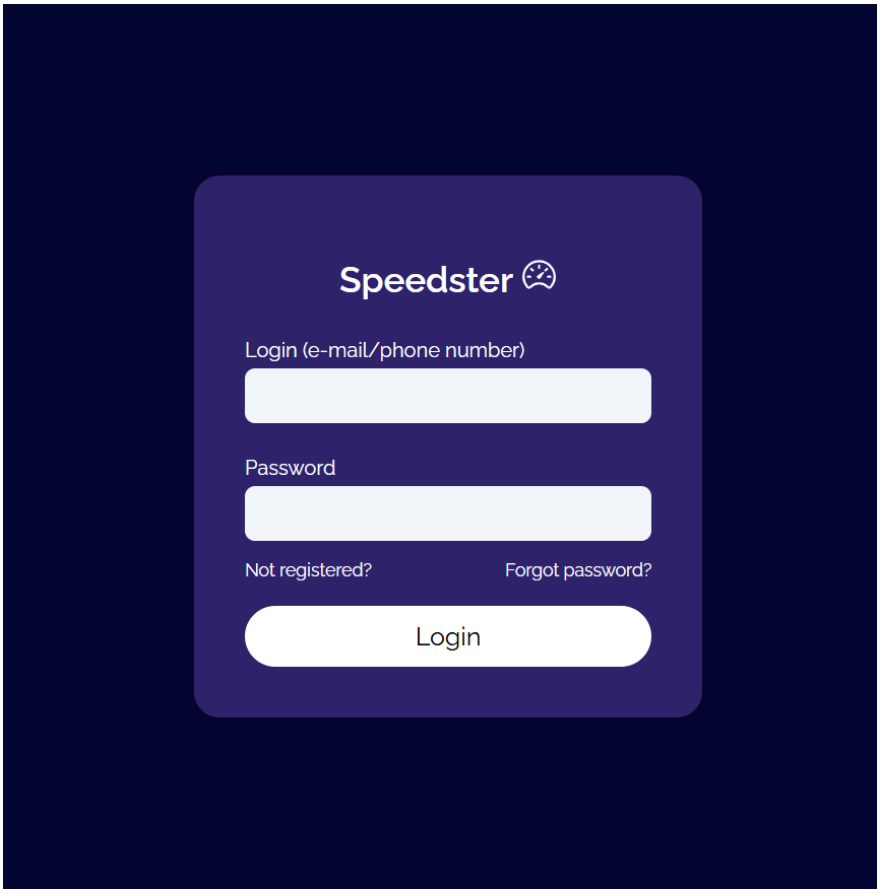


Рисунок 3.11 – Розроблений дизайн сторінки авторизації

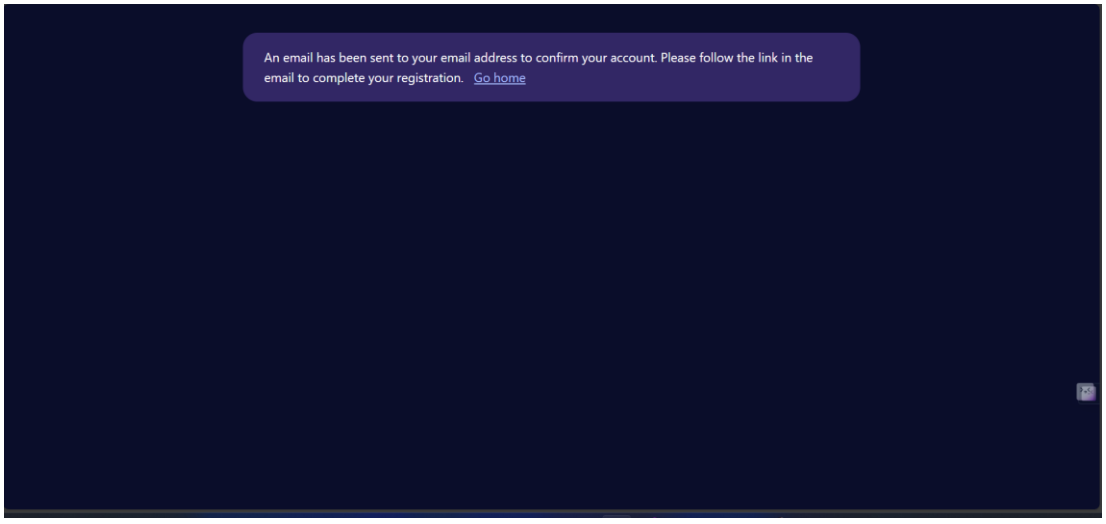


Рисунок 3.12 – Сторінка з повідомленням про подальші дії для завершення реєстрації

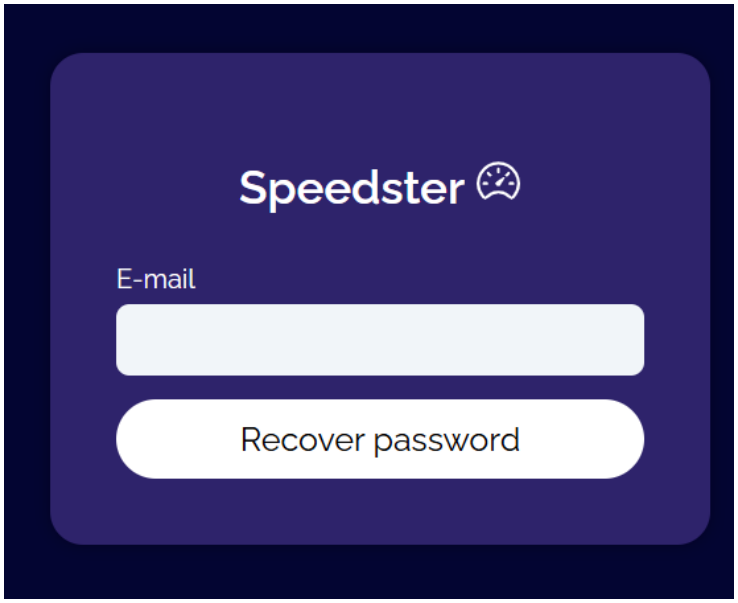


Рисунок 3.13 – Сторінка для відновлення пароля

Слід зазначити, що кінцевий вигляд застосунку буде продемонстрований пізніше, оскільки це не є фінальним дизайном.

Лістинг коду компонентів, які зображені на рис. 3.9-3.12, розміщено в додатку А.

3.3 Розробка навігації сайту

Для переходу між створеними компонентами розроблено навігацію, яка реалізована за допомогою бібліотеки фреймворку React під назвою React Router. Ця бібліотека дозволяє налаштувати перехід з однієї сторінки на іншу в односторінкових застосунках [21].

Наприклад, була поставлена задача створити перехід між головною сторінкою та сторінкою з інформацією про компанію. Для цього в файлі index.js строка, яка відповідає за рендеринг всього застосунку, «обгортається» компонентом BrowserRouter для того, щоб бібліотека React Router працювала справно. Далі в файлі App.js прописуються основні шляхи (рути), за якими користувач буде переходити з однієї сторінки в іншу. «Обгортка» та приклад створених шляхів у застосунку зображені у вигляді коду на рис. 3.12-3.13.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <HeadProvider>
    <App />
    </HeadProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Рисунок 3.12 – «Обгортка» застосунку компонентом BrowserRouter

```
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<MainPage />} />
        <Route path="/register" element={<RegisterPage />} />
        <Route path="/login" element={<LoginPage />} />
      </Routes>
    </div>
  );
}
```

Рисунок 3.13 – Створення шляхів для навігації між створеними компонентами (сторінками)

Як результат, створена навігація виведена в блок на головній сторінці сайту (рис. 3.14).

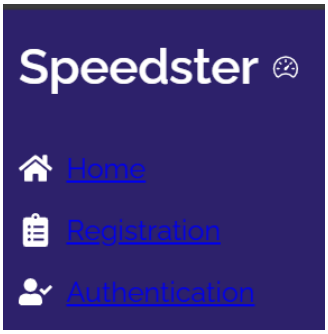


Рисунок 3.14 – Створена навігація застосунку

3.4 Створення боту в Telegram

Для повної реалізації заявленого функціоналу при постановці задачі розроблено бот у соціальній мережі Telegram. Цей бот відповідає за повідомлення про перевищення ліміту швидкості лише для тих користувачів, які успішно пройшли реєстрацію на основному сайті. Ті користувачі, які не були зареєстровані або не пройшли аутентифікацію, не зможуть користуватись функціоналом бота.

Для безпосереднього створення бота використано головного бота для їх створення – BotFather. За його допомогою будь-який користувач соціальної мережі Telegram може створити бота для власних потреб для його подальшої розробки [22].

Процес створення бота у Telegram зазначено на рис. 3.15-3.16.

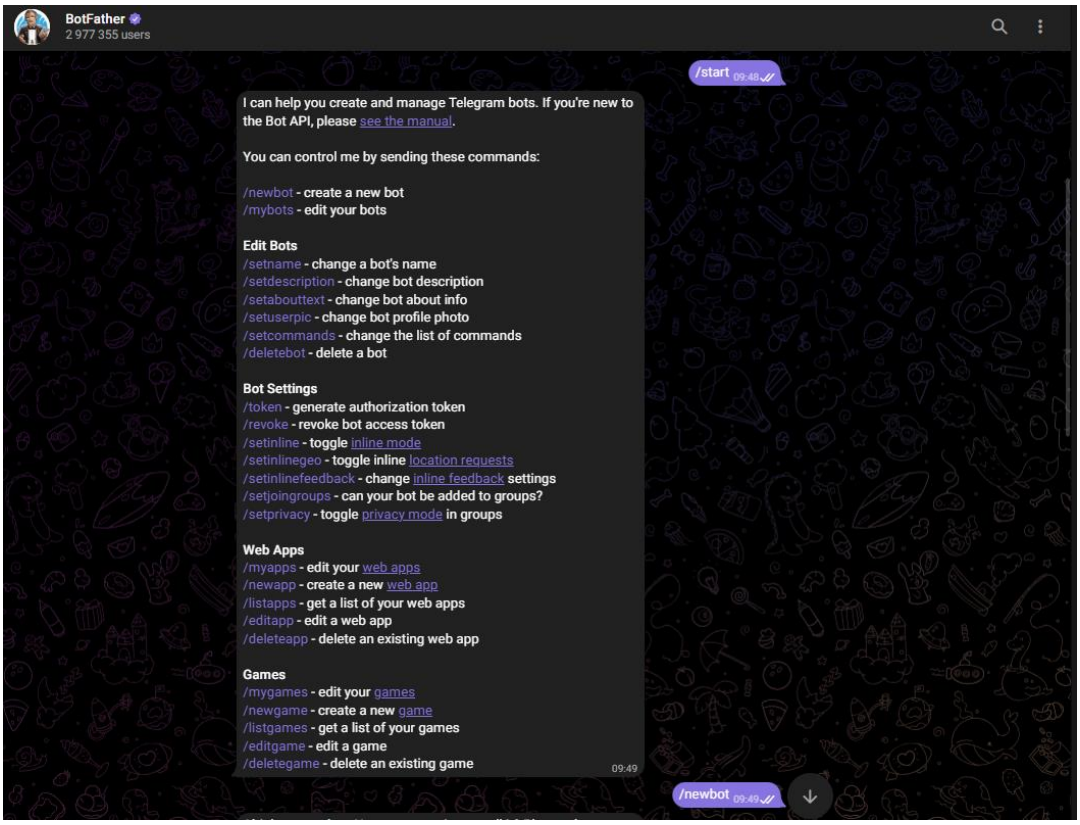
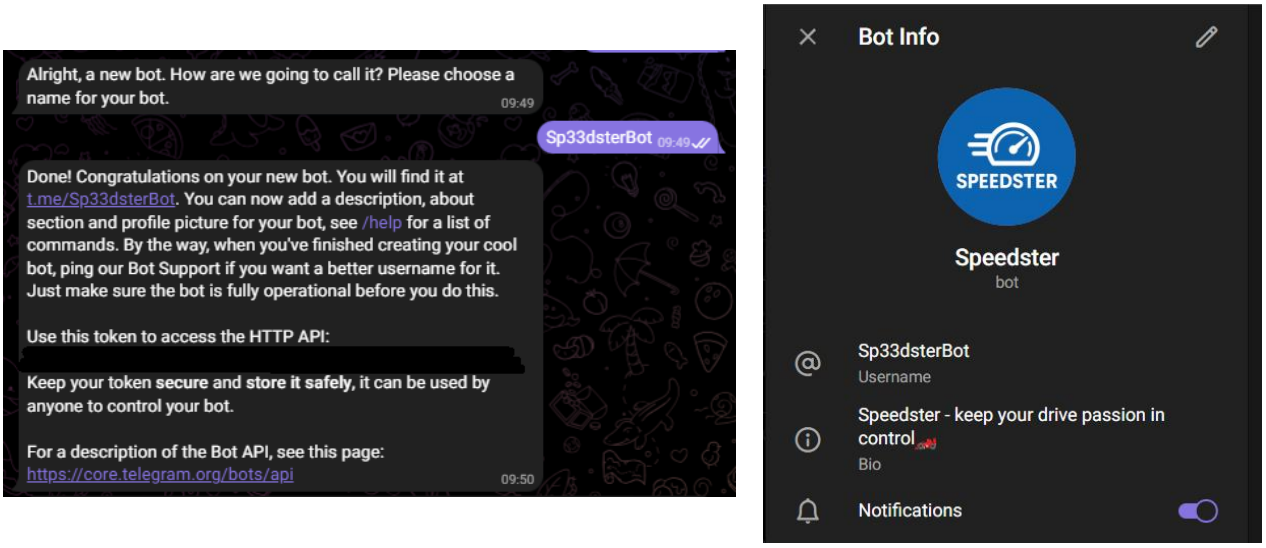


Рисунок 3.15 – Створення бота



а) б)

Рисунок 3.16 – Створений бот та API-ключ (а), оформлення бота та інформація про нього (б)

Для коректної розробки бота необхідно створити подібну до вебзастосунку блок-схему, в якій буде зображено всі можливі взаємодії користувача з ботом. Схематичне дерево рішень у боті Telegram зазначено на рис. 3.17.

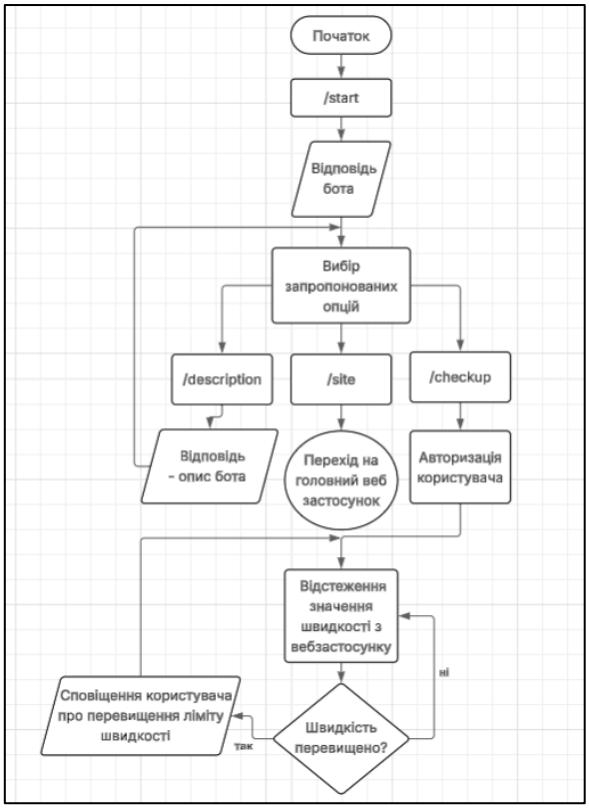


Рисунок 3.17 – Дерево рішень для бота у Telegram

Реалізація перших трьох команд (/start, /description, /site) є досить простою, оскільки потребує небагато зусиль для справної роботи. Фрагменти коду з командами /start, /description та /site зображені на рис. 3.18-3.20.

```
@bot.message_handler(commands=['start'])
def main(message):
    bot.send_message(
        message.chat.id,
        f'Hello, {message.from_user.username}! This is your assistant
Speedster for controlling your speed on streets. How can I help?'
    )
    markup = types.InlineKeyboardMarkup()
    markup.add(types.InlineKeyboardButton('Go on Speedster website',
url='https://www.webpagetest.org/'))
                markup.add(types.InlineKeyboardButton('Description',
callback_data='description'))
        bot.send_message(message.chat.id,   "Choose   an   option:",
reply_markup=markup)
```

Рисунок 3.17 – Обробка команди /start

```
@bot.message_handler(commands=['description'])
def description(message):
    bot.send_message(
        message.chat.id,
        'Speedster is a bot whose main goal is to help drivers control
their speed while driving through city streets. All you need is to be
registered on the main web application Speedster.'
    )
    markup = types.InlineKeyboardMarkup()
    markup.add(types.InlineKeyboardButton('Go on Speedster website',
url='https://www.webpagetest.org/'))
                markup.add(types.InlineKeyboardButton('Description',
callback_data='description'))
        bot.send_message(message.chat.id,   "Choose   an   option:",
reply_markup=markup)
```

Рисунок 3.18 – Обробка команди /description

```
@bot.message_handler(commands=['site', 'website'])
def site(message):
    webbrowser.open('https://www.webpagetest.org/')
```

Рисунок 3.19 – Обробка команди /site

Ці команди мають схожу структуру. При обробці команди виводиться повідомлення з текстом у чат з ботом, а потім виводяться кнопки з вибором подальших дій для користувача. Команда /site має найпростішу структуру з усіх вищезазначених, оскільки для її обробки використовується лише декілька строк коду.

Обробка команд /checkup та робота боту будуть показані пізніше після підключення боту до вебзастосунку.

3.5 Розробка БД та її підключення до вебзастосунку

В постановці задачі було окреслено, що базою даних, яка буде використовуватись для реалізації вебзастосунку, буде MongoDB. Ця БД якнайкраще взаємодіє з застосунками, написаними мовою Node.js, оскільки в ній інформація зберігається не в табличному виді, а в якості об’єктів формату JSON.

Для розробки БД необхідно інсталювати на машину застосунок MongoDB Compass для більш зручної роботи з застосунком. Під час створення в ньому бази даних видано логін та пароль власника, який пізніше знадобиться для підключення.

Створена база даних зазначена на рис. 3.20.

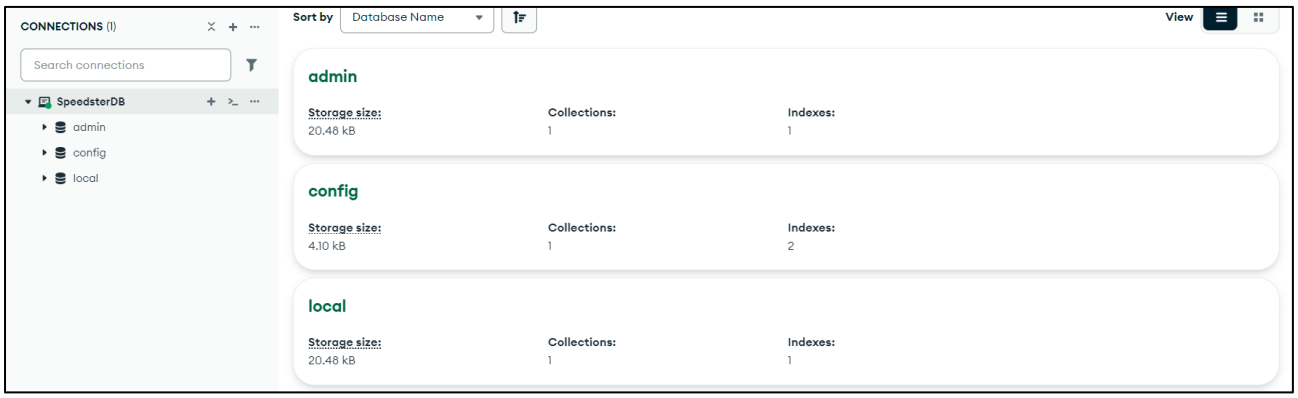


Рисунок 3.20 – Створена база даних

За замовчуванням, створюються 3 БД – admin для можливих налаштувань адміністратора, config для створення пресету налаштувань та local зі

збереженням логів про операції в застосунку. Формат даних виглядає так (рис. 3.21):

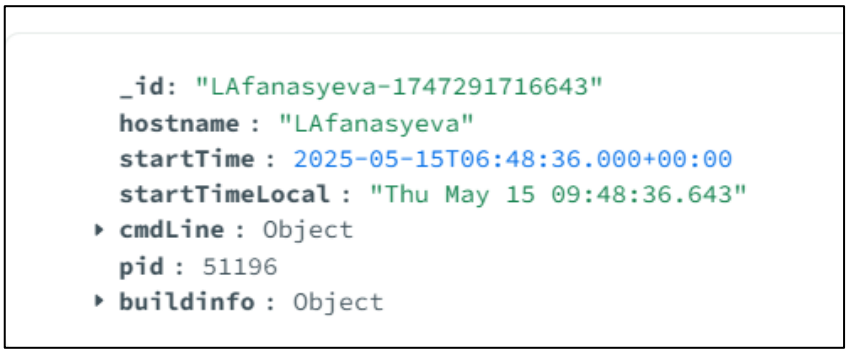


Рисунок 3.21 – Формат запису об’єкта (JSON)

Для даного застосунку достатньо створити колекцію, в якій будуть зберігатись дані користувача, задані при реєстрації (ім’я, номер телефону, електронна пошта, пароль).

Коректний формат запису даних налаштовується безпосередньо при написанні коду. Для виконання цього етапу необхідно створити «модель» користувача, в якій буде вказані необхідні поля для заповнення в форматі схеми MongoDB. Приклад такого запису наведено нижче на рис. 3.22.

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  phone: { type: String, required: true },
  password: { type: String, required: true },
});
```

Рисунок 3.22 – Запис схеми для заповнення колекції в MongoDB

Підключення до БД відбувається досить просто – створюється окремий файл, наприклад під назвою server.js, в якому розробник прописує всі необхідні умови (порт серверу, валідація, винятки тощо). Ключовим елементом для підключення є та сама строка, яка надавалась при створенні бази даних в MongoDB Compass разом з логіном та паролем власника. Приклад такої строки зазначено на рис. 3.23.

```
mongoose.connect('mongodb://localhost:27017/Speedster', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
```

Рисунок 3.23 – Підключення до БД через сервер

Використання моделі User.js, зазначеної на рис. 3.22, відбувається під час реєстрації, коли користувач зробив ввід даних, які пройшли валідацію. Після натискання кнопки «Complete registration» ці дані у форматі JSON записуються в колекцію users у вказану базу даних. Навіть якщо в базі даних відсутня жодна колекція, система автоматично створить таку колекцію, що робить процес роботи максимально зручним і швидким. Приклад успішно пройденої реєстрації та занесених даних у базу даних зазначено на рис. 3.24-3.25.

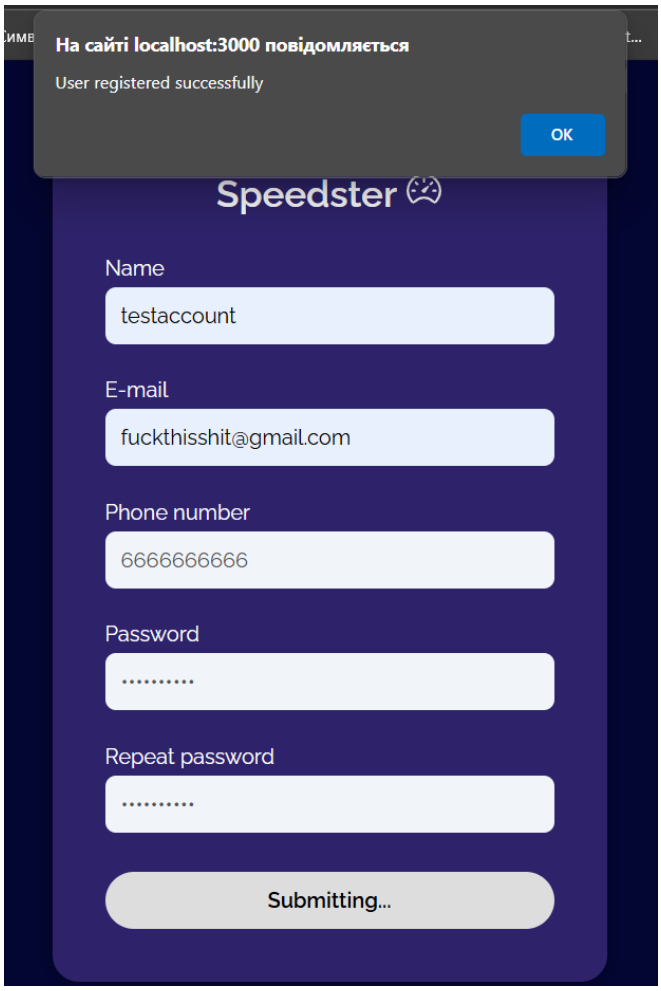


Рисунок 3.24 – Реєстрація користувача пройшла успішно

Рисунок 3.25 – Записані дані в БД

3.6 Підтвердження користувача за допомогою електронної пошти

Після реєстрації користувач перенаправляється на іншу сторінку з приблизним описом «На вашу електрону пошту надійшло письмо з посиланням для підтвердження реєстрації. Будь ласка, перейдіть по ньому для закінчення реєстрації.». Це стандартна практика у багатьох сучасних сервісах та стандартна задача с точки зору розробника.

Для реалізації використана бібліотека nodemailer, яка допомогає надсилати розсилку з поштового ящика кожного разу, коли хтось реєструється або змінює пароль [23]. Щоб все це працювало, розробнику необхідно мати пустий акаунт Google Gmail [24] з налаштованою подвійною аутентифікацією та ключем для входу в застосунок. Всі ці дані вказуються у строках коду, щоб налаштувати транспортер писем. Приклад підключення вказано на рис. 3.26.

```
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'noreplysp33dster@gmail.com',
    pass: 'xxxx xxxx xxxx xxxx',
  },
});
```

Рисунок 3.26 – Приклад підключення пошти для розсилки через nodemailer

Після того, як користувач пройшов реєстрацію та переадресацію на наступну сторінку, на його пошту, яку він вказав під час реєстрації, надсилається

письмо з проханням перейти за посиланням для підтвердження акаунту. У випадку, якщо користувач не підтвердив свій акаунт та намагається пройти логін на сайті, спрацьовує валідація і користувач отримує повідомлення «Будь ласка, пройдіть підтвердження свого акаунту» (рис 3.27). Також про це свідчить значення змінної isVerified у БД (рис. 3.28).

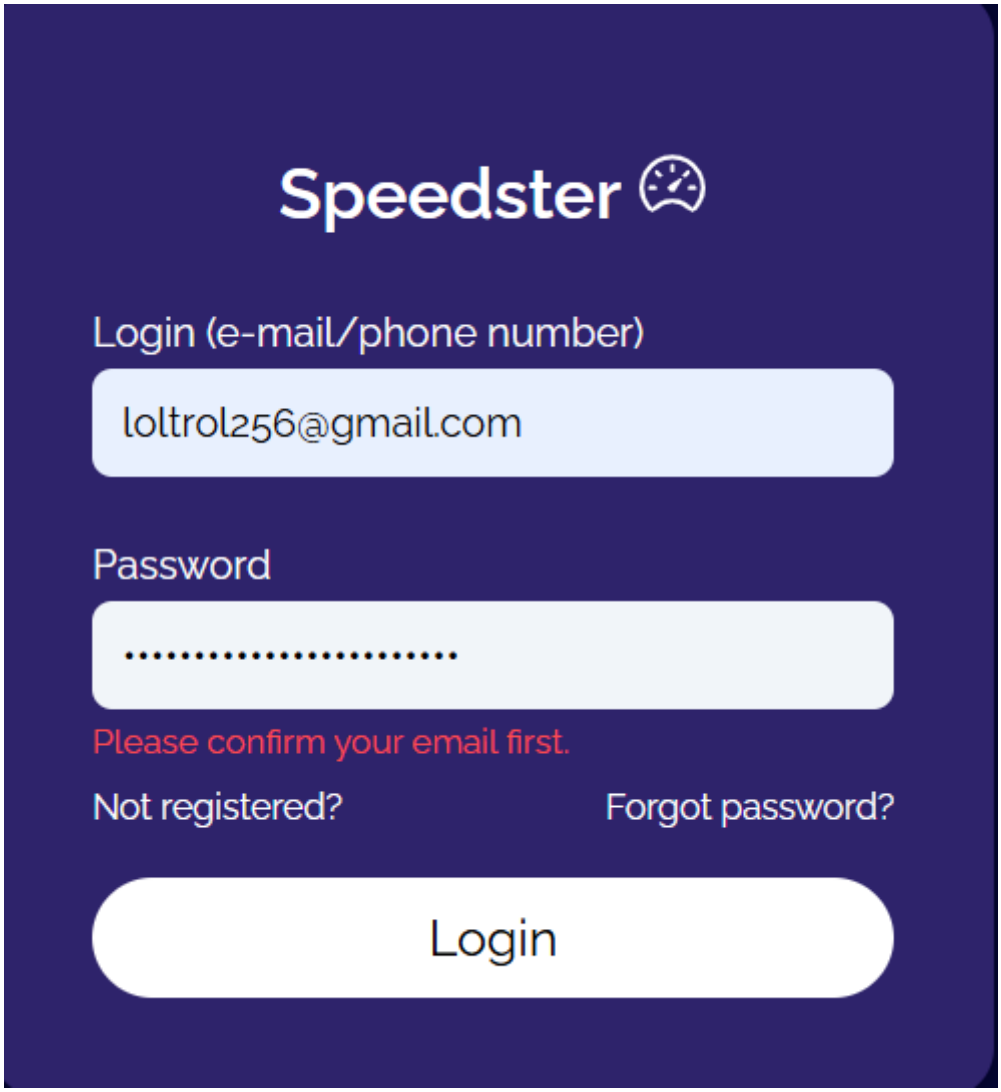


Рисунок 3.27 – Повідомлення про відсутність підтвердження електронної пошти

Рисунок 3.28 – Запис у базі даних про наявність верифікації

3.7 Зміна паролю у випадку його втрати

Зміна паролю у випадку, якщо користувач його забув або втратив, відбувається схожим чином з механізмом підтвердження акаунту через електронну пошту. Механізм реалізовано таким чином: користувач при логіні натискає на кнопку “Forgot password?”, після чого йде переадресація на компонент, в якому користувач вказує свою електронну пошту. У випадку, якщо вона не співпадає з тою, яка була вказана при реєстрації та записана у БД, користувач отримує повідомлення про помилку. В іншому випадку користувач отримує повідомлення про те, що було надіслане письмо на електронну пошту з посиланням для того, щоб оновити втрачений пароль. Процес зміни паролю показано на рис. 3.29-3.32.

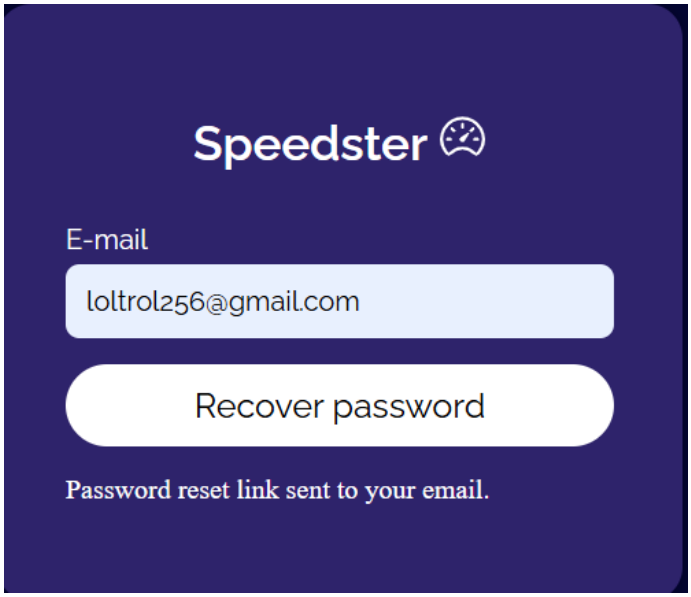


Рисунок 3.29 – Повідомлення про надіслане письмо на електронну пошту

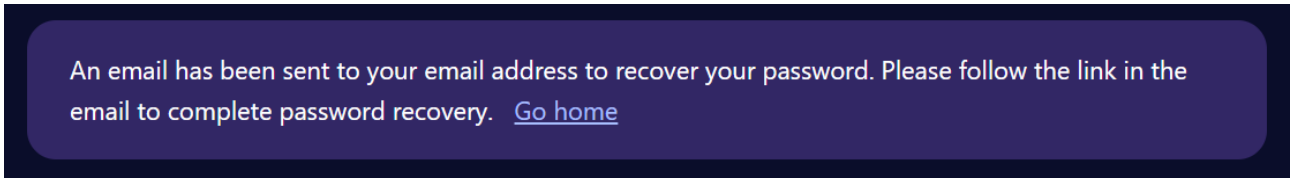


Рисунок 3.30 – Компонент з повідомленням про надіслане письмо

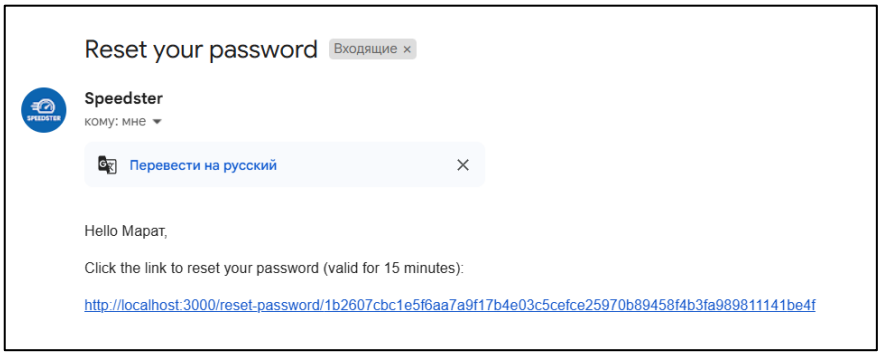


Рисунок 3.31 – Письмо з посиланням для оновлення паролю

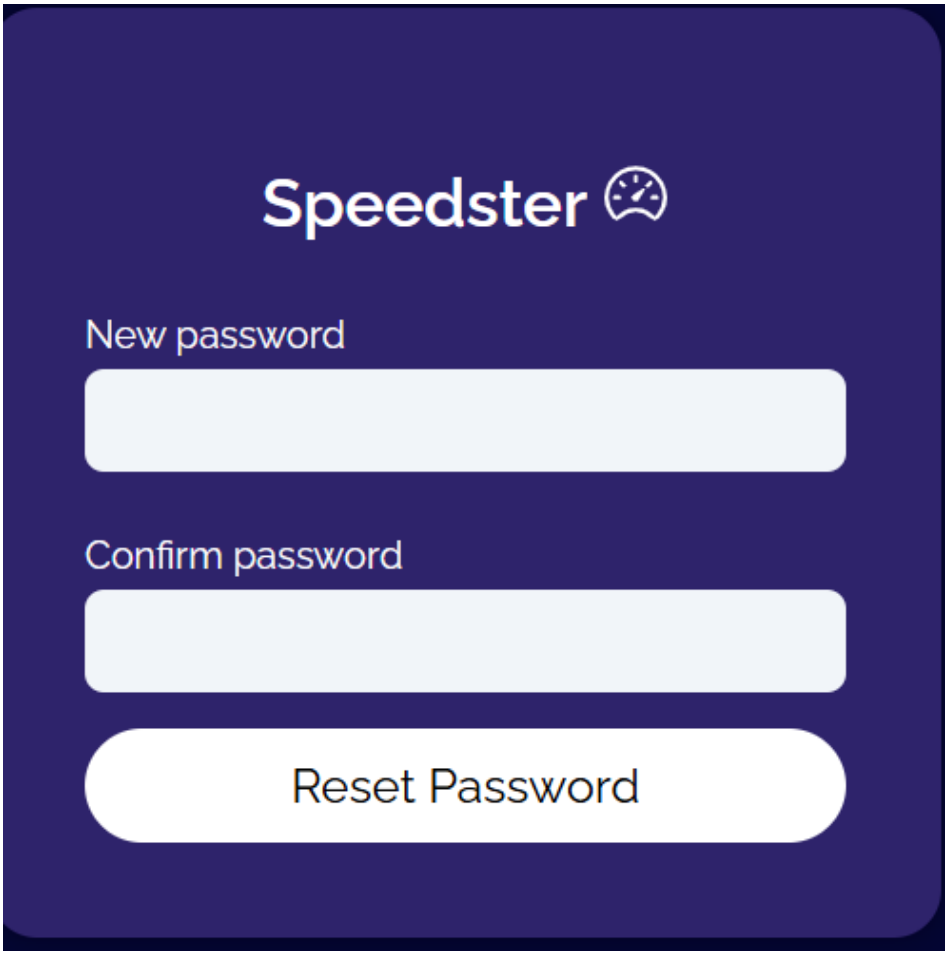


Рисунок 3.32 – Компонент оновлення паролю

Після того, як письмо надійшло на електронну пошту, у користувача є 15 хвилин на оновлення пароля. В іншому випадку токен для оновлення паролю не буде дійсним, і користувачеві необхідно буде знову надіслати запит на оновлення пароля. Токен в БД, який відповідає за можливий час оновлення паролю, зазначено на рис. 3.33.

Рисунок 3.33 – Токен для оновлення паролю

3.8 Підв’язка геолокації в застосунок

Основним завданням вебзастосунку контролю швидкості дорожнього руху є розрахування швидкості користувача під час його руху та підтримання безпечної швидкості на дорогах. Як було описано в постановці задачі, все це реалізовано через взаємодію з геолокацією в реальному часі. Для цього в компонент Speedometer.jsx, який відповідає за візуальний компонент та працездатність функції виміру швидкості, додано функцію калькуляції швидкості в залежності від змін у просторі на основі даних геолокації. Фрагмент коду з функцією зазначено на рис. 3.34.

```
        watchId.current = navigator.geolocation.watchPosition(
          (position) => {
            const { latitude, longitude } = position.coords;
            const currentTime = position.timestamp;

            if (prevPos.current) {
                const { latitude: prevLat, longitude: prevLon, time:
prevTime } = prevPos.current;

              const dist = calculateDistance(prevLat, prevLon, latitude,
longitude);
              const timeElapsed = (currentTime - prevTime) / 1000;
              if (timeElapsed > 0) {
                const speedMps = dist / timeElapsed;
                const speedKph = speedMps * 3.6;
                setSpeed(speedKph);

                if (speedKph > speedLimit) {
                  playAlert();
                } else {
                  stopAlert();
                }
              }
            }
```

Рисунок 3.34 – Функція для вирахування швидкості на основі наданих координат геолокації

У випадку, якщо користувач перейде ліміт у 60 км/год, кожної секунди буде програватись гучний сигнал, який буде повідомляти користувача про пораду зниження швидкості. Фрагмент коду, який відповідає за програвання звукового сигналу, зазначено на рис. 3.35.

```
const playAlert = () => {
   if (!isAlertPlaying.current && audioRef.current) {
     audioRef.current.play();
     isAlertPlaying.current = true;
   }
 };

 const stopAlert = () => {
   if (isAlertPlaying.current && audioRef.current) {
     audioRef.current.pause();
     audioRef.current.currentTime = 0;
     isAlertPlaying.current = false;
   }
 };
```

Рисунок 3.35 – Функції програвання сигналу про перевищення швидкості

3.9 Зв’язка вебзастосунку з ботом в Telegram

Оскільки весь функціонал вебзастосунку, передбачений під час постановки задачі готовий, необхідно зробити прив’язку вебзастосунку до бота у Telegram для того, щоб під час перевищення ліміту швидкості ті користувачі, які зареєстровані та пройшли аутентифікацію, могли не тримати застосунок відкритим весь час та отримувати повідомлення про перевищення швидкості у соцмережу Telegram.

Вищезазначений функціонал реалізовано через API бота, який було надано під час створення бота в самому початку. Для роботи написана функція, суть якої полягає у надсиланні POST-запиту до бота у Telegram через його токен з повідомленням про перевищення швидкості. Дана серверна функція зазначена на рис. 3.36.

```
    app.post('/api/alert', async (req, res) => {
      const { speed } = req.body;
      if (!speed) return res.status(400).json({ error: 'Speed is
required' });

      const message = `🚨 Limit of speed exceeded! Speed at the
moment: ${speed.toFixed(2)} kph`;

      try {
                    const       response       =       await
fetch(`https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessa
ge`, {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            chat_id: TELEGRAM_CHAT_ID,
            text: message,
          }),
        });

        if (!response.ok) throw new Error('Telegram API error');
        res.status(200).json({ message: 'Alert sent to Telegram'
});
      } catch (err) {
        console.error('Telegram alert error:', err);
        res.status(500).json({ message: 'Failed to send alert to
Telegram' });
      }
    });
```

Рисунок 3.36 – Фрагмент коду з функцією сповіщення водія про перевищення швидкості

У Telegram робота даної функції виглядає так (рис. 3.37):

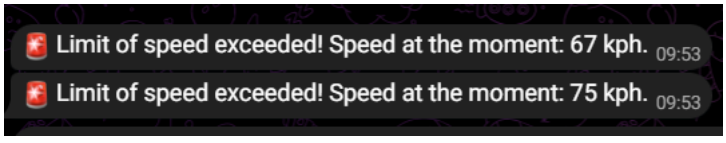


Рисунок 3.37 – Робота функції сповіщення водія про перевищення швидкості у Telegram

3.10 Сесія користувача у застосунку

Під час роботи застосунку було помічено одну з критичних невистачаючих функцій – коректна робота збереження логіну. Ця проблема вирішується інтеграцією JWT-токенів у вебзастосунок [25]. Їх принцип роботи полягає у тому, що під час логіну користувача буде створюватись токен, який буде відповідати за час сесії користувача у застосунок. У випадку, якщо у токена сплинув час дійсності, користувачеві необхідно буде знову пройти логін для того, щоб користуватись повним функціоналом. Фрагмент коду з реалізацією технології JWT-токенів зазначений на рис. 3.38.

```
    const authenticateToken = async (req, res, next) => {
      const authHeader = req.headers['authorization'];
      const token = authHeader?.split(' ')[1];
      if (!token) return res.status(401).json({ message: 'Access
token missing' });

      try {
        const decoded = jwt.verify(token, JWT_SECRET);
        const user = await User.findById(decoded.id);
        if (!user || !user.activeTokens?.includes(token)) {
          return res.status(403).json({ message: 'Invalid or revoked
token' });
        }

        req.user = decoded;
        req.token = token;
        next();
      } catch (err) {
        return res.status(403).json({ message: 'Invalid or expired
token' });
      }
    };
```

Рисунок 3.38 – Фрагмент коду з реалізацією JWT-токенів

Токен зберігається у змінну activeTokens, яка представляє собою масив об’єктів. Активний токен сесії зображено на рис. 3.39.

```
activeTokens : Array (1)
  0: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4MmFkZGIyZmNlNTE4ZTFiY…"
createdAt : 2025-05-19T07:28:50.760+00:00
updatedAt : 2025-05-19T07:41:52.545+00:00
```

Рисунок 3.39 – Активний JWT-токен сесії

3.11 Конфіденційність чутливих даних в коді

Окрім проблеми з відсутністю коректного поняття сесій в застосунку, було виявлено проблему відсутності конфіденційних даних в коді. Раніше паролі від пошти для розсилки, її ключ, API-токен бота у Telegram, порт серверу бази даних та JWT-токен зберігались фактично у відкритому доступі в коді. Таким чином, у випадку зламу розробника та зливу вихідного коду зловмисник отримав би повний контроль над поштою для розсилки, застосунком та ботом у Telegram. Цю проблему було вирішено створенням файлу розширення .env, який містить у собі всі конфіденційні дані, необхідні для контролювання системою загалом [26].

Плюсом до того, налаштовано доступ до файлу за допомогою ключа-пароля, який буде знати лише закрите коло (розробник та адміністратори). Приклад заповнення такого файлу наведено на рис. 3.40.

Рисунок 3.40 – Файл .env з чутливими даними

Також відкореговано серверний код і замінено всі конфіденційні дані на змінні з файлу .env.

3.12 Тестування окремих функцій застосунку. Перспектива вигрузки на хостинг

3.12.1 Тестування

У світі можливої перспективи релізу даного вебзастосунку на ринок необхідно провести тестування для знаходження помилок або недоліків, які не були виявлені на стадії розробки.

Для цього було проведено огляд застосунку в тестовому режимі під час його безпосередньої роботи.

З усіх видів тестування, визначених міжнародною класифікацією ISTQB[27], до вебзастосунку було використано такі види тестування:

– функціональне тестування для перевірки роботи всіх заявлених функцій відповідно до документації;

– підвиди нефункціонального тестування (тестування стабільності, зручності, ефективності та портативності створеного застосунку);

– структурне тестування з використанням методу покриття шляху.

Під час функціонального тестування було виявлено один з недоліків роботи застосунку, а саме низьку частоту опитування геолокації для розрахунку коректної швидкості.

Плюсом до того, під час нефункціонального тестування було виявлено вивід розраховуваємої швидкості у вигляді числа з точкою, а не цілого числа, що також було усунуто (рис. 3.41).

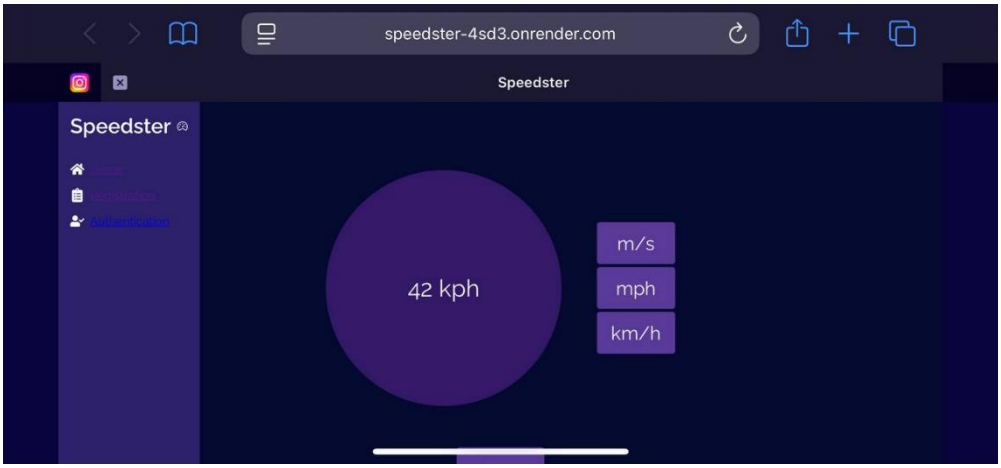


Рисунок 3.41 – Коректне відображення швидкості

3.12.2 Перспектива вигрузки на хостинг

Для підтримки застосунку у довгостроковій перспективі його необхідно вигрузити на домен. Більшість онлайн-сервісів пропонують домени за певну суму грошей, однак в дослідницьких цілях достатньо буде використати один з сервісів, який надає безкоштовний домен на певний час.

Одним з таких сервісів є онлайн-хостинг Render, який надає можливість безкоштовної вигрузки проєкту з його подальшою компоновкою в машинний код та його запуском на наданому домені. [28] Існує також аналог такого сервісу під назвою GitHub Pages, який робить аналогічні дії, однак не дає можливості безкоштовної аренди домену на довгострокову перспективу, і слугує лише демонстрацією візуальної частини [29].

Принцип роботи Render можна описати наступним чином:

застосунок бере заздалегідь створений репозиторій у GitHub, клонує всі створені в ньому файли, та компілює сам застосунок цілком на базі скриптів файлу package.json в корні директорії проєкту. Після вдалої компіляції вебзастосунок вивантажується на згенерований домен, і будь-яка людина може зайти за посиланням і побачити створений вебзастосунок.

Для створеного вебзастосунку по контролю швидкості дорожнього руху було прорблено тестове вивантаження на домен за таким самим принципом.

Скрипти файлу package.json, використані для налаштування компіляції, зазначені на рис. 3.42. Налаштування самого «компілятора» Render зазначені на рис. 3.43. Процес вивантаження з компіляцією зображено на рис. 3.44. Результат вдалого вивантаження зображено на рис. 3.45.

```
        "scripts": {
          "start": "concurrently \"npm run start:backend\" \"npm run
start:frontend\"",
          "start:frontend": "npx cross-env PORT=3006 react-scripts
start",
          "start:backend": "concurrently \"python ../telegram-
bot/main.py\" \"node server.js\"",
          "build": "react-scripts build",
          "test": "react-scripts test",
          "eject": "react-scripts eject"
        },
```

Рисунок 3.42 – Фрагмент коду package.json зі скриптами для компіляції

Рисунок 3.43 – Базові налаштування для вивантаження у середовищі Render

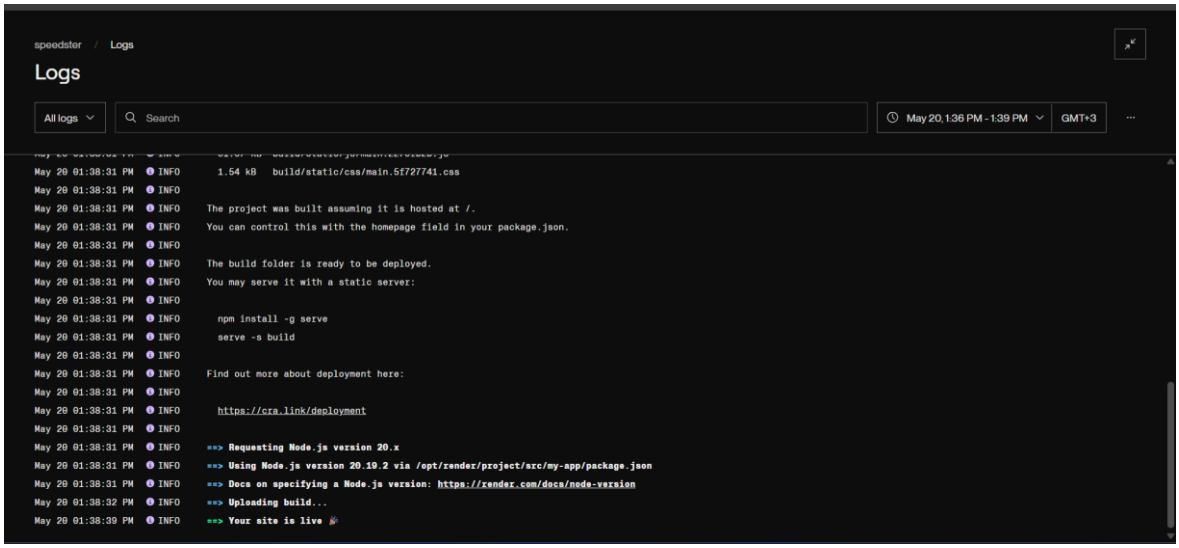


Рисунок 3.44 – Процес вивантаження вебзастосунку на домен

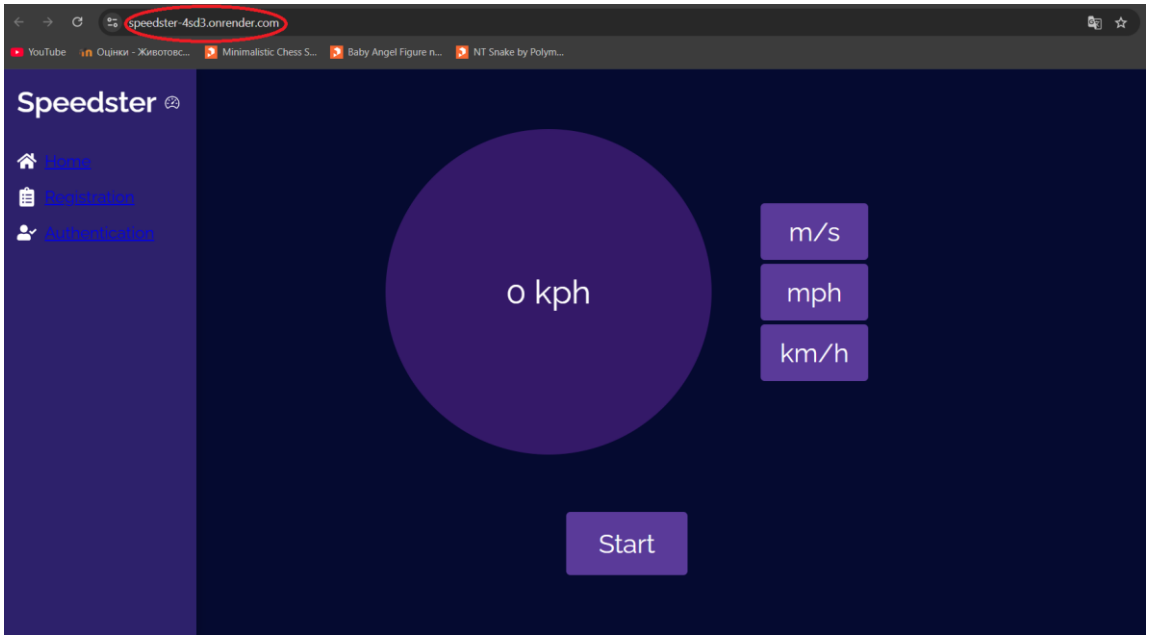


Рисунок 3.45 – Результати вивантаження на домен

Висновки до розділу 3

В даному розділі описано основні сегменти роботи, які стосуються реалізації вебзастосунку для контролю швидкості дорожнього руху.

Створено макет дизайну вебзастосунку, який був орієнтиром для створення подібного дизайну застосунку. Наведено структуру застосунку і бота Telegram, їх логіку у вигляді блок-схем з описаними шляхами. Створено бота у Telegram і прописано базову логіку для реалізації заявленого функціоналу. Розроблено базу даних за допомогою MongoDB для реалізації збереження даних користувачів. Впроваджено базові функції для налагодження системи:

– реєстрація користувача та валідація даних при реєстрації;

– підтвердження акаунту за допомогою посилання на електронній пошті;

– аутентифікація користувача (перевірка вхідних даних з даними, занесеними у БД при реєстрації);

– можливість зміни пароля у випадку його втрати;

– визначена сесія користувача при його аутентифікації тривалістю 2 години.

Реалізовано відстеження швидкості користувача шляхом підключення геолокації до вебзастосунку. Виконано синхронізацію застосунку зі створеним ботом у Telegram для сповіщення зареєстрованих користувачів про перевищення ліміту швидкості.

Виконано рефакторинг коду та винесено конфіденційні дані в окремий файл з обмеженим доступом.

Також виконано тестування застосунку згідно з міжнародною класифікацією ISTQB, усунення виявлених недоліків та тестове вивантаження застосунку на хостинг.

4 РЕЗУЛЬТАТИ РЕАЛІЗАЦІЇ ВЕБЗАСТОСУНКУ

4.1 Окреслення етапів розробки

Розробка програмного забезпечення завжди включає в себе послідовне виконання низки етапів для найбільш ефективного та оптимального процесу розробки. Оскільки всю розробку на момент написання цього розділу вже завершено, є доречним окреслити всі пройдені етапи розробки вебзастосунку для контролю швидкості дорожнього руху.

1. Аналіз вимог та постановка задачі. Проведено дослідження в області інтелектуальних рішень по контролю швидкості дорожнього руху. Серед них обрано три таких рішення, на базі яких сформовано порівняльну таблицю з їх характеристиками. Відповідно до порівняння, сформовано вимоги до вебзастосунку, визначено його функціонал та критерії оцінювання.

2. Проєктування системи. На базі отриманих вимог обрано архітектуру програмного рішення, в основі якої лежить взаємодія зі структурами бази даних та інтерфейсом для користувача. Окреслено технології, мови програмування, фреймворки, бібліотеки та інші інструменти розробки.

3. Реалізація. Виконано програмування основного функціоналу застосунку, заявленого під час постановки задачі. Код написаний з урахуванням принципів чистої структури та з використанням всіх зазначених технологій.

4. Тестування. Проведено проміжкове тестування функцій під час розробки для вчасного коригування неточностей та усунення можливих помилок. Також, після розробки проведено тестування за трьома видами, визначеними міжнародною класифікацією ISTQB [27] – функціональне тестування, нефункціональне тестування та структурне тестування для виявлення помилок, які були пропущені під час розробки. В результаті тестування знайдені помилки в роботі функцій та візуального компоненту, які були усунуті.

5. Можливе впровадження. Виконано тестове вивантаження готового застосунку на хостинг для перевірки його роботи. В результаті тестування

недоліків не виявлено. З вищезазначеного можна зробити висновок, що створений вебзастосунок повністю готовий для часткового впровадження у існуючі системи контролю дорожнього руху для тестування обмеженим колом користувачів на рівні альфа-тестування, щоб виявити помилки, які не були помічені розробником, а також запропонувати власні ідеї щодо можливих допрацювань та покращень вебзастосунку для контролю швидкості дорожнього руху.

Для простішого сприйняття інформація нижче зазначена у вигляді схеми (рис. 4.1).

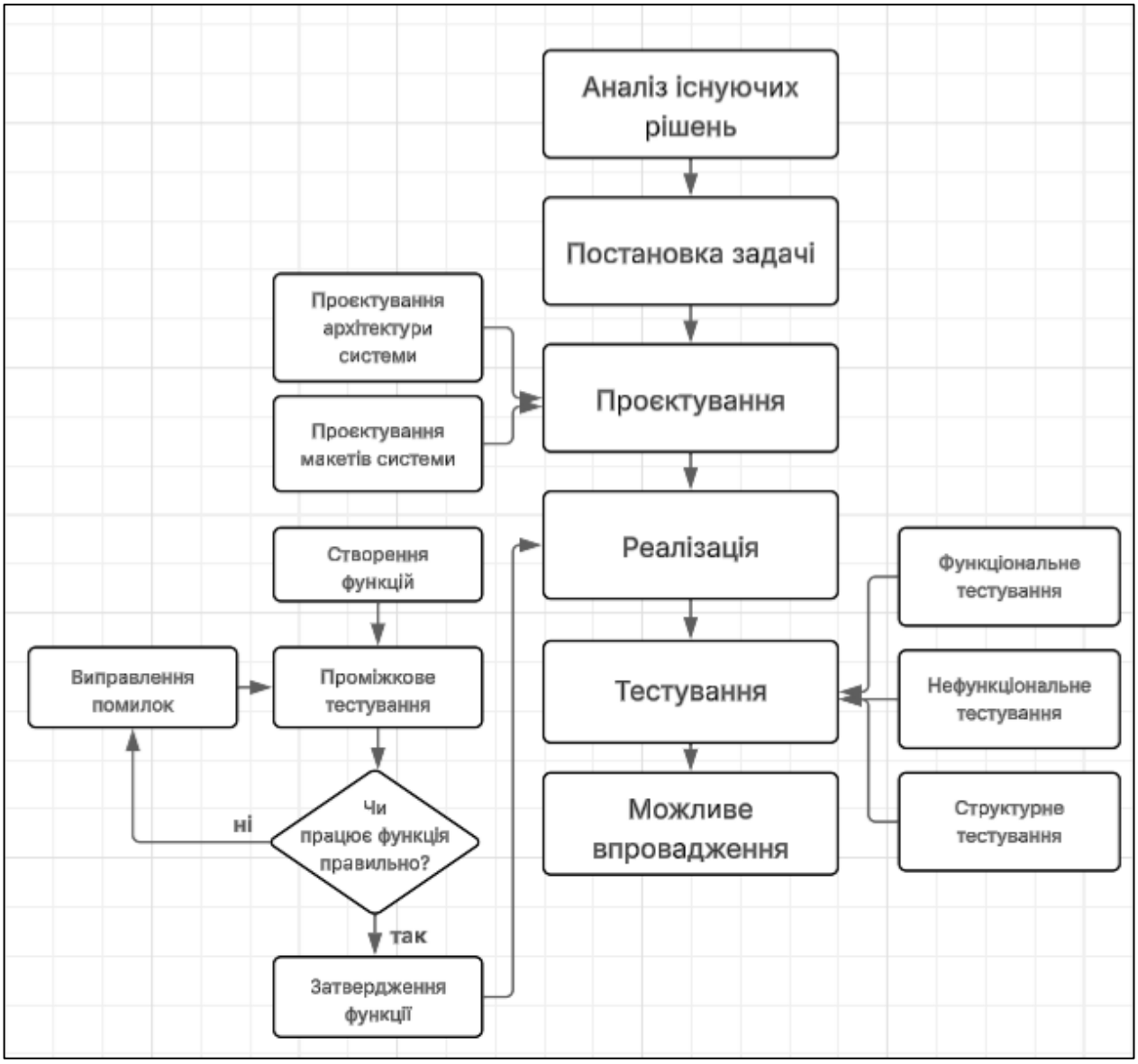


Рисунок 4.1 – Схема пройдених етапів роботи

4.2 Демонстрація результатів розробки

Результатом розробки є готовий вебзастосунок для контролю швидкості дорожнього руху.

Головна сторінка вебзастосунку виглядає наступним чином (рис. 4.2).

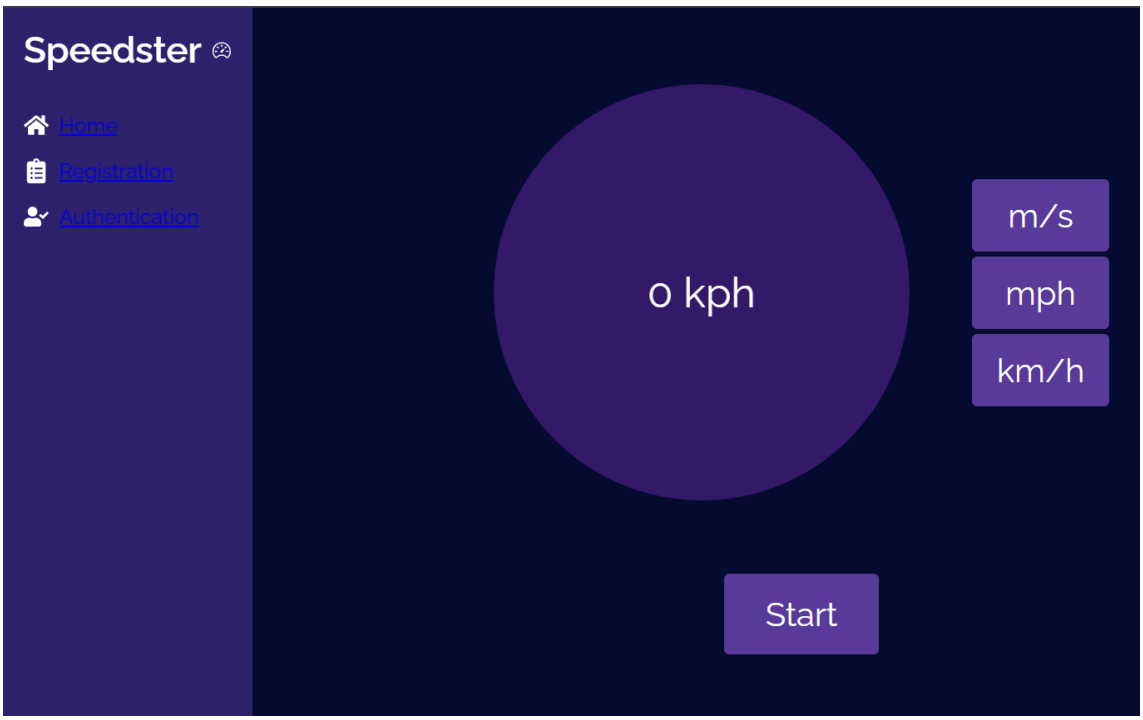


Рисунок 4.2 – Головна сторінка вебзастосунку

На головній сторінці є сегмент з цифровим спідометром, де користувач може змінити одиницю виміру з запропонованих (км/год, м/с, мілі/год). При натисканні кнопки «Start» у користувача буде запитано дозвіл до геолокації, після надання дозволу якої почнеться зчитування змін у геолокації користувача, вирахування швидкості та її виведення на сторінку. Користувач може в будь-який момент змінити одиниці виміру, навіть під час роботи, і це ніяк не вплине на розрахунки, оскільки з технічної точки зору, при отриманні даних геолокації швидкість розраховується у всіх одиницях одночасно. При цьому виводиться лише вказані одиниці. Приклад запиту геолокації зазначено на рис. 4.3, робота спідометру вказана на рис. 4.4.

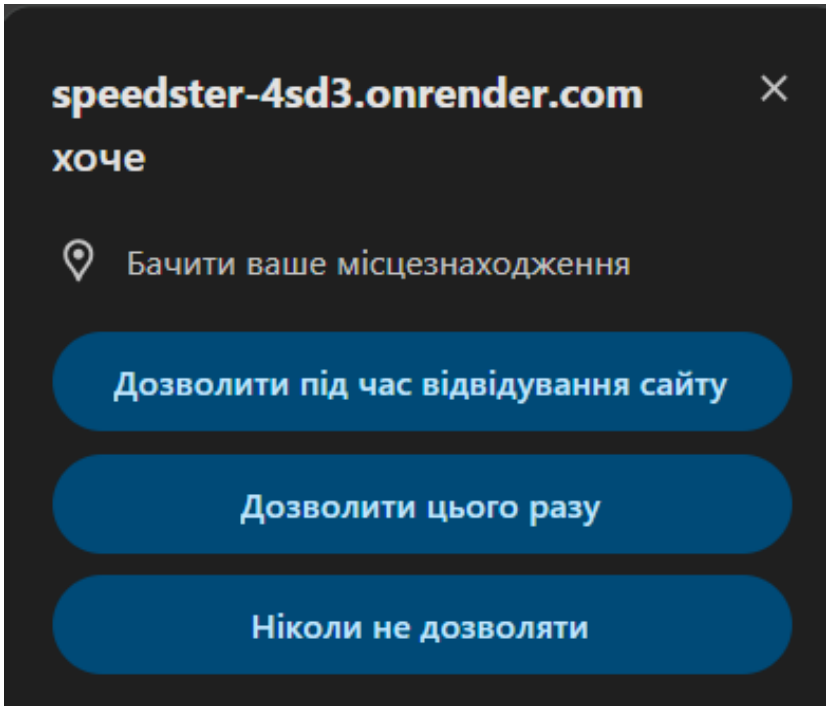


Рисунок 4.3 – Запит геолокації від застосунку

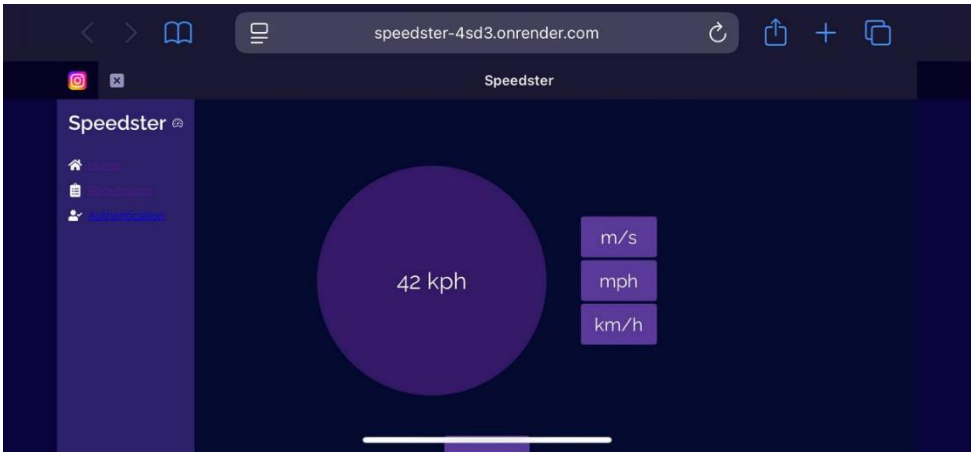


Рисунок 4.4 – Приклад роботи спідометра

У випадку, якщо користувач хоче пройти реєстрацію, він переходить за посиланням «Registration» і бачить перед собою сторінку реєстрації. Виглядає вона наступним чином (рис. 4.5):

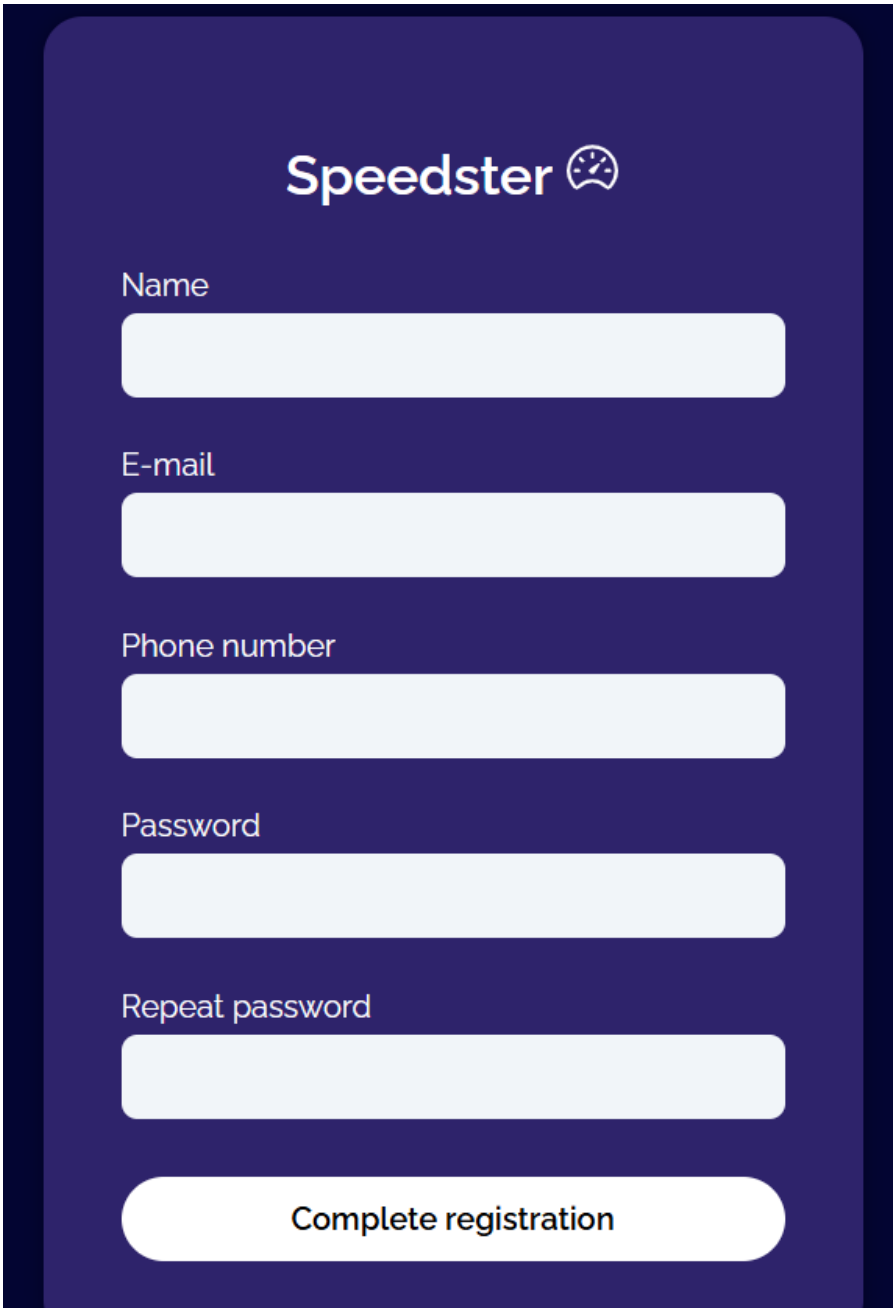


Рисунок 4.5 – Сторінка реєстрації користувача

Користувач вносить свої дані для реєстрації і натискає кнопку «Complete registration». В цей момент проходить перевірка на валідацію даних. У випадку непроходження перевірки користувачеві запропонується змінити дані в тому полі, в якому дані не є правильними і не пройшли валідацію. У випадку, якщо користувач намагається зареєструвати акаунт на ту електронну пошту, на якій вже прив’язаний існуючий акаунт Speedster [30], йому видається помилка і пропозиція пройти аутентифікацію на іншій сторінці. Якщо всі дані правильні і

їх ще немає в базі даних, відбувається переадресація на іншу сторінку з повідомленням про те, що на вказану електронну пошту надіслано письмо з посиланням для завершення реєстрації та підтвердження дійсності акаунта. Приклад сторінки з повідомленням наведено на рис. 4.6, приклад з отриманим письмом наведено на рис. 4.7.

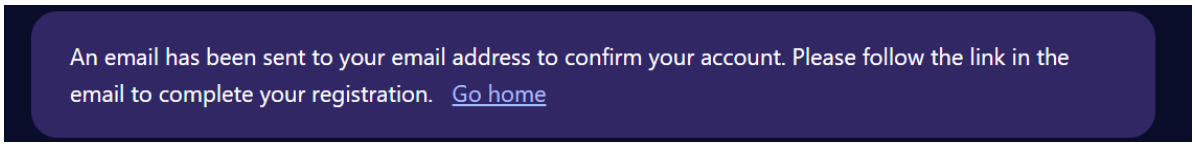


Рисунок 4.6 – Сторінка з повідомленням про письмо з підтвердженням

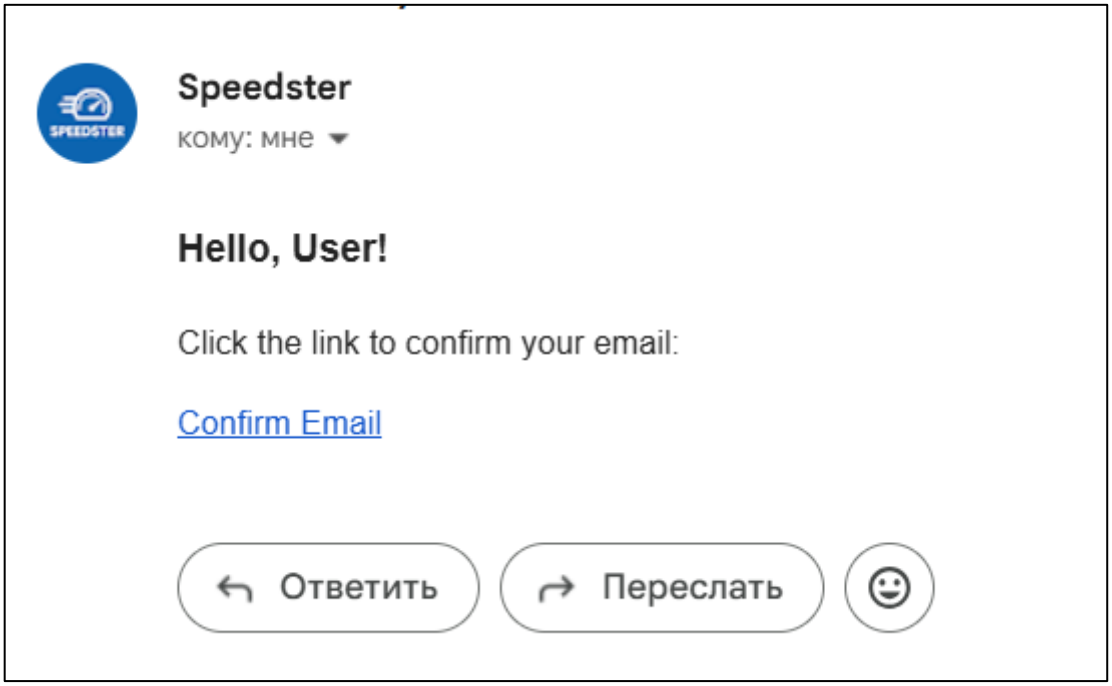


Рисунок 4.7 – Письмо з посиланням для підтвердження акаунту

Після підтвердження йде переадресація на сторінку аутентифікації, виглядає вона наступним чином (рис. 4.8):

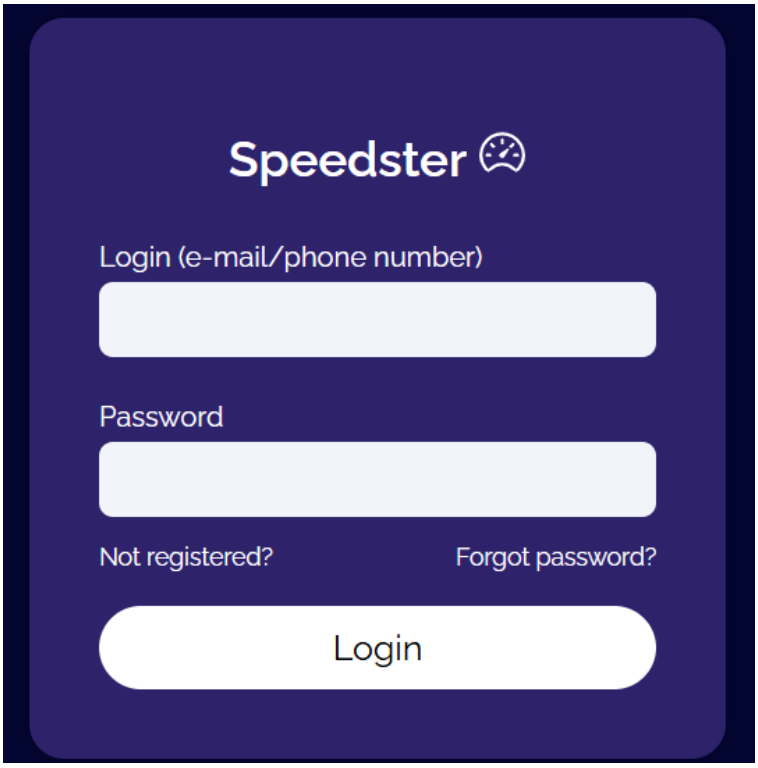


Рисунок 4.8 – Сторінка авторизації

Користувач вводить дані, які він писав при реєстрації (електронна пошта та пароль), після чого успішно авторизується на сайт. Авторизація необхідна користувачеві лише в тому випадку, якщо він хоче користуватись ботом у Telegram, в іншому випадку буде достатньо базового функціоналу сайту.

Базовий функціонал спідометру також включає в себе повідомлення користувача про перевищення швидкості. У випадку перевищення швидкості користувач отримує про це повідомлення шляхом звукового сигналу, який буде програватись постійно до тих пір, поки користувач не знизить її до максимально допустимої – 60 км/год.

У випадку, якщо користувач авторизувався на вебзастосунку і вийшов з браузеру, відстеження його швидкості продовжується, і у випадку перевищення йому в Telegram буде надіслано повідомлення про перевищення (рис. 4.9).

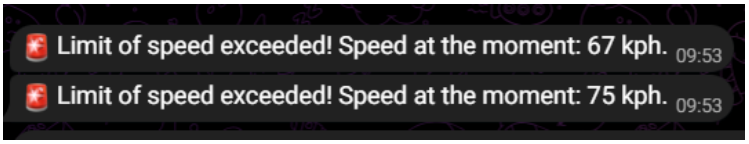


Рисунок 4.9 – Повідомлення про перевищення швидкості у Telegram

4.3 Порівняння створеного застосунку з існуючими аналогами

Для об’єктивної оцінки створеного застосунку необхідно здійснити його порівняння з вже існуючими інтелектуальними рішеннями. В рамках коректного порівняння створений застосунок буде порівнюватись з інтелектуальним рішенням, маючим схожий функціонал та структуру. На основі цих критеріїв було порівняно розроблений вебзастосунок та застосунок Speed Control App [7]. Результати порівняння наведені в таблиці 4.1.

Таблиця 4.1 – Порівняння створеного вебзастосунку Speedster та застосунку Speed Control App

Характеристика/ назва застосунку	**Radarbot: Speed Cameras | GPS**	**Speed Control App**
Характеристики системи	Інтелектуальна система, яка розраховує швидкість водія та попереджує його про перевищення швидкості, базуючись на його геолокації	Інтелектуальна система, яка відстежує швидкість водія та показує діючі обмеження на дорозі, базуючись на геолокації користувача.
Платформа	Будь-яка (мінімальні вимоги – пристрій, який має доступ до Інтернету, влаштований браузер та модуль геолокації)	Android
Функціонал	Контроль швидкості транспортного засобу зі встановленим обмеженням для водія; сповіщення про перевищення ліміту швидкості	Контроль швидкості транспортного засобу з можливістю встановлення обмежень та отримання сповіщень про їх перевищення; надає статистику поїздок, допомагаючи аналізувати стиль водіння та покращувати безпеку на дорозі.
Переваги	Простота реалізації, повний функціонал є безкоштовним	Простота реалізації

Недоліки	Недостатній функціонал, порівнюючи з іншими інтелектуальними рішеннями на ринку	Необхідність заплатити гроші за повний функціонал застосунку, недостатній функціонал
Можливість масштабування	Високий рівень	Високий рівень

За результатами здійсненого порівняння, застосунок Speedster перевершує застосунок Speed Control App [7] за двома критеріями – кросплатформність та безкоштовність. Завдяки відсутності необхідності платити за повний функціонал та доступ до застосунку з будь-якого пристрою, який має дисплей, Інтернет-модуль та модуль геолокації.

Висновки до розділу 4

В даному розділі наведено відомості про результати реалізації застосунку. Наведено стислий опис етапів розробки, які були пройдені та їх характеристики.

Створено схему, яка візуалізує всі етапи розробки та коротко характеризує їх. Представлено результати фінальної версії застосунку, а саме продемонстровано візуальну частину та описано функціонал, який був передбачений під час постановки задачі.

Здійснене порівняння створеного вебзастосунку для контролю швидкості дорожнього руху з подібним існуючим рішенням – застосунком Speed Control App. Виявленими перевагами розробки є безкоштовність повного функціоналу та кросплатформеність, в той час як недоліком є недостатній функціонал порівняно з усіма інтелектуальними рішеннями даної сфери.

ВИСНОВКИ

В результаті бакалаврської роботи було ретельно проаналізовано наявні на ринку інформаційні системи контролю швидкості дорожнього руху, що дозволило виявити їхні сильні та слабкі сторони. Було встановлено, що більшість сучасних систем використовують мікросервісну архітектуру для покращення масштабованості та надійності. Вони також активно інтегруються з різноманітними джерелами даних, такими як дорожні камери, GPS-трекери та датчики швидкості.

На основі проведеного аналізу був розроблений власний вебзастосунок для контролю швидкості дорожнього руху, який також базується на мікросервісній архітектурі. Система включає кілька ключових сервісів: сервіс збору та обробки даних про швидкість та геолокацію, сервіс сповіщення про порушення, сервіс аутентифікації та сервіс конфігурацій.

Крім того, була розроблена frontend та backend частини застосунку. Для цього використовувалися такі технології, як середовище виконання Node.js, мова JavaScript, фреймворк React та низка бібліотек, зокрема axios, bcrypt, cors, crypto, express, jsonwebtoken, nodemailer та інші. Для зберігання та адміністрування даних користувачів була створена та підключена база даних MongoDB. Додатково, було розроблено Telegram-бот, який повністю синхронізується з основним вебзастосунком та дублює його функціонал.

Розроблена система дозволяє автоматизувати процеси контролю швидкості, що суттєво зменшує час на обробку даних, підвищує точність виявлення порушень та мінімізує ймовірність помилок. Це досягається завдяки інтеграції різних сервісів та застосуванню сучасних технологій обробки та візуалізації геопросторових даних.

Особливу увагу було приділено безпеці даних та захисту інформації. Використання високого рівня валідації даних та шифрування конфіденційної інформації шляхом її конвертації у токени забезпечує надійний захист від

несанкціонованого доступу, а також конфіденційність персональних даних відповідно до чинного законодавства.

Система розроблена з акцентом на зручність для користувача. Інтуїтивно зрозумілий інтерфейс та висока швидкість обробки даних сприяють підвищенню ефективності роботи та загальному покращенню безпеки на дорогах через контроль швидкості водіїв.

Отже, розроблений вебзастосунок по контролю швидкості дорожнього руху відповідає сучасним вимогам ринку та має задовільну функціональність, що дозволяє його ефективно використовувати в умовах реальної дорожньої інфраструктури для підвищення безпеки та дисципліни на дорогах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. BankID НБУ. *Національний банк України*. URL: https://bank.gov.ua/ua/bank-id-nbu/ (дата звернення: 27.04.2025).

2. Дія – Державні послуги онлайн. *Державні послуги онлайн | Дія*. URL: https://diia.gov.ua/ (дата звернення: 27.04.2025).

3. Bevor Sie zu Google Maps weitergehen. *Google*. URL: https://www.google.com/maps (last accessed: 27.04.2025).

4. Вказівки руху, затори наживо та поточна ситуація на дорогах - Waze. *Waze*. URL: https://www.waze.com/uk/live-map (дата звернення: 27.04.2025).

5. Company R. Radarbot speed camera detector - apps on google play. *Android Apps on Google Play*. URL: https://surl.li/ftlrkj (last accessed: 27.04.2025).

6. Speedometer 55 GPS speed & HUD by stanislav dvoychenko. *AppAdvice*. URL: https://surl.li/egorju (last accessed: 27.04.2025).

7. Speed control app for android - free app download. *AppBrain*. URL: https://surl.li/dftbgl (last accessed: 27.04.2025).

8. Що краще моноліт чи мікросервіси? | IAMPM. *IAMPM*. URL: https://surl.li/pdicyq (дата звернення: 27.04.2025).

9. Монолітна архітектура ПЗ. - qalight. *QALight*. URL: https://surl.li/xbgfyj (дата звернення: 27.04.2025).

10. Монолітна архітектура проти архітектури мікросервісів - javascript.org.ua - JS communities. *javascript.org.ua - JS Communities*. URL: https://surli.cc/fhivtk (дата звернення: 27.04.2025).

11. Поняття та функції СКБД. компоненти СКБД. – вікі ЦДУ. *Вікі ЦДУ*. URL: https://surl.li/ppzrsq (дата звернення: 27.04.2025).

12. JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (last accessed: 26.05.2025).

13. Node.js – Запускайте JavaScript будь-де. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/uk (дата звернення: 26.05.2025).

14. Quick Start – React. *React*. URL: https://react.dev/learn (last accessed: 26.05.2025).

15. Python 3.13 documentation. *Python documentation*. URL: https://docs.python.org/3/ (last accessed: 26.05.2025).

16. Welcome to AIOHTTP – aiohttp 3.12.0 documentation. *Welcome to AIOHTTP – aiohttp 3.12.0 documentation*. URL: https://surl.li/sptyti (last accessed: 26.05.2025).

17. What is MongoDB? | Geekboots. *Geekboots for Web Developer Programmer and Tech Enthusiast*. URL: https://www.geekboots.com/story/what-is-mongodb (last accessed: 07.05.2025).

18. Lucidchart | diagramming powered by intelligence. *Lucidchart*. URL: https://www.lucidchart.com/pages/ (last accessed: 26.05.2025).

19. What is canva? 2025 guide | firebear. *Magento 2 Import Extension & Integration solutions | Firebear Studio*. URL: https://surl.lu/yubxkp (last accessed: 26.05.2025).

20. React. *React*. URL: https://react.dev/ (last accessed: 26.05.2025).

21. React router official documentation. *React Router Official Documentation*. URL: https://reactrouter.com/ (last accessed: 26.05.2025).

22. BotFather. *Expertflow CX*. URL: https://surl.lu/irjkcz (last accessed: 26.05.2025).

23. Nodemailer | nodemailer. *Nodemailer | Nodemailer*. URL: https://nodemailer.com/ (last accessed: 26.05.2025).

24. Gmail. *Gmail*. URL: https://mail.google.com/mail/ (last accessed: 26.05.2025).

25. Jsonwebtoken. *npm*. URL: https://surli.cc/jhoxwe (last accessed: 26.05.2025).

26. What is .env ? How to Set up and run a .env file in Node? | Codementor. *Codementor | Get live 1:1 coding help, hire a developer, & more*. URL: https://surl.li/fqewsk (last accessed: 26.05.2025).

27. International Software Testing Qualifications Board. *International Software Testing Qualifications Board*. URL: https://www.istqb.org/ (last accessed: 26.05.2025).

28. Cloud Application Platform | Render. *Render*. URL: https://render.com/ (last accessed: 26.05.2025).

29. GitHub Pages. *GitHub Pages*. URL: https://pages.github.com/ (last accessed: 26.05.2025).

30. Speedster. *Speedster*. URL: https://speedster-4sd3.onrender.com/ (last accessed 26.05.2025).

ДОДАТОК А

Програмна реалізація вебзастосунку

Лістинг коду головного файлу App.js

```
      import { Routes, Route } from 'react-router-dom';
      import MainPage from './components/MainPage/MainPage';
      import LoginPage from './components/LoginPage/LoginPage';
      import RegisterPage from './components/RegisterPage/RegisterPage';
      import PasswordRenewal from
'./components/PasswordRenewal/PasswordRenewal';
      import EmailPasswordRecovery from './components/E-
mailPasswordRecovery/EmailPasswordRecovery';
      import EmailConfirmLetter from './components/E-
mailConfirmLetter/EmailConfirmLetter';
      import Dashboard from './components/Dashboard/Dashboard';
      import ResetPassword from './components/PasswordRenewal/ResetPassword';

      function App() {
        return (
          <div className="App">
            <Routes>
              <Route path="/" element={<MainPage />} />
              <Route path="/register" element={<RegisterPage />} />
              <Route path="/login" element={<LoginPage />} />
              <Route path ="/passwordrecovery" element = {<PasswordRenewal/>} />
              <Route path ="/emailpasswordrecovery" element =
{<EmailPasswordRecovery/>} />
              <Route path ="/emailconfirmationletter" element =
{<EmailConfirmLetter/>} />
              <Route path = "/dashboard" element = {<Dashboard/>} />
              <Route path="/reset-password/:token" element={<ResetPassword />}
/>
            </Routes>
          </div>
        );
      }

      export default App;
```

Лістинг коду структурного файлу index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom'
import App from './App';
import { HeadProvider } from 'react-head';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <HeadProvider>
    <App />
    </HeadProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Лістинг коду компонента MainPageLayout.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { FaHome, FaClipboardList, FaUserCheck } from 'react-icons/fa';
import { IoSpeedometerOutline } from 'react-icons/io5';
import '../../styles/MainPage/MainPageLayout.css';

const MainPageLayout = ({ children }) => {
  return (
    <div className="main-page-container">
      <div className="side-panel">
        <div className="side-panel-header">
          <span>Speedster</span>
          <IoSpeedometerOutline style={{ fontSize: '32px', color: 'white'
}} />
        </div>

        <nav className="side-panel-nav">
          <div className="side-panel-nav-item">
            <FaHome className="side-panel-icon" />
            <Link to="/">Home</Link>
          </div>

          <div className="side-panel-nav-item">
            <FaClipboardList className="side-panel-icon" />
            <Link to="/register">Registration</Link>
          </div>

          <div className="side-panel-nav-item">
            <FaUserCheck className="side-panel-icon" />
            <Link to="/login">Authentication</Link>
          </div>
        </nav>
      </div>
      <div className="background">{children}</div>
    </div>
  );
};

export default MainPageLayout;
```

Лістинг коду компонента Speedometer.jsx

```
import React, { useState, useEffect, useRef } from 'react';
import '../../styles/MainPage/Speedometer.css';

const Speedometer = () => {
  const [speed, setSpeed] = useState(0); // в km/h
  const [unit, setUnit] = useState('kph');
  const watchId = useRef(null);
  const prevPos = useRef(null);

  const getSpeedInUnit = () => {
    switch (unit) {
      case 'mps':
        return Math.round(speed / 3.6);
      case 'mph':
        return Math.round(speed * 0.621371);
      default:
        return Math.round(speed);
    }
  };

  const changeUnit = (newUnit) => {
    setUnit(newUnit);
  };

  const calculateDistance = (lat1, lon1, lat2, lon2) => {
    const toRad = (val) => (val * Math.PI) / 180;
    const R = 6371000;
    const dLat = toRad(lat2 - lat1);
    const dLon = toRad(lon2 - lon1);
    const a =
      Math.sin(dLat / 2) ** 2 +
      Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) * Math.sin(dLon
/ 2) ** 2;
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return R * c;
  };

  const startTracking = () => {
    if (!navigator.geolocation) {
      alert('Geolocation is not supported by your browser');
      return;
    }

    if (watchId.current !== null) {
      navigator.geolocation.clearWatch(watchId.current);
      watchId.current = null;
      setSpeed(0);
      prevPos.current = null;
      return;
    }

    watchId.current = navigator.geolocation.watchPosition(
      (position) => {
```

```
            const { latitude, longitude } = position.coords;
            const currentTime = position.timestamp;

            if (prevPos.current) {
              const { latitude: prevLat, longitude: prevLon, time:
prevTime } = prevPos.current;

              const dist = calculateDistance(prevLat, prevLon,
latitude, longitude);
              const timeElapsed = (currentTime - prevTime) / 1000;

              if (timeElapsed > 0) {
                const speedMps = dist / timeElapsed;
                const speedKph = speedMps * 3.6;
                setSpeed(speedKph);
              }
            }

            prevPos.current = { latitude, longitude, time: currentTime
};
          },
          (error) => {
            if (error.code === error.TIMEOUT) {
              console.warn('Geolocation timeout - position did not
change');
              return;
            }
            console.error('Error getting position:', error);
            alert(`Error of getting geolocation');
          },
          {
            enableHighAccuracy: true,
            maximumAge: 0,
            timeout: 1000
          }
        );
      };

      useEffect(() => {
        return () => {
          if (watchId.current !== null) {
            navigator.geolocation.clearWatch(watchId.current);
          }
        };
      }, []);

      return (
        <div className="app-container">
          <main className="content">
            <div className="speedometer-section">
              <div className="speedometer-container">
                <div className="speedometer-unit-wrapper">
                  <div className="speedometer-display">
                    <div className="digital-speedometer">
                      <p>
                        {getSpeedInUnit()} {unit}
```

```
                    </p>
                  </div>
                </div>
                <aside className="unit-selector">
                  <button onClick={() =>
changeUnit('mps')}>m/s</button>
                  <button onClick={() =>
changeUnit('mph')}>mph</button>
                  <button onClick={() =>
changeUnit('kph')}>km/h</button>
                </aside>
              </div>
            </div>

            <div className="speed-controls">
              <button className="start-button"
onClick={startTracking}>
                {watchId.current === null ? 'Start' : 'Stop'}
              </button>
            </div>
          </div>
        </main>
      </div>
    );
  };

  export default Speedometer;
```

Лістинг коду компонента PasswordRenewal.jsx

```
import React, { useState } from 'react';
import "../../styles/LoginPage/LoginPage.css";
import { IoSpeedometerOutline } from "react-icons/io5";
import { Meta } from 'react-head';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const PasswordRenewal = () => {
  const [email, setEmail] = useState('');
  const [message, setMessage] = useState('');
  const navigate = useNavigate();

  const handleRecoverPassword = async () => {
    try {
      const res = await axios.post('http://localhost:3005/forgot-
password', { email });
      setMessage(res.data.message);

      setTimeout(() => {
        navigate('/emailpasswordrecovery');
      }, 3000);
    } catch (err) {
      setMessage(err.response?.data?.message || 'Error occurred');
    }
  };

  return (
    <>
      <Meta name="viewport" content="width=device-width, initial-
scale=1.0" />
      <div className="login-background">
        <div className="login-container">
          <h2 className="login-title">
            Speedster <IoSpeedometerOutline style={{ fontSize:
'28px', color: 'white' }} />
          </h2>

          <label className="login-label">E-mail</label>
          <input
            type="email"
            className="login-input"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
          />

          <button className="login-button"
onClick={handleRecoverPassword}>
            Recover password
          </button>

          {message && <p className="login-message">{message}</p>}
        </div>
```

```
        </div>
      </>
    );
};

export default PasswordRenewal;
```

Лістинг коду компонента ResetPassword.jsx

```
import React, { useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { IoSpeedometerOutline } from "react-icons/io5";
import axios from 'axios';
import "../../styles/LoginPage/LoginPage.css";
import { Meta } from 'react-head';

const ResetPassword = () => {
  const { token } = useParams();
  const navigate = useNavigate();
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [message, setMessage] = useState('');

  const handleReset = async () => {
    if (password !== confirmPassword) {
      setMessage('✕ Passwords do not match');
      return;
    }

    try {
      const res = await axios.post(`http://localhost:3005/reset-
password/${token}`, { password });
      setMessage(res.data.message);
      setTimeout(() => navigate('/login'), 3000);
    } catch (err) {
      setMessage(err.response?.data?.message || '✕ Error
occurred');
    }
  };

  return (
    <>
      <Meta name="viewport" content="width=device-width, initial-
scale=1.0" />
      <div className="login-background">
        <div className="login-container">
          <h2 className="login-title">
            Speedster <IoSpeedometerOutline style={{ fontSize:
'28px', color: 'white' }} />
          </h2>

          <label className="login-label">New password</label>
          <input
            type="password"
            className="login-input"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />

          <label className="login-label">Confirm password</label>
          <input
            type="password"
```

```
            className="login-input"
            value={confirmPassword}
            onChange={(e) => setConfirmPassword(e.target.value)}
          />

          <button className="login-button" onClick={handleReset}>
            Reset Password
          </button>

          {message && <p className="login-message">{message}</p>}
        </div>
      </div>
    </>
  );
};

export default ResetPassword;
```

Лістинг коду компонента RegisterPage.jsx

```
import React, { useState } from 'react';
import '../../styles/RegisterPage/RegisterPage.css';
import { IoSpeedometerOutline } from "react-icons/io5";
import { Meta } from 'react-head';
import { useNavigate } from 'react-router-dom';

const RegisterPage = () => {
  const navigate = useNavigate();

  const [form, setForm] = useState({
    name: '',
    email: '',
    phone: '',
    password: '',
    repeatPassword: '',
  });

  const [errors, setErrors] = useState({});
  const [serverError, setServerError] = useState('');
  const [loading, setLoading] = useState(false);

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const validate = () => {
    const newErrors = {};

    if (!form.name.trim()) newErrors.name = 'Enter your name';
    if (!form.email.trim()) newErrors.email = 'Enter e-mail';
    else if (!/\S+@\S+\.\S+/.test(form.email)) newErrors.email =
'Uncorrect e-mail';

    if (!form.phone.trim()) newErrors.phone = 'Enter your phone
number';
    if (!form.password) {
      newErrors.password = 'Enter your password';
    } else {
      if (form.password.length < 8) {
        newErrors.password = 'Password must be at least 8
characters long';
      } else if (!/[a-z]/.test(form.password)) {
        newErrors.password = 'Password must contain at least one
lowercase letter';
      } else if (!/[A-Z]/.test(form.password)) {
        newErrors.password = 'Password must contain at least one
uppercase letter';
      } else if (!/[0-9]/.test(form.password)) {
        newErrors.password = 'Password must contain at least one
digit';
      } else if (!/[!@#$%^&*(),.?":{}|<>]/.test(form.password)) {
        newErrors.password = 'Password must contain at least one
special character';
```

```
            } else if (/(.)\1\1/.test(form.password)) {
              newErrors.password = 'Password cannot contain the same
character 3 or more times in a row';
            }
          }

          if (form.repeatPassword !== form.password) {
            newErrors.repeatPassword = 'Passwords don`t match';
          }

          setErrors(newErrors);
          return Object.keys(newErrors).length === 0;
        };

        const handleSubmit = async () => {
          setServerError('');
          if (validate()) {
            setLoading(true);
            try {
              const response = await
fetch('http://localhost:3005/register', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({
                  name: form.name,
                  email: form.email,
                  phone: form.phone,
                  password: form.password,
                }),
              });

              const data = await response.json();

              if (!response.ok) {
                setServerError(data.message || 'Error of connection');
              } else {
                alert(data.message || 'Registration completed
successfully!');
                navigate('/emailconfirmationletter');
              }
            } catch (error) {
              setServerError('Error connecting to the server');
            } finally {
              setLoading(false);
            }
          }
        };

        return (
          <>
            <Meta name="viewport" content="width=device-width, initial-
scale=1.0" />

            <div className="registration-background">
              <div className="registration-container">
                <h2 className="registration-title">
```

```
                  Speedster <IoSpeedometerOutline style={{ fontSize:
'28px', color: 'white' }} />
                </h2>

                <label className="registration-label">Name</label>
                <input
                  type="text"
                  name="name"
                  className="registration-input"
                  value={form.name}
                  onChange={handleChange}
                  disabled={loading}
                />
                {errors.name && <div className="error-
text">{errors.name}</div>}

                <label className="registration-label">E-mail</label>
                <input
                  type="email"
                  name="email"
                  className="registration-input"
                  value={form.email}
                  onChange={handleChange}
                  disabled={loading}
                />
                {errors.email && <div className="error-
text">{errors.email}</div>}

                <label className="registration-label">Phone
number</label>
                <input
                  type="tel"
                  name="phone"
                  className="registration-input"
                  value={form.phone}
                  onChange={handleChange}
                  disabled={loading}
                />
                {errors.phone && <div className="error-
text">{errors.phone}</div>}

                <label className="registration-label">Password</label>
                <input
                  type="password"
                  name="password"
                  className="registration-input"
                  value={form.password}
                  onChange={handleChange}
                  disabled={loading}
                />
                {errors.password && <div className="error-
text">{errors.password}</div>}

                <label className="registration-label">Repeat
password</label>
                <input
```

```
                  type="password"
                  name="repeatPassword"
                  className="registration-input"
                  value={form.repeatPassword}
                  onChange={handleChange}
                  disabled={loading}
                />
                {errors.repeatPassword && <div className="error-
text">{errors.repeatPassword}</div>}

                {serverError && <div className="error-text" style={{
marginBottom: '10px' }}>{serverError}</div>}

                <button
                  className="registration-button"
                  onClick={handleSubmit}
                  disabled={loading}
                >
                  {loading ? 'Submitting...' : 'Complete registration'}
                </button>
              </div>
            </div>
          </>
        );
      };

      export default RegisterPage;
```

Лістинг коду компонента LoginPage.jsx

```
import React, { useState } from "react";
import "../../styles/LoginPage/LoginPage.css";
import { IoSpeedometerOutline } from "react-icons/io5";
import { Meta } from 'react-head';
import { Link, useNavigate } from 'react-router-dom';

const LoginForm = () => {
  const navigate = useNavigate();

  const [form, setForm] = useState({ login: '', password: '' });
  const [error, setError] = useState('');

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleLogin = async () => {
    setError('');

    try {
      const response = await fetch('http://localhost:3005/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          login: form.login,
          password: form.password,
        }),
      });

      const result = await response.json();

      if (response.ok) {
        navigate('/dashboard');
      } else {
        setError(result.message || 'Login failed');
      }
    } catch (err) {
      setError('Server error');
      console.error('Login error:', err);
    }
  };

  return (
    <>
      <Meta name="viewport" content="width=device-width, initial-
scale=1.0" />
      <div className="login-background">
        <div className="login-container">
          <h2 className="login-title">
            Speedster <IoSpeedometerOutline style={{ fontSize:
'28px', color: 'white' }} />
```

```
                </h2>

                <label className="login-label">Login (e-mail/phone
number)</label>
                <input
                  type="text"
                  name="login"
                  className="login-input"
                  value={form.login}
                  onChange={handleChange}
                />

                <label className="login-label">Password</label>
                <input
                  type="password"
                  name="password"
                  className="login-input"
                  value={form.password}
                  onChange={handleChange}
                />

                {error && <div className="error-text">{error}</div>}

                <div className="login-links">
                  <Link to="/register">Not registered?</Link>
                  <Link to="/passwordrecovery">Forgot password?</Link>
                </div>

                <button className="login-button" onClick={handleLogin}>
                  Login
                </button>
              </div>
            </div>
          </>
        );
      };

      export default LoginForm;
```

Лістинг коду компонента DashboardLayout.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { FaHome } from 'react-icons/fa';
import { IoSpeedometerOutline } from 'react-icons/io5';
import { FaTelegram } from "react-icons/fa6";
import '../../styles/MainPage/MainPageLayout.css';

const DashboardLayout = ({ children }) => {
  return (
    <div className="main-page-container">
      <div className="side-panel">
        <div className="side-panel-header">
          <span>Speedster</span>
          <IoSpeedometerOutline style={{ fontSize: '32px', color:
'white' }} />
        </div>

        <nav className="side-panel-nav">
          <div className="side-panel-nav-item">
            <FaHome className="side-panel-icon" />
            <Link to="/">Home</Link>
          </div>

          <div className="side-panel-nav-item">
            <FaTelegram className="side-panel-icon" />
            <a href='https://t.me/Sp33dsterBot'>Telegram Bot</a>
          </div>
        </nav>
      </div>
      <div className="background">{children}</div>
    </div>
  );
};

export default DashboardLayout;
```

Лістинг коду моделі користувача User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');

const userSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  phone: String,
  password: String,
  isVerified: { type: Boolean, default: false },
  verificationToken: String,
  resetToken: String,
  resetTokenExpiry: Date,
  activeTokens: [String],
}, { timestamps: true });


userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  try {
    const salt = await bcrypt.genSalt(10);
    const hashed = await bcrypt.hash(this.password, salt);
    this.password = hashed;
    next();
  } catch (err) {
    next(err);
  }
});

module.exports = mongoose.model('User', userSchema);
```

Лістинг коду серверного файлу server.js

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bcrypt = require('bcrypt');
const nodemailer = require('nodemailer');
const crypto = require('crypto');
const jwt = require('jsonwebtoken');
const fetch = (...args) => import('node-fetch').then(({ default:
fetch }) => fetch(...args));
require('dotenv').config();

const User = require('./src/models/User.js');

const app = express();
const port = process.env.PORT || 3005;

const JWT_SECRET = process.env.JWT_SECRET;
const TELEGRAM_BOT_TOKEN = process.env.TELEGRAM_BOT_TOKEN;
const TELEGRAM_CHAT_ID = process.env.TELEGRAM_CHAT_ID;

mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => console.log('✅ Connected to MongoDB'))
  .catch(err => console.error('❌ MongoDB connection error:',
err));

app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: false }));

const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader?.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'Access token
missing' });

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    const user = await User.findById(decoded.id);
    if (!user || !user.activeTokens?.includes(token)) {
      return res.status(403).json({ message: 'Invalid or revoked
token' });
    }

    req.user = decoded;
    req.token = token;
    next();
  } catch (err) {
    return res.status(403).json({ message: 'Invalid or expired
token' });
  }
};
```

```
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS,
  },
});

// 🚨 Telegram alert
app.post('/api/alert', async (req, res) => {
  const { speed } = req.body;
  if (!speed) return res.status(400).json({ error: 'Speed is
required' });

  const message = `🚨 Speed limit exceeded! Speed:
${speed.toFixed(2)} kph`;

  try {
    const response = await
fetch(`https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ chat_id: TELEGRAM_CHAT_ID, text:
message }),
    });

    if (!response.ok) throw new Error('Telegram API error');
    res.status(200).json({ message: 'Alert sent to Telegram' });
  } catch (err) {
    console.error('Telegram alert error:', err);
    res.status(500).json({ message: 'Failed to send alert to
Telegram' });
  }
});

// 📝 Register
app.post('/register', async (req, res) => {
  try {
    const { name, email, phone, password } = req.body;

    const existingUser = await User.findOne({ email });
    if (existingUser) return res.status(400).json({ message: 'User
already exists' });

    const verificationToken =
crypto.randomBytes(32).toString('hex');

    const user = new User({
      name,
      email,
      phone,
      password,
      verificationToken,
    });
```

```
          await user.save();

          await transporter.sendMail({
            from: '"Speedster" <noreplysp33dster@gmail.com>',
            to: email,
            subject: 'Confirm your email',
            html: `
              <h3>Hello, ${name}!</h3>
              <p>Click the link to confirm your email:</p>
              <a
href="http://localhost:3005/confirm/${verificationToken}">Confirm
Email</a>
            `,
          });

          res.status(201).json({ message: 'Registration successful. Check
your email.' });
        } catch (err) {
          console.error('Registration error:', err);
          res.status(500).json({ message: 'Registration error' });
        }
      });

      // ✅ Confirm email
      app.get('/confirm/:token', async (req, res) => {
        try {
          const user = await User.findOne({ verificationToken:
req.params.token });
          if (!user) return res.redirect('http://localhost:3006/invalid-
token');

          user.isVerified = true;
          user.verificationToken = undefined;
          await user.save();

          res.redirect('http://localhost:3006/email-confirmed');
        } catch (err) {
          console.error(err);
          res.redirect('http://localhost:3006/error');
        }
      });

      // 🔐 Login
      app.post('/login', async (req, res) => {
        try {
          const { login, password } = req.body;

          const user = await User.findOne({ $or: [{ email: login }, {
phone: login }] });
          if (!user) return res.status(404).json({ message: 'User not
found' });
          if (!user.isVerified) return res.status(403).json({ message:
'Email not verified' });

          const match = await bcrypt.compare(password, user.password);
```

```
        if (!match) return res.status(401).json({ message: 'Incorrect
password' });

        const token = jwt.sign({ id: user._id, email: user.email },
JWT_SECRET, { expiresIn: '2h' });

        user.activeTokens = user.activeTokens || [];
        user.activeTokens.push(token);
        await user.save();

        res.status(200).json({ message: 'Login successful', token });
      } catch (err) {
        console.error(err);
        res.status(500).json({ message: 'Login error' });
      }
    });

    // 🚪 Logout
    app.post('/logout', authenticateToken, async (req, res) => {
      try {
        await User.findByIdAndUpdate(req.user.id, {
          $pull: { activeTokens: req.token },
        });
        res.status(200).json({ message: 'Logged out successfully' });
      } catch (err) {
        res.status(500).json({ message: 'Logout failed' });
      }
    });

    // 👤 Get profile
    app.get('/api/profile', authenticateToken, async (req, res) => {
      try {
        const user = await User.findById(req.user.id).select('-
password');
        if (!user) return res.status(404).json({ message: 'User not
found' });
        res.status(200).json(user);
      } catch (err) {
        res.status(500).json({ message: 'Error fetching profile' });
      }
    });

    // 🔄 Forgot password
    app.post('/forgot-password', async (req, res) => {
      const { email } = req.body;
      const user = await User.findOne({ email });
      if (!user) return res.status(404).json({ message: 'No user with
that email' });

      const token = crypto.randomBytes(32).toString('hex');
      user.resetToken = token;
      user.resetTokenExpiry = Date.now() + 15 * 60 * 1000;
      await user.save();

      const resetLink = `http://localhost:3005/reset-
password/${token}`;
```

```
        try {
          await transporter.sendMail({
            from: '"Speedster" <noreplysp33dster@gmail.com>',
            to: user.email,
            subject: 'Reset your password',
            html: `<p>Hello ${user.name},</p><p>Reset link (valid 15
min): <a href="${resetLink}">${resetLink}</a></p>`,
          });

          res.status(200).json({ message: 'Reset link sent' });
        } catch (err) {
          console.error('Email error:', err);
          res.status(500).json({ message: 'Failed to send reset email'
});
        }
      });

      // ✅ Redirect to frontend reset page
      app.get('/reset-password/:token', (req, res) => {
        const { token } = req.params;
        res.redirect(`http://localhost:3006/reset-password/${token}`);
      });

      // ✅ Reset password logic
      app.post('/reset-password/:token', async (req, res) => {
        const { token } = req.params;
        const { password } = req.body;

        try {
          const user = await User.findOne({ resetToken: token,
resetTokenExpiry: { $gt: Date.now() } });
          if (!user) return res.status(400).json({ message: 'Invalid or
expired token' });

          user.password = password;
          user.resetToken = undefined;
          user.resetTokenExpiry = undefined;
          await user.save();

          res.status(200).json({ message: 'Password updated successfully'
});
        } catch (err) {
          console.error('Reset error:', err);
          res.status(500).json({ message: 'Error resetting password' });
        }
      });

      app.listen(port, () => {
        console.log(`🚀 Server running at http://localhost:${port}`);
      });
```

Лістинг коду файлу main.py

```
import telebot
from telebot import types
import webbrowser
bot = telebot.TeleBot('8112834276:AAFbAJrvTBHEabWb-
pL7Oztum8YkikUTLNI')
@bot.message_handler(commands=['start'])
def main(message):
    bot.send_message(
        message.chat.id,
        f'Hello, {message.from_user.username}! This is your
assistant Speedster for controlling your speed on streets. How can I
help?'
    )
    markup = types.InlineKeyboardMarkup()
    markup.add(types.InlineKeyboardButton('Go on Speedster
website', url='https://speedster-4sd3.onrender.com'))
    markup.add(types.InlineKeyboardButton('Description',
callback_data='description'))
    bot.send_message(message.chat.id, "Choose an option:",
reply_markup=markup)

@bot.message_handler(commands=['description'])
def description(message):
    bot.send_message(
        message.chat.id,
        'Speedster is a bot whose main goal is to help drivers
control their speed while driving through city streets. All you need is
to be registered on the main web application Speedster.'
    )
    markup = types.InlineKeyboardMarkup()
    markup.add(types.InlineKeyboardButton('Go on Speedster
website', url='https://speedster-4sd3.onrender.com'))
    markup.add(types.InlineKeyboardButton('Description',
callback_data='description'))
    bot.send_message(message.chat.id, "Choose an option:",
reply_markup=markup)

@bot.message_handler(commands=['site', 'website'])
def site(message):
    webbrowser.open('https://speedster-4sd3.onrender.com')

@bot.callback_query_handler(func=lambda call: call.data ==
'description')
def callback_description(call):
    description(call.message)

bot.polling(none_stop=True)
```