МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

01 червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Мобільний застосунок контролю автомобільних витрат»

здобувачки групи ІН – 01 Гріцун Лариси Олександрівни

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

Лариса ГРІЦУН

(підпис)

Керівник,

асистент кафедри комп'ютерних наук,

кандидат фізико-математичних наук Олександр ВЛАСЕНКО

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувачки групи ІН-01 Гріцун Лариси Олександрівни

1. Тема роботи: «Мобільний застосунок контролю автомобільних витрат» затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки мобільних застосунків. 3) Розробка мобільного застосунку контролю автомобільних витрат. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «08» квітня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.05.24-09.05.24	
2	*Огляд технологій, що використовуються для створення мобільних додатків*	10.05.24-12.05.24	
3	*Розробка інтерфейсу користувача*	13.05.24-15.05.24	
4	*Розробка серверної частини*	15.05.24-20.05.24	
5	*Розробка функціональності додатку*	20.05.24-25.05.24	
6	*Тестування додатку та аналіз отриманих результатів*	26.05.24	
7	*Оформлення пояснювальної записки до кваліфікаційної роботи*	27.05.24	

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 90 стр., 22 рис., 1 додаток, 17 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки надає зручний інструмент для відстеження, аналізу та планування витрат на автомобіль, відповідає потребам сучасного ринку та допомагає водіям краще керувати своїми фінансами.

Об'єкт дослідження – процес контролю витрат на експлуатацію автомобілів за допомогою мобільних додатків.

Мета роботи – розробка мобільного додатку для ефективного контролю та управління автомобільними витратами для покращення фінансового планування власників автомобілів.

Методи дослідження – дослідження існуючих ринкових, аналіз сучасних технологій і фреймворків для створення мобільних додатків, розробка прототипів для перевірки функціональності та зручності інтерфейсу, тестування продукту з метою його вдосконалення.

Результати – розроблено мобільний застосунок для відстежування витрат пального та сервісу автомобіля, що дозволяє користувачеві вносити інформацію про автомобіль, надає можливість перегляду останніх витрат та заправлень, додавання записів про заправлення та витрати на сервіс. Застосунок надає можливість перегляду статистики витрат у вигляді графіка, що допомагає користувачам краще розуміти та планувати свої фінанси. Тестування застосунку на реальних даних автомобільних витрат, дозволило перевірити правильність функціонування та відповідність очікуваним результатам.

АВТОМОБІЛЬНІ ВИТРАТИ, КЕРУВАННЯ ФІНАНСАМИ, МОБІЛЬНИЙ ДОДАТОК, IONIC, NESTJS, REACT

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи. Автомобілі стали базовою потребою багатьох людей, забезпечуючи зручний та швидкий спосіб пересування. Контроль над витратами, пов'язаними з експлуатацією автомобіля, є невід'ємною частиною успішного володіння транспортним засобом.

Для підтримки автомобіля в належному стані, своєчасного проходження технічного огляду, заміни витратних матеріалів та контролю витрат палива виникає потреба в розробці зручного мобільного застосунку, де всі дані про авто будуть зберігатися та відслідковуватися в одному місці.

Мобільний застосунок для контролю автомобільних витрат допоможе власникам авто оптимізувати витрати, підвищити продуктивність та забезпечити безпеку експлуатації транспортного засобу.

Актуальність. Розробка мобільного застосунку для контролю автомобільних витрат є актуальною та перспективною темою, яка задовольняє потреби широкої аудиторії. Багато водіїв відчувають потребу у зручному інструменті для відстеження своїх витрат на транспорт. Зростання цін на пальне та запчастини підвищує важливість ведення обліку витрат на автомобіль. Нині багато водіїв ведуть облік витрат вручну, записуючи дані в блокноти або таблиці. Цей метод є незручним, трудомістким і не забезпечує можливості отримання детальної статистики та аналітики. Мобільний застосунок, який завжди під рукою на смартфоні, дозволяє швидко та легко вносити дані про витрати та заправки. Такий застосунок може допомогти водіям економити гроші, планувати бюджет, оптимізувати витрати та краще контролювати експлуатацію своїх транспортних засобів.

Об’єкт дослідження. Об'єктом дослідження є процеси контролю витрат на експлуатацію автомобілів за допомогою мобільних додатків. Дослідження включає аналіз потреб користувачів, огляд існуючих рішень, визначення

функціональних вимог та розробку інтуїтивного інтерфейсу. Особлива увага приділяється аналізу технологічним можливостям та необхідним функціям для ефективного управління витратами. Метою є створення інноваційного інструменту, який полегшить водіям контроль за витратами на транспортні засоби, сприяючи оптимізації витрат та ефективному плануванню бюджету.

Предмет дослідження. Предметом дослідження є кросплатформенний мобільний додаток для контролю автомобільних витрат, розроблений на базі Ionic SDK з використанням React для фронтенд-розробки та Nest.js для бекенд-розробки з інтеграцією PouchDB. Додаток дозволяє користувачам ефективно відстежувати витрати на транспортні засоби, включаючи витрати на пальне, технічне обслуговування та страхування.

Гіпотеза. Впровадження кросплатформенного мобільного застосунку для контролю автомобільних витрат дозволить водіям ефективніше керувати витратами на транспортні засоби. Це досягатиметься через спрощення процесу обліку витрат, забезпечення доступу до даних у режимі офлайн, а також надання інтуїтивно зрозумілого інтерфейсу для введення та аналізу даних.

Новизна. Новизна розробки мобільного застосунку для контролю автомобільних витрат полягає у поєднанні Ionic для створення клієнтської частини та Nest.js для розробки серверної, а також в інтеграції PouchDB для забезпечення локального сховища даних. Таке поєднання технологій дозволить створити потужний та функціональний додаток для контролю автомобільних витрат, забезпечуючи при цьому зручність користування та можливість працювати без підключення до мережі завдяки використанню сучасних вебтехнологій.

Структура. Дана робота складається зі вступу, інформаційного огляду, постановки задачі, вибору методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Вибір типу застосунку

Мобільний застосунок – це програмне забезпечення, спеціально розроблене для використання на мобільних пристроях. Мобільні додатки дозволяють користувачам виконувати різноманітні завдання, такі як доступ до інформації, взаємодія з іншими користувачами, розваги, продуктивність, навчання, бізнес тощо. Зазвичай вони встановлюються на мобільний пристрій і працюють незалежно від інтернет-браузера.[1]

Мобільні застосунки за своїм призначенням та цільовою аудиторією поділяються на кілька типів:

1. Ігрові.
2. Додатки для бізнесу.
3. Навчальні.
4. Для повсякденних задач.
5. Комерційні.
6. Розважальні.
7. Утилітарні.
8. Для подорожей.

Додаток про відстеження витрат пального належить саме до додатків, що допомагають користувачам спростити та відстежувати виконання повсякденних задач.

За технологіями та методами розробки існують 3 основних типи мобільних додатків:

Progressive Web Apps – це вебпрограми, які працюють на різних платформах і пристроях. Доступ до них здійснюється через веббраузер і забезпечує роботу, схожу на програму, з автономною функцією та можливістю надсилання push-повідомлень. Програми цього типу розроблено з

використанням таких технологій як HTML, CSS і JavaScript, але пропонують функції, традиційно пов’язані з рідними мобільними програмами.

Переваги: висока продуктивність та якісний UI.

Недоліки: висока вартість розробки, відсутність кросплатформеності та складнощі з підтримкою одразу для Android та iOS.

Native Applications розробляються для певної платформи, наприклад Android або iOS, з використанням рідних мов програмування платформи та інструментів розробки, таких як Swift для iOS і Java або Kotlin для Android. Ці типи додатків пропонують найкращу продуктивність і доступ до функцій пристрою, але потребують окремої розробки для кожної платформи.

Переваги: відносна простота розробки, підтримується і в браузері і в будь-якій іншій платформі, потребують мінімальну кількість пам’яті девайсу.

Недоліки: потребують постійного інтернет-з’єднання, не мають всього доступу до API.

Hybrid Applications поєднують елементи як веб, так і нативних програм. Подібно до PWA, ці програми розробляються з використанням HTML, CSS і JavaScript, а потім поміщаються в нативний контейнер, який дозволяє їм працювати на кількох платформах.

Переваги: простота та швидкість розробки, база для програмування на всіх платформах, доступ майже до всього API.

Недоліки: недостатня швидкість роботи та продуктивність у порівнянні з нативними додатками.[2]

Для розробки додатку про відстеження витрат пального доцільно обрати саме гібридний тип додатку, так як це дозволить використовуючи технології HTML, CSS, Javascript та розробити кросплатформенний додаток зі швидкою роботою.

1.2 Постановка задачі

Метою роботи є розробка мобільного додатку для ефективного контролю та управління автомобільними витратами для покращення фінансового планування власників автомобілів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Аналіз потреб користувачів – визначення потреб та очікувань цільової аудиторії від додатку, включаючи вимоги до функціональності та інтерфейсу;

2) Визначення функціональних вимог – визначення технологій для розробки додатку, вибір мов програмування, фреймворків, інструментів та бази даних;

3) Розробка інтерфейсу користувача – проєктування інтуїтивно зрозумілого та зручного інтерфейсу, який дозволить користувачам легко вводити дані про витрати та отримувати доступ до статистики;

4) Розробка функціональності – реалізація функцій, визначених на попередніх етапах, включаючи зберігання та обробку даних, генерацію звітів та графіків.

5) Тестування – проведення тестування додатку для виявлення помилок та недоліків.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Огляд архітектури клієнтської частини

Архітектура додатку визначає як будуть взаємодіяти компоненти між собою, організацію логіки та забезпечення продуктивності та стабільності роботи. Вибір правильної архітектури та технологій є ключовим для досягнення поставлених цілей та задоволення потреб користувачів. Розглянемо відомості про актуальні технології розробки мобільних додатків у порівняльній таблиці:

Таблиця 2.1. Характеристика основних технологій[5, 8, 17]

Назва технології	**Мова програмування**	**Переваги**	**Недоліки**
React Native	JavaScript	- Висока продуктивність; - Можливість використовувати одну кодову базу для обох платформ; - Зручна розробка; - Швидке оновлення; - Велика спільнота розробників;	- Потребує написання коду для певних функцій; - Обмежений доступ до деяких функцій пристрою; - Проблеми при сумісності при виході оновлень; - Відносно невелика кількість користувацьких модулів;
Flutter	Dart	- Єдина кодова база; - Швидка розробка; - Висока продуктивність; - Схожість з нативним дизайном; - Кросплатформенність; - Велика кількість пакетів та бібліотек для інтегрування в проєкт;	- Великий розмір проєкту; - Не підтримує інтеграцію з деякими зовнішніми модулями та API; - Високий поріг входження в розуміння архітектури інструментарію;
Xamarin	C#	- Сумісність з MVC(Model-View-Controller) та MVVM(Model-View-ViewModel) архітектурами;	- Потребує проміжного рівня для зв'язку з нативним кодом;

		- Спрощена система підтримки інструментарію; - Велика та різноманітна кількість компонентів; - Якість та ефективність додатків;	- Не досить сильно розвинена спільнота розробників;
Ionic	Angular, React, Vue	- Швидка розробка та час виходу на ринок порівняно з різними програмами для iOS/Android; - Можна розробляти в браузері за винятком функціоналу, що доступний тільки на мобільному девайсі; - Багато доступних і простих у використанні компонентів інтерфейсу користувача. - Одна кодова база одразу для двох платформ (iOS та Android):	- Обмежений доступ до деяких функцій. - Швидкість роботи може бути меншою порівняно з додатками; - Нестабільність додатку при збірці проєкту (висока можливість збоїв); - Конфлікт та нестабільність зовнішніх доповнень при інтеграції з базовими вбудованими компонентами інструментарію;
Native Script	Angular, Typescript, JavaScript, CSS, Vue.js	- Нативний інтерфейс без використання WebViews. - Повний прямий доступ до Android та iOS API. - Крос-платформенність. - Підтримка важливих бізнес-функцій. - Надійна підтримка бекенду.	- Відсутність великої користувацької бази порівняно з іншими фреймворками.

З-поміж усіх технологій для розробки додатків оберемо Ionic SDK (React), який має численні переваги, які роблять його доречним вибором для створення мобільних додатків.

По-перше, це кросплатформенність додатку, що дозволяє створювати додатки, які працюють як на Android, так і на iOS, а також як вебсайти, використовуючи єдину кодову базу. Це значно полегшує розробку та підтримку додатків для різних платформ.

Другою важливою перевагою є використання стандартних вебтехнологій, таких як HTML, CSS і JavaScript/TypeScript.

Окрім того, Ionic надає широкий вибір готових компонентів і інструментів, що спрощує створення інтерфейсу користувача та робить розробку більш швидкою та ефективною. Загалом, обираючи Ionic, розробник отримує потужний інструмент для швидкого створення якісних крос-платформених додатків без необхідності вивчення нових технологій і мов програмування.

2.2 Огляд архітектури серверної частини

При розробці мобільного додатку, серверна частина грає критичну роль у забезпеченні ефективної та безпечної роботи додатку. Фреймворк для серверної частини мобільного додатку має відповідати ряду вимог, які забезпечують ефективну та безпечну роботу додатку. Один з ключових аспектів – ефективна обробка запитів та відповідей, щоб забезпечити швидку реакцію додатку на взаємодію користувачів. Фреймворк повинен бути готовий до масштабування, здатний забезпечити стабільну роботу додатку при зростанні навантаження. Наявність вбудованих механізмів захисту від різноманітних атак, таких як XSS, CSRF, SQL ін'єкції, є критичною для запобігання порушенням безпеки.

Фреймворк повинен надавати зручні інструменти для швидкого розгортання та розробки функціоналу, щоб забезпечити ефективний процес розробки. Важливо мати можливість легко інтегрувати різні типи баз даних та використовувати їх в додатку, для забезпечення оптимального управління даними.[11-12]

Таблиця 2.2 Характеристика типів серверних технологій

Назва технології	Мова програмування	Переваги	Недоліки
Express.js	JavaScript	- Легкий у використанні, швидкий розгортання, велика спільнота;	- Обмежений вбудований набір модулів, що потребує використання додаткових модулів;
Nest.js	TypeScript	- Модульність, вбудована підтримка TypeScript, розширені можливості для тестування, структурований підхід до розробки; - Підтримка мікросервісів; - Ядро базується Typescript; - Підтримка GraphQL; - Велика спільнота розробників;	- Витрачає багато ресурсів на підітримку високої продуктивності;
Django	Python	- Швидка розробка, велика кількість готових рішень, вбудована аутентифікація та авторизація; - Легко масштабується; - Швидка розробка; - Велика спільнота розробки;	- Не підходить для маленьких проєктів; - Є монолітним та не має гнучких можливостей в кастомізації

Nest.js – це модульний фреймворк для побудови ефективних та масштабованих серверних додатків на Node.js. Він має ряд переваг, які роблять його ідеальним вибором для реалізації серверної частини мобільного додатку:

1. Модульність – Nest.js пропонує модульну структуру, що дозволяє легко організувати код та робити його повторне використання.

2. Вбудована підтримка TypeScript – має вбудовану підтримку цієї мови програмування, що дозволяє писати більш безпечний та зрозумілий код.

3. Структурований підхід до розробки – пропонує структурований підхід до розробки, що сприяє підтримці та розвитку проєкту з плинністю.

Компоненти Nest.js є ключовими елементами для побудови серверної частини додатку і допомагають організувати код та робити його повторне використання. Компоненти, які можна використати, включають:

1. Контролери (Controllers) відповідають за обробку вхідних запитів та відправку відповідей. Вони приймають запити від клієнта і викликають відповідні методи сервісів для обробки цих запитів. Контролери визначаються за допомогою декораторів та класів.

2. Провайдери (Providers) відповідають за створення та управління залежностями в застосунку. Вони надають іншим компонентам доступ до різних ресурсів, таких як сервіси, бази даних тощо. Провайдери використовуються для ін'єкції залежностей в інші компоненти.

3. Модулі (Modules) дозволяють групувати пов'язаний функціонал разом і організовувати додаток у логічні блоки. Кожен модуль може містити контролери, провайдери, сервіси та інші компоненти, необхідні для реалізації певного функціоналу.

4. Middleware у Nest.js є функціями, які виконуються перед або після обробки кожного запиту. Вони дозволяють виконувати певні дії на кожному етапі обробки запиту, такі як логування, перевірка аутентифікації, обробка помилок тощо.

5. Сервіси (Services) використовуються для реалізації бізнес-логіки додатку. Вони містять методи, які виконують певні операції та взаємодіють з базою даних, іншими сервісами чи провайдерами.

6. Пайплайни (Pipes) дозволяють валідувати та обробляти дані, які надходять від клієнта перед їх обробкою. Вони можуть виконувати перевірки, очищення та трансформації даних, щоб забезпечити їхню коректність та безпеку.

7. Фільтри (Filters) дозволяють перехоплювати та обробляти винятки, які виникають під час обробки запиту. Вони дозволяють зробити глобальну обробку помилок та виконати певні дії, такі як логування помилок чи відправка спеціалізованих відповідей.

8. Винятки (Exceptions) використовуються для керування помилками та винятками, які виникають в додатку. Вони дозволяють створювати та

обробляти різноманітні типи помилок та надавати коректні відповіді клієнтам в разі їх виникнення.

2.3 Огляд бази даних

При виборі бази даних для розробки мобільного застосунку з використанням Ionic, React, Typescript та Nest.js, важливо враховувати різноманітні фактори, такі як типи даних, швидкодія, масштабованість, надійність та інші.

При виборі бази даних для мобільного застосунку слід звернути увагу на такі аспекти:

1. обрати базу даних, яка найбільше підходить для структури даних, що використовується в додатку;
2. переконатися, що база даних може ефективно обробляти очікуване навантаження та масштабуватися з ростом додатку;
3. важливо, щоб база даних була надійною та стійкою до відмов, особливо при роботі в умовах мобільного середовища;
4. переконатися, що база даних може легко інтегруватися з іншими компонентами додатку, такими як серверна частина(Nest.js) та клієнтська сторона(Ionic + React + Typescript). Нижче (табл.2.3) представлений огляд декількох популярних баз даних.

Також не менш важливо обрати тип бази даних, що найбільш зручно та ефективно реалізує зберігання та керування інформацією в додатку. При обиранні між реляційними та NoSQL базами даних важливо враховувати різні фактори, такі як потреби проєкту, структура даних, типи операцій, швидкодія, масштабованість та інші. Реляційні бази даних, які включають такі системи, як PostgreSQL, MySQL та SQLite, відомі своєю структурованістю та підтримкою транзакційної цілісності даних. Вони надійні, мають широкий набір функцій та добре підходять для проєктів зі складними зв'язками між даними.

Таблиця 2.3 – Характеристика типів баз даних

Назва технології	**Мова програмування**	**Переваги**	**Недоліки**
MongoDB	NoSQL	Гнучкість схеми, підтримка розподіленості, добре підходить для обробки великих обсягів даних	Може виникнути проблема з консистентністю даних при великому обсязі транзакцій, висока вартість масштабування
PostgreSQL	Relational	Висока надійність, підтримка транзакцій та цілісності даних, широкі можливості для оптимізації	Схема даних є статичною та важко змінюється, менша гнучкість порівняно з NoSQL
Firebase	NoSQL	Простота використання, реальний час оновлення даних, інтегрована аутентифікація та інші сервіси Google	Обмежені можливості запитів та складних операцій, залежність від сервісів Google
PouchDB	NoSQL	Офлайн синхронізація, вбудована підтримка CouchDB, можливість працювати на різних платформах	Обмежені можливості для великих обсягів даних, не підтримує складних операцій із запитами

З іншого боку, NoSQL бази даних, такі як MongoDB, PouchDB та Firebase, надають гнучкість у роботі з неструктурованими даними, швидкість доступу до інформації та простоту масштабування. Ці бази даних не вимагають фіксованої схеми, що дозволяє зберігати та керувати великими обсягами даних, які можуть мати різну структуру і динамічно змінюватися.

Наприклад, MongoDB використовує документно-орієнтовану модель, де дані зберігаються у форматі JSON-подібних документів. Це дозволяє зручно зберігати складні і вкладені структури даних, що робить MongoDB ідеальним вибором для додатків, які вимагають швидкої та гнучкої обробки даних. PouchDB, у свою чергу, є клієнтською NoSQL базою даних, яка підтримує синхронізацію з CouchDB або будь-яким сервером, що реалізує протокол CouchDB. Це робить PouchDB відмінним вибором для офлайн-додатків, де

важливо мати можливість працювати з даними локально, а потім синхронізувати їх з сервером.

NoSQL бази даних особливо корисні для проєктів з великим обсягом даних або потребами в швидкому доступі до даних, таких як мобільні додатки, де часто виникає необхідність в роботі з неструктурованою або змінною інформацією. У таких випадках, можливість швидкого масштабування та ефективного оброблення великих обсягів неструктурованих даних стає критичною.

NoSQL БД забезпечують високу доступність і стійкість до збоїв, що робить їх надійним вибором для розподілених систем. Наприклад, розподілена архітектура MongoDB дозволяє легко додавати нові вузли до кластера для горизонтального масштабування і підтримує реплікацію для забезпечення відмовостійкості.У нашому проєкті було обрано NoSQL БД PouchDB так як вона найбільше відповідає вимогам до мобільного додатку, побудованому на технологіях Ionic, React, Typescript, Nest.js. PouchDB може працювати як в браузері, так і на сервері, і має вбудовану підтримку CouchDB. [16]

Нижче представлений огляд компонентів та можливостей PouchDB:

1. Компоненти:

- Сховище даних PouchDB надає можливість зберігати дані в локальному сховищі браузера або синхронізувати їх з віддаленим сервером CouchDB.
- Офлайн синхронізація – автоматично синхронізує дані між клієнтом та сервером, навіть коли клієнт працює в офлайн-режимі.
- Безсхемна база даних дозволяє зберігати дані без жорсткої схеми, що робить його гнучким для роботи з різними типами даних.
- Запити та індекси – PouchDB підтримує складні запити та може створювати індекси для швидкого доступу до даних.

2. Приклад використання:

```
import PouchDB from 'pouchdb';
```

```
// Ініціалізація бази даних
const db = new PouchDB('my_database');

// Додавання документу до бази даних
db.put({
    _id: 'my_document',
    title: 'Hello, World!',
    content: 'This is a sample document.'
});

// Отримання документу за його ідентифікатором
db.get('my_document').then(console.log);
```

3. Структура:

- База даних – кожен екземпляр PouchDB відповідає окремій базі даних.

Документи – дані зберігаються у вигляді JSON документів з унікальним ідентифікатором.

2.4 Проєктування функціоналу

Основний акцент при проєктуванні функціональної частини повинен бути спрямований на покриття критично важливих бізнес-функцій мобільного застосунку. Середньостатистичний користувач додатку робитиме стандартні для ведення історії обслуговування автомобіля дії: додаватиме запис про заправку, заміну моторної оливи, гальмівних колодок, антифризу, заміна гуми тощо. Окрім цього, користувач зможе вносити дані пробігу автомобіля для кожного з вищеперерахованих записів, що допоможе своєчасно зробити наступну заміну. Для того, щоб побачити інфографіку витрат за певний період існуватиме можливість переглянути статистику, що стане візуально підсумовувати сукупність записів та витрат в одній шкалі.

Нижче розглянемо детально пріоритетні функції при реалізації додатку:

1. Внесення інформації про автомобіль:

- Назва автомобілю: користувач може ввести назву свого авто для ідентифікації транспортного засобу.

- Рік випуску: запис року випуску авто для належного відстеження його старіння.
- Пробіг: користувач може вносити запис про поточний пробіг автомобіля.
- Зображення: користувач може завантажити зображення автомобіля до системи.

2. Перегляд останніх витрат:

- Додаток дозволяє переглядати історію останніх витрат та заправлень.

3. Додавання заправлення:

- Загальна сума: користувач може вносити загальну суму грошей, витрачену на заправлення.
- Кількість літрів: запис кількості літрів, заправлених автомобілем.
- Дата заправки.

4. Перегляд заправлень:

- Користувач може переглядати історію заправлень, включаючи інформацію про кожне заправлення.

5. Додавання витрати:

- Назва витрати: Введення назви витрати або послуги, на яку були витрачені кошти.
- Пробіг авто на момент створення витрати: Вказання пробігу автомобіля на момент витрати.
- Дата витрати: Фіксація дати, коли була проведена витрата.
- Ціна за роботу: Внесення вартості послуг або роботи.
- Ціна матеріалу: Внесення вартості матеріалів, які були придбані.

6. Перегляд статистики:

- Графік витрат: Додаток надає можливість побудувати графік витрат, який може бути відображений за різними періодами – тиждень, місяць, рік.

2.5 Проєктування дизайну

Для кожного користувача дуже важливо орієнтуватися в інтерфейсі застосунку без будь-яких перешкод і багів. Навігація має бути простою та інтуїтивно зрозумілою.

При розробці мобільного застосунку ми будемо користуватися технікою функціонального мінімалізму, коли зовнішній вигляд програми зведений до мінімальної кількості елементів, відсутня перевантаженість текстом та кнопками, користувач бачить лише найважливіші елементи та функції. Це повинно значним способом покращити користувацький досвід (UX) та вплинути на зручність користування.

Створимо назву додатку, розробимо сторінку завантаження та логотип програми, що зображено на рисунку 2.1. Назва додатку: “Vehicle Wallet”:

Рисунок 2.1 – Логотип додатку

Головна сторінка міститиме дані про автомобіль та історію останніх витрат користувача, що зображено на рисунку 2.2.

З нижньої панелі навігації користувач може перейти на розділ витрат пального. На ній відображаються внесені записи про заправки та можливість додати заправлення, що зображено на рисунках 2.3 – 2.4.

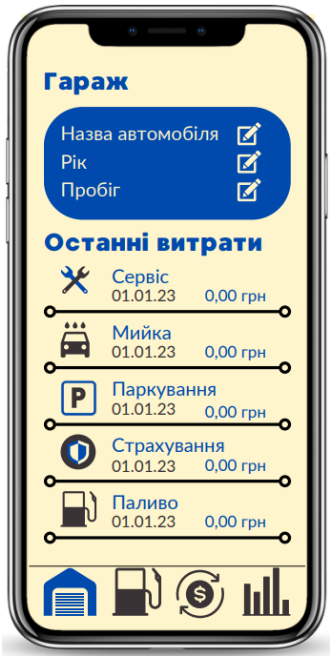


Рисунок 2.2 – Домашня сторінка

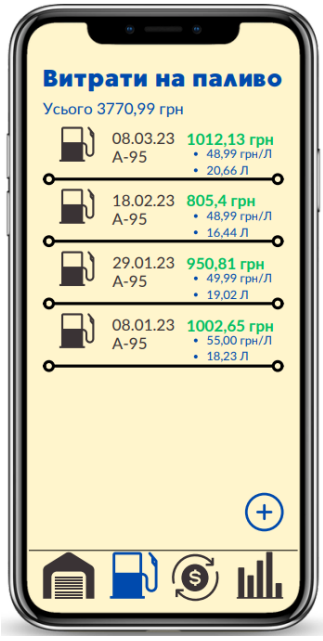


Рисунок 2.3 – Розділ витрати палива

Рисунок 2.3 – Додавання нового заправлення

Основні витрати винесені в окремий розділ. У залежності вид типу витрат при їх створенні обирається відповідне зображення, що зображено на рисунку 2.5.

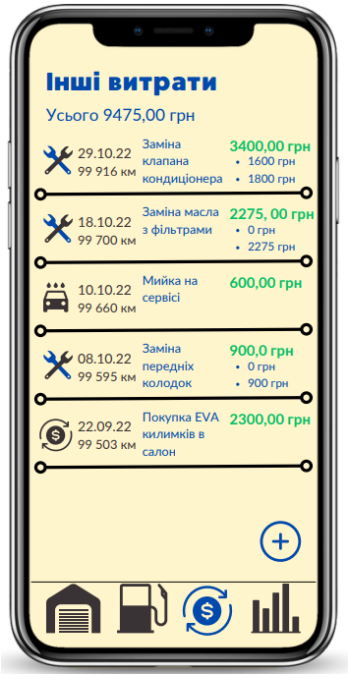


Рисунок 2.4. – Розділ витрат автомобіля

При створенні витрати користувачеві буде запропоновано ввести назву витрати, пробіг автомобіля, дату витрати, ціну роботи якщо вона була виконана та ціну матеріалу, що зображено на рисунку 2.6.

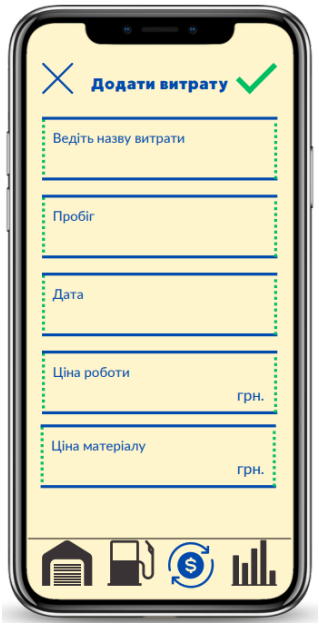


Рисунок 2.5. – Створення нової витрати

Розділ статистика дає можливість переглянути витрати за тиждень, місяць або рік, що зображено на рисунку 2.7.

Рисунок 2.6. – Розділ статистики

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Архітектура проєкту

Проєкт розроблено з використанням наступних технологій, бібліотек та фреймворків:

- React − це JavaScript-бібліотека для побудови користувацьких інтерфейсів. Вона дозволяє розробникам створювати великі вебзастосунки, які можуть змінювати дані, не перезавантажуючи сторінку.

- Vite − це інструмент для збірки, який надає швидке середовище розробнику. Він пропонує швидке завантаження, миттєве оновлення модулів та мульти-сторінкову підтримку з переходом між сторінками.

- TypeScript: це надбудова над JavaScript, яка додає статичні типи.

- Capacitor − кросплатформенний інструмент, який дозволяє створити збірку додатку (.apk файл) або дати можливість для запуску на потрібній операційній системі, наприклад, через Android Studio або XCode.

- Ionic − набір інструментів для розробки гібридних мобільних додатків. Він використовує вебтехнології, такі як HTML, CSS та TypeScript, для створення додатків, які можуть працювати на різних платформах.

- Chart.js − бібліотека для візуалізації даних, яка дозволяє створювати графіки та діаграми в браузері.

- React Router − стандартна бібліотека маршрутизації для React. Вона дозволяє створювати додатки, які можуть переходити між різними сторінками без перезавантаження сторінки, що зображено на рисунку 3.1:

```
<CarImageContext.Provider value={{ carImage, setCarImage }}>
  {showWelcome ? (
    <IonRouterOutlet>
      <Route exact path="/welcome" component={WelcomePage} />
      <WelcomePage />
      <Redirect exact from="/" to="/garage" />
    </IonRouterOutlet>
  ) : (
    <IonTabs onIonTabsDidChange={(e: CustomEvent) => setSelectedTab(e.detail.tab)}>
      <IonRouterOutlet>
        <Route exact path="/settings" component={SettingsPage} />
        <Route exact path="/garage" component={GaragePage} />
        <Route exact path="/fuel" component={FuelPage} />
        <Route exact path="/service" component={ServicePage} />
        <Route exact path="/statistics" component={StatisticsPage} />
        <Route exact path="/add-expense" component={AddExpensePage} />
        <Route exact path="/add-service" component={AddServicePage} />
      </IonRouterOutlet>
```

Рисунок 3.1 – Маршрутизація додатку

3.2 Реалізація основних компонентів

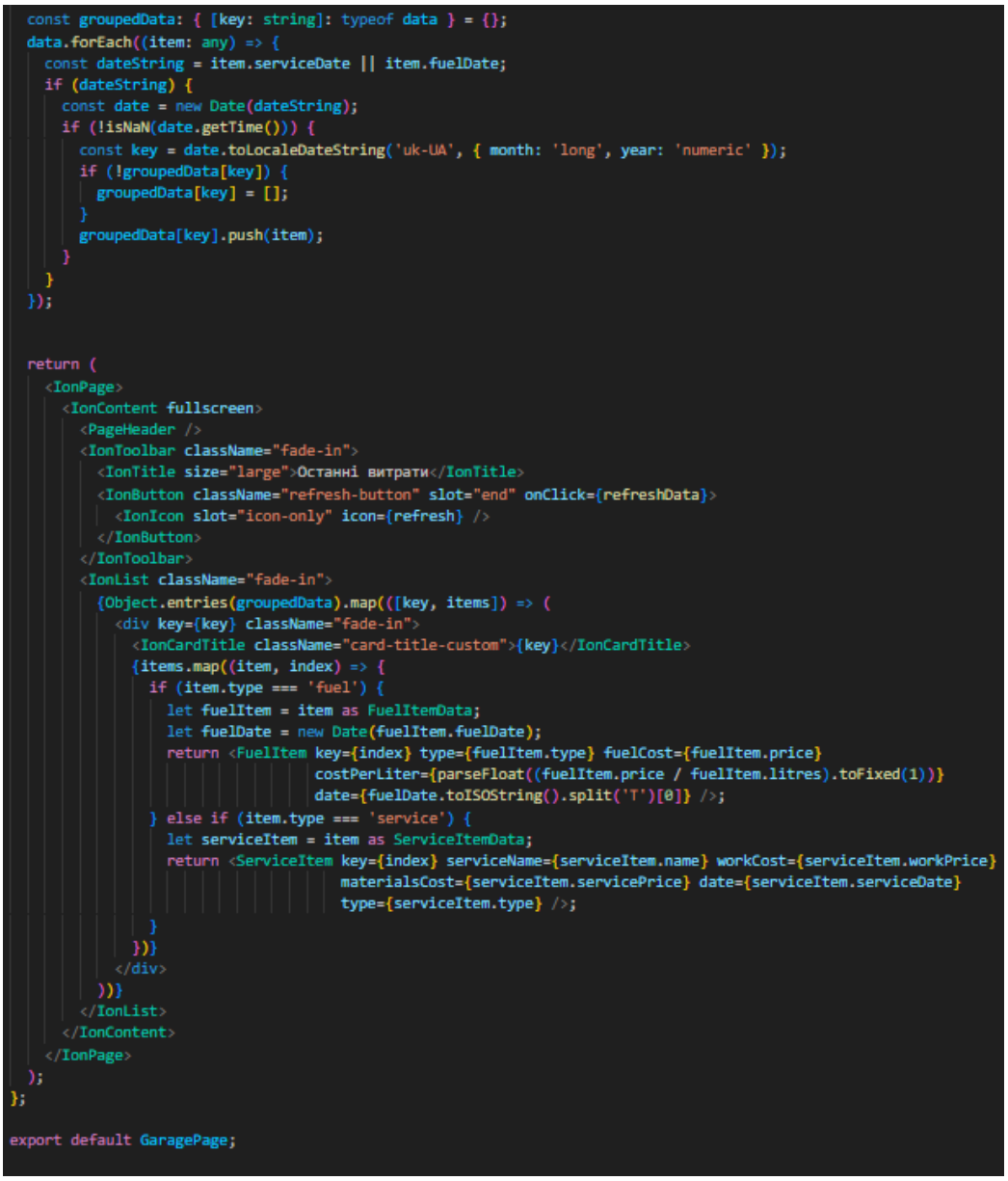
App.tsx є головним компонентом додатку. Він включає в себе всі інші компоненти та відповідає за маршрутизацію між різними сторінками, що зображено на рисунку 3.2.

```
return (
  <IonApp>
    <IonReactRouter>
      <CarImageContext.Provider value={{ carImage, setCarImage }}>
        {showWelcome ? (
          <IonRouterOutlet>
            <Route exact path="/welcome" component={WelcomePage} />
            <WelcomePage />
            <Redirect exact from="/" to="/garage" />
          </IonRouterOutlet>
        ) : (
          <IonTabs onIonTabsDidChange={(e: CustomEvent) => setSelectedTab(e.detail.tab)}>
            <IonRouterOutlet>
              <Route exact path="/settings" component={SettingsPage} />
              <Route exact path="/garage" component={GaragePage} />
              <Route exact path="/fuel" component={FuelPage} />
              <Route exact path="/service" component={ServicePage} />
              <Route exact path="/statistics" component={StatisticsPage} />
              <Route exact path="/add-expense" component={AddExpensePage} />
              <Route exact path="/add-service" component={AddServicePage} />
            </IonRouterOutlet>
            <IonTabBar slot="bottom" className="custom-tab-bar">
              <IonTabButton tab="garage" href="/garage">
                <IonIcon className={`my-icon ${selectedTab === 'garage' ? 'active-tab-icon' : ''}`}
                         icon={GarageIcon}
                         color={selectedTab === 'garage' ? activeColor : defaultColor} />
                <IonLabel color={selectedTab === 'garage' ? activeColor : ''}>Гараж</IonLabel>
              </IonTabButton>
              <IonTabButton tab="fuel" href="/fuel">
                <IonIcon className={`my-icon ${selectedTab === 'fuel' ? 'active-tab-icon' : ''}`}
                         icon={FuelIcon}
                         color={selectedTab === 'fuel' ? activeColor : defaultColor} />
                <IonLabel color={selectedTab === 'fuel' ? activeColor : ''}>Паливо</IonLabel>
              </IonTabButton>
              <IonTabButton tab="service" href="/service">
                <IonIcon className={`my-icon ${selectedTab === 'service' ? 'active-tab-icon' : ''}`}
                         icon={ServiceIcon}
                         color={selectedTab === 'service' ? activeColor : defaultColor} />
                <IonLabel color={selectedTab === 'service' ? activeColor : ''}>Сервіс</IonLabel>
              </IonTabButton>
              <IonTabButton tab="statistics" href="/statistics">
                <IonIcon className={`my-icon ${selectedTab === 'statistics' ? 'active-tab-icon' : ''}`}
                         icon={StatisticsIcon}
                         color={selectedTab === 'statistics' ? activeColor : defaultColor} />
                <IonLabel color={selectedTab === 'statistics' ? activeColor : ''}>Статистика</IonLabel>
              </IonTabButton>
            </IonTabBar>
          </IonTabs>
        )}
      </CarImageContext.Provider>
    </IonReactRouter>
  </IonApp>
);
};

export default App;
```

Рисунок 3.2 – Файл проєкту App.tsx

GaragePage.tsx представляє сторінку "Гараж". Він відображає інформацію про витрати на паливо та сервіс, рисунок 3.3.



```
  const groupedData: { [key: string]: typeof data } = {};
  data.forEach((item: any) => {
    const dateString = item.serviceDate || item.fuelDate;
    if (dateString) {
      const date = new Date(dateString);
      if (!isNaN(date.getTime())) {
        const key = date.toLocaleDateString('uk-UA', { month: 'long', year: 'numeric' });
        if (!groupedData[key]) {
          groupedData[key] = [];
        }
        groupedData[key].push(item);
      }
    }
  });


  return (
    <IonPage>
      <IonContent fullscreen>
        <PageHeader />
        <IonToolbar className="fade-in">
          <IonTitle size="large">Останні витрати</IonTitle>
          <IonButton className="refresh-button" slot="end" onClick={refreshData}>
            <IonIcon slot="icon-only" icon={refresh} />
          </IonButton>
        </IonToolbar>
        <IonList className="fade-in">
          {Object.entries(groupedData).map(([key, items]) => (
            <div key={key} className="fade-in">
              <IonCardTitle className="card-title-custom">{key}</IonCardTitle>
              {items.map((item, index) => {
                if (item.type === 'fuel') {
                  let fuelItem = item as FuelItemData;
                  let fuelDate = new Date(fuelItem.fuelDate);
                  return <FuelItem key={index} type={fuelItem.type} fuelCost={fuelItem.price}
                                   costPerLiter={parseFloat((fuelItem.price / fuelItem.litres).toFixed(1))}
                                   date={fuelDate.toISOString().split('T')[0]} />;
                } else if (item.type === 'service') {
                  let serviceItem = item as ServiceItemData;
                  return <ServiceItem key={index} serviceName={serviceItem.name} workCost={serviceItem.workPrice}
                                      materialsCost={serviceItem.servicePrice} date={serviceItem.serviceDate}
                                      type={serviceItem.type} />;
                }
              })}
            </div>
          ))}
        </IonList>
      </IonContent>
    </IonPage>
  );
};

export default GaragePage;
```

Рисунок 3.3 – Файл GaragePage.tsx

PageHeader.tsx представляє заголовок сторінки. Він використовується на всіх сторінках для забезпечення єдиного вигляду, рисунок 3.4.

```
  return (
    <IonHeader className="header-class">
      <IonToolbar>
        <div className="nav-bar-container">
          <img src={logo} alt="Company Logo" className="logo" />
          <Link to="/settings">
            <IonIcon icon={settingsOutline} className="settings-icon" onClick={handleEditClick} />
          </Link>
        </div>
        <div className="content-container">
          <div className="inner-container">
            <div className="text-container">
              {isEditing ? (
                <>
                  <IonInput className="fixed-height-input" value={tempInputData.name} onIonChange={handleInputChange}
                            name="name" />
                  <IonInput className="fixed-height-input" value={tempInputData.mileage} onIonChange={handleInputChange}
                            name="mileage" />
                </>
              ) : (
                <>
                  <div className="field-value">{carData.name}</div>
                  <div className="field-value">{carData.mileage} км</div>
                </>
              )}
              <div className="field-value">{carData.consumption} л/100км</div>
            </div>
            <div className="image-container">
              <div className="car-image"
                   style={{ backgroundImage: `url(${carImage ? carImage : carDefaultImage})` }} />
              <input type="file" accept="image/*" onChange={handleImageUpload} className="hidden-input" />
            </div>
            <IonIcon icon={pencilOutline} className="edit-icon" onClick={handleEditClick} />
            {isEditing && (
              <IonButton className="floating-save-button" onClick={handleSaveClick}>Зберегти</IonButton>
            )}
          </div>
        </div>
      </IonToolbar>
    </IonHeader>

  );
};

export default PageHeader;
```

Рисунок 3.4 – Файл GaragePage.tsx

3.3 Імплементація серверної частини

Імплементація серверної частини є ключовим етапом у розробці мобільного застосунку для контролю автомобільних витрат. Серверна частина імплементована за допомогою фреймворку NestJS, який забезпечує модульну архітектуру, що спрощує розробку, тестування та підтримку додатку. Основними компонентами серверної частини є контролери, сервіси та базові налаштування для роботи з базою даних і забезпечення безпеки.

Основні модулі та компоненти, що входять до складу серверної частини, наведені нижче:

- **AppModule:** Основний модуль додатку, який імпортує всі необхідні контролери та сервіси, що зображено на рисунку 3.5.

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { DatabaseService } from './services/database.service';
import { CarService } from './services/car.service';
import { CarController } from './controllers/car.controller';
import { FuelService } from './services/fuel.service';
import { FuelController } from './controllers/fuel.controller';
import { ServiceController } from './controllers/service.controller';
import { ServiceService } from './services/service.service';

@Module({
  imports: [],
  controllers: [AppController, CarController, FuelController, ServiceController],
  providers: [AppService, DatabaseService, CarService, FuelService, ServiceService],
})
export class AppModule {
}
```

Рисунок 3.5 – Файл app.module.ts

Контролери: Використовуються для обробки HTTP-запитів та взаємодії з клієнтською частиною. На рисунку 3.6 файл визначає контролер, який обробляє HTTP-запити, пов'язані з операціями з паливом. Контролер взаємодіє з відповідним сервісом FuelService, який містить бізнес-логіку для роботи з даними.

Метод create обробляє HTTP POST запити на маршрут /fuels. Декоратор @Post() вказує на тип запиту. Метод приймає дані про паливо в тілі запиту (@Body() fuelData) і передає їх у сервіс FuelService для створення нового запису. Аргумент fuelData містить такі поля: _id, price, litres, fuelDate, _rev.

Метод get обробляє HTTP GET запити на маршрут /fuels/:id. Декоратор @Get(':id') вказує, що метод приймає параметр id з URL. Метод використовує цей id для отримання відповідного запису про паливо через сервіс FuelService.

Метод getAll обробляє HTTP GET запити на маршрут /fuels. Декоратор @Get() вказує на тип запиту. Метод викликає сервіс FuelService для отримання всіх записів про паливо.

```
import { Body, Controller, Get, Param, Post } from '@nestjs/common';
import { FuelService } from '../services/fuel.service';

@Controller('fuels')
export class FuelController {
  constructor(private readonly fuelService: FuelService) {
  }

  @Post()
  async create(@Body() fuelData: { _id: string;
    price: number; litres: number; fuelDate: Date; _rev: string }) {
    return this.fuelService.create(fuelData);
  }

  @Get(':id')
  async get(@Param('id') id: string) {
    return this.fuelService.get(id);
  }

  @Get()
  async getAll() {
    return this.fuelService.getAll();
  }
}
```

Рисунок 3.6 – Файл fuel.controller.ts

Сервіси містять бізнес-логіку додатку та взаємодіють з базою даних. Вони інкапсулюють основні операції та забезпечують розширюваність і підтримку коду. Сервіс fuel.service імпортує модель Fuel, яка визначає структуру даних про паливо, сервіс DatabaseService для взаємодії з базою даних, декоратор Injectable з NestJS для оголошення сервісу, та функцію uuidv4 для генерації унікальних ідентифікаторів, що продемонстровано на рисунку 3.7.

Метод create приймає дані про паливо (price, litres, fuelDate), створює новий об'єкт Fuel з унікальним ідентифікатором _id, а потім намагається отримати існуючий документ з бази даних за цим ідентифікатором. Якщо документ існує, він оновлюється. Якщо документ не знайдено, створюється новий запис у базі даних.

Метод get приймає ідентифікатор id і використовує DatabaseService для отримання документа з бази даних за цим ідентифікатором. Повертається об'єкт Fuel.

Метод getAll використовує DatabaseService для пошуку всіх документів типу fuel у базі даних. Повертається масив об'єктів Fuel.

```
import * as PouchDB from 'pouchdb';
import { Injectable } from '@nestjs/common';
import * as PouchDBFind from 'pouchdb-find';

PouchDB.plugin(PouchDBFind);

@Injectable()
export class DatabaseService {
  private db: PouchDB.Database<any>;

  constructor() {
    this.db = new PouchDB<any>('vehicle_wallet_db');
    this.db.info().then(() => console.log('Database is successfully connected'));
  }

  async get<T>(id: string): Promise<T> {
    return this.db.get(id);
  }

  async put<T>(id: string, document:
    PouchDB.Core.PutDocument<T & PouchDB.Core.ExistingDocument<T>>): Promise<T> {
    await this.db.put({ _id: id, ...document });
    return this.db.get(id);
  }

  async find<T>(selector: PouchDB.Find.FindRequest<NonNullable<unknown>>): Promise<T[]> {
    const result = await this.db.find(selector);
    return result.docs;
  }

  async clear(): Promise<void> {
    await this.db.destroy();
    this.db = new PouchDB<any>('vehicle_wallet_db');
  }

  async remove(id: string, rev: string): Promise<void> {
    await this.db.remove(id, rev);
  }
}
```

Рисунок 3.7 – Файл fuel.service.ts

3.4 Робота додатку

Для того щоб запустити застосунок в браузері або створити збірку для запуску на смартфоні було написано інструкції для проєкту в README.MD, що зображено на рисунку 3.8:

Vehicle Wallet

Introduction

Vehicle Wallet is a mobile application designed to track and manage your vehicle's service history and expenses. It provides functionalities such as recording refueling, maintenance activities, viewing recent expenses, and generating expense charts for different periods.

Installation and Running

1. Clone the repository using the command `git clone <repository url>`.
2. Navigate to the project directory using the command `cd ./vehicle-wallet`.
3. Install the dependencies using the command `npm install`.
4. Start the application using the command `vite`.

Running on Android

To run the application on an Android device, you need to have an Android emulator or a physical device connected to your computer.

1. Run `tsc && vite build` to build the project.
2. Run `npx cap copy android` to copy the build files to the Android project.
3. Run `npx cap open android` to open the Android project in Android Studio.
4. Connect your Android device to your computer or create virtual device in Android Studio.
5. Run the application on the connected device or emulator using the `Run` button in Android Studio.

Building APK

1. Run `tsc && vite build` to build the project.
2. Run `npx cap copy android` to copy the build files to the Android project.
3. Run `cd android` to navigate to the Android project directory.
4. Run `.\gradlew.bat assembleDebug` to build the APK file.
5. You can find the APK file in the `android/app/build/outputs/apk/debug` directory.

NPM Scripts

This project uses several npm scripts for development and building the application:

- `dev` : Runs the Vite development server.
- `build` : Compiles the TypeScript code and builds the application using Vite.
- `preview` : Runs the Vite preview server on the built application(need to build firstly).
- `android:build` : This is a composite script that performs several actions in sequence:
  - Compiles the TypeScript code.
  - Builds the application using Vite.
  - Copies the built files to the Android project using Capacitor.
  - Opens the Android project in Android Studio.

Рисунок 3.8 – Інструкції для запуску проєкту

При відкритті додатку відображається сторінка «Гараж», що зображено на рисунку 3.9:

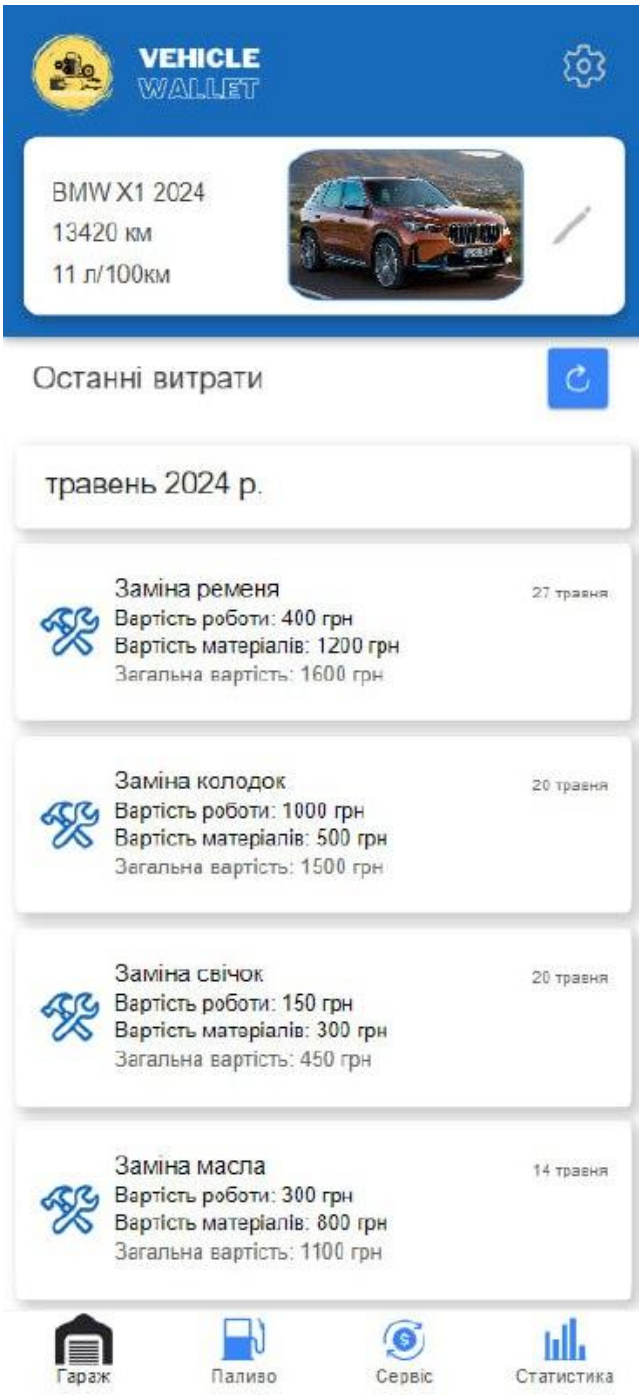


Рисунок 3.9 – Сторінка «Гараж»

На сторінці паливо є можливість переглянути заправлення, відсортовані та згруповані за місяцями, що зображено на рисунку 3.10:

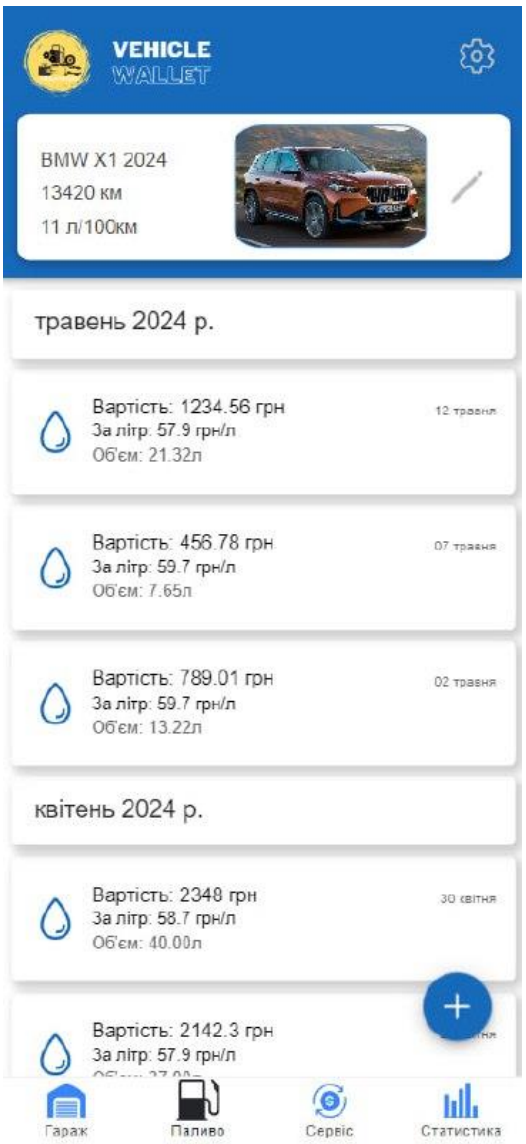


Рисунок 3.10 – Сторінка «Паливо»

На сторінці сервіс є можливість переглянути заправлення, відсортовані та згруповані за місяцями, що зображено на рисунку 3.11:

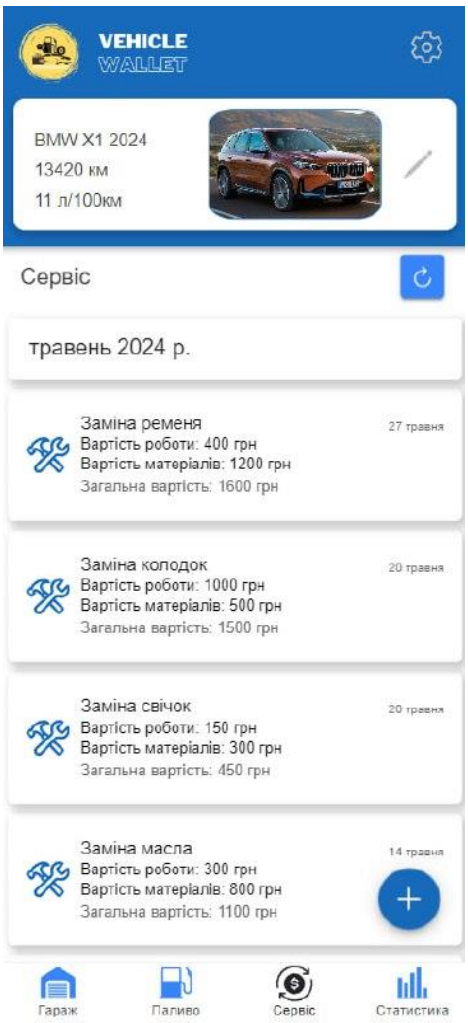


Рисунок 3.11 – Сторінка «Сервіс»

Користувач може редагувати дані про авто у верхній навігаційній панелі, що зображено на рисунку 3.12:

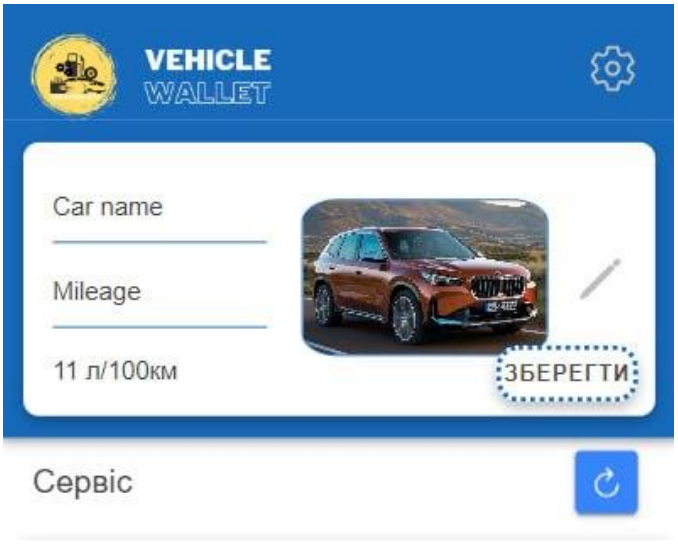


Рисунок 3.12 – Редагування даних в навігаційній панелі

На рисунку 3.13 зображена сторінка додавання заправки:

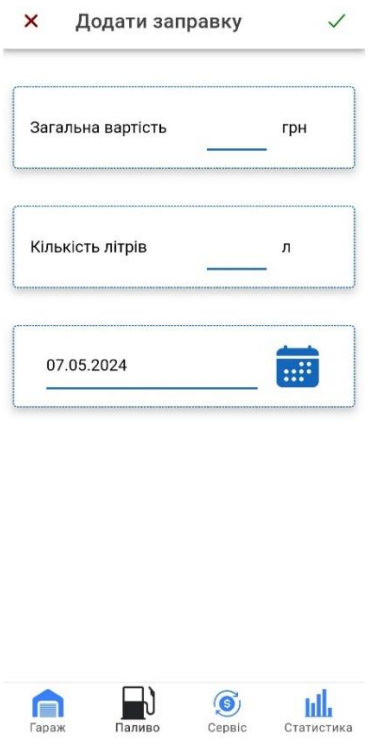


Рисунок 3.13 – Сторінка додавання заправки

На рисунку 3.14 зображена сторінка додавання сервісу:

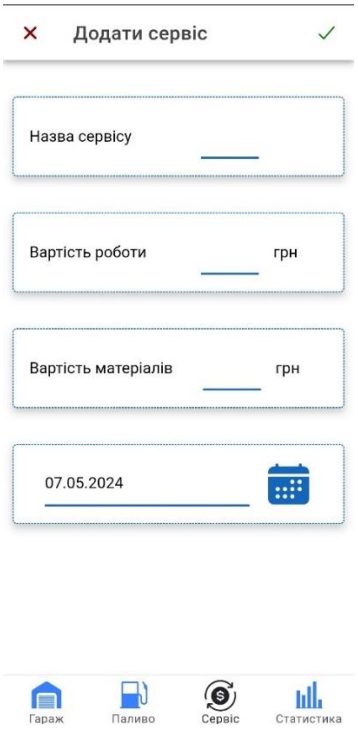


Рисунок 3.14 – Сторінка додавання витрати на сервісне обслуговування

На сторінці статистики у вигляді лінійного графіку вартість витрат на заправки згруповано за місяцями, що відображено на сторінці 3.15:

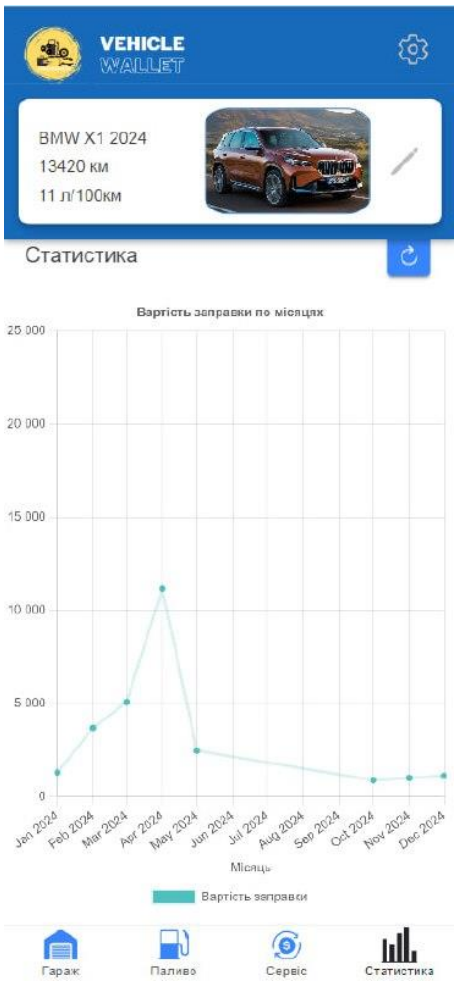


Рисунок 3.15 – Сторінка додавання витрати на сервісне обслуговування

ВИСНОВКИ

У процесі роботи були використані сучасні технології та фреймворки, що дозволили створити надійний і функціональний продукт. Зокрема, для розробки фронтенд-частини додатку було обрано Ionic SDK з використанням React, що забезпечує високий рівень інтерактивності та зручності користувацького інтерфейсу. Використання Nest.js для бекенд-частини додатку надало можливість створити гнучку та масштабовану архітектуру сервісів, яка легко піддається розширенню та модифікації. PouchDB був обраний як база даних завдяки його можливостям для роботи в офлайн-режимі та синхронізації даних, що значно покращує користувацький досвід, дозволяючи зберігати дані локально та синхронізувати їх з сервером при наявності інтернет-з'єднання.

У ході виконання кваліфікаційної роботи було виконано такі завдання:

1. Аналіз потреб користувачів: Проведено дослідження та аналіз потреб і очікувань цільової аудиторії. Визначено основні вимоги до функціональності та інтерфейсу мобільного додатку, що дозволило чітко окреслити завдання для наступних етапів розробки.

2. Визначення функціональних вимог: Визначено технології для розробки додатку, включаючи використання Ionic SDK (React) для фронтенду, Nest.js для бекенду та PouchDB для зберігання даних. Обрані мови програмування та фреймворки дозволили створити надійну та ефективну систему для контролю автомобільних витрат.

3. Розробка інтерфейсу користувача: Створено інтуїтивно зрозумілий та зручний інтерфейс, що дозволяє користувачам легко вводити дані про витрати, переглядати історію витрат, заправлень та сервісних робіт, а також отримувати детальну статистику та звіти.

4. Розробка функціональності: Реалізовано основні функції додатку, включаючи введення інформації про автомобіль, додавання та перегляд витрат

на пальне та сервіс, генерацію графіків та звітів. Використання Nest.js для серверної частини та PouchDB для бази даних забезпечило ефективне зберігання та обробку даних.

5. Тестування: Проведено тестування додатку для виявлення помилок та недоліків. Тестування показало, що додаток працює стабільно та відповідає вимогам користувачів щодо функціональності та зручності використання.

Надалі планується подальше вдосконалення додатку, включаючи додавання нових функцій, оптимізацію продуктивності та інтеграцію з іншими сервісами для ще більш ефективного управління автомобільними витратами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільний застосунок – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Мобільний_застосунок (дата звернення: 25.05.2024).

2. The Different Types of Mobile Apps: Native, Hybrid, PWA. The forefront of innovation | Flatirons. URL: https://flatirons.com/blog/the-different-types-of-mobile-apps-native-hybrid-pwa/ (date of access: 25.05.2024).

3. Frain B. Responsive web design with HTML5 and CSS: build future-proof responsive websites using the latest HTML5 and CSS techniques. Packt Publishing, Limited, 2022.

4. Freeman E., Robson E. Head First HTML and CSS. O'Reilly Media, Incorporated, 2012.

5. Introduction to ionic | ionic documentation. *Ionic Framework - The Cross-Platform App Development Leader*. URL: https://ionicframework.com/docs/.

6. Meloni J., Kyrnin J. HTML, CSS, and javascript all in one: covering HTML5, CSS3, and ES6, sams teach yourself. Pearson Education, Limited, 2018.

7. Ravulavaru A. Learning ionic. Packt Publishing, Limited, 2015.

8. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. *React*. URL: https://uk.legacy.reactjs.org/.

9. Saini G. Hybrid mobile development with ionic. Packt Publishing, Limited, 2017.

10. Tyagi P. Pragmatic flutter: building cross-platform mobile apps for android, IOS, web and desktop. Taylor & Francis Group, 2021. 337 p.

11. NestJS vs. Express.js - LogRocket Blog. LogRocket Blog. URL: https://blog.logrocket.com/nestjs-vs-express-js/ (date of access: 05.05.2024).

12. NestJS vs Django: Which Framework Should You Use? - Frontend Mag. Frontend Mag. URL: https://www.frontendmag.com/insights/nestjs-vs-django/ (date of access: 27.05.2024).

13. Goyal A. PouchDB tutorial for beginners : learn pouchdb tutorial from scratch: learn pouchdb tutorial step by step. Independently Published, 2018.

14. Griffiths D., Griffiths D. React cookbook: recipes for mastering the react framework. O'Reilly Media, Incorporated, 2021. 500 p.

15. JavaScript with syntax for types. TypeScript: JavaScript With Syntax For Types. URL: https://www.typescriptlang.org/ (date of access: 06.05.2024).

16. PouchDB, the JavaScript Database that Syncs!. PouchDB, the JavaScript Database that Syncs!. URL: https://pouchdb.com/.

17. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/.

ДОДАТОК

Main.tsx:

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import App from './App';

const container = document.getElementById('root');
const root = createRoot(container!);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Images.d.ts:

```
declare module "*.svg" {
  const content: any;
  export default content;
}

declare module "*.png" {
  const content: any;
  export default content;
}

declare module "*.jpg" {
  const content: any;
  export default content;
}

declare module "*.jpeg" {
  const content: any;
  export default content;
}
```

```
declare module "*.gif" {
  const content: any;
  export default content;
}

declare module "*.bmp" {
  const content: any;
  export default content;
}

declare module "*.tiff" {
  const content: any;
  export default content;
}
```

App.tsx:

```
import React, { useEffect, useState } from 'react';
import { Redirect, Route } from 'react-router-dom';
import {
  IonApp,
  IonIcon,
  IonLabel,
  IonRouterOutlet,
  IonTabBar,
  IonTabButton,
  IonTabs,
  setupIonicReact,
} from '@ionic/react';
import { IonReactRouter } from '@ionic/react-router';
import GaragePage from './pages/Garage/GaragePage';
import FuelPage from './pages/Fuel/FuelPage';
import ServicePage from './pages/Service/ServicePage';
import StatisticsPage from './pages/Statistics/StatisticsPage';
import WelcomePage from './pages/Welcome/WelcomePage';
import SettingsPage from './pages/SettingsPage/SettingsPage';
```

```
/* Core CSS required for Ionic components to work properly */
import '@ionic/react/css/core.css';

/* Basic CSS for apps built with Ionic */
import '@ionic/react/css/normalize.css';
import '@ionic/react/css/structure.css';
import '@ionic/react/css/typography.css';

/* Optional CSS utils that can be commented out */
import '@ionic/react/css/padding.css';
import '@ionic/react/css/float-elements.css';
import '@ionic/react/css/text-alignment.css';
import '@ionic/react/css/text-transformation.css';
import '@ionic/react/css/flex-utils.css';
import '@ionic/react/css/display.css';

/* Theme variables */
import './theme/variables.css';
import './App.css';
import { CarImageContext } from './utils/constants';

/* Import SVGs */
import GarageIcon from '../resources/android/menuItems/garage.svg';
import FuelIcon from '../resources/android/menuItems/fuel.svg';
import ServiceIcon from '../resources/android/menuItems/service.svg';
import StatisticsIcon from '../resources/android/menuItems/statistics.svg';
import AddExpensePage from './pages/AddExpensePage/AddExpensePage';
import AddServicePage from './pages/AddServicePage/AddServicePage';

setupIonicReact();

const App: React.FC = () => {
  const [showWelcome, setShowWelcome] = useState(true);
```

```
const [selectedTab, setSelectedTab] = useState('garage');
const [carImage, setCarImage] = useState<string | null>(null);

const activeColor = 'dark';
const defaultColor = 'primary';

useEffect(() => {
  setTimeout(() => {
    setShowWelcome(false);
  }, 200);
}, []);

return (
  <IonApp>
    <IonReactRouter>
      <CarImageContext.Provider value={{ carImage, setCarImage }}>
        {showWelcome ? (
          <IonRouterOutlet>
            <Route exact path="/welcome" component={WelcomePage} />
            <WelcomePage />
            <Redirect exact from="/" to="/garage" />
          </IonRouterOutlet>
        ) : (
          <IonTabs onIonTabsDidChange={(e: CustomEvent) => setSelectedTab(e.detail.tab)}>
            <IonRouterOutlet>
              <Route exact path="/settings" component={SettingsPage} />
              <Route exact path="/garage" component={GaragePage} />
              <Route exact path="/fuel" component={FuelPage} />
              <Route exact path="/service" component={ServicePage} />
              <Route exact path="/statistics" component={StatisticsPage} />
              <Route exact path="/add-expense" component={AddExpensePage} />
              <Route exact path="/add-service" component={AddServicePage} />
            </IonRouterOutlet>
            <IonTabBar slot="bottom" className="custom-tab-bar">
```

```
            <IonTabButton tab="garage" href="/garage">
              <IonIcon className={`my-icon ${selectedTab === 'garage' ? 'active-tab-icon' : ''}`}
                    icon={GarageIcon}
                    color={selectedTab === 'garage' ? activeColor : defaultColor} />
              <IonLabel color={selectedTab === 'garage' ? activeColor : ''}>Гараж</IonLabel>
            </IonTabButton>
            <IonTabButton tab="fuel" href="/fuel">
              <IonIcon className={`my-icon ${selectedTab === 'fuel' ? 'active-tab-icon' : ''}`}
                    icon={FuelIcon}
                    color={selectedTab === 'fuel' ? activeColor : defaultColor} />
              <IonLabel color={selectedTab === 'fuel' ? activeColor : ''}>Паливо</IonLabel>
            </IonTabButton>
            <IonTabButton tab="service" href="/service">
              <IonIcon className={`my-icon ${selectedTab === 'service' ? 'active-tab-icon' : ''}`}
                    icon={ServiceIcon}
                    color={selectedTab === 'service' ? activeColor : defaultColor} />
              <IonLabel color={selectedTab === 'service' ? activeColor : ''}>Сервіс</IonLabel>
            </IonTabButton>
            <IonTabButton tab="statistics" href="/statistics">
              <IonIcon className={`my-icon ${selectedTab === 'statistics' ? 'active-tab-icon' : ''}`}
                    icon={StatisticsIcon}
                    color={selectedTab === 'statistics' ? activeColor : defaultColor} />
              <IonLabel color={selectedTab === 'statistics' ? activeColor :
''}>Статистика</IonLabel>
            </IonTabButton>
          </IonTabBar>
        </IonTabs>
      )}
    </CarImageContext.Provider>
  </IonReactRouter>
 </IonApp>
);
};
```

```
export default App;
```

App.css:

```
.my-icon {
    font-size: 2rem;
    transition: color 0.7s;
    margin-bottom: 0.5vh;
    margin-top: 0.4vh;
}

.fade-in {
    animation: fadeIn ease 1s;
}

@keyframes fadeIn {
    0% {
        opacity: 0;
    }
    100% {
        opacity: 1;
    }
}

.active-tab-icon {
    transform: scale(1.2);
    transition: transform 0.3s ease-in-out;
}

.custom-tab-bar {
    --height: 105px !important;
}
```

CarService.ts:

```
export class CarService {
  static async saveCar(car: any, carImage: string) {
    const response = await fetch('http://localhost:3000/cars', {
```

```
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ ...car, image: carImage }),
   });
   return response.json();
  }

  static async getCar(id: string) {
   const response = await fetch(`http://localhost:3000/cars/${id}`);
   return response.json();
  }
}
```

WelcomePage.tsx:

```
import {IonContent, IonPage} from '@ionic/react';
import './WelcomePage.css';
import React from "react";
import welcomeImage from '../../../resources/android/welcomeImage/welcome.svg';

const WelcomePage: React.FC = () => {
   return (
      <IonPage>
         <IonContent fullscreen>
            <div className="welcome-content">
               <img src={welcomeImage} alt="WelcomePage Image"/>
            </div>
         </IonContent>
      </IonPage>
   );
};

export default WelcomePage;
```

WelcomePage.css:

```
.welcome-content {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
    background: #1566bd;
}

img {
    width: 80%;
}
```

StatisticsPage.tsx:

```
import React, { useEffect, useState } from 'react';
import { IonButton, IonContent, IonIcon, IonPage, IonTitle, IonToolbar } from '@ionic/react';
import { Line } from 'react-chartjs-2';
import './StatisticsPage.css';
import PageHeader from '../../components/PageHeader/PageHeader';
import { Chart, registerables } from 'chart.js';
import 'chartjs-adapter-date-fns';
import { refresh } from 'ionicons/icons';

Chart.register(...registerables);

interface ChartData {
  labels: Date[];
  datasets: {
    label?: string;
    data: number[];
    fill?: boolean;
    backgroundColor?: string;
    borderColor?: string;
  }[];
}
```

```
const StatisticsPage: React.FC = () => {
  const [chartData, setChartData] = useState<ChartData>({
    labels: [],
    datasets: [
      {
        data: [],
      },
    ],
  });


  const fetchData = async () => {
    const response = await fetch('http://localhost:3000/fuels');
    const fuelData = await response.json();

    const groupedData: { [key: number]: number } = {};

    fuelData.forEach((item: any) => {
      const date = new Date(item.fuelDate);
      const month = date.getMonth();

      if (groupedData[month]) {
        groupedData[month] += item.price;
      } else {
        groupedData[month] = item.price;
      }
    });

    const months = Object.keys(groupedData).map(month => {
      const date = new Date(2023, parseInt(month));
      date.setFullYear(new Date().getFullYear());
      return date;
    });
```

```
  const costs = Object.values(groupedData);

  setChartData({
    labels: months,
    datasets: [
      {
        label: 'Вартість заправки',
        data: costs,
        fill: false,
        backgroundColor: 'rgb(75, 192, 192)',
        borderColor: 'rgba(75, 192, 192, 0.2)',
      },
    ],
  });
};

useEffect(() => {
  fetchData();
}, []);

return (
  <IonPage>
    <IonContent fullscreen>
      <PageHeader />
      <IonToolbar>
        <IonTitle size="large">Статистика</IonTitle>
        <IonButton className="refresh-button" slot="end" onClick={fetchData}>
          <IonIcon slot="icon-only" icon={refresh} />
        </IonButton>
      </IonToolbar>
      <div style={{ height: '600px' }}>
        <Line
          className="chart"
          data={chartData}
```

```
      options={{
        maintainAspectRatio: false, // Add this line
        scales: {
          x: {
            type: 'time',
            time: {
              unit: 'month',
              displayFormats: {
                month: 'MMM yyyy',
              },
            },
            title: {
              display: true,
              text: 'Місяць',
            },
          },
          y: {
            min: 0,
            max: 25000,
          },
        },
        plugins: {
          title: {
            display: true,
            text: 'Вартість заправки по місяцях',
          },
          legend: {
            position: 'bottom',
          },
        },
      }}
    />
  </div>
</IonContent>
```

```
    </IonPage>
  );
};

export default StatisticsPage;
```

StatisticsPage.css:

```
.refresh-button {
    margin-right: 25px !important;
}

.chart {
    margin-top: 10px;
}
```

SettingsPage.tsx:

```
import React from 'react';
import {IonContent, IonItem, IonLabel, IonList, IonPage, IonTitle, IonToolbar} from "@ionic/react";
import '../../App.css';

const SettingsPage: React.FC = () => {
    const settings = ['Налаштування 1', 'Налаштування 2', 'Налаштування 3'];

    return (
        <IonPage>
            <IonToolbar>
                <IonTitle size="large">Налаштування</IonTitle>
            </IonToolbar>
            <IonContent fullscreen>
                <IonList>
                    {settings.map((setting, index) => (
                        <IonItem key={index} className="fade-in">
                            <IonLabel>{setting}</IonLabel>
                        </IonItem>
                    ))}
```

```
            </IonList>
          </IonContent>
        </IonPage>
    );
};

export default SettingsPage;
```

ServicePage.tsx:

```
import { IonButton, IonCardTitle, IonContent, IonIcon, IonPage, IonTitle, IonToolbar } from '@ionic/react';
import { refresh } from 'ionicons/icons';
import './ServicePage.css';
import React, { useEffect, useState } from 'react';
import PageHeader from '../../components/PageHeader/PageHeader';
import ServiceItem from '../../components/ServiceItem/ServiceItem';
import AddServiceButton from '../../components/AddButton/AddButton';

interface ServiceItemData {
  _id: string;
  type: string;
  name: string;
  workPrice: number;
  servicePrice: number;
  serviceDate: Date;
}

const ServicePage: React.FC = () => {
  const [serviceData, setServiceData] = useState<ServiceItemData[]>([]);

  useEffect(() => {
    fetchServices();
  }, []);

  const fetchServices = async () => {
```

```
  const response = await fetch('http://localhost:3000/services');
  const data = await response.json();
  setServiceData(data);
 };

 serviceData.sort((a: ServiceItemData, b: ServiceItemData) => new
Date(b.serviceDate).getTime() - new Date(a.serviceDate).getTime());

 const groupedData: { [key: string]: typeof serviceData } = {};
 serviceData.forEach((item: ServiceItemData) => {
  const date = new Date(item.serviceDate);
  const key = date.toLocaleDateString('uk-UA', { month: 'long', year: 'numeric' });
  if (!groupedData[key]) {
   groupedData[key] = [];
  }
  groupedData[key].push(item);
 });

 return (
  <IonPage>
   <IonContent fullscreen>
    <PageHeader />
    <IonToolbar className="fade-in">
     <IonTitle size="large">Сервіс</IonTitle>
     <IonButton className="refresh-button" slot="end" onClick={fetchServices}>
      <IonIcon slot="icon-only" icon={refresh} />
     </IonButton>
    </IonToolbar>
    {Object.entries(groupedData).map(([key, items]) => (
     <div key={key} className="fade-in">
      <IonCardTitle className="card-title-custom">{key}</IonCardTitle>
      {items.map((item: ServiceItemData, index) => (
       <ServiceItem key={index} serviceName={item.name} workCost={item.workPrice}
                    materialsCost={item.servicePrice}   date={item.serviceDate}   type={item.type}
```

```
/>
          ))}
        </div>
      ))}
      <AddServiceButton link="/add-service" />
    </IonContent>
  </IonPage>
 );
};

export default ServicePage;
```

ServicePage.css:

```
.refresh-button {
   margin-right: 25px !important;
}
```

GaragePage.tsx:

```
import { IonButton, IonCardTitle, IonContent, IonIcon, IonList, IonPage, IonTitle, IonToolbar }
from '@ionic/react';
import './GaragePage.css';
import React, { useEffect, useState } from 'react';
import FuelItem from '../../components/FuelItem/FuelItem';
import ServiceItem from '../../components/ServiceItem/ServiceItem';
import PageHeader from '../../components/PageHeader/PageHeader';
import { refresh } from 'ionicons/icons';

interface ServiceItemData {
  _id: string;
  type: string;
  name: string;
  workPrice: number;
  servicePrice: number;
  serviceDate: Date;
}
```

```
interface FuelItemData {
  _id: string;
  type: string;
  price: number;
  litres: number;
  fuelDate: Date;
}

const GaragePage: React.FC = () => {
  const [data, setData] = useState<(ServiceItemData | FuelItemData)[]>([]);

  const fetchData = async () => {
    const response1 = await fetch('http://localhost:3000/services');
    const services = await response1.json();

    const response2 = await fetch('http://localhost:3000/fuels');
    const fuel = await response2.json();

    setData([...services, ...fuel]);
  };

  useEffect(() => {
    fetchData();
  }, []);

  const refreshData = () => {
    fetchData();
  };

  data.sort((a: any, b: any) => new Date(b.serviceDate || b.fuelDate).getTime() - new Date(a.serviceDate || a.fuelDate).getTime());

  const groupedData: { [key: string]: typeof data } = {};
  data.forEach((item: any) => {
```

```
  const dateString = item.serviceDate || item.fuelDate;
  if (dateString) {
    const date = new Date(dateString);
    if (!isNaN(date.getTime())) {
      const key = date.toLocaleDateString('uk-UA', { month: 'long', year: 'numeric' });
      if (!groupedData[key]) {
        groupedData[key] = [];
      }
      groupedData[key].push(item);
    }
  }
});


return (
  <IonPage>
    <IonContent fullscreen>
      <PageHeader />
      <IonToolbar className="fade-in">
        <IonTitle size="large">Останні витрати</IonTitle>
        <IonButton className="refresh-button" slot="end" onClick={refreshData}>
          <IonIcon slot="icon-only" icon={refresh} />
        </IonButton>
      </IonToolbar>
      <IonList className="fade-in">
        {Object.entries(groupedData).map(([key, items]) => (
          <div key={key} className="fade-in">
            <IonCardTitle className="card-title-custom">{key}</IonCardTitle>
            {items.map((item, index) => {
              if (item.type === 'fuel') {
                let fuelItem = item as FuelItemData;
                let fuelDate = new Date(fuelItem.fuelDate);
                return <FuelItem key={index} type={fuelItem.type} fuelCost={fuelItem.price}
                                 costPerLiter={parseFloat((fuelItem.price / fuelItem.litres).toFixed(1))}
```

```
                        date={fuelDate.toISOString().split('T')[0]} />;
            } else if (item.type === 'service') {
              let serviceItem = item as ServiceItemData;
              return <ServiceItem key={index} serviceName={serviceItem.name}
workCost={serviceItem.workPrice}
                        materialsCost={serviceItem.servicePrice}
date={serviceItem.serviceDate}
                        type={serviceItem.type} />;
            }
           })}
          </div>
        ))}
      </IonList>
    </IonContent>
  </IonPage>
 );
};

export default GaragePage;
```

GaragePage.css:

```
.content-container {
  align-items: center;
}
```

FuelPage.tsx:

```
import React, { useEffect, useState } from 'react';
import { IonButton, IonCardTitle, IonContent, IonHeader, IonIcon, IonPage, IonTitle, IonToolbar
} from '@ionic/react';
import './FuelPage.css';
import FuelItem from '../../components/FuelItem/FuelItem';
import AddExpenseButton from '../../components/AddButton/AddButton';
import PageHeader from '../../components/PageHeader/PageHeader';
import { refresh } from 'ionicons/icons';

interface FuelItemData {
```

```
  _id: string;
  type: string;
  price: number;
  litres: number;
  fuelDate: Date;
}

const FuelPage: React.FC = () => {
  const [fuelData, setFuelData] = useState<{ [key: string]: FuelItemData[] }>({});

  const fetchFuelData = () => {
    fetch('http://localhost:3000/fuels')
      .then(response => response.json())
      .then(data => {
        // Sort by date
        data.sort((a: FuelItemData, b: FuelItemData) => {
          let aDate = new Date(a.fuelDate).getTime();
          let bDate = new Date(b.fuelDate).getTime();
          return bDate - aDate;
        });

        const groupedData: { [key: string]: FuelItemData[] } = {};
        data.forEach((item: FuelItemData) => {
          const fuelDate = new Date(item.fuelDate);
          const formattedDate = fuelDate.toLocaleDateString('uk-UA', { month: 'long', year: 'numeric' });
          if (!groupedData[formattedDate]) {
            groupedData[formattedDate] = [];
          }
          item.fuelDate = fuelDate;
          groupedData[formattedDate].push(item);
        });

        setFuelData(groupedData);
```

```
    });
  };

  useEffect(() => {
    fetchFuelData();
  }, []);

  return (
    <IonPage>
      <PageHeader />
      <IonContent fullscreen>
        <IonHeader collapse="condense">
          <IonToolbar className="fade-in">
            <IonTitle size="large">Заправки</IonTitle>
            <IonButton className="refresh-button" slot="end" onClick={fetchFuelData}>
              <IonIcon slot="icon-only" icon={refresh} />
            </IonButton>
          </IonToolbar>
        </IonHeader>
        {Object.entries(fuelData).map(([key, items]) => (
          <div key={key}>
            <IonCardTitle className="card-title-custom">{key}</IonCardTitle>
            {items.map((item: FuelItemData, index: number) => (
              <FuelItem key={index} type={item.type} fuelCost={item.price}
                        costPerLiter={parseFloat((item.price / item.litres).toFixed(1))}
                        date={item.fuelDate.toISOString().split('T')[0]} />
            ))}
          </div>
        ))}
        <AddExpenseButton link="/add-expense" />
      </IonContent>
    </IonPage>
  );
};
```

```
export default FuelPage;
```

FuelPage.css:

```
.card-title-custom {
    margin: 1vh 1vh 1vh;
    box-shadow: 4px 4px 8px 2px rgba(0, 0, 0, 0.2);
    text-align: left;
    padding-left: 2vh;
    padding-top: 1.5vh;
    padding-bottom: 4vh;
    height: 6vh;
    border-radius: 5px;
    font-size: 20px;
    font-family: "Roboto", sans-serif;
}

.refresh-button {
    margin-right: 25px !important;
}
```

AddServicePage.tsx:

```
import {
  IonButton,
  IonButtons,
  IonCard,
  IonCardContent,
  IonContent,
  IonDatetime,
  IonHeader,
  IonIcon,
  IonInput,
  IonItem,
  IonLabel,
  IonModal,
  IonPage,
```

```
  IonTitle,
  IonToolbar,
} from '@ionic/react';
import { calendar, checkmark, close } from 'ionicons/icons';
import React, { useEffect, useState } from 'react';
import './AddServicePage.css';
import { useHistory } from 'react-router';

const AddServicePage: React.FC = () => {
  const [serviceName, setServiceName] = useState<string>();
  const [workCost, setWorkCost] = useState<number>();
  const [materialsCost, setMaterialsCost] = useState<number>();
  const [date, setDate] = useState<string>(new Date().toLocaleDateString('uk-UA'));
  const [tempDate, setTempDate] = useState<string>(date);
  const [showDatetime, setShowDatetime] = useState<boolean>(false);

  const handleSave = async () => {
    if (!serviceName || !workCost || !materialsCost || !date) {
      alert('Будь ласка, заповніть всі поля');
      return;
    }

    const serviceData = {
      name: serviceName,
      workPrice: workCost,
      servicePrice: materialsCost,
      serviceDate: date,
    };

    try {
      const response = await fetch('http://localhost:3000/services', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
```

```
      },
      body: JSON.stringify(serviceData),
    });

    if (!response.ok) {
      throw new Error('Не вдалося створити запис про сервіс');
    }

    setServiceName(undefined);
    setWorkCost(undefined);
    setMaterialsCost(undefined);
    setDate(new Date().toISOString());

    history.push('/service');
  } catch (error) {
    console.error('Failed to save service data:', error);
  }
};

const handleDateClick = () => {
  setShowDatetime(true);
};

const handleDateChange = (e: any) => {
  if (typeof e.detail.value === 'string') {
    const selectedDate = new Date(e.detail.value.split('.').reverse().join('-'));
    setTempDate(selectedDate.toISOString());
  }
};

const handleSelectDate = () => {
  setDate(tempDate);
  setShowDatetime(false);
};
```

```
  const history = useHistory();

  useEffect(() => {
    return history.listen(() => {
      setServiceName(undefined);
      setWorkCost(undefined);
      setMaterialsCost(undefined);
      setDate(new Date().toLocaleDateString('uk-UA'));
    });
  }, [history]);

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot="start">
            <IonButton routerLink="/service">
              <IonIcon slot="icon-only" icon={close} className="large-icon close-icon" />
            </IonButton>
          </IonButtons>
          <IonTitle>Додати сервіс</IonTitle>
          <IonButtons slot="end">
            <IonButton onClick={handleSave}>
              <IonIcon slot="icon-only" icon={checkmark} className="large-icon checkmark-icon"
/>
            </IonButton>
          </IonButtons>
        </IonToolbar>
      </IonHeader>
      <IonContent fullscreen>
        <IonCard className="large-card">
          <IonItem className="flex-row full-width no-wrap" lines="none">
            <IonLabel slot="start">Назва сервісу</IonLabel>
```

```
        <IonInput className="fixed-width-input highlight-input input-padding"
value={serviceName}
                onIonChange={e => {
                 if (e.detail.value !== null) {
                  setServiceName(e.detail.value);
                 }
                }} />
        </IonItem>
      </IonCard>
      <IonCard className="large-card">
        <IonItem className="flex-row full-width no-wrap" lines="none">
          <IonLabel slot="start">Вартість роботи</IonLabel>
          <IonInput className="fixed-width-input highlight-input input-padding"
value={workCost}
                onIonChange={e => setWorkCost(Number(e.detail.value))} />
          <IonLabel>грн</IonLabel>
        </IonItem>
      </IonCard>
      <IonCard className="large-card">
        <IonItem className="flex-row full-width no-wrap" lines="none">
          <IonLabel slot="start">Вартість матеріалів</IonLabel>
          <IonInput className="fixed-width-input highlight-input input-padding"
value={materialsCost}
                onIonChange={e => setMaterialsCost(Number(e.detail.value))} />
          <IonLabel>грн</IonLabel>
        </IonItem>
      </IonCard>
      <IonCard className="large-card">
        <IonCardContent>
          <IonItem className="flex-row full-width no-wrap" lines="none">
            <IonInput className="highlight-input" value={date} placeholder="Виберіть дату"
                 readonly onClick={handleDateClick} />
            <IonIcon slot="end" icon={calendar} className="calendar-icon"
onClick={handleDateClick} />
```

```
          </IonItem>
        </IonCardContent>
      </IonCard>
      <IonModal isOpen={showDatetime} onDidDismiss={() => setShowDatetime(false)}>
        <IonDatetime
          value={tempDate}
          onIonChange={handleDateChange}
          presentation="date"
        />
        <IonButton onClick={handleSelectDate}>Обрати</IonButton>
      </IonModal>
    </IonContent>
  </IonPage>
 );
};

export default AddServicePage;
```

AddExpensePage.tsx:

```
import {
 IonButton,
 IonButtons,
 IonCard,
 IonCardContent,
 IonContent,
 IonDatetime,
 IonHeader,
 IonIcon,
 IonInput,
 IonItem,
 IonLabel,
 IonModal,
 IonPage,
 IonTitle,
 IonToolbar,
```

```
} from '@ionic/react';
import { calendar, checkmark, close } from 'ionicons/icons';
import React, { useEffect, useState } from 'react';
import './AddExpensePage.css';
import { useHistory } from 'react-router';

const AddExpensePage: React.FC = () => {
  const [fuelCost, setFuelCost] = useState<number>();
  const [liters, setLiters] = useState<number>();
  const [date, setDate] = useState<string>(new Date().toISOString());
  const [tempDate, setTempDate] = useState<string>(date);
  const [showDatetime, setShowDatetime] = useState<boolean>(false);

  const handleSave = async () => {
    if (!fuelCost || !liters || !date) {
      alert('Будь ласка, заповніть всі поля');
      return;
    }

    const fuelData = {
      price: fuelCost,
      litres: liters,
      fuelDate: date,
    };

    try {
      const response = await fetch('http://localhost:3000/fuels', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(fuelData),
      });
```

```
    if (!response.ok) {
      throw new Error('Не вдалося створити запис про заправлення');
    }

    setFuelCost(undefined);
    setLiters(undefined);
    setDate(new Date().toLocaleDateString('uk-UA'));

    history.push('/fuel');
  } catch (error) {
    console.error('Failed to save fuel data:', error);
  }
};

const handleDateClick = () => {
  setShowDatetime(true);
};

const handleDateChange = (e: any) => {
  if (typeof e.detail.value === 'string') {
    setTempDate(e.detail.value);
  }
};

const handleSelectDate = () => {
  setDate(tempDate);
  setShowDatetime(false);
};

const history = useHistory();

useEffect(() => {
  return history.listen(() => {
    setFuelCost(undefined);
```

```
      setLiters(undefined);
      setDate(new Date().toLocaleDateString('uk-UA'));
    });
  }, [history]);

  return (
    <IonPage>
      <IonHeader>
        <IonToolbar>
          <IonButtons slot="start">
            <IonButton routerLink="/fuel">
              <IonIcon slot="icon-only" icon={close} className="large-icon close-icon" />
            </IonButton>
          </IonButtons>
          <IonTitle>Додати заправку</IonTitle>
          <IonButtons slot="end">
            <IonButton onClick={handleSave}>
              <IonIcon slot="icon-only" icon={checkmark} className="large-icon checkmark-icon"
/>
            </IonButton>
          </IonButtons>
        </IonToolbar>
      </IonHeader>
      <IonContent fullscreen>
        <IonCard className="large-card">
          <IonItem className="flex-row full-width no-wrap" lines="none">
            <IonLabel slot="start">Загальна вартість</IonLabel>
            <IonInput          className="fixed-width-input          highlight-input          input-padding"
value={fuelCost}
                      onIonChange={e => setFuelCost(Number(e.detail.value))} />
            <IonLabel>грн</IonLabel>
          </IonItem>
        </IonCard>
        <IonCard className="large-card">
```

```
        <IonItem className="flex-row full-width no-wrap" lines="none">
          <IonLabel slot="start">Кількість літрів</IonLabel>
          <IonInput className="fixed-width-input highlight-input input-padding" value={liters}
                onIonChange={e => setLiters(Number(e.detail.value))} />
          <IonLabel>л</IonLabel>
        </IonItem>
      </IonCard>
      <IonCard className="large-card">
        <IonCardContent>
          <IonItem className="flex-row full-width no-wrap" lines="none">
            <IonInput className="highlight-input" value={date} placeholder="Виберіть дату"
                readonly onClick={handleDateClick} />
            <IonIcon slot="end" icon={calendar} className="calendar-icon"
onClick={handleDateClick} />
          </IonItem>
        </IonCardContent>
      </IonCard>
      <IonModal isOpen={showDatetime} onDidDismiss={() => setShowDatetime(false)}>
        <IonDatetime
          value={tempDate}
          onIonChange={handleDateChange}
          presentation="date"
        />
        <IonButton onClick={handleSelectDate}>Обрати</IonButton>
      </IonModal>
    </IonContent>
  </IonPage>
 );
};

export default AddExpensePage;
```

AddExpensePage.css:

```
.fixed-width-input {
  width: 8vh;
```

```
    float: right;
}

.flex-row {
    display: flex;
    justify-content: normal;
}

.large-card {
    height: 11vh;
    margin-bottom: 5vh;
    margin-top: 5vh;
    display: flex;
    align-items: center;
    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
    border: 1px dashed #1566bd;
}

.full-width {
    width: 100%;
}

.no-wrap {
    white-space: nowrap;
}

.highlight-input {
    border-bottom: 2px solid #1566bd;
}

.input-padding {
    margin-left: 2vh;
    margin-right: 2vh;
}
```

```
.calendar-icon {
    font-size: 2em;
    color: #1566bd;
}

.calendar-icon {
    font-size: 3em;
    color: #1566bd;
}

.close-icon {
    color: darkred;
}

.checkmark-icon {
    color: green;
}
```

ServiceItem:

```
import { IonIcon, IonItem, IonLabel, IonNote } from '@ionic/react';
import { constructOutline } from 'ionicons/icons';
import React from 'react';
import './ServiceItem.css';

interface ServiceItemProps {
  type: string;
  serviceName: string;
  workCost: number;
  materialsCost: number;
  date: Date;
}

const ServiceItem: React.FC<ServiceItemProps> = ({ serviceName, workCost, materialsCost, date }) => {
```

```
  const totalCost = workCost + materialsCost;

  const serviceDate = new Date(date);
  const formattedDate = serviceDate.toLocaleDateString('uk-UA', { day: '2-digit', month: 'long' });

  return (
    <IonItem className="service-item-card" lines="none">
      <IonIcon className="service-icon" icon={constructOutline} slot="start" />
      <IonLabel>
        <h2>{serviceName}</h2>
        <h3>Вартість роботи: {workCost} грн</h3>
        <h3>Вартість матеріалів: {materialsCost} грн</h3>
        <p>Загальна вартість: {totalCost} грн</p>
      </IonLabel>
      <IonNote slot="end">{formattedDate}</IonNote>
    </IonItem>
  );
};

export default ServiceItem;
```

ServiceItem.css:

```
.service-item-card {
    display: flex;
    align-items: center;
    padding-bottom: 1vh;
    padding-top: 1vh;
    margin: 1vh;
    box-shadow: 4px 4px 8px 2px rgba(0, 0, 0, 0.2);
    border-radius: 5px;
}

.service-icon {
    margin-right: 1vh;
    font-size: 2.5em;
```

```
  color: #1566bd;
  order: -1; /* Додаємо цю властивість */
}
```

PageHeader.tsx:

```
import { IonButton, IonHeader, IonIcon, IonInput, IonToolbar } from '@ionic/react';
import React, { useContext, useEffect, useState } from 'react';
import './PageHeader.css';
import logo from '../../../resources/android/logo/wlcome.png';
import carDefaultImage from '../../../resources/android/garageSection/car.png';
import { CarImageContext } from '../../utils/constants';
import { InputChangeEventDetail } from '@ionic/core';
import { pencilOutline, settingsOutline } from 'ionicons/icons';
import { CarService } from '../../services/CarService';
import { useLocation } from 'react-router';
import { Link } from 'react-router-dom';

const PageHeader: React.FC = () => {
  const [carData, setCarData] = useState({
    _id: '1',
    name: 'Car name',
    mileage: 'Mileage',
    consumption: '11',
  });

  const [isEditing, setIsEditing] = useState(false);
  const { carImage, setCarImage } = useContext(CarImageContext);

  const location = useLocation();

  useEffect(() => {
    const fetchCarData = async () => {
      try {
        const fetchedCarData = await CarService.getCar(carData._id);
        setCarData(fetchedCarData);
```

```
      setCarImage(fetchedCarData.image);
    } catch (error) {
      console.error('Failed to fetch car data:', error);
    }
  };

  fetchCarData().then(() => {
    console.log('Car data fetched successfully');
  });
}, [location.pathname, isEditing]);

const handleImageUpload = (event: React.ChangeEvent<HTMLInputElement>) => {
  const file = event.target.files?.[0];
  if (file) {
    const reader = new FileReader();
    reader.onloadend = () => {
      setCarImage(reader.result as string);
    };
    reader.readAsDataURL(file);
  }
};

const handleEditClick = () => {
  setIsEditing(!isEditing);
};

const [tempInputData, setTempInputData] = useState(carData);

const handleInputChange = (event: CustomEvent<InputChangeEventDetail>) => {
  const name = event.target && (event.target as any).name;
  const updatedTempInputData = {
    ...tempInputData,
    [name]: event.detail.value,
  };
```

```
    setTempInputData(updatedTempInputData);
  };

  const handleSaveClick = async () => {
    const id = tempInputData._id || '1';
    const name = tempInputData.name;
    const mileage = tempInputData.mileage;
    const consumption = tempInputData.consumption;

    setCarData({ _id: id, name, mileage, consumption });

    try {
      const savedCar = await CarService.saveCar(tempInputData, carImage || '');
      console.log('Car saved:', savedCar);
      setIsEditing(false);

      const updatedCarData = await CarService.getCar(savedCar._id);
      setCarData(updatedCarData);
      setTempInputData(updatedCarData);
    } catch (error) {
      console.error('Failed to save car data:', error);
    }
  };

  return (
    <IonHeader className="header-class">
      <IonToolbar>
        <div className="nav-bar-container">
          <img src={logo} alt="Company Logo" className="logo" />
          <Link to="/settings">
            <IonIcon icon={settingsOutline} className="settings-icon" onClick={handleEditClick}
/>
          </Link>
        </div>
```

```
      <div className="content-container">
        <div className="inner-container">
          <div className="text-container">
            {isEditing ? (
              <>
                <IonInput          className="fixed-height-input"          value={tempInputData.name}
onIonChange={handleInputChange}
                        name="name" />
                <IonInput        className="fixed-height-input"        value={tempInputData.mileage}
onIonChange={handleInputChange}
                        name="mileage" />
              </>
            ) : (
              <>
                <div className="field-value">{carData.name}</div>
                <div className="field-value">{carData.mileage} км</div>
              </>
            )}
            <div className="field-value">{carData.consumption} л/100км</div>
          </div>
          <div className="image-container">
            <div className="car-image"
                style={{ backgroundImage: `url(${carImage ? carImage : carDefaultImage})` }} />
            <input          type="file"          accept="image/*"          onChange={handleImageUpload}
className="hidden-input" />
          </div>
          <IonIcon icon={pencilOutline} className="edit-icon" onClick={handleEditClick} />
          {isEditing && (
            <IonButton                                                  className="floating-save-button"
onClick={handleSaveClick}>Зберегти</IonButton>
          )}
        </div>
      </div>
    </IonToolbar>
```

```
    </IonHeader>

  );
};

export default PageHeader;
```

PageHeader.css:

```
.header-class {
    position: sticky;
    top: 0;
    z-index: 1000;
}

.content-container {
    display: flex;
    justify-content: space-between;
    padding: 1.5vh;
    background-color: #1566bd;
}

.inner-container {
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items: center;
    padding: 2vh;
    background-color: #ffffff;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.25);
    border-radius: 10px;
    width: 100%;
}

.logo {
    margin-top: 2vh;
```

```
    margin-left: 2vh;
    width: 20vh;
}

.nav-bar-container {
    display: flex;
    align-items: center;
    width: 100%;
    background-color: #1566bd;
}

.text-content {
    display: flex;
    flex-direction: column;
    color: white;
}

.settings-icon {
    font-size: 2em;
    position: fixed;
    top: 25px;
    right: 25px;
    color: #bebebe;
}

.edit-icon {
    font-size: 2em;
    color: #bebebe;
}

.text-container {
    display: flex;
    flex-direction: column;
}
```

```
.field-value {
    margin-top: 0.6em;
}

.image-container {
    position: absolute;
    width: 12vh;
    height: 12vh;
    margin-left: 17vh;
    margin-top: 1vh;
}

.car-image {
    position: relative;
    width: 140%;
    height: 90%;
    object-fit: cover;
    border-radius: 15%;
    border: 1px solid #1566bd;
    background-size: cover;
    background-position: center;
}

.hidden-input {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    opacity: 0;
    cursor: pointer;
}
```

```
@keyframes clickAnimation {
   0% {
      transform: scale(1);
   }
   50% {
      transform: scale(0.9);
   }
   100% {
      transform: scale(1);
   }
}

.animate {
   animation: clickAnimation 0.2s;
}

.fixed-height-input {
   border-bottom: 1px solid #1566bd;
   margin-bottom: 1vh;
   width: 70% !important;
}

.floating-save-button {
   position: fixed;
   bottom: 19px;
   right: 23px;
   --box-shadow: 0 4px 8px rgba(0, 0, 0, 0.25);
   --background: #ffffff;
   --background-activated-opacity: 0.8;
   --border-radius: 14px;
   font-size: 14px; /* Adjust this value as needed */
   --color: #504b4b;
   font-weight: bold;
   width: 90px; /* Adjust this value as needed */
```

```
    height: 30px; /* Adjust this value as needed */
    --border-style: dotted;
    --border-color: #1566bd;
}
```

LastExpenseItem.tsx:

```
import { IonItem, IonLabel } from '@ionic/react';
import React from 'react';

interface LastExpenseItemProps {
    label: string;
    price: number;
}

const LastExpenseItem: React.FC<LastExpenseItemProps> = ({ label, price }) => {
    return (
        <IonItem>
            <IonLabel>{label}</IonLabel>
            <IonLabel slot="end">{price} грн</IonLabel>
        </IonItem>
    );
};

export default LastExpenseItem;
```

FuelItem.tsx:

```
import {IonIcon, IonItem, IonLabel, IonNote} from '@ionic/react';
import {waterOutline} from 'ionicons/icons';
import React from "react";
import './FuelItem.css';

interface FuelItemProps {
    type: string;
    fuelCost: number;
    costPerLiter: number;
    date: string;
```

```
}

const FuelItem: React.FC<FuelItemProps> = ({fuelCost, costPerLiter, date}) => {
    const liters = fuelCost / costPerLiter;

    const fuelDate = new Date(date);
    const formattedDate = fuelDate.toLocaleDateString('uk-UA', {day: '2-digit', month: 'long'});

    return (
        <IonItem className="fuel-item-card" lines="none">
            <IonIcon className="fuel-icon" icon={waterOutline} slot="start"/>
            <IonLabel>
                <h2>Вартість: {fuelCost} грн</h2>
                <h3>За літр: {costPerLiter} грн/л</h3>
                <p>Об&#39;єм: {liters.toFixed(2)}л</p>
            </IonLabel>
            <IonNote slot="end">{formattedDate}</IonNote>
        </IonItem>
    );
};

export default FuelItem;
```

FuelItem.css:

```
.fuel-item-card {
    display: flex;
    align-items: center;
    padding-bottom: 1vh;
    padding-top: 1vh;
    margin: 1vh;
    box-shadow: 4px 4px 8px 2px rgba(0, 0, 0, 0.2);
    border-radius: 5px;
}

.fuel-icon {
```

```
    margin-right: 1vh;
    font-size: 2.5em;
    color: #1566bd;
    order: -1;
}
```

AddButton.tsx:

```
import {IonFab, IonFabButton, IonIcon} from '@ionic/react';
import {add} from "ionicons/icons";
import {Link} from "react-router-dom";
import React from "react";
import './AddButton.css';

interface AddExpenseButtonProps {
    link: string;
}

const AddExpenseButton: React.FC<AddExpenseButtonProps> = ({link}) => {
    return (
        <IonFab className="fab-button" vertical="bottom" horizontal="end" slot="fixed">
            <Link to={link}>
                <IonFabButton className="fab-button-color">
                    <IonIcon className="fab-icon" icon={add}/>
                </IonFabButton>
            </Link>
        </IonFab>
    );
};

export default AddExpenseButton;
```

AddButton.css:

```
.fab-button {
    margin-bottom: 2vh;
    margin-right: 2vh;
}
```

```
.fab-button-color {
   --background: #1566bd;
}

.fab-icon {
   font-size: 2.4em;
}
```

Main.ts:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as express from 'express';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.use(express.json({ limit: '50mb' }));
  app.enableCors();
  await app.listen(3000);
}

bootstrap();
```

app.module.ts:

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { DatabaseService } from './services/database.service';
import { CarService } from './services/car.service';
import { CarController } from './controllers/car.controller';
import { FuelService } from './services/fuel.service';
import { FuelController } from './controllers/fuel.controller';
import { ServiceController } from './controllers/service.controller';
import { ServiceService } from './services/service.service';

@Module({
```

```
  imports: [],
  controllers: [AppController, CarController, FuelController, ServiceController],
  providers: [AppService, DatabaseService, CarService, FuelService, ServiceService],
})
export class AppModule {
}
```

App.controller.ts:

```
import { Controller, Delete } from '@nestjs/common';
import { DatabaseService } from './services/database.service';

@Controller()
export class AppController {
  constructor(private readonly databaseService: DatabaseService) {
  }

  @Delete('clear-database')
  async clearDatabase() {
    await this.databaseService.clear();
    return { message: 'Database cleared successfully' };
  }
}
```

Service.service.ts:

```
import { DatabaseService } from './database.service';
import { Injectable } from '@nestjs/common';
import { v4 as uuidv4 } from 'uuid';
import { Service } from '../models/service.model';

@Injectable()
export class ServiceService {
  constructor(private readonly databaseService: DatabaseService) {
  }

  async create(serviceData: { name: string; servicePrice: number; workPrice: number; serviceDate:
Date }) {
```

```
  const service = new Service();
  service._id = uuidv4();
  service.name = serviceData.name;
  service.servicePrice = serviceData.servicePrice;
  service.workPrice = serviceData.workPrice;
  service.serviceDate = serviceData.serviceDate;

  try {
    const doc = await this.databaseService.get<Service>(service._id);
    if (doc._rev) {
      return await this.databaseService.put(service._id, { ...doc, ...service });
    }
  } catch (error) {
    if (error.name === 'not_found') {
      return await this.databaseService.put(service._id, service);
    } else {
      console.error(`Failed to get document with id ${service._id}:`, error);
    }
  }
}

async get(id: string) {
  return (await this.databaseService.get(id)) as unknown as Promise<Service>;
}

async getAll() {
  return this.databaseService.find<Service>({ selector: { type: 'service' } });
}

async deleteAll() {
  const services = await this.getAll();
  for (const service of services) {
    await this.databaseService.remove(service._id, service._rev);
  }
```

```
  }
}
```

Fuel.service.ts:

```
import { Fuel } from '../models/fuel.model';
import { DatabaseService } from './database.service';
import { Injectable } from '@nestjs/common';
import { v4 as uuidv4 } from 'uuid';

@Injectable()
export class FuelService {
  constructor(private readonly databaseService: DatabaseService) {
  }

  async create(fuelData: { price: number; litres: number; fuelDate: Date }) {
    const fuel = new Fuel();
    fuel._id = uuidv4();
    fuel.price = fuelData.price;
    fuel.litres = fuelData.litres;
    fuel.fuelDate = fuelData.fuelDate;

    try {
      const doc = await this.databaseService.get<Fuel>(fuel._id);
      if (doc._rev) {
        return await this.databaseService.put(fuel._id, { ...doc, ...fuel });
      }
    } catch (error) {
      if (error.name === 'not_found') {
        return await this.databaseService.put(fuel._id, fuel);
      } else {
        console.error(`Failed to get document with id ${fuel._id}:`, error);
      }
    }
  }
```

```
  async get(id: string) {
    return (await this.databaseService.get(id)) as unknown as Promise<Fuel>;
  }

  async getAll() {
    return this.databaseService.find<Fuel>({ selector: { type: 'fuel' } });
  }
}
```

Database.service.ts:

```
import * as PouchDB from 'pouchdb';
import { Injectable } from '@nestjs/common';
import * as PouchDBFind from 'pouchdb-find';

PouchDB.plugin(PouchDBFind);

@Injectable()
export class DatabaseService {
  private db: PouchDB.Database<any>;

  constructor() {
    this.db = new PouchDB<any>('vehicle_wallet_db');
    this.db.info().then(() => console.log('Database is successfully connected'));
  }

  async get<T>(id: string): Promise<T> {
    return this.db.get(id);
  }

  async put<T>(id: string, document:
    PouchDB.Core.PutDocument<T & PouchDB.Core.ExistingDocument<T>>): Promise<T> {
    await this.db.put({ _id: id, ...document });
    return this.db.get(id);
  }
```

```
  async find<T>(selector: PouchDB.Find.FindRequest<NonNullable<unknown>>):
Promise<T[]> {
    const result = await this.db.find(selector);
    return result.docs;
  }

  async clear(): Promise<void> {
    await this.db.destroy();
    this.db = new PouchDB<any>('vehicle_wallet_db');
  }

  async remove(id: string, rev: string): Promise<void> {
    await this.db.remove(id, rev);
  }
}
```