ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку «Автомобільний менеджер» за допомогою фреймворку Vue.js»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав КУЙДІН
(підпис)

Виконав: здобувач(ка) вищої освіти групи ПД-43

Владислав КУЙДІН

Керівник: Владислав ЯСКЕВИЧ
кандидат технічних наук

Рецензент:

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Куйдіну Владиславу Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку «Автомобільний менеджер» за допомогою фреймворку Vue.js»
керівник кваліфікаційної роботи к.т.н., доцент Владислав Яскевич,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про облік обслуговування автомобілів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій для обліку обслуговування автомобілів.
2. Проектування застосунку для удосконалення обліку обслуговування автомобілів.
3. Програмна реалізація та опис функціонування застосунку для обліку обслуговування автомобілів.
4. Тестування Web-застосунку для обліку обслуговування автомобілів.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Блок-схема роботи програми.
6. Діаграма класів Web-застосунку.
7. Екранні форми.
8. Відео-демонстрація роботи застосунку.
9. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих методів обліку обслуговування автомобілів	14.03-16.05.2024	
4	Проектування застосунку для удосконалення обліку обслуговування автомобілів	17.05-19.05.2024	
5	Програмна реалізація та опис функціонування застосунку для обліку обслуговування автомобілів	19.05-31.05.2024	
6	Тестування Web-застосунку для удосконалення обліку обслуговування автомобілів	19.05-31.05.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач(ка) вищої освіти _____ Владислав КУЙДІН
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 1 табл., 27 рис., 11 джерел.

Мета роботи – удосконалення обліку обслуговування автомобілів за рахунок Web-застосунку “Автомобільний менеджер” з використанням фреймворку Vue.js.

Об’єкт дослідження – процес обліку обслуговування автомобілів.

Предмет дослідження – Web-застосунок для обліку обслуговування автомобілів.

Короткий зміст роботи: В роботі проаналізовано потреби в інструментах для обліку обслуговування автомобілів. Проаналізовано інструментальні засоби для обліку обслуговування автомобілів: Simply Auto: Car Maintenance, My Car – пальне, My Car Service - Керування автомобілем. Розроблено алгоритм роботи Web-застосунку та програмно реалізовані ключові функціональні можливості, зокрема: авторизація, додавання автомобілів, додавання задач, підказки по обслуговуванню на основі виконаних задач. Проведено тестування Web-застосунку. В роботі використано середовище розробки Visual Studio Code, мову програмування JavaScript, фреймворк Vue.js, базу даних Google Firebase, інструмент створення прототипів Figma.

Сферою використання застосунку є облік обслуговування автомобілів.

КЛЮЧОВІ СЛОВА: ОБЛІК ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ, JAVASCRIPT, VUE, VUEX, VUE ROUTER, FIREBASE, WEB.

ЗМІСТ

ВСТУП

Проблема наявності інструментів для ефективного менеджменту обслуговування автомобілів завжди була актуальною, але зростання кількості автомобілів та збільшення потреби у підтримці їх у належному стані зробило її ще більш нагальною. Це спричинило необхідність у створенні зручних та функціональних рішень для власників автомобілів. Існуючі інструменти, такі як Simply Auto: Car Maintenance, My Car – пальне та My Car Service - Керування автомобілем, пропонують широкий спектр функцій, багато з яких є зайвими для користувача. Для власників автомобілів першочерговим є задоволення потреб у менеджменті обслуговування деталей, рідин та інших ключових аспектів експлуатації автомобіля.

Розроблюване програмне забезпечення "Автомобільний менеджер" має на меті спростити процес організації обслуговування автомобілів, забезпечивши користувачів зручним інтерфейсом для додавання, видалення та зміни даних про автомобілі, ведення історії всіх виконаних та запланованих завдань, а також підказками для майбутніх обслуговувань. Наприклад, якщо в історії буде зазначено, що було замінено мастило, застосунок автоматично нагадає, що через умовно 8 тисяч км потрібно буде знову замінити мастило.

Таким чином, "Автомобільний менеджер" спеціалізується на конкретних аспектах менеджменту обслуговування автомобілів, таких як заміна, встановлення, лагодження деталей. На відміну від багатьох існуючих додатків, що пропонують широкий спектр функцій, не всі з яких є необхідними для користувача, "Автомобільний менеджер" зосереджений на покритті основних потреб автовласників.

Об’єкт дослідження – процес обліку обслуговування автомобілів.

Предмет дослідження – Web-застосунок для обліку обслуговування автомобілів.

Мета роботи – удосконалення обліку обслуговування автомобілів за рахунок Web-застосунку “Автомобільний менеджер” з використанням фреймворку Vue.js.

Завдання роботи:

- Проаналізувати та порівняти існуючі рішення менеджменту обслуговування автомобілів.
- Скласти вимоги до розроблюваного програмного забезпечення.
- Вибрати інструменти розробки програмного забезпечення.
- Визначити архітектуру та структуру проекту.
- Створити дизайн інтерфейсу користувача.
- Розробити застосунок.
- Протестувати програмне забезпечення.

Результати виконаної роботи можуть вільно використовуватися навчальними закладами, фахівцями, та окремим особами для вирішення перерахованих проблем.

Деякі результати дослідження доповідались на студентських наукових конференціях: V Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT» (м.Київ. 2024); Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ» (м.Київ. 2024). Опубліковано у матеріалах конференцій [398].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ СИСТЕМИ ОБЛІКУ ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

1.1 Опис задач

Менеджмент автомобілів – це комплексна система управління, що охоплює всі аспекти експлуатації, технічного обслуговування та ремонту автотранспортних засобів. Ця система передбачає планування, організацію, контроль та аналіз всіх процесів, пов'язаних з використанням автомобілів. Основною метою менеджменту автомобілів є забезпечення безпеки, ефективності та тривалості експлуатації транспортних засобів.

Основні завдання менеджменту автомобілів:

- Планування технічного обслуговування:

Регулярне технічне обслуговування автомобілів є критично важливим для підтримки їх у належному технічному стані. Планування включає визначення графіку заміни мастил, перевірок гальм, огляду шин та інших важливих компонентів.

1. Організація ремонтних робіт:

Менеджмент автомобілів передбачає організацію та координацію всіх необхідних ремонтних робіт. Це може включати як планові ремонти, так і непередбачені роботи, що виникають у процесі експлуатації.

- Моніторинг експлуатаційних показників:

Система менеджменту автомобілів включає моніторинг таких показників, як пробіг, частота ремонтів та інші параметри, що дозволяють оцінити ефективність використання транспортних засобів.

- Ведення облікової документації:

Всі операції, пов'язані з технічним обслуговуванням та ремонтом автомобілів, повинні бути задокументовані. Це дозволяє мати повну історію

обслуговування кожного автомобіля. [1]

Переваги менеджменту автомобілів:

- Збільшення рівня безпеки:

Регулярне технічне обслуговування та своєчасний ремонт значно знижують ризик виникнення аварійних ситуацій.

- Зниження витрат:

При регулярному обслуговуванні автомобіля знижуються витрати на його експлуатацію, оскільки своєчасне виявлення та усунення несправностей допомагає уникнути великих ремонтів

- Збільшення терміну експлуатації автомобіля:

Своєчасне технічне обслуговування та ремонт допомагають зберегти автомобілі в належному технічному стані, що продовжує їх термін служби.

Використання сучасних технологій:

- Інтернет речей (IoT):

Використання IoT технологій дозволяє відстежувати стан автомобілів у режимі реального часу, отримуючи дані з датчиків про стан різних систем автомобіля.

- Блокчейн технології:

Використання блокчейн технологій може забезпечити прозорість та безпеку облікової документації, а також захистити дані про історію обслуговування автомобілів від фальсифікації, наприклад, при покупці або продажі транспортного засобу, гарантуючи достовірність наданої інформації про його стан та обслуговування.

Для ефективного управління обслуговуванням автомобілів за допомогою застосунку необхідно мати доступ до різноманітного функціоналу, який включає такі можливості:

1) перегляд списку автомобілів;
2) планування розкладу ремонтних робіт;
3) заповнення та перегляд історії обслуговування;

4) отримання підказок про майбутні технічні роботи.

Наразі існують застосунки під ключ, які значно полегшують процес управління обслуговуванням автомобілів для їх власників. Такі застосунки, як MyMazda, Mercedes me, FordPass, HondaLink та BMW ConnectedDrive, забезпечують доступ до широкого спектру функцій. Ці функції включають можливість відстеження графіку технічного обслуговування, запису на сервісні роботи, отримання оповіщень про необхідність технічного обслуговування, перегляд історії обслуговування автомобіля, а також віддалене керування функціями автомобіля, такими як запуск двигуна та блокування/розблокування дверей. Завдяки цим застосункам, власники автомобілів можуть зручно керувати своїм транспортним засобом, забезпечуючи його надійну роботу та знижуючи витрати на обслуговування.

Враховуючи, що не всі автомобілі маю власні додатки для управління технічним обслуговуванням та ремонтом, було прийнято рішення про створення універсального застосунку, який буде мати основні функції для менеджменту обслуговування забезпечити зручне управління для власників автомобілів різних марок і моделей.

Цільовою аудиторією універсального застосунку для управління автомобілями є широкий спектр користувачів, включаючи власників автомобілів, менеджерів автопарків, автосервісів, а також інших осіб, які займаються управлінням транспортними засобами.

Опис користувача як власника автомобіля: від 18 років і старше, скоріше за все має базові знання про технічне обслуговування автомобілів. Бажання покращити управління своїм транспортним засобом, знижуючи витрати на обслуговування та покращуючи безпеку.

Середній вік власників автомобілів сьогодні варіюється залежно від конкретної демографічної групи. Наприклад, у програмі «Date My Car» учасниками є, як правило, жінки віком від 20 до 35 років і чоловіки старше 20 років. З іншого боку, дослідження водіїв похилого віку показало, що популяція літніх водіїв зростає, у Великобританії водійські права мають понад 15 мільйонів

людей старше 60 років [2].

Оскільки застосунок розробляється з максимально простим інтерфейсом, додаткове навчання може обмежитися 1 відео-інструкцією.

Розробка даного програмного забезпечення відбувається у декілька етапів:

- розробка прототипу інтерфейсу користувача
- розробка Web-застосунку
- тестування програмного забезпечення

Для розробки такої системи необхідно ретельно підібрати інструменти. Необхідно вибрати платформу для створення прототипу, базу даних, середовище розробки для застосунку, мову програмування, хостинг та інше. Детально про вибрані інструменти розробки у підрозділі 3 цього розділу.

1.2 Огляд аналогів

Серед аналогів програмного забезпечення для організації дистанційного навчання безперечно необхідно розглянути Simply Auto: Car Maintenance, My Car – пальне, My Car Service - Керування автомобілем.

1.2.1 Simply Auto: Car Maintenance

Simply Auto: Car Maintenance – це зручний застосунок для управління та відстеження технічного обслуговування вашого автомобіля. Він допомагає користувачам зберігати важливу інформацію про автомобіль, включаючи витрати на паливо, обслуговування та ремонти.

Реалізована цей застосунок у 3-х варіантах:

- Android застосунок;
- IOS застосунок;
- Web-застосунок(бета-версія).

Основні функції застосунку:

- Можливість записувати витрати на паливо, щоб стежити за ефективністю використання пального вашого автомобіля.
- Застосунок нагадує про заплановані технічні обслуговування, такі як

заміна масла, шин, акумулятора тощо.

- Всі записи про обслуговування та ремонти зберігаються в одному місці, що полегшує доступ до них.
- Дані можуть бути синхронізовані між різними пристроями, що дозволяє мати доступ до них з будь-якого пристрою.
- Можливість експортувати дані у форматі PDF або Excel для аналізу або надання сервісному центру.
- Застосунок допомагає відстежувати всі витрати, пов'язані з автомобілем, включаючи страховку, кредити та інші витрати.

Недоліки застосунку(базовано на коментарях та власному досвіді використання):

- Застосунок має безліч функцій, що може ускладнити його використання для нових користувачів. На початку може бути важко зрозуміти, як користуватися всіма можливостями.
- Величезна кількість опцій та налаштувань може створити перевантаження інтерфейсу, що ускладнює навігацію та пошук потрібної функції.
- У деяких випадках користувачі стикаються з проблемами синхронізації даних між різними пристроями, що може викликати плутанину або втрату інформації.[3]

Враховуючи ці недоліки, користувачі можуть зіткнутися з певними труднощами під час використання застосунку Simply Auto: Car Maintenance. Проте, для багатьох користувачів його переваги можуть переважити ці недоліки, особливо якщо вони готові витратити трохи більше часу на освоєння застосунку.

Екран з наявними автомобілями у застосунку Simply Auto: Car Maintenance представлений на рисунку 1.1. Панель приладів представлена на рисунку 1.2.

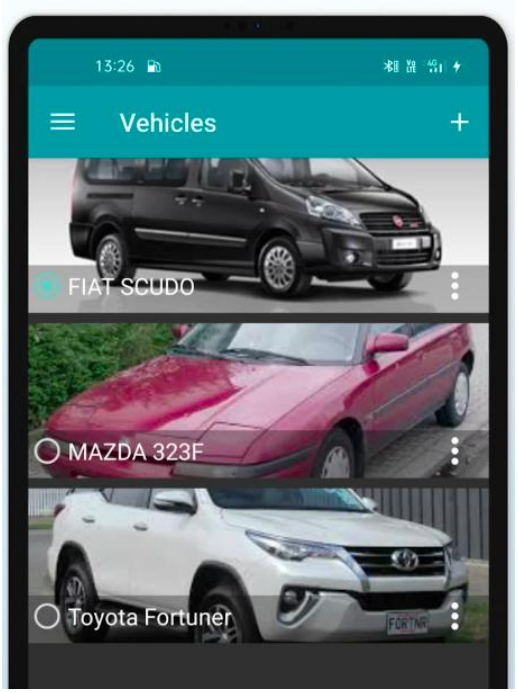


Рис.1.1. Наявний транспорт Simply Auto: Car Maintenance

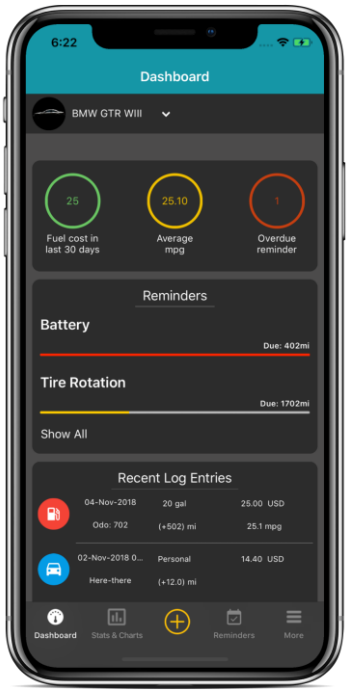


Рис.1.2. Панель приладів Simply Auto: Car Maintenance

1.2.2 My Car – пальне

My Car – Пальне - це зручний застосунок для відстеження витрат на пальне та управління технічним обслуговуванням вашого автомобіля. Реалізована ця система у 3-х варіантах:

- Android застосунок;
- IOS застосунок;
- Web-застосунок.

Відстеження витрат на пальне:

- Легкий запис та аналіз витрат на пальне, щоб краще розуміти витрати на використання автомобіля.
- Зберігання інформації про кількість заправленого пального, вартість та пробіг автомобіля на момент заправки.

Розрахунок витрат пального:

- Автоматичне розрахування середньої витрати пального на кілометр або милю.
- Аналіз ефективності використання пального за різні періоди.

Журнали заправок:

- Зберігання історії заправок з інформацією про дату, місце, кількість пального та вартість.
- Записи про різні види пального, якщо використовується більше одного автомобіля або види пального.

Нагадування про обслуговування:

- Нагадування про необхідність проведення технічного обслуговування автомобіля, включаючи заміну масла, шин, фільтрів тощо.
- Встановлення інтервалів для обслуговування та отримання сповіщення про наближення дати обслуговування.

Звіти та графіки:

- Генерація детальних звітів та графіків про витрати на пальне та технічне обслуговування, щоб мати повну картину про автомобіль.
- Аналіз витрат та використання автомобіля.

Синхронізація даних:

- Синхронізація даних між різними пристроями для доступу до інформації з

будь-якого місця.

- Забезпечення надійного зберігання даних у хмарі для запобігання їх втраті.

Додаткові можливості:

- Імпорт та експорт даних у форматах CSV для подальшого аналізу або резервного копіювання.
- Відстеження витрат на пальне та обслуговування для кількох автомобілів в одному застосунку.
- Розрахунок вартості поїздок[4]

Враховуючи ці недоліки, користувачі можуть зіткнутися з певними труднощами під час використання застосунку My Car – пальне. Проте, для багатьох користувачів його переваги можуть переважити ці недоліки, особливо якщо вони готові приділити трохи більше часу для освоєння застосунку.

Екран історії у застосунку My Car – Пальне представлена на рисунку 1.3, а на рисунку 1.4 відображено статистику витрат пального.

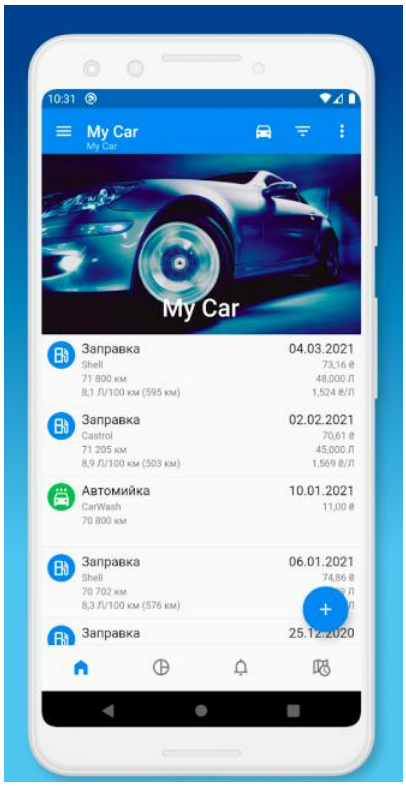


Рис.1.3. Історія My Car – Пальне

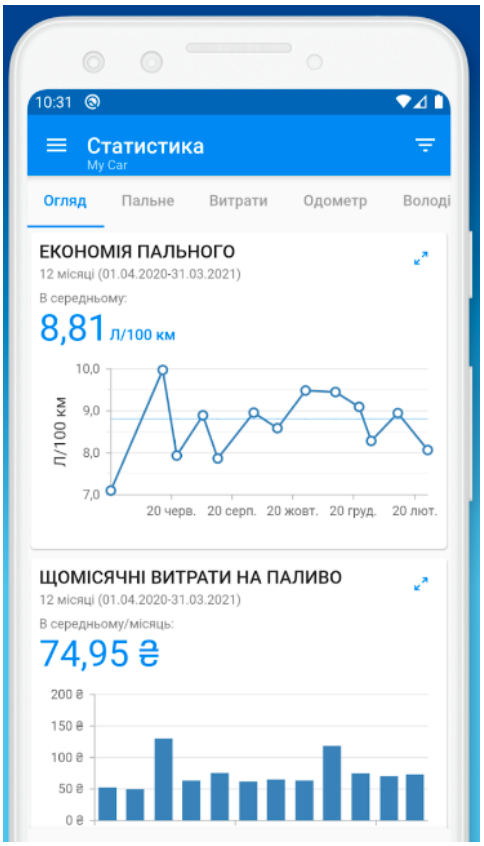


Рис.1.4. Статистика My Car – Пальне

1.2.3 My Car Service - Керування автомобілем

My Car Service - Car Manage - це потужний інструмент для управління технічним обслуговуванням та витратами на автомобіль, що допомагає водіям бути в курсі всіх важливих аспектів обслуговування їхнього транспорту.

Реалізована ця система у 3-х варіантах:

- Android застосунок;
- IOS застосунок;

Основні функції:

Управління технічним обслуговуванням:

- Відстеження подій технічного обслуговування, такі як заміна масла, шин, фільтрів, гальмівних колодок і так далі.
- Зберігання історії про обслуговування автомобіля з датами, пробігом, типом робіт та вартістю.
- Керування кількома автомобілями одночасно в одному застосунку.

Нагадування про обслуговування:

- Автоматичні нагадування про майбутні події технічного обслуговування, щоб не пропустити жодного важливого заходу.

Витрати на автомобіль:

- Відстеження витрат на автомобіль, включаючи пальне, технічне обслуговування, страхування, податки та інші витрати.
- Генерування звітів про витрати на автомобіль та їх аналіз, щоб мати повну картину витрат і економічності використання автомобіля.

Синхронізація з хмарою:

- Синхронізація всіх даних між різними пристроями, забезпечуючи надійне зберігання та доступ до них з будь-якого місця.

Сканування документів:

- Зберігання копій важливих документів, таких як квитанції, страхові поліси та інших документів, за допомогою функції сканування.[5]

Приклади екранів застосунку My Car Service - Car Manage представлені на рисунках 1.5 та 1.6.

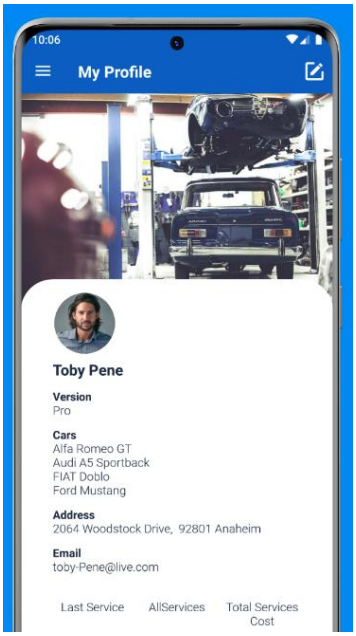


Рис.1.5. Профіль My Car Service

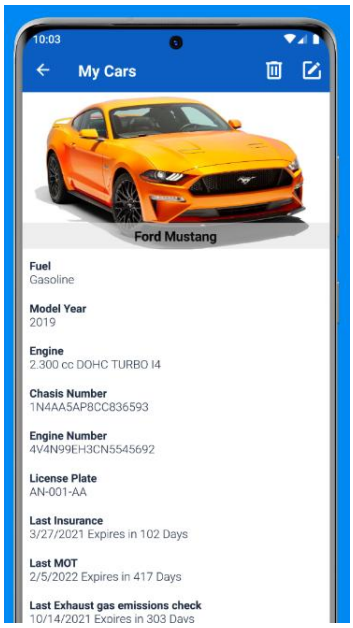


Рис.1.6. Інформація про автомобіль My Car Service

1.2.4 Порівняння

Для визначення переваг, недоліків та ключових можливостей проведено порівняння застосунків Simply Auto: Car Maintenance, My Car – пальне, My Car Service - Керування автомобілем та розроблюваного застосунку – Автомобільний менеджер. Результати порівняння продемонстровано у таблиці 1.1.

Таблиця 1.1

Порівняння існуючих аналогів

Характеристики	Simply Auto: Car Maintenance	My Car – пальне	My Car Service - Керування автомобілем	Автомобільний менеджер
Платформи	ОС Android, IOS, Web-застосунок(beta)	ОС Android, IOS, Windows, Web-застосунок	IOS та Android	Web-застосунок

Продовження таблиці 1.1

Порівняння існуючих аналогів

Характеристики	Simply Auto: Car Maintenance	My Car – пальне	My Car Service - Керування автомобілем	Автомобільний менеджер
Додавання/редагування/ видалення автомобілів	+	+	+	+
Підказки по обслуговуванню	+	-	-	+
Синхронізація даних	+	+	+	+
Автентифікація	+	+	+	+
Націлення на звужений спектр послуг	-	-	-	+

2 ПРОЕКТУВАННЯ ТА ВИЗНАЧЕННЯ ВИМОГ ДО СИСТЕМИ ОБЛІКУ ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

2.1 Вимоги до системи обліку обслуговування автомобілів

У результаті аналізу предметної області та існуючих рішень було сформульовано функціональні та нефункціональні вимоги. Функціональні вимоги визначають мінімальний набір функцій розроблюваного застосунку, які повинні задовольняти потреби користувачів. Нефункціональні вимоги описують характеристики програмного забезпечення, які не можуть бути представлені у вигляді функцій, але є важливими загальними властивостями.

Функціональні вимоги:

1) Користувач повинен мати можливість увійти в систему, використовуючи свої дані для входу за допомогою Firebase Auth.
2) Користувач повинен мати можливість додавати, редагувати та видаляти інформацію про транспортний засіб.
3) Користувач повинен мати можливість додавати, редагувати та видаляти інформацію про задачі, які потрібно виконати або були виконані в транспортному засобі.
4) Система враховує дані про виконані задачі для генерації підказок для заміни деталей(рідин, фільтрів) після виконання подібної задачі.

Нефункціональні вимоги включають:

1) Інтерфейс користувача за стандартами проектування Material design.
2) Захист від несанкціонованого доступу за допомогою технології Firebase Auth.
3) Локалізація українською мовою.

Для кращого розуміння функціональних вимог на рисунку 2.1 представлено діаграму варіантів використання. Ця діаграма ілюструє варіанти використання застосунку, взаємодію користувачів із серверною частиною та самим

застосунком. На рисунку 2.2 показана блок-схема роботи програми, яка допомагає зрозуміти потік роботи системи. Діаграма демонструє, як одна дія переходить в іншу.

Рис.2.1. Діаграма використання

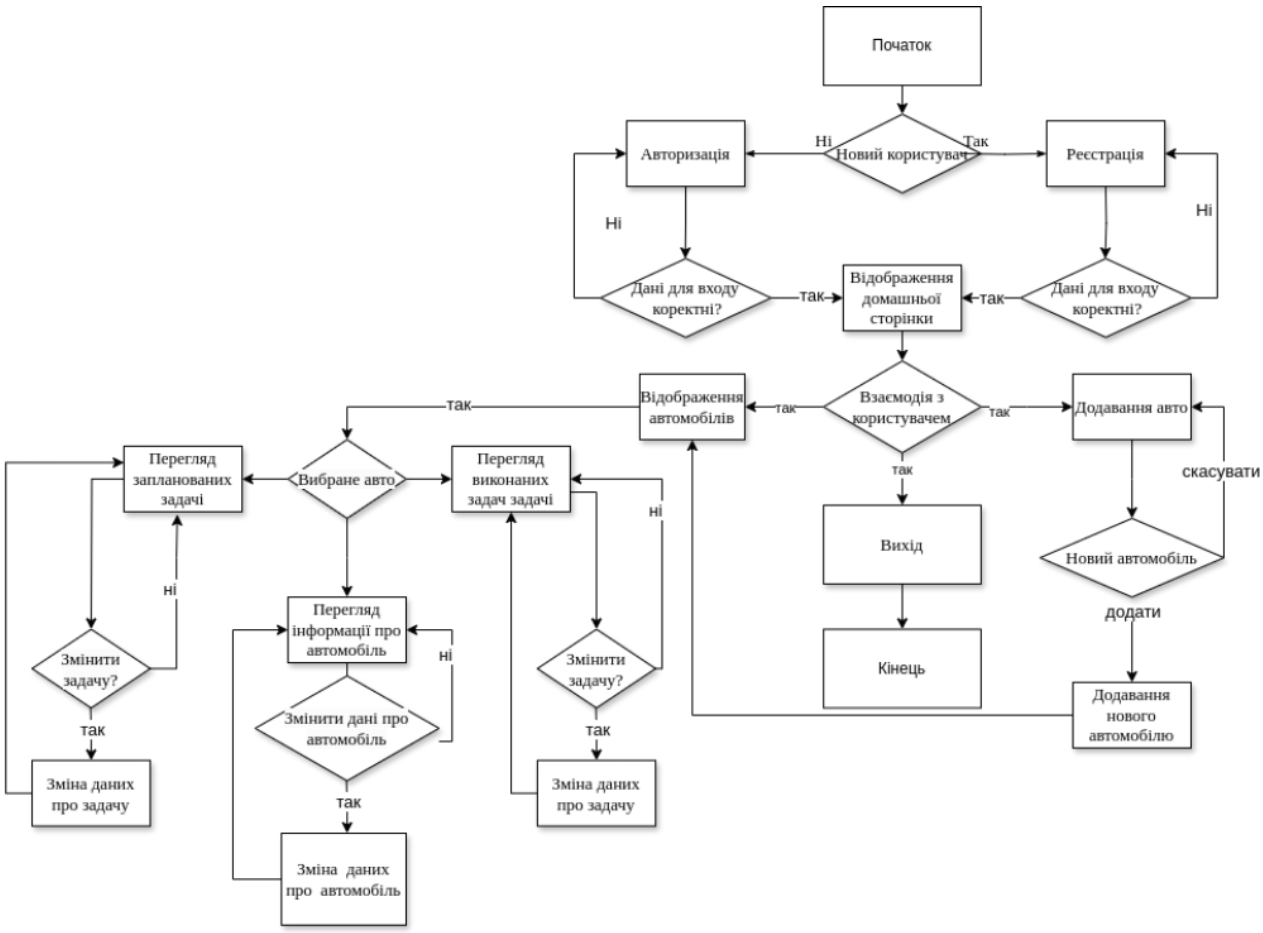


Рис.2.2. Блок-схема роботи програми

2.2 Проектування архітектури системи обліку обслуговування автомобілів

Будь-який програмний продукт повинен мати чітко визначену архітектуру. Архітектура програмного продукту – це структура системи, яка охоплює її компоненти, взаємозв'язки між ними та принципи їхнього проектування та розвитку. Це своєрідний план або каркас, на якому будується програмне забезпечення, і який визначає, як різні частини системи взаємодіють одна з одною. Основні складові архітектури програмного продукту включають:

- компоненти як окремі частини системи, які виконують певні функції. Це можуть бути модулі, сервіси, класи, бібліотеки тощо;

- способи взаємозв'язків, якими компоненти взаємодіють і обмінюються даними;
- правила та шаблони, які використовуються для створення архітектури(парадигми програмування, управління даними);
- середовище, в якому працює програмне забезпечення, включаючи сервери, мережеві компоненти, бази даних та інші рішення;

Архітектура програмного продукту – це критично важливий елемент процесу розробки програмного забезпечення, який забезпечує структурованість, якість, безпеку та ефективність системи. Вона допомагає планувати, організовувати і контролювати процес розробки, забезпечуючи можливість масштабування та адаптації до змін, а також сприяє кращій комунікації та документації.

Щоб уникнути необхідності створювати архітектуру системи з нуля, можна використовувати архітектурні патерни – готові рішення, які можна адаптувати під вимоги проекту. Найпоширеніші з них – це MVC (Model-View-Controller), MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel).

Види архітектури:

- монолітна;
- мікросервісна;
- модульна (багаторівнева).

При розробці застосунків на Vue.js часто використовують паттерн архітектури MVVM (Model-View-ViewModel), який забезпечує розділення логіки застосунку на три основні частини:

1. Model (Модель)
    - Модель відповідає за управління даними та бізнес-логікою застосунку.
    - У Vue.js модель представлена об'єктами даних, які зберігаються в стані компонента або у Vuex (якщо використовується централізоване управління станом).

2. View (Представлення)
    - Представлення відповідає за відображення інтерфейсу користувача.
    - У Vue.js представлення визначається за допомогою шаблонів (templates) в компонентах, де використовуються HTML, CSS та директиви Vue.js для рендерингу даних.
3. ViewModel (Модель-Представлення)
    - Модель-представлення забезпечує зв'язок між моделлю та представленням, керуючи логікою відображення та реактивністю.
    - У Vue.js ViewModel реалізовано через Vue компоненти, які включають реактивні дані, методи та обчислювальні властивості (computed properties).

Паттерн архітектури MVVM у Vue.js забезпечує структурований підхід до розробки додатків, розділяючи логіку, управління даними та представлення. Це полегшує розробку, тестування та підтримку додатків, забезпечуючи високу продуктивність та модульність.

У файлова структура застосунку представлена на рисунку 2.3.

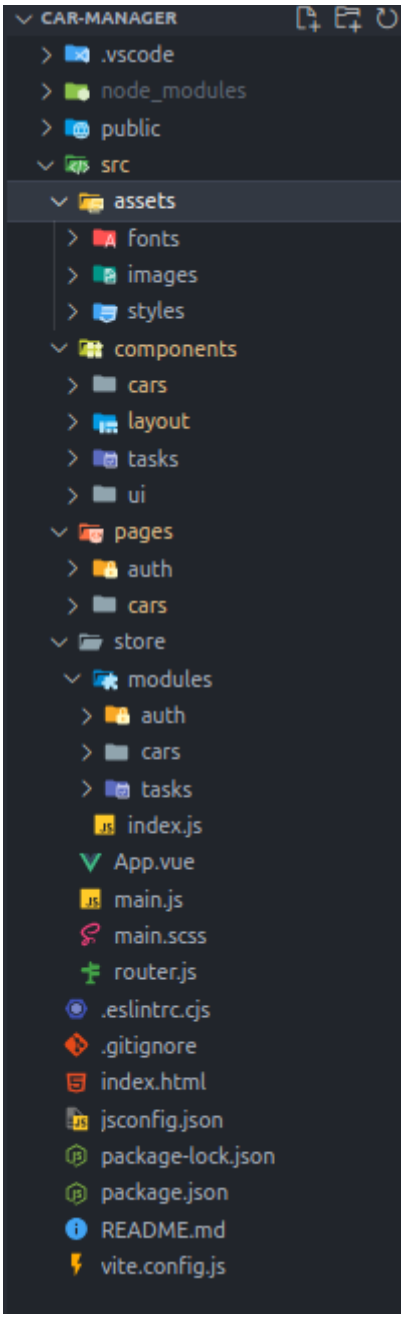


Рис.2.3. Файлова структура застосунку

Дана файлова структура проекту демонструє організований підхід до розробки на Vue.js, з використанням концепції розділення логіки та компонентів. Розглянемо кожен каталог і файл детальніше:

Основні каталоги та файли:

- .vscode – налаштування редактора Visual Studio Code для даного проекту.
- node_modules – каталог зі встановленими залежностями проекту.

- public – містить статичні файли, такі як index.html

Каталог src:

- assets – містить статичні ресурси, які використовуються у застосунку.
- components – містить Vue компоненти, що повторно використовуються в різних частинах застосунку.
- pages – містить Vue компоненти для різних сторінок застосунку.

Каталог store:

- modules – модулі стану для різних частин застосунку.
- App.vue: головний компонент застосунку, який об'єднує всі інші компоненти та сторінки.
- main.js: головний файл для ініціалізації Vue застосунку.
- main.scss: головний файл стилів для застосунку.
- router.js: файл конфігурації маршрутизатора для налаштування маршрутизації у застосунку.

Ця файлова структура демонструє добре організований підхід до розробки Vue.js додатків з використанням компонентів, сторінок та модулів стану. Вона забезпечує чітке розділення обов'язків, що полегшує підтримку та масштабування проекту. Основні каталоги та файли забезпечують логічну організацію коду, що спрощує навігацію та розвиток проекту.

На рисунку 2.4 зображена діаграма класів, на якій видно основні класи застосунку та їхні зв’язки.

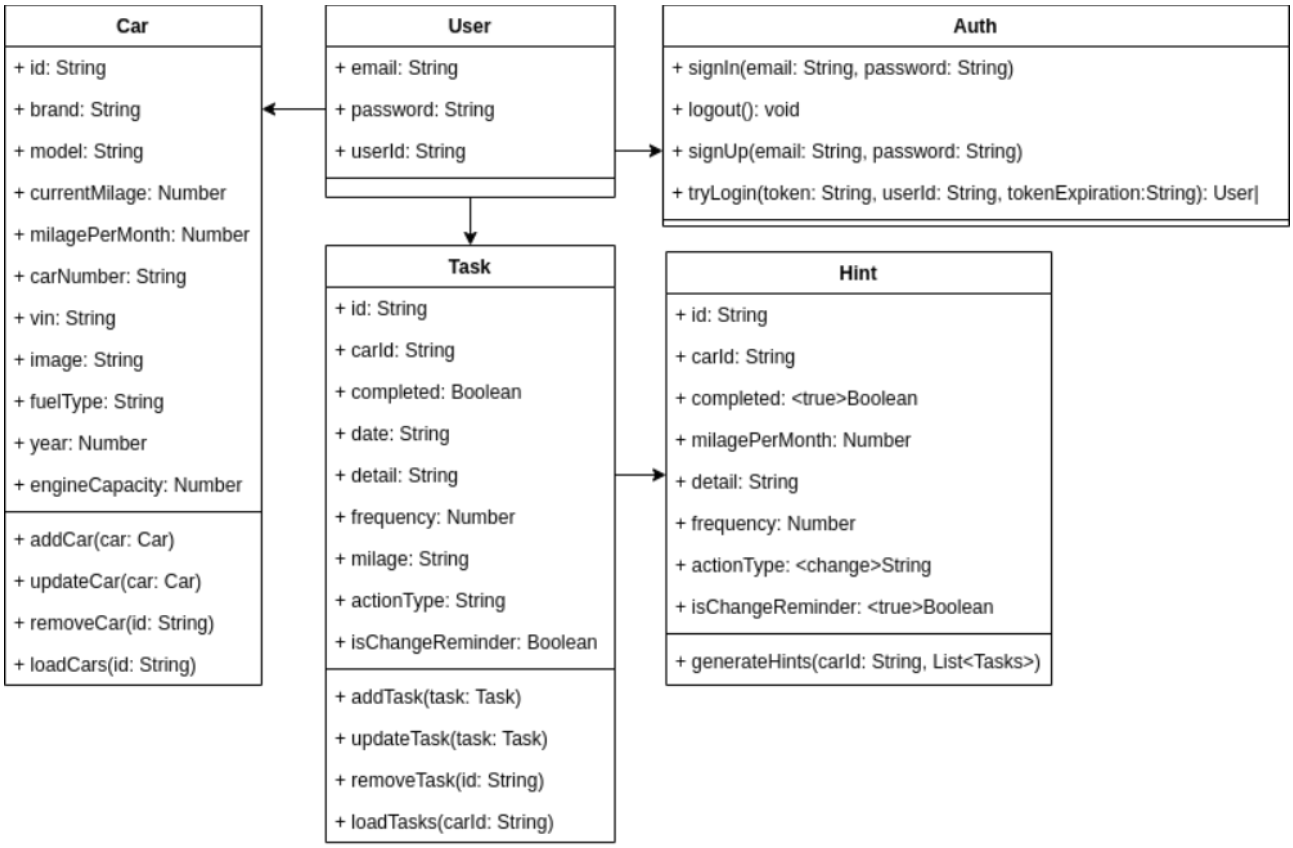


Рис.2.4. Діаграма класів

2.3 Розробка серверної частини системи обслуговування автомобілів

Для забезпечення клієнт-серверного зв’язку в проекті було обрано Firebase REST API. Це потужний інструмент, який надає можливість взаємодіяти з різними сервісами Firebase через стандартні HTTP-запити. Використання Firebase REST API дозволяє інтегрувати такі функціональні можливості, як реальна база даних, автентифікація користувачів, зберігання файлів та відправка повідомлень, що робить його ідеальним рішенням для розробки сучасних веб та мобільних додатків.

Основні Сервіси Firebase через REST API:

- Firebase Authentication API;
- Firebase Firestore API;
- Firebase Realtime Database API;
- Firebase Cloud Messaging API;

- Firebase Cloud Storage API.

В нашому випадку було обрано два основні сервіса: Firebase Authentication API та Firebase Realtime Database API.

Також варто зазначити, що REST API ґрунтується на декількох принципах:

1. Кожен запит, відправлений на сервер, має включати всі дані, необхідні для його обробки.
2. Підтримується кешування.
3. Використовуються уніфіковані інтерфейси (GET, POST, PUT, DELETE).
4. Сервер працює незалежно від клієнта.
5. Система має багатошарову архітектуру. [10]

2.4 Розробка структури бази даних для обліку обслуговування автомобілів

Вибрана база даних Firebase – це NoSQL база даних, яка забезпечує зберігання даних у вигляді JSON-структур (Realtime Database). Така структура дозволяє зберігати дані в гнучкий та масштабований спосіб. Документи та колекції, або JSON-структури, представляють собою моделі даних із серверної частини. Тому необхідно ретельно спроектувати базу даних, щоб уникнути конфліктів між моделями та даними під час розробки.

Проектування бази даних у Firebase включає визначення структури даних, організації колекцій і документів (у випадку Firestore) або JSON-структур (у випадку Realtime Database). Це забезпечує ефективне зберігання даних та доступ до них, а також спрощує подальший розвиток і масштабування додатку.

Опис моделі:

1. Користувачі (Users)
2. Автомобілі (Cars)
3. Задачі (Tasks)

Користувачі:

- user_id: STRING - Унікальний ідентифікатор користувача
- e-mail: STRING - Електронна пошта користувача
- password: STRING - Пароль користувача

Автомобілі:

- car_id: STRING - Унікальний ідентифікатор автомобіля
- user_id: STRING - Ідентифікатор користувача (власника автомобіля)
- brand: STRING - Марка автомобіля
- carNumber: STRING - Номер автомобіля
- currentMilage: INTEGER - Поточний пробіг
- engineCapacity: INTEGER - Об'єм двигуна
- fuelType: STRING - Тип палива
- image: STRING - URL зображення автомобіля
- milagePerMonth: INTEGER - Пробіг за місяць
- model: STRING - Модель автомобіля

Задачі:

- task_id: STRING - Унікальний ідентифікатор задачі
- car_id: STRING - Ідентифікатор автомобіля
- user_id: STRING - Ідентифікатор користувача
- actionType: STRING - Тип дії (наприклад, "change")
- completed: BOOLEAN - Статус виконання задачі
- date: STRING - Дата виконання задачі
- detail: STRING - Деталі задачі
- frequency: INTEGER - Частота виконання (наприклад, кожні 8000 км)
- isChangeReminder: BOOLEAN - Нагадування про зміну

- mileage: INTEGER - Пробіг на момент виконання задачі

Інфологічна модель бази даних визначає ключові сутності, їхні атрибути та взаємозв'язки між ними. Ця модель допоможе при проектуванні та реалізації бази даних для зберігання інформації про користувачів, їхні автомобілі та відповідні задачі. Такий підхід забезпечує структурованість і логічну організацію даних, що полегшить подальшу розробку та підтримку системи.

2.5 Розробка інтерфейсу системи обслуговування автомобілів

При проектуванні інтерфейсу користувача важливо дотримуватися певних стандартів та рекомендацій. Це допомагає уніфікувати інтерфейс, створити єдину тему та покращити користувацький досвід. Використання усталених стандартів забезпечує послідовність у дизайні, що робить взаємодію з додатком інтуїтивно зрозумілою та зручною для користувачів. Дотримання рекомендацій дозволяє уникнути поширених помилок у дизайні та сприяє створенню більш естетично привабливих і функціональних інтерфейсів.

Переваги Material Design:

- єдність і консистентність
- візуальна привабливість
- кращий користувацький досвід
- простота впровадження

Для проектування веб-застосунку використано Figma. Під час проектування використовувалися графічні матеріали (векторні іконки) з колекції Material Design. Було визначено стилі інтерфейсу та вибрано відповідні шрифти.

Перелік кольорів, які було використано:

- оранжевий: E27437;
- темно-сірий: 232323;
- світло-сірий: 717171;

- сірий: 3C3C3C;
- білий: FFFFF;
- білий прозорий: FFFFF 12%
- червоний: FF505C;

Шрифти(Fixel Display):

- заголовок: 20 жирний;
- титул: 18 напівжирний;
- текст: 18 середній;
- поля вводу: 18 тонкий;
- кнопки: 16 напівжирний;

При проектуванні було створено 10 варіантів екранних форм та 2 діалогових вікна.

На рисунку 2.5 зображено екран входу(реєстрації).

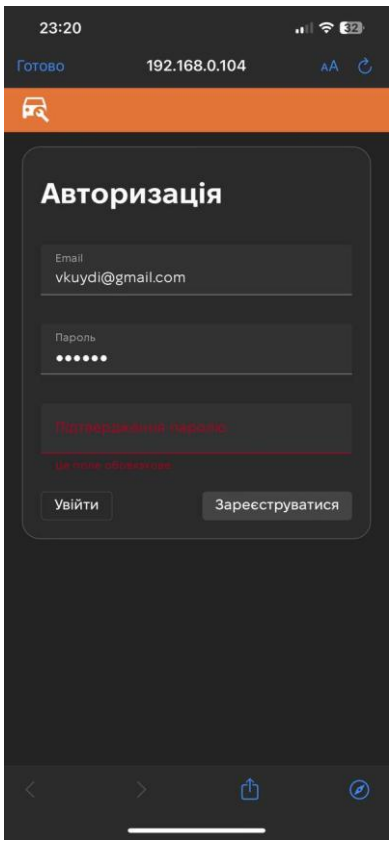


Рис.2.5. Екран входу та реєстрації

На рисунку 2.6 зображено головний екран, де знаходяться всі додані автомобілі та після натискання на “+” відображено форму додавання автомобіля.

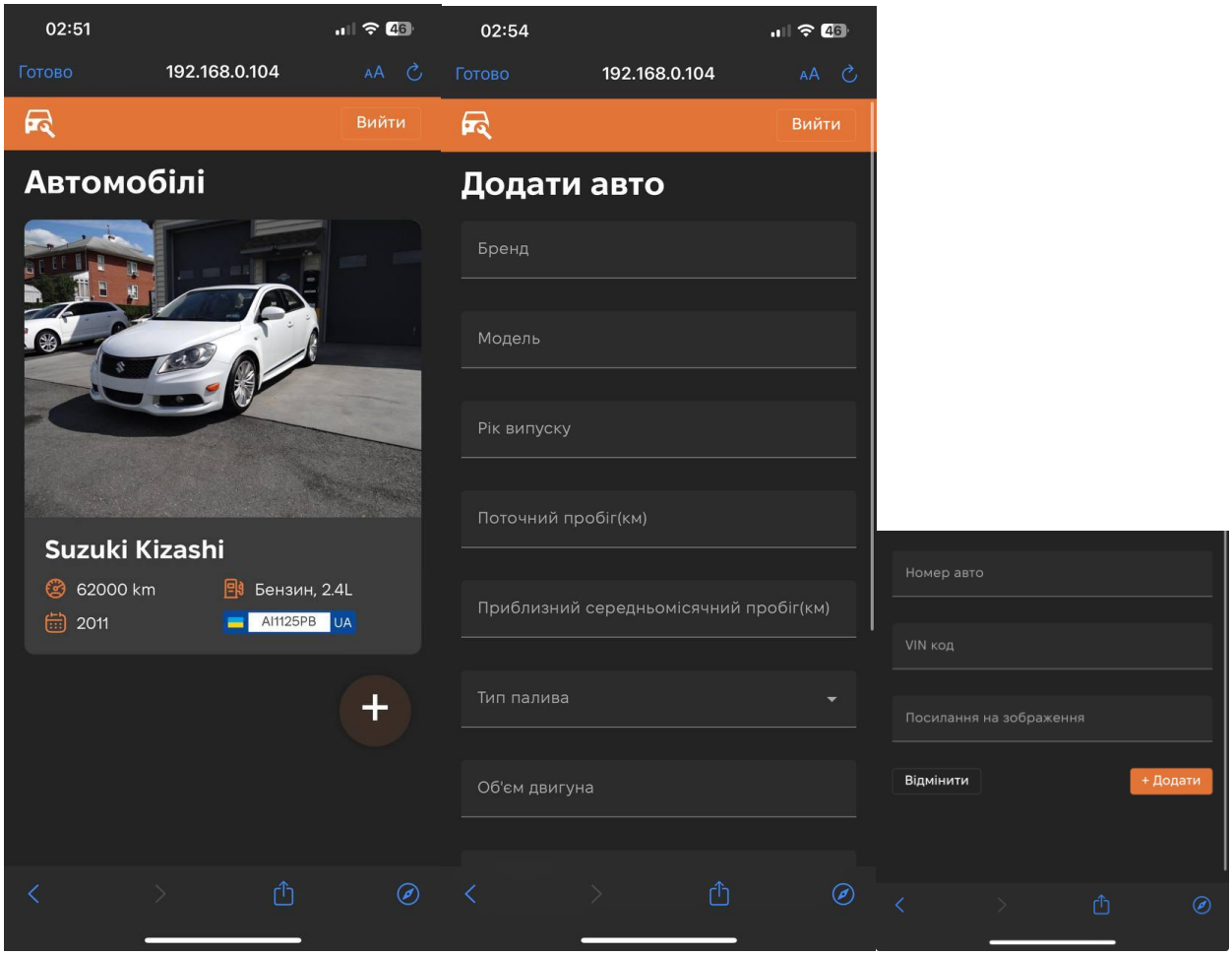


Рис.2.6. Екран головної сторінки

На рисунку 2.7 відображено обраний автомобіль, випадне меню для переключення між автомобілями та форма для зміни інформації про транспортний засіб. На рисунку 2.8 відображено задачі обраного автомобіля та форма додавання.

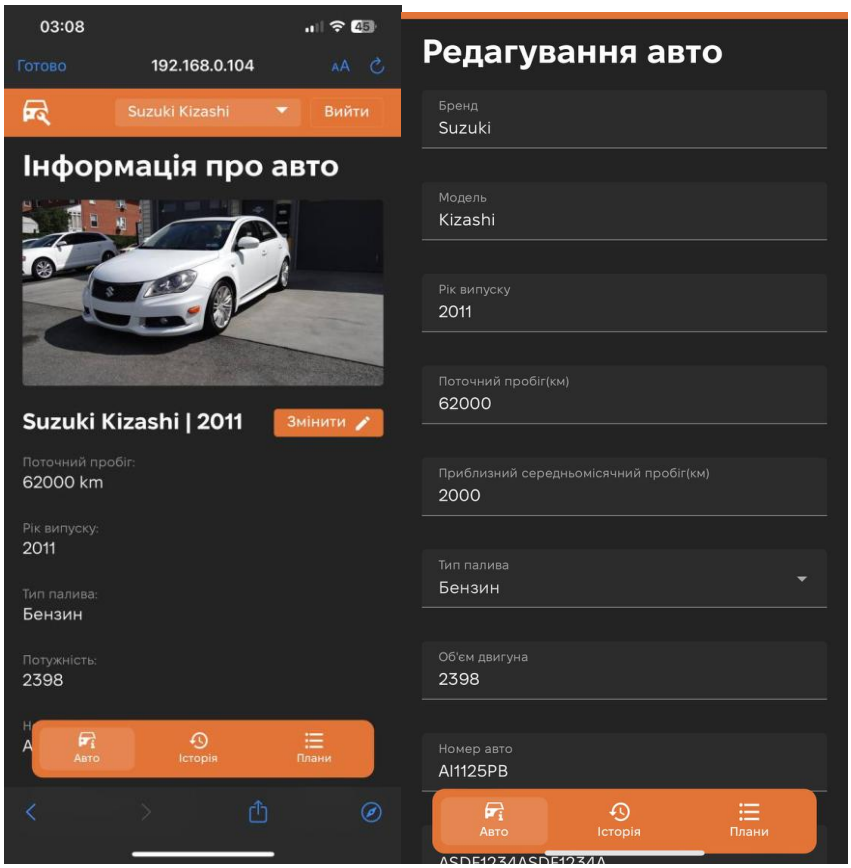


Рис.2.7. Екран обраного автомобілю та форма зміни інформації

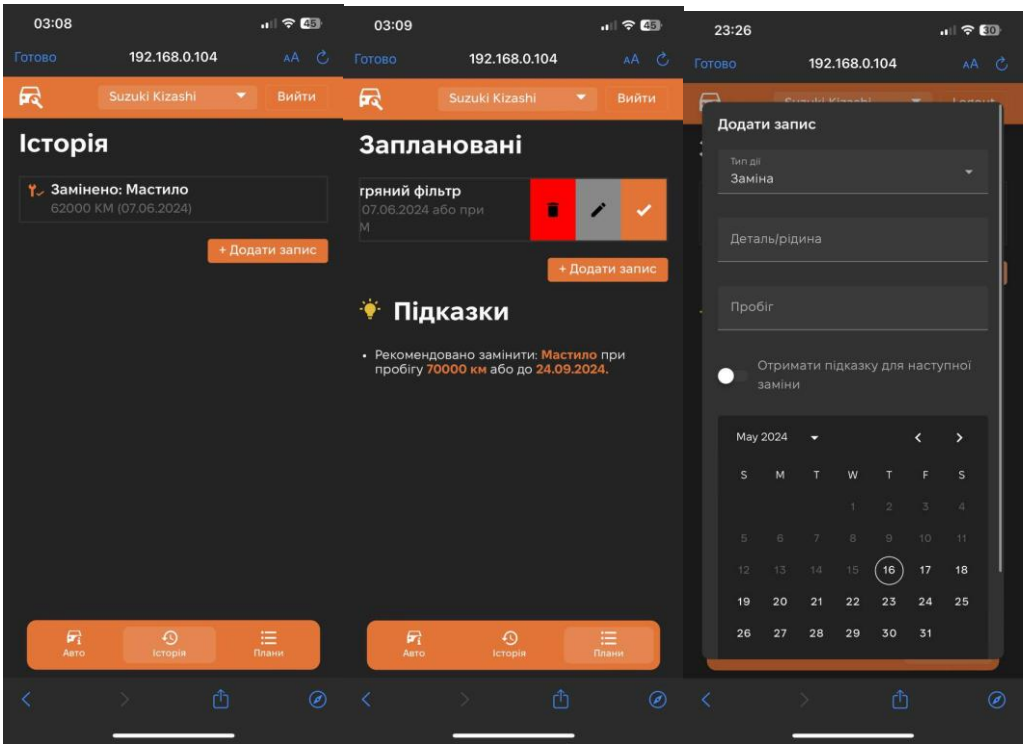


Рис.2.8. Екран задач та форма зміни/додавання інформації

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Технології розробки

Розробка відбувається за допомогою фреймворку. Фреймворк – це набір інструментів та бібліотек, які спрощують процес розробки програмного забезпечення. Компанії витрачають значні кошти на IT-відділи, частково через тривалий час розробки програмного забезпечення. Фреймворки допомагають скоротити цей час, стандартизуючи підходи до створення веб-додатків. З точки зору компаній, фреймворк є цінним інструментом. Він підтримує архітектуру програмного забезпечення як сервіс (SOA), яка організовує та використовує розподілені можливості, що можуть бути під контролем різних доменів. SOA забезпечує швидші цикли розробки, легкість у підтримці та можливість повторного використання різних сервісів у різних ситуаціях. Фреймворки також сприяють досягненню SOA архітектури з меншими витратами. Фреймворки також забезпечують інформаційну безпеку, запобігаючи таким атакам, як XSS та SQL ін'єкції. Розробники фреймворків вбудовують заходи безпеки, що знижує необхідність додаткових зусиль з боку розробників.

Проведено аналіз та порівняння інших відомих фреймворків.

У сучасному світі веб-розробки існує багато фреймворків, які допомагають розробникам створювати ефективні та масштабовані додатки. Серед них особливо виділяються Vue.js, React та Angular. Кожен з цих фреймворків має свої унікальні особливості, переваги та недоліки, що робить їх більш придатними для різних типів проектів та команд розробників.

REACT

React – це JavaScript-бібліотека для створення користувацьких інтерфейсів, розроблена та підтримувана Facebook. Вона широко використовується такими компаніями, як Uber, Netflix, Twitter, Udemy, і Paypal. Основні переваги React включають:

- React дозволяє розробляти інтерфейси користувача за допомогою компонентів, що полегшує повторне використання коду;
- Віртуальний DOM забезпечує швидке оновлення користувацького інтерфейсу, мінімізуючи зміни в реальному DOM;
- React використовує JSX, що дозволяє писати компоненти, використовуючи синтаксис, схожий на HTML;
- React легко інтегрується з іншими бібліотеками або фреймворками для управління станом, маршрутизації тощо;

React орієнтований на розробку односторінкових додатків (SPA) і забезпечує високу продуктивність завдяки використанню віртуального DOM. React підтримує універсальні (ізоморфні) додатки, що дозволяє рендеринг на сервері для покращення SEO та продуктивності.

React потребує значних зусиль для налаштування проекту, оскільки він є бібліотекою, а не фреймворком. Це означає, що розробники мають самостійно обирати інші інструменти для управління станом (наприклад, Redux) та маршрутизації (наприклад, React Router).[7]

ANGULAR

Angular – це повноцінний фреймворк для розробки веб-додатків, створений і підтримуваний Google. Основні переваги Angular включають:

- Angular дозволяє розбивати додаток на модулі, що покращує організацію коду та його повторне використання;
- Angular використовує TypeScript, що додає статичну типізацію та покращує розробку великих проектів;
- Angular має вбудовані інструменти для управління станом, маршрутизації, форм, запитів HTTP та інше;
- Angular використовує шаблони на основі HTML з додаванням директив і двостороннього зв'язку даних;

Angular є повноцінним фреймворком для створення масштабованих і

складних додатків. Він включає потужні інструменти для тестування, а також засоби для спрощення створення форм і обробки валідації. Angular CLI спрощує налаштування і управління проектами, забезпечуючи високий рівень продуктивності.

Angular має круту криву навчання, особливо для новачків, через свою складність і велику кількість концепцій, таких як модулі, компоненти, директиви, сервісні ін'єкції тощо. Крім того, велика кількість абстракцій може зробити проект важчим для підтримки, якщо не дотримуватись чіткої структури.[8]

VUE

Для розробки застосунку використано Vue.js. Він широко використовується для розробки веб-додатків завдяки його простоті, швидкості та адаптивності. Vue.js є легким JavaScript фреймворком, який сприяє швидшій і надійнішій розробці фронтенду. Він відрізняється від інших фреймворків своєю поступовою адаптивністю, зосереджуючись на шарі представлення і легко поєднуючись з іншими бібліотеками або проектами. Vue.js дозволяє створювати потужні односторінкові веб-додатки при інтеграції з сучасними стеком та бібліотеками. Його простота використання, малий розмір та швидка продуктивність роблять його популярним серед розробників, які прагнуть створювати інтерактивні, гнучкі та орієнтовані на дані додатки. Vue.js також популярний серед стартапів і великих підприємств завдяки зручному інтерфейсу та мінімальній кривій навчання, що робить його чудовим вибором для програмістів, які прагнуть впроваджувати нові технології у свої проекти. На рисунку 3.2 відображено схему роботи реактивності у Vue.js.

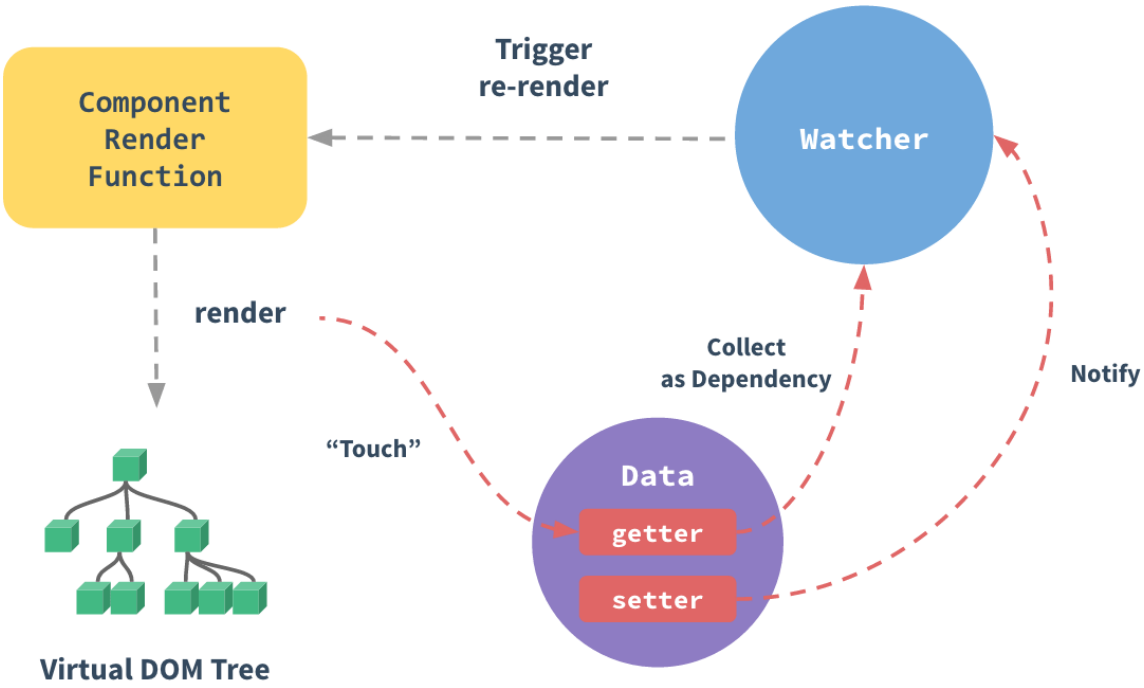


Рис.3.2. Реактивність Vue.js

У цій роботі було обрано Vue.js, оскільки, з моєї точки зору, це найбільш зручне середовище для розробників. Vue.js є дуже легким для вивчення, що робить його ідеальним для навчальних цілей. Однак є й інші позитивні аспекти цього фреймворку, такі як:

- відмінний інструмент командного рядка (CLI), який постачається з Vue;
- швидкість;
- офіційні пакети, такі як Vuex і Vue Router;
- компоненти в одному файлі;
- малий розмір;
- відмінна та легко зрозуміла документація;
- можливість поступової інтеграції на веб-сторінку;

Завдяки цим характеристикам Vue.js є чудовим вибором, для розробників прагнуть покращити свої Web-застосунки, зберігаючи при цьому існуючу функціональність та структуру. Завдяки цим характеристикам Vue.js є чудовим

вибором, для розробників прагнуть покращити свої Web-застосунки, зберігаючи при цьому існуючу функціональність та структуру.

Висновки на користь використання Vue.js:

- Vue.js відзначається інтуїтивно зрозумілим синтаксисом та низьким бар'єром входу, що робить його ідеальним для новачків та навчальних цілей. Це значно спрощує процес початкового освоєння у порівнянні з Angular, який має круту криву навчання через свою складну архітектуру.
- Vue.js є поступово адаптивним фреймворком, що дозволяє легко інтегрувати його в існуючі проекти без необхідності повної переробки. Ця особливість вигідно відрізняє Vue.js від Angular, який є більш монолітним, та React, який потребує більшого налаштування для інтеграції з іншими інструментами.
- Завдяки невеликому розміру та використанню віртуального DOM, Vue.js забезпечує високу продуктивність та швидке завантаження додатків. У порівнянні з React, Vue.js виграє за рахунок своєї компактності та меншої складності налаштувань.
- Vue.js надає офіційні пакети для управління станом (Vuex) та маршрутизації (Vue Router), що значно спрощує процес розробки та забезпечує всебічну підтримку. Це є перевагою над React, де аналогічні можливості досягаються за рахунок сторонніх бібліотек, що може ускладнити налаштування проекту.
- Документація Vue.js відзначається високою якістю та зрозумілістю, що сприяє швидкому навчанню та ефективному використанню фреймворку. Активна спільнота забезпечує підтримку новачків та надає численні ресурси для навчання, що є перевагою над Angular, який має складнішу документацію.
- Vue.js дозволяє поступово впроваджувати нові функціональності в існуючі проекти, зберігаючи при цьому їхню роботу. Це особливо важливо для компаній, що вже мають функціонуючі веб-сайти і не можуть дозволити

собі тривалих періодів простою для повної міграції на новий фреймворк.[6]

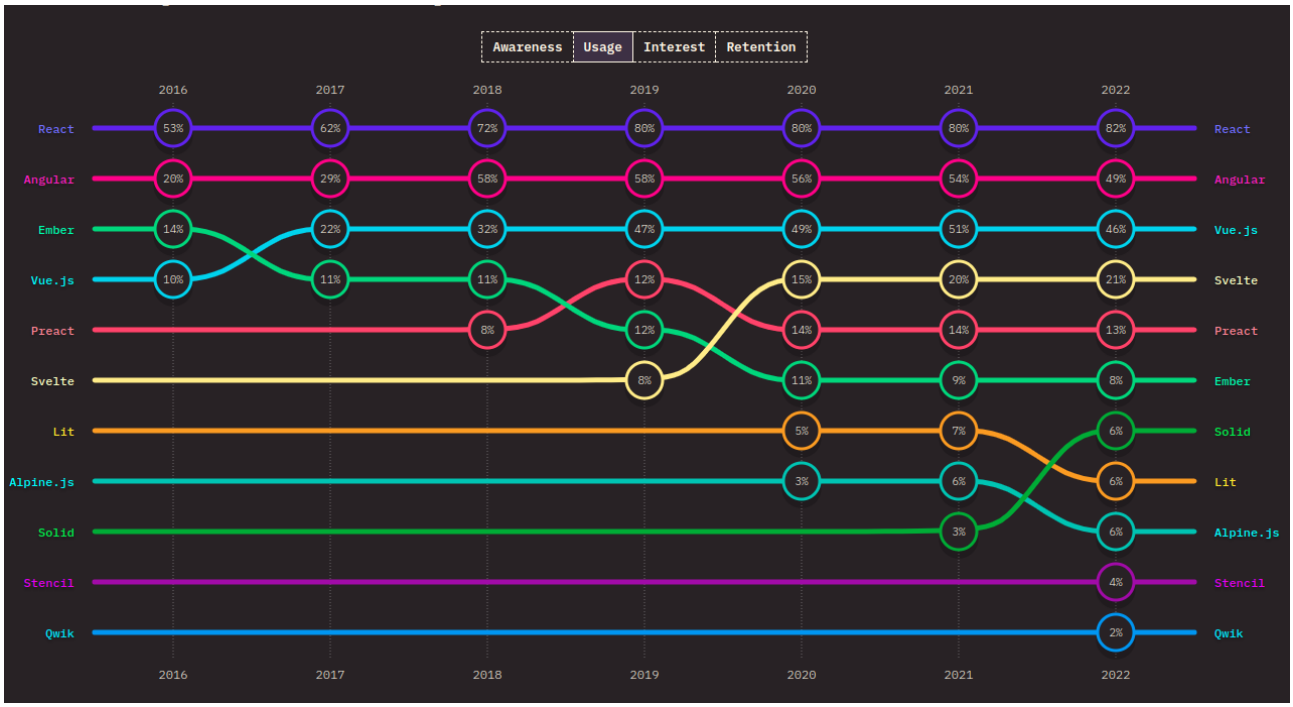


Рис.3.3 Рейтинг State of JS 2022

Діаграма на рисунку 3.3 ілюструє зміну популярності використання різних JavaScript-фреймворків у період з 2016 по 2022 рік.[9] Загальні Тенденції:

- React зберігає провідні позиції та показує стабільне зростання у використанні протягом всього періоду.
- Angular також залишається одним із популярних фреймворків, хоч і з меншим зростанням у порівнянні з React.
- Vue.js демонструє значне зростання з 2016 по 2022 рік, особливо після 2017 року.
- Svelte показує швидке зростання після 2019 року, що свідчить про зростаючу популярність цього фреймворка.
- Preact стабільно зростає, хоч і на нижчому рівні, ніж інші фреймворки.
- Нові фреймворки, такі як Solid, Lit, Alpine.js, Stencil та Qwik, починають

з'являтися в останні роки і показують зростання інтересу та використання.

За результатами проведеного аналізу можна зробити висновок, що Vue.js є оптимальним вибором для багатьох проектів завдяки своїй простоті, гнучкості, високій продуктивності та відмінній документації. Він забезпечує швидкий старт для новачків і надає всі необхідні інструменти для створення сучасних веб-додатків, що робить його привабливим як для малих стартапів, так і для великих корпоративних проектів.

GIT

Також було використано Git. Це широко використовувана система контролю версій, яка служить багатьом цілям. Він діє як система резервного копіювання, гарантуючи, що файли зберігаються на сервері, щоб запобігти втраті даних у разі збоїв машини. Git також підтримує історію версій файлів, захищаючи від втрати попередніх змін. Крім того, Git полегшує співпрацю, дозволяючи розробникам працювати над одним проектом одночасно, відстежуючи зміни та ефективно їх об’єднуючи. Він важливий у безперервних процесах інтеграції, де артефакти розробки можуть бути автоматично протестовані та випущені в виробничі системи. Крім того, Git використовується в освітніх установах для розповсюдження вправ, рішень та оцінок, демонструючи його універсальність, окрім розробки програмного забезпечення. Загалом, функціональні можливості Git роблять його безцінним інструментом для керування кодом, сприяння співпраці та забезпечення цілісності проекту. [11]

FIGMA

Для розробки прототипу інтерфейсу було обрано Figma.

Figma – це потужний інструмент для дизайну інтерфейсів користувача (UI) та прототипування, який став дуже популярним серед дизайнерів і розробників. Вибір Figma для розробки прототипу інтерфейсу має багато переваг:

- Figma є хмарним інструментом, що означає, що всі файли зберігаються онлайн і можуть бути доступні з будь-якого місця, де є інтернет-з’єднання. Це значно полегшує спільну роботу над проектами, оскільки дизайнери можуть легко ділитися своїми файлами і співпрацювати в режимі реального

часу.

- Figma дозволяє кільком користувачам одночасно працювати над одним файлом. Це означає, що дизайнери, розробники та інші учасники проекту можуть бачити зміни в реальному часі, коментувати та надавати зворотний зв'язок без необхідності постійно пересилати файли.
- Figma має вбудовані інструменти для створення інтерактивних прототипів. Ви можете легко створювати взаємодії між різними екранами та компонентами, що дозволяє створити реалістичні макети інтерфейсу. Це важливо для тестування UX (користувацького досвіду) і отримання зворотного зв’язку від користувачів.
- Figma підтримує створення компонентів, що дозволяє використовувати один і той самий елемент в різних частинах проекту. Це допомагає підтримувати консистентність дизайну і значно полегшує внесення змін. При зміні одного компонента, всі його екземпляри автоматично оновлюються.
- Figma дозволяє створювати та використовувати дизайн-системи, що включають стандартизовані компоненти, стилі, кольорові схеми та типографіку. Це забезпечує єдиний стиль для всіх елементів інтерфейсу і сприяє узгодженості та ефективності у створенні дизайну.
- Figma інтегрується з багатьма іншими інструментами для розробки, управління проектами та тестування. Наприклад, вона підтримує інтеграції з Zeplin, Trello, Slack та іншими сервісами, що полегшує робочий процес і співпрацю між командами.
- Figma дозволяє користувачам переглядати дизайни і залишати коментарі безпосередньо на макеті. Це робить процес отримання зворотного зв’язку швидшим та ефективнішим, оскільки всі коментарі зберігаються в контексті відповідних елементів дизайну.
- Завдяки своїй архітектурі, Figma добре підходить як для невеликих проектів, так і для великих команд з складними дизайн-системами. Вона

підтримує безліч файлів та проектів, що дозволяє масштабувати робочий процес разом із ростом команди чи проекту.

- Оскільки Figma працює в браузері, вона доступна на всіх основних операційних системах, включаючи Windows, macOS, Linux та навіть на мобільних пристроях. Це забезпечує велику гнучкість для команд, які працюють на різних платформах.
- Figma забезпечує високий рівень безпеки, використовуючи шифрування даних та інші методи захисту інформації. Це важливо для компаній, які працюють з конфіденційними даними.

Figma є відмінним вибором для розробки прототипів інтерфейсів завдяки своїм потужним інструментам для спільної роботи, інтерактивності, підтримці компонентів та дизайн-систем, інтеграціям з іншими інструментами та доступності на різних платформах. Це робить її незамінним інструментом для сучасних дизайнерів та розробників, які прагнуть створювати якісні та узгоджені інтерфейси користувача.[10]

GOOGLE FIREBASE

В якості бази даних вибрано Google Firebase. Майже кожен сучасний сайт, додаток чи програмне забезпечення потребує можливості віддаленого доступу до бази даних, а також функцій аутентифікації та авторизації. Віддалений доступ дозволяє користувачам отримувати доступ до своїх даних з різних пристроїв, потребуючи лише підключення до Інтернету. При цьому дані користувачів захищені за допомогою процесів авторизації та автентифікації.

Такі можливості забезпечуються за допомогою серверів та хмарних технологій. Власний сервер облаштовувати економічно невигідно, тому хмарні технології набули великої популярності. Одним з найпоширеніших рішень є Firebase від Google. Воно надає широкий спектр інструментів для розробників, зокрема:

- аналітику;
- відслідковування помилок;
- аутентифікацію;

- базу даних в реальному часі;
- тестову лабораторію;
- хмарні повідомлення;
- сховище;

так багато інших.

Розглянемо основні інструменти, які були використані.

Аутентифікація – інструмент для ідентифікації користувачів. Firebase підтримує різні методи аутентифікації, такі як електронна пошта і пароль, Google-акаунт, Facebook, X (Twitter), GitHub, вхід за номером телефону, а також власні рішення.[12]

База даних в реальному часі (Realtime Database) – це хмарна NoSQL база даних, яка дозволяє зберігати та синхронізувати дані між користувачами в режимі реального часу.

3.2 Ініціалізація проекту

Для створення та розробки додатку "Автомобільний менеджер" було використано Vite. Vite — це сучасний інструмент для збірки фронтенд-проектів, створений для забезпечення високої продуктивності та зручності розробки. Він забезпечує швидкий запуск проекту, миттєве оновлення модулів (HMR), ефективну збірку і підтримку сучасних функцій.

Конфігураційний файл Vite, представлений на рисунку 3.4, налаштовує проект на Vue.js, включає плагіни та додаткові параметри, які спрощують розробку. Потрібно виділити плагін @vitejs/plugin-vue для підтримки Vue.js у проекті. Цей плагін забезпечує обробку .vue файлів та інших специфічних для Vue функцій.

```
import { URL, fileURLToPath } from 'node:url'

import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
  plugins: [
    vue(),
  ],
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src', import.meta.url)),
    },
  },
  css: {
    preprocessorOptions: {
      scss: {
        additionalData: '@import "@/assets/styles/_variables.scss";',
      },
    },
  },
})
```

Рис.3.4. Конфігурація Vite

3.3 Створення та ініціалізація проекту у Firebase

Етапи створення проекту:

1) Відкрити консоль Firebase;
2) Створити обліковий запис та проект;
3) Додати веб-застосунок до проекту Firebase;
4) Завантажити та налаштувати конфігураційний файл;
5) Інтегрувати Firebase SDK;

Вікно створення нового проекту у Firebase представлена на рисунку 3.5.

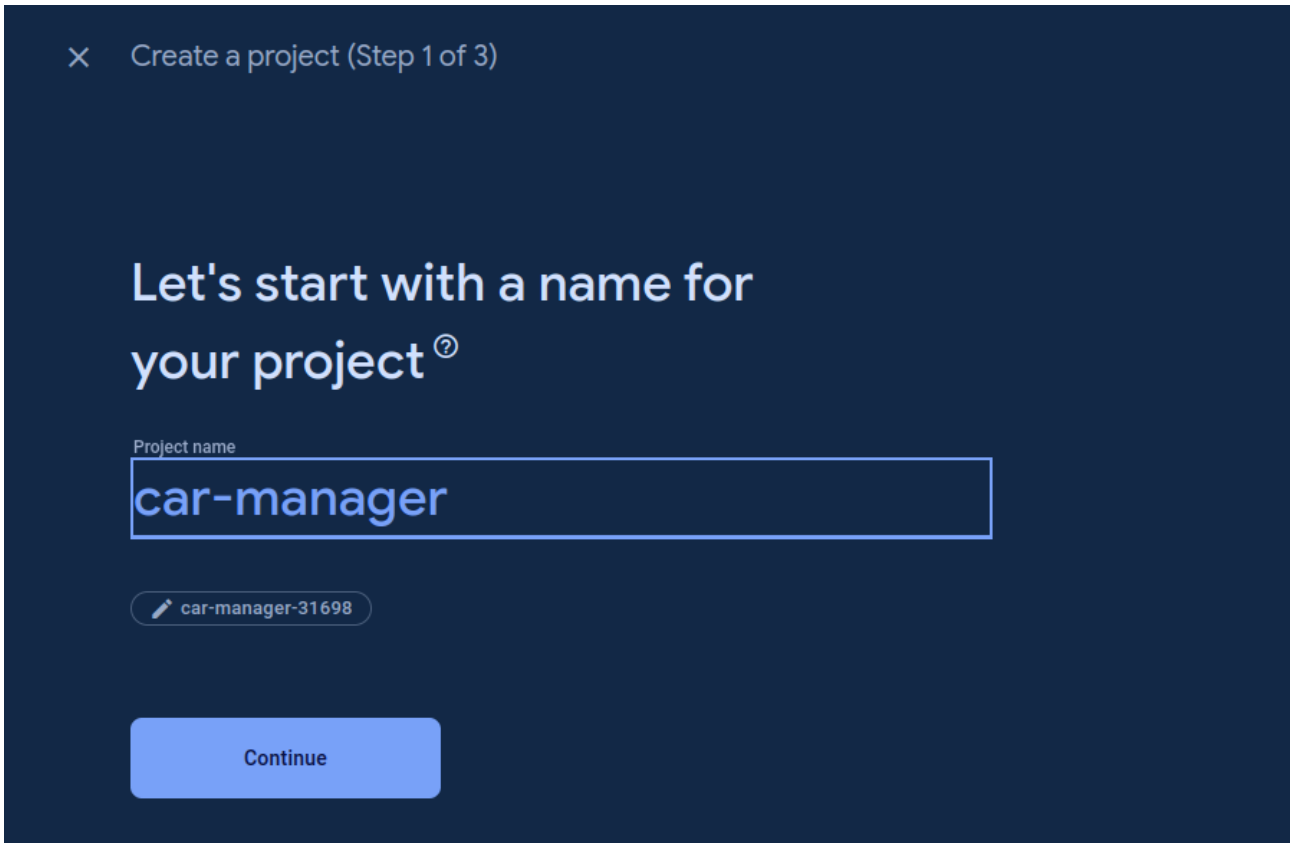


Рис.3.5 Додавання нового проекту до Firebase

На рисунку 3.6 зображено доданий проект з обраними сервісами Firebase:

- Authentication
- Realtime Database

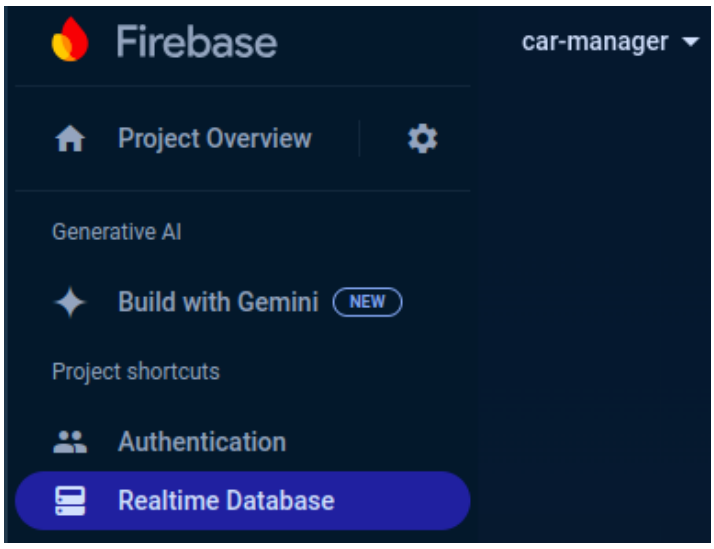


Рис.3.6 Проект Firebase

Далі необхідно ініціалізувати Firebase в нашому проекті:

- Встановити Firebase;
- Імпортувати та ініціалізувати Firebase у проекті рисунку 3.7.

```
import firebase from 'firebase/app';
import 'firebase/auth';
import 'firebase/database';

const firebaseConfig = {
  apiKey: "AIzaSyBOaKgaarVkossxQl9aMcSVFRGV0mRm8FY",
  authDomain: "car-manager-5d9e6.firebaseapp.com",
  databaseURL: "https://car-manager-5d9e6.firebaseio.com",
  projectId: "car-manager-5d9e6",
  appId: "478784615266"
};

firebase.initializeApp(firebaseConfig);
```

Рис.3.7. Конфігурація проекту

Після успішної ініціалізації проекту важливо забезпечити безпеку системи та користувачів, тому було використано Firebase Authentication. Firebase Authentication з використанням email та паролю є одним з найбільш популярних методів автентифікації. Це забезпечує зручний та безпечний спосіб входу для користувачів. Цей метод дозволяє користувачам створювати облікові записи, входити в систему, змінювати паролі та багато іншого. На рисунку 3.9 зображено шаблон форми автентифікації. На рисунку 3.8 зображено метод авторизації користувача - login. Також цей самий метод використовується для реєстрації нового користувача.

```
async Login(context, payload) {
  const responseAfterLogin = await fetch('https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIzaSyBOaKgaarVkossxQl9aMcSVFRGV0mRm8FY', {
    method: 'POST',
    body: JSON.stringify({
      email: payload.email,
      password: payload.password,
      returnSecureToken: true,
    }),
  })

  const responseData = await responseAfterLogin.json()
  if (!responseAfterLogin.ok) {
    const error = new Error(responseData.message || 'Failed to authenticate. Check your login data.')
    throw error
  }

  const expiresIn = +responseData.expiresIn * 1000
  const expirationDate = new Date().getTime() + expiresIn

  localStorage.setItem('token', responseData.idToken)
  localStorage.setItem('userId', responseData.localId)
  localStorage.setItem('tokenExpiration', expirationDate)

  timer = setTimeout(() => {
    context.dispatch('autoLogout')
  }, expiresIn)

  context.commit('setUser', {
    token: responseData.idToken,
    userId: responseData.localId,
  })
},
```

Рис.3.8. Метод авторизації користувача.

```
<template>
  <BaseCard>
    <section class="auth">
      <div class="title">
        Авторизація
      </div>
      <VForm
        class="auth-form"
        fast-fail
        @submit.prevent="submitForm"
      >
        <div class="auth-form__group">
          <VTextField
            v-model.trim="email"
            :rules="[rules.required, rules.email]"
            label="Email"
          />
        </div>
        <div class="auth-form__group">
          <VTextField
            v-model.trim="password"
            label="Пароль"
            :rules="[rules.required, rules.password]"
            type="password"
          />
        </div>
        <div v-if="!isLoginMode" class="auth-form__group">
          <VTextField
            v-model.trim="confirmPassword"
            label="Підтвердження паролю"
            :rules="[rules.required, rules.password, rules.passwordMatch]"
            type="password"
          />
        </div>
        <div class="auth-form__controls">
          <BaseButton
            type="button"
            mode="outline-btn"
            @click="switchMode"
          >
            {{ switchModeButtonCaption }}
          </BaseButton>
          <BaseButton>
            {{ submitButtonCaption }}
          </BaseButton>
        </div>
      </VForm>
    </section>
  </BaseCard>
</template>
```

Рис.3.9. Шаблон форми для автентифікації

Після успішної автентифікації користувача потрібно зберігати дані про автомобілі та задачі, тому для цього було обрано Firebase Realtime Database.

Особливості Firebase Realtime Database:

- синхронізація даних у реальному часі;
- підтримка офлайн-режиму;
- безпека
- масштабованість

На рисунку 3.10 зображено метод додавання даних про автомобіль.

```
async addCar(context, data) {
  const userId = context.rootGetters.userId
  const token = context.rootGetters.token

  const response = await fetch(`https://car-manager-5d9e6-default-rtdb.firebaseio.com/cars/${userId}.json?auth=${token}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(data),
  })

  const responseData = await response.json()

  if (!response.ok) {
    const error = new Error(responseData.message || 'Failed to add car')
    throw error
  }

  context.commit('addCar', {
    ...data,
    id: responseData.name,
  })
},
```

Рис.3.10. Метод додавання даних про автомобіль

Якщо дані про автомобіль було успішно додано, то користувач має можливість додати задачі, які потрібно виконати або були виконані. Для організації функціональності було імплементовано наступний метод, зображений на рисунку 3.11. Він є доволі комплексним, оскільки містить в адресі запиту два важливих поля userId та selectedCarId. Чітко вибудувана ієрархія задач, де кожен користувач має свої автомобілі, а кожен автомобіль містить список задач, дозволяє легко знаходити та обробляти потрібні дані. На рисунку 3.12 показано приклад структури бази даних: tasks => userId => selectedCarId => taskId.

```
async addTask(context, task) {
  const userId = context.rootGetters.userId
  const token = context.rootGetters.token
  const selectedCarId = context.rootGetters.selectedCar.id

  const response = await fetch(`https://car-manager-5d9e6-default-rtdb.firebaseio.com/tasks/${userId}/${selectedCarId}.json?auth=${token}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(task),
  })

  const responseData = await response.json()

  if (!response.ok) {
    const error = new Error(responseData.message || 'Failed to add task')
    throw error
  }

  context.commit('addTask', {
    carId: selectedCarId,
    task: {
      ...task,
      id: responseData.name,
    },
  })
},
```

Рис.3.11 Метод додавання задачі для автомобіля

Рис.3.12. Структура гілки tasks

3.4 Відображення даних на стороні користувача

Веб-застосунок "Автомобільний менеджер", надає користувачам можливість ефективно управляти своїми транспортними засобами, зберігаючи та переглядаючи детальну інформацію про кожен автомобіль і пов'язані з ним задачі. Ця функціональність дозволяє користувачам бачити повний список їхніх автомобілів, включаючи марку, модель, рік випуску та інші деталі. Крім того, користувачі можуть переглядати задачі, які потрібно виконати або вже виконані

для кожного автомобіля. Користувачі можуть легко додавати нові автомобілі та задачі, а також оновлювати існуючі дані, що забезпечує високу продуктивність та зручність використання. На рисунку 3.13 зображено компонент CarsList та HomeCar, які відображають усі наявні автомобілі користувача та інформацію про них. Варто зазначити, що одним із найпотужніших інструментів Vue.js для умовного рендерингу компонентів є директива v-if. Вона дозволяє умовно відображати або приховувати елементи на основі виразу, який повертає логічне значення (true або false).

```
<template>
  <BaseCard>
    <section class="cars-home">
      <div class="title">
        Автомобілі
      </div>
      <ul v-if="userCars.length > 0" class="cars-list">
        <HomeCar
          v-for="car in userCars"
          :id="car.id"
          :key="car.id"
          :brand="car.brand"
          :model="car.model"
          :milage="car.currentMilage"
          :year="car.year"
          :fuel-type="car.fuelType"
          :image="car.image"
          :capacity="car.engineCapacity"
          :car-number="car.carNumber"
        />
      </ul>
      <div v-else class="cars-home__no-cars">
        У вас поки немає автомобілів!
      </div>
      <BaseButton
        mode="add-car"
        link
        :to="addCarLink"
      >
        +
      </BaseButton>
    </section>
  </BaseCard>
</template>
```

```
<template>
  <li class="home-car">
    <RouterLink
      :to="{ name: 'CarInfo', params: { carId: id, id: userId } }"
    >
      <div class="image-wrapper">
        <img
          class="image-wrapper__img"
          :src="image"
          :alt="brand"
        >
      </div>
      <div class="short-info-wrapper">
        <div class="short-info-wrapper__title">
          {{ brand }} {{ model }}
        </div>
        <div class="info-items">
          <div class="info-item">
            <img src="@/assets/images/home-car-icons/milage.svg" alt="milage">
            <span>{{ milage }} km</span>
          </div>
          <div class="info-item">
            <img src="@/assets/images/home-car-icons/fuel.svg" alt="fuel">
            <span>{{ engine }}</span>
          </div>
          <div class="info-item">
            <img src="@/assets/images/home-car-icons/calendar.svg" alt="fuel">
            <span>{{ year }}</span>
          </div>
          <div class="info-item info-items__car-number">
            {{ carNumber }}
          </div>
        </div>
      </div>
    </RouterLink>
  </li>
</template>
```

Рис.3.13. Шаблони компонентів CarsList та HomeCar

Компонент CarInfo, зображений на рисунку 3.14, містить всю надану користувачем інформацію про автомобіль. Цей компонент забезпечує зручний інтерфейс для перегляду та редагування інформації про автомобіль, включаючи поточний пробіг, рік випуску, тип палива, потужність, номер авто та VIN-код.

```vue
<template>
  <BaseCard v-if="selectedCar">
    <div class="title">
      {{ titleCaption }}
    </div>
    <div v-if="!isEditing" class="car-info">
      <img…
      >
      <div class="header-wrapper">
        <h2>
          {{ title }}
        </h2>
        <BaseButton mode="primary-btn" @click="toggleEdit">
          <span>Змінити</span>
          <svg…
          </svg>
        </BaseButton>
      </div>
      <div class="info-wrapper car-info__wrapper">
        <div class="info-item">
          <div class="info-item__hint">
            Поточний пробіг:
          </div>
          <div class="info-item__value">
            {{ milage }} km
          </div>
        </div>
        <div class="info-item">
          <div class="info-item__hint">
            Рік випуску:
          </div>
          <div class="info-item__value">
            {{ year }}
          </div>
        </div>
        <div class="info-item">
          <div class="info-item__hint">
            Тип палива:
          </div>
          <div class="info-item__value">
            {{ fuelType }}
          </div>
        </div>
        <div class="info-item">
          <div class="info-item__hint">
            Потужність:
          </div>
          <div class="info-item__value">
            {{ capacity }}
          </div>
        </div>
        <div class="info-item">
          <div class="info-item__hint">
            Номер авто:
          </div>
          <div class="info-item__value">
            {{ carNumber }}
          </div>
        </div>
        <div class="info-item info-wrapper__vin">
          <div class="info-item__hint info-wrapper__vin__hint">
            VIN-код:
          </div>
          <div class="info-item__value info-wrapper__vin__value">
            {{ vinCode }}
          </div>
          <BaseButton
            mode="outline-btn"
            class="info-wrapper__vin__btn"
            @click="copyToClipboard(vinCode)"
          >
            <svg…
            </svg>

            <span>Копіювати</span>
          </BaseButton>
        </div>
      </div>
      <BaseButton mode="remove-btn">
        <span>Видалити</span>
      </BaseButton>
    </div>
    <CarForm
      v-else
      :car="selectedCar"
      @submit-data="submitForm"
      @cancel="cancelForm"
    >
      <template #cancel>
        Відмінити
      </template>
      <template #submit>
        Зберегти
      </template>
    </CarForm>
  </BaseCard>
</template>
```

Рис.3.14 Шаблон компоненту CarInfo

4 ТЕСТУВАННЯ

4.1 Підходи тестування

Для перевірки застосунку «Автомобільний менеджер» обрано мануальний підхід тестування. Особливістю даного тестування є те, що перевірка відбувається без застосування додаткових інструментів, тестувальник вручну вводить тестові дані, виявляє помилки та дефекти, перевіряє взаємодію модулей та компонентів, та всієї програми. Це дозволить перевірити, як кінцевий користувач буде взаємодіяти із застосунком та які ризики можуть виникнути.

Під час мануального тестування перевіряються наступні аспекти:

- Правильність роботи функцій та їх реакції на дії користувача.
- Відповідність інтерфейсу користувача до прототипу дизайну.
- Підтримка web-застосунку різними браузерами.
- Перевірка авторизації та аутентифікації.

Перевірка інтерфейсу включала у себе порівняння зовнішній вигляд готового застосунку із макетами та прототипами, що були розроблені на етапі планування та визначення вимог. Використовувався підхід візуального порівняння. Кожна сторінка та форма були перевірені на відповідність. Окрім цього, було надано доступ групі користувачів, що тестували зручність використання об’єктів інтерфейсу та надавали зворотній зв’язок про їх досвід користування.

Для перевірки на правильність роботи функцій використано тестування чек-листів та дослідницьке тестування.

Тестування на основі чек-листів використовує готові чек-листи, що описують перевірки, що необхідно провести для визначення правильності роботи функцій та компонентів програми. Список чек-листів складається з номеру, назви кроку, статусу, тестових даних та коментаря, за необхідністю. Використання

даного підходу забезпечить легкий підхід до тестування web-застосунку, оскільки чітко описує його структуру та визначає дані, що мають бути введені для перевірки. Також, це забезпечує ефективність у процесі виявлення помилок та дефектів.

Тестування безпеки перевіряє вразливість системи на отримання несанкціонованого доступу до даних користувачів. У випадку тестування web-застосунку «Автомобільний менеджер», необхідно перевірити авторизацію та аутентифікацію користувача у застосунку.

Окрім цього, необхідно перевірити, що застосунок відкривається через такі браузери як: Google Chrome, Firefox, Opera. Важливо, щоб у кожному браузері web-застосунок відображався коректно та працював відповідно вимогам.

Після виправлення помилок, що були виявлені під час функціонального тестування, вирішено провести дослідницьке тестування, яке представляє собою перевірку, що не ґрунтуються на створених сценаріях. Під час даного підходу, тестувальники самостійно досліджують програму, намагаючись знайти та вгадати вразливі частини програмного забезпечення, що залишились після основного етапу тестування.

4.2 Опис процесу тестування

Для перевірки реєстрації та авторизації створено чек-листи:

Номер: 1

Назва кроку: Зареєструватись як новий користувач з валідними даними

Статус: пройшов

Тестові дані: email: testmail@gmail.com, пароль: password

Коментар: Google Chrome версія 125.0.6422.33

Номер: 2

Назва кроку: Зареєструватись як новий користувач без вводу даних

Статус: пройшов

Тестові дані: нічого не вводити

Коментар: Google Chrome версія 125.0.6422.33

Номер: 3

Назва кроку: Зареєструватись як новий користувач з невалідним email

Статус: пройшов

Тестові дані: email: testmailgmail.com, пароль: password

Коментар: Opera версія 110.0.5130.23

Номер: 4

Назва кроку: Зареєструватись як новий користувач з невалідним паролем

Статус: пройшов

Тестові дані: email: testmail@gmail.com, пароль: pass

Коментар: Firefox версія 115.0

Номер: 5

Назва кроку: Авторизуватись з валідними даними

Статус: пройшов

Тестові дані: email: testmail@gmail.com, пароль: password

Коментар: Google Chrome версія 125.0.6422.33

Номер: 6

Назва кроку: Авторизуватись з невалідним email

Статус: пройшов

Тестові дані: email: testmail@gmailcom, пароль: password

Коментар: Google Chrome версія 125.0.6422.33

Номер: 7

Назва кроку: Авторизуватись з невалідним паролем

Статус: пройшов

Тестові дані: email: testmail@gmailcom, пароль: word

Коментар: Google Chrome версія 125.0.6422.33

Для перевірки функції створення та редагування машини є наступні чек-листи:

Номер: 8

Назва кроку: Додати нову машину

Статус: пройшов

Тестові дані: Бренд: Toyota, Модель: Tundra TRD Pro, Рік випуску: 2021, Поточний пробіг(км): 54000, Приблизний середньомісячний пробіг(км): 1500, Тип палива: бензин, Об’єм двигуна: 3400, Номер авто: AA1234BB, VIN код: 2FMZA51695BA06174, Посилання: https://www.google.com/imgres

Коментар: Firefox версія 115.0

Номер: 9

Назва кроку: Додати нову машину без вводу даних

Статус: пройшов

Тестові дані: нічого не вводити

Коментар: Firefox версія 115.0

Номер: 10

Назва кроку: Додати нову машину без бренду

Статус: пройшов

Тестові дані: Модель: Tundra TRD Pro, Рік випуску: 2021, Поточний пробіг(км): 54000, Приблизний середньомісячний пробіг(км): 1500, Тип палива: бензин, Об’єм двигуна: 3400, Номер авто: AA1234BB, VIN код: 2FMZA51695BA06174, Посилання на зображення: https://www.google.com/imgres.

Коментар: Firefox версія 115.0

Номер: 11

Назва кроку: Додати нову машину без номеру авто

Статус: пройшов

Тестові дані: Модель: Tundra TRD Pro, Рік випуску: 2021, Поточний пробіг(км): 54000, Приблизний середньомісячний пробіг(км): 1500, Тип палива: бензин, Об’єм двигуна: 3400, VIN код: 2FMZA51695BA06174, Посилання на зображення: https://www.google.com/imgres.

Коментар: Firefox версія 115.0

Номер: 12

Назва кроку: Редагувати інформацію про машину

Статус: пройшов

Тестові дані: Рік випуску: 2022, Тип палива: газ/бензин, VIN код: 1FUJA6CKX7LX09358.

Коментар: Opera версія 110.0.5130.23

Перевірка створення та редагування записів відображена у чек-листах:

Номер: 13

Назва кроку: Створити запис

Статус: пройшов

Тестові дані: Тип дії: заміна, Деталь/рідина: свічка запалювання, Пробіг:43000, увімкнути отримання підказки, Дата: 31.05.2024

Коментар: Opera версія 110.0.5130.23

Номер: 14

Назва кроку: Створити запис без вводу даних

Статус: пройшов

Тестові дані: нічого не вводити

Коментар: Google Chrome версія 125.0.6422.33

Номер: 15

Назва кроку: Створити запис без пробігу

Статус: пройшов

Тестові дані: Тип дії: заміна, Деталь/рідина: свічка запалювання, увімкнути отримання підказки, Дата: 24.05.2024

Коментар: Google Chrome версія 125.0.6422.33

Номер: 16

Назва кроку: Редагувати запис

Статус: пройшов

Тестові дані: Тип дії: встановлення.

Коментар: Firefox версія 115.0

Перевірка зміни статусу створеного запису:

Номер: 17

Назва кроку: Відмітити запису на «виконаний»

Статус: пройшов

Тестові дані: натиснути на галочку для зміни статусу на «виконано»

Коментар: Firefox версія 115.0

Номер: 18

Назва кроку: Видалити запис

Статус: пройшов

Тестові дані: натиснути на смітник для видалення

Коментар: Firefox версія 115.0

Після проведення основного тестування із застосування чек-листів, проведено дослідницьке тестування для виявлення дефектів, що могли залишитись. За результатом проведеного тестування не виявлено жодних невиправлених дефектів.

Етап тестування завершено успішно. За результатами проведеної перевірки визначено, що додаток відповідає поставленим умовам та вимогам. Інтерфейс користувача відповідає створеним на етапі планування макетам. Усі чек-листи були виконанні. Застосунок готовий до представлення для кінцевого користувача. Всі функції працюють правильно.

ВИСНОВКИ

Проаналізовано існуючі рішення для обліку обслуговування автомобілів, зокрема, такі популярні застосунки, як Simply Auto: Car Maintenance, My Car – пальне, My Car Service - Керування автомобілем. В результаті аналізу було визначено, що ці застосунки мають широкий спектр функцій, але не завжди повністю відповідають індивідуальним потребам користувачів. Аналіз цих рішень допоміг визначити ключові вимоги до розробки власного Web-застосунку.

У процесі аналізу було встановлено основні переваги та недоліки існуючих рішень. Переваги включають широкий набір функцій, легкість у використанні та мобільність. Однак, недоліками виявилися обмежені можливості персоналізації, відсутність деяких специфічних функцій та потреба у постійному інтернет-з'єднанні. Ця інформація була використана для покращення функціональності та зручності нового Web-застосунку.

Сформульовано функціональні та нефункціональні вимоги до Web-застосунку "Автомобільний менеджер". Основні функціональні вимоги включають можливість додавання, редагування та видалення даних про автомобілі, ведення історії обслуговування, надання підказок для планового обслуговування, автентифікацію користувачів та зберігання даних у реальному часі. Нефункціональні вимоги включають високу продуктивність, безпеку даних, зручний інтерфейс та адаптивний дизайн для різних пристроїв.

Проведено ретельний аналіз інструментів для створення Web-застосунку. Вибір фреймворку Vue.js для фронтенду був обґрунтований його простотою, продуктивністю та активною підтримкою спільноти. Для бекенду обрано Google Firebase через його потужні можливості зберігання даних у реальному часі, автентифікації користувачів та легкість інтеграції з фронтендом.

На основі проведених досліджень та аналізу було розроблено Web-застосунок "Автомобільний менеджер". Застосунок дозволяє користувачам легко управляти даними про свої автомобілі, вести облік історії обслуговування,

отримувати підказки для планового обслуговування, а також забезпечує зручний та інтуїтивно зрозумілий інтерфейс.

Після завершення розробки було проведено комплексне тестування застосунку для забезпечення його стабільної роботи та відповідності заданим вимогам. Тестування включало перевірку функціональності, продуктивності, безпеки та зручності використання.

Усі задачі дипломної роботи виконані. Дипломна робота пройшла апробацію на конференціях:

1. Куйдін В.С., Яскевич В.О. Доцільність розробки Web-застосунку “Автомобільний менеджер”. Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT», 18 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез, К.: ДУІКТ, 2024. С. 71-72.
2. Куйдін В.С., Яскевич В.О. Інструменти хмарних технологій Firebase. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2024. С. 398-399.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jinpyo H. The Relationship Benefits of Auto Maintenance and Repair Service: A Case Study of Korea / H. Jinpyo, K. Boyoung, O. Sungho. // Behavioral sciences. – 2020. – №115. – С. 5–7.
2. A Comparison Study of The Brand Loyalty of Perodua and Proton Car Owners: A Case Study in Kota Bharu / Siti Haslini Binti Zakaria, Siti Hasma Hajar Binti Mat Zin, Nur Syafizan binti Mohd Sufter, Nurain binti Samsury. // International Journal of Academic Research in Business and Social Sciences. – 2023. – №4. – С. 330 – 346.
3. Simply Auto: Car Maintenance. URL: https://simplyauto.app/.
4. My Car – пальне. URL: https://play.google.com/store/apps/details?id=com.kineapps.mycar&hl=uk.
5. My Car Service - Керування автомобілем. URL: https://play.google.com/store/apps/details?id=com.MyCarService.mycarservice&hl=en_US.
6. THE USAGE OF VUE JS FRAMEWORK FOR WEB APPLICATION CREATION / P. Peter, T. Matúš. // Mesterseges intelligencia. – 2020. – №2. – С. 61–72.
7. A preliminary empirical study of react library related questions shared on stack overflow / Ganno Tribuana Kurniaji, Yusuf Sulistyo Nugroho, Syful Islam. // Computer Science and Information Technologies. – 2023. – №1. – С. 14–23.
8. Web Development Using Angular: A Case Study / Amarpreet Kaur Sahani, Pawan Singh, VijiPriya Jeyamani. // A2Z Journals. – 2020. – №2. – С. 1–7.
9. State of JavaScript. URL: https://stateofjs.com/en-US.
10. Figma. URL: https://www.figma.com.
11. GIT. URL: https://git-scm.com/

12. Pankaj Chougale. FIREBASE - OVERVIEW AND USAGE / Pankaj Chougale, Vaibhav Yadav, Dr. Anil Gaikwad. // International Research Journal of Modernization in Engineering Technology and Science. – 2021. – №3. – С. 1179.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку «Автомобільний менеджер» за допомогою фреймворку Vue.js

Виконав студент 4 курсу
групи ПД-43
Куйдін Владислав Сергійович
Керівник роботи
к.т.н., доцент кафедри Яскевич Владислав Олександрович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – удосконалення обліку обслуговування автомобілів за рахунок Web-застосунку “Автомобільний менеджер” з використанням фреймворку Vue.js.

Об’єкт дослідження – процес обліку обслуговування автомобілів.

Предмет дослідження – Web-застосунок для обліку обслуговування автомобілів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати інструменти розробки Web-застосунку та порівняти існуючі рішення менеджменту обслуговування автомобілів.
2. Скласти вимоги до програмного забезпечення.
3. Вибрати інструменти розробки Web-застосунку “Автомобільний менеджер”.
4. Створити прототип інтерфейсу користувача для Web-застосунку “Автомобільний менеджер”.
5. Розробити застосунок “Автомобільний менеджер” з використанням фреймворку Vue.js.
6. Протестувати Web-застосунок для обліку обслуговування автомобілів.

АНАЛІЗ АНАЛОГІВ

Застосунки / Характеристики	**Simply Auto: Car Maintenance**	**My Car – пальне**	**My Car Service - Керування автомобілем**	**Автомобільний менеджер**
Платформи	ОС Android, IOS, Web-застосунок(beta)	ОС Android, IOS, Windows, Web-застосунок	IOS та Android	Web-застосунок
Додавання/редагування/ видалення автомобілів	+	+	+	+
Підказки по обслуговуванню	+	-	-	+
Синхронізація даних	+	+	+	+
Автентифікація	+	+	+	+
Націлення на звужений спектр послуг	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Користувач повинен мати можливість увійти в систему, використовуючи свої дані для входу за допомогою Firebase Auth.
2. Користувач повинен мати можливість додавати, редагувати та видаляти інформацію про транспортний засіб.
3. Користувач повинен мати можливість додавати, редагувати та видаляти інформацію про задачі, які потрібно виконати або були виконані в транспортному засобі.
4. Система враховує дані про виконані задачі для генерації підказок для заміни деталей(рідин, фільтрів) після виконання подібної задачі.

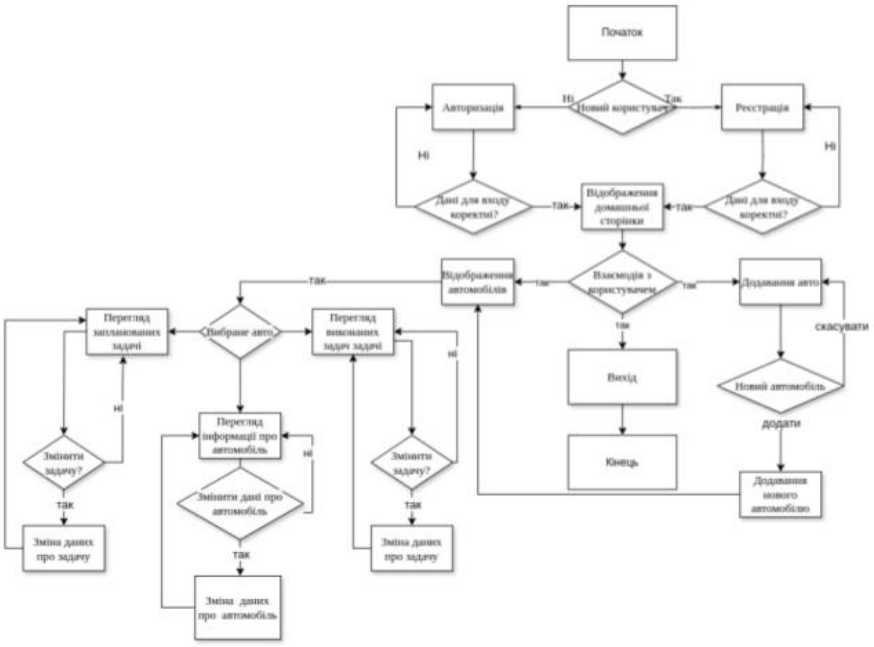
Нефункціональні вимоги:

1. Створити інтерфейс користувача за стандартами проектування Material design.
2. Захист від несанкціонованого доступу за допомогою технології Firebase Auth.
3. Локалізація українською мовою.

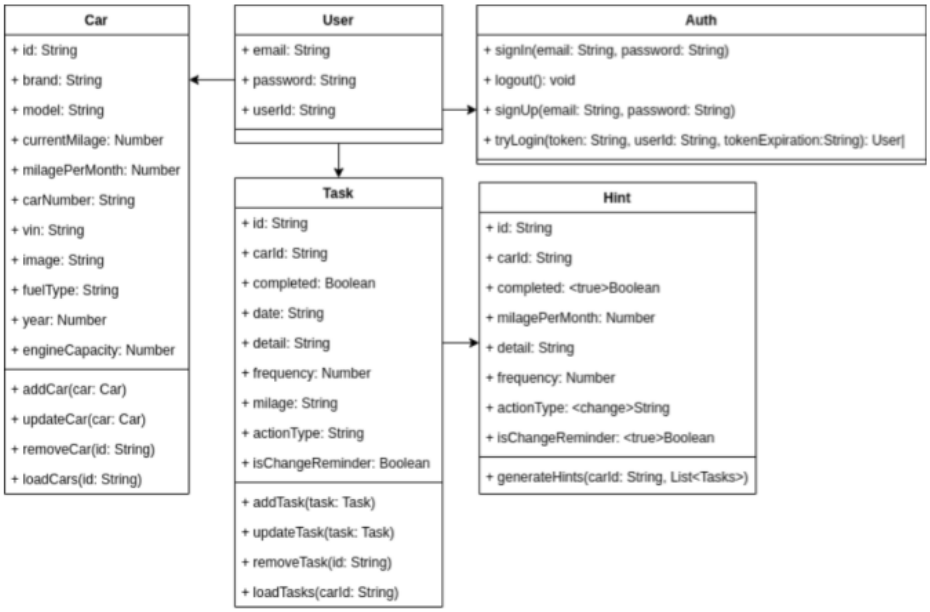
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

БЛОК-СХЕМА РОБОТИ ПРОГРАМИ

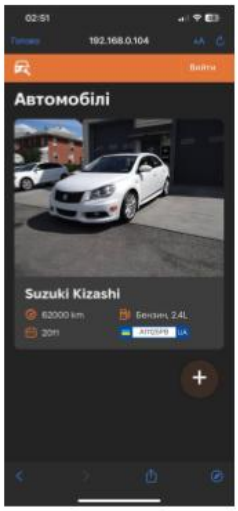


ДІАГРАМА КЛАСІВ



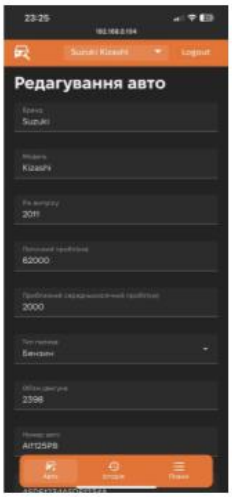
ЕКРАННІ ФОРМИ

Вхід у систему



Головна сторінка

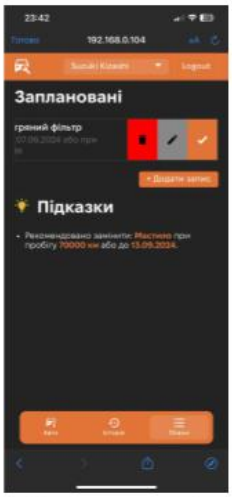
Інформація про вибране авто



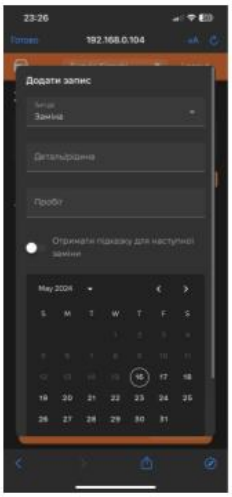
Зміна інформації про авто

ЕКРАННІ ФОРМИ

Виконані задачі



Заплановані задачі та підказки



Діалогове вікно для додавання записів

Відео-демонстрація роботи застосунку

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Яскевич В.О., Куйдін В.С.Доцільність розробки web-застосунку «автомобільний менеджер». Сучасний стан та перспективи розвитку IoT: Матеріали п’ятої міжнародної науково-технічної конференції. Збірник тез. 18.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. Подано до друку.
2. Яскевич В.О., Куйдін В.С. Інструменти хмарних технологій Firebase.Застосування програмного забезпечення в інформаційно-комунікаційних технологіях: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 398.

ВИСНОВКИ

1.Проаналізовано існуючі рішення для обліку обслуговування автомобілів: Simply Auto: Car Maintenance, My Car – пальне, My Car Service - Керування автомобілем.

2.Встановлено переваги та недоліки існуючих Web-застосунків.

3.Сформульовано функціональні та нефункціональні вимоги до Web-застосунку “Автомобільний менеджер”. Застосунок забезпечує додавання, редагування та видалення даних про автомобілі, історію обслуговування та підказки для планового обслуговування.

4.Проаналізовано інструменти розробки програмного забезпечення та використано Vue.js для фронтенду та Firebase для бекенду.

5.Розроблено Web-застосунку “Автомобільний менеджер” за допомогою фреймворку Vue.js.

6.Проведено тестування Web-застосунку “Автомобільний менеджер”.

ДЯКУЮ ЗА УВАГУ!