Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмний засіб ведення обліку обслуговування автомобілів»

Виконав: студент 2 курсу, групи 2КІ-21м
напряму підготовки (спеціальності)
123 — «Комп'ютерна інженерія»
_____ Капличний О. С.

Керівник: к.т.н., доц. каф. ОТ
_____ Колесник І. С.
«21» 12 2022 р.

Опонент: к.т.н., доцент каф. ПЗ
_____ Черноволик Г. О.
«22» 12 2022 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«23» 12 2022 р.

Вінниця ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 — Інформаційні технології
Спеціальність 123 — «Комп'ютерна інженерія»
Освітня програма — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
проф., д.т.н. О.Д. Азаров
«15» 09 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

студенту Капличному Олегу Сергійовичу

1 Тема роботи «Програмний засіб ведення обліку обслуговування автомобілів» керівник роботи Колесник Ірина Сергіївна, к.т.н., доцент, затверджені наказом вищого навчального закладу від 15.09.2022 р. № 205-А

2 Строк подання студентом роботи 16 грудня 2022 року

3 Вихідні дані до роботи: дані, що отримуються з автомобіля з використанням діагностичного модуля, реєстр витратних матеріалів та помилок.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз способів побудови програмних засобів, проектування програмного засобу, розробка програмного засобу, тестування створеного програмного засобу, розрахунок економічної доцільності створення програми ведення обліку обслуговування автомобілів.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): план розподілу полів головної сторінки, загальний вигляд головної сторінки додатка, структурна схема додатка, сторінка реєстрації авто, діагностичний сканер-адаптер.

3

6 Консультанти розділів роботи представлені в таблиці1.

Таблиця 1— Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесник І. С., к.т.н., доц. каф. ОТ		
5	Небава М.І., к.т.н., професор кафедри ЕПВМ		

7 Дата видачі завдання 25.03.2022 р.

8 Календарний план наведено в таблиці 2.

Таблиця 2—Календарний план

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Пошук та огляд інформаційних джерел	16.09.22р.	вик.
2	Огляд та аналіз існуючих підходів щодо створення web додатків	14.10.22р.	вик.
3	Результати проведених досліджень та архітектура проекту	05.11.22р.	вик.
4	Проектування програмного засобу	15.11.22р.	вик.
5	Економічна частина	20.11.22р.	вик.
6	Попередній захист	25.11.22р.	вик.
7	Оформлення пояснювальної записки і презентації	10.12.22р.	вик.
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	16.12.22р.	вик.

Студент _____ Капличний О. С.

Керівник роботи _____ Колесник І. С.

АНОТАЦІЯ

УДК 004.93

Капличний О. С. Програмний засіб ведення обліку обслуговування автомобілів. Магістерська кваліфікаційна робота зі спеціальності 123 — комп'ютерна інженерія, освітня програма комп'ютерна інженерія. Вінниця: ВНТУ, 2022. 114 с.

На укр. мові. Бібліогр.: 19, рис. 32; табл. 5.

Магістерську кваліфікаційну роботу присвячено створенню мобільного додатка по веденню обліку обслуговування автомобілів.

Здійснено огляд існуючих сервісів, які дозволяють вести облік обслуговування авто, визначено їх переваги та недоліки. Проведено аналіз найсучасніших програмних рішень та технологій для написання власного програмного забезпечення для ведення обліку обслуговування автомобілів. Проведено дослідження, що дозволили проаналізувати існуючі, та запропонувати власний сервіс для автомобілістів. На основі отриманих знань, розроблено повністю працюючий прототип, який можна випускати в користування. Виконано тестування та проведено експерименти на працездатність розробленого програмного засобу, та його енергоефективність на фізичному пристрої. Проведено економічний розрахунок доцільності розробки.

В результаті розробки створено кросплатформний мобільний додаток, який дозволяє вести облік обслуговування автомобілів із модифікованим програмним модулем зв'язку, що дозволяє зробити додаток більш енергоефективним, ніж аналоги. Також проведено оптимізацію додатку під різні екрани смартфонів.

Ключові слова: кросплатформний додаток, Bluetooth, OBD2, React Native, Javascript, TypeScript, мобільна розробка, обслуговування автомобіля, Redux, WebStorm, облік.

ANNOTATION

Kaplychny O. S. Software for keeping track of car maintenance. Master's qualification work on specialty 123 — computer engineering, computer engineering educational program. Vinnytsia: VNTU, 2022. 114 p.

In Ukrainian speech Bibliogr.: 19, fig. 32; table 5.

The master's thesis is devoted to the creation of a mobile application for keeping records of car maintenance.

An overview of the existing services that allow you to keep records of car maintenance was carried out, and their advantages and disadvantages were determined. An analysis of the most modern software solutions and technologies for writing your own software for keeping track of car maintenance was carried out. Research was carried out, which allowed to analyze the existing ones, and to offer our own service for motorists. Based on the knowledge gained, a fully working prototype has been developed that can be put into use. Testing and experiments were carried out on the performance of the developed software and its energy efficiency on the physical device. An economic calculation of the feasibility of the development was carried out.

As a result of the development, a cross-platform mobile application was created, which allows you to keep track of car maintenance with a modified communication software module, which allows you to make the application more energy efficient than its counterparts. The application has also been optimized for different smartphone screens.

Keywords: Cross-Platform App, Bluetooth, OBD2, React Native, Javascript, TypeScript, Mobile Development, Car Service, Redux, WebStorm, Accounting.

ЗМІСТ

ВСТУП

В наші часи інтеграція мобільних пристроїв у всі сфери не є новинкою, кожна поважаюча себе компанія випускає свій додаток, в якому є певний функціонал для її клієнтів. Додатки доставки, платформи для продажі, Інтернет магазини, всі вони мають основний трафік користувачів саме завдяки мобільним додаткам, деякі компанії взагалі існують завдяки наявності мобільних додатків та мобільних пристроїв [1].

Основна ставка на користувачів ставиться на те, що функціоналом можна користуватися в будь-якому місці і в будь який час. Також все зводиться до автоматизації, дуже багато служб підтримки фільтрують людей через ботів, а вже потім беруть заявки в обробку. Таким чином мобільні гаджети та Інтернет технології полонили весь світ, всі сфери обслуговування і продажу, оскільки це банально зручно в користуванні. Зараз відсутність сайту або мобільного додатку викликає недовіру до компанії, ніби ця компанія настільки неуспішна, що не може навіть зробити нормальний додаток.

Існують компанії, які існують тільки в телефоні, прикладом такої компанії є Монобанк. Це банк, у якого відділень немає фізично, він існує тільки завдяки мобільному додатку. На перший погляд це звучить трішки дивно, але якщо розібратись в цьому питанні, то це несе тільки позитивні моменти в користуванні цим банком. Відсутність черг, фізичних документів, які можуть губитись, не потрібно більше тратити час на те, щоб поїхати в іншу частину міста та перевідкрити договір або відкрити ФОП, все можна зробити за кілька хвилин у вашому смартфоні.

Не є виключенням для поширення мобільних технологій і Інтернету й автомобільна індустрія. На даний момент у найсучасніших машин є додатки, які дозволяють виконувати паркування машини дистанційно, вмикати обігрів, заводити, провітрювати салон в спекотний день, коли ви стоїте у черзі до каси в якомусь магазині і багато чого іншого. Раніше деякі компанії робили спеціальні опціональні ключі до машини, які коштували значну кількість коштів, але на

відміну від звичайних мали сенсорний екран із вбудованим міні додатком. який дозволяв користуватись деякими функціями машини вмикаючи їх дистанційно. Потім це переросло в звичайні додатки в телефоні із тими самими функціями, оскільки для виробника дешевше оплатити розробку додатка на смартфон ніж оплатити розробку ключа, знайти виробника, який виготовить ці ключі по замовленню, розробити програмне забезпечення на цей ключ і так далі. Також перевагою додатка є те, що в ньому не може бути браку. Ключ може бути неякісним, може бути погано зібраним, і взагалі може бути багато нюансів, які підпадають під гарантійний випадок, з додатком такого бути не може, просто буде невелика команда розробників, які підтримуватимуть цей додаток та випускатимуть для нього оновлення [2].

Таким чином, приходимо до висновку, що зараз мобільні додатки є основою бізнесу та багатьох великих компаній, це найзручніший спосіб для кінцевого користувача отримувати інформацію.

Візьмемо автомобільну галузь, користувачі тут ще не такі вибагливі до таких речей як мобільні додатки. На даному етапі розвитку інтеграція автомобіля зі смартфоном є більше рідкістю ніж буденністю, але ж тут також є багато речей, які можна спростити просто інтегрувавши сюди мобільні технології.

Невід’ємною частиною експлуатації авто є його обслуговування, саме від нього залежить можливість довгого користування та відсутність проблем у власника транспортного засобу. Але зазвичай людям досить незручно вести облік обслуговування авто, тому заміна деяких розхідних матеріалів в більшості випадків несвоєчасна. Деякі люди записують ще в блокнот на листок або ще кудись, що не є зручним оскільки не завжди під рукою. Дехто записує в нотатки в телефоні, але теперішній інформаційний світ наскільки бурний, що ця інформація також втрачається. Але незаперечним є той факт, що в кожного незамінним інструментом є смартфон, який завжди є під рукою, також зараз досить важко уявити життя без мобільного Інтернету. Тож було вирішено створити програмний засіб, який би автоматизував ведення обліку обслуговування автомобіля, створював нагадування про необхідність

обслуговування, та міг би підказати найближчі станції технічного обслуговування, на яких можна провести чергове обслуговування транспортного засобу.

Також важливим фактором на сьогоднішній день є кросплатформність додатку, оскільки частина людей користуються Android, а частина використовує IOS і випускаючи додаток тільки для якоїсь однієї платформи інша платформа лишається можливості користуватися ним. Питання кросплатформності можна вирішити двома способами, можна зробити два ідентичних додатки на обидві платформи окремо, а можна скористатись веб-технологіями, які дозволяють писати один код, який буде компілюватись в два додатки на дві платформи. Із обмеженим бюджетом перший варіант стає не реалізованим, оскільки коштів на дві команди розробників необхідно більше ніж на одну команду, а недоліки додатка, який буде написаний на кросплатформній технології незначні і у випадку цієї роботи неважливі взагалі.

Отже, виходячи із слів наведених вище, можна прийти до висновку, що тема є актуальною на даний час.

Актуальність дослідження полягає у створенні додатку, який об’єднає в собі функції що у значній мірі полегшать процедуру ведення обліку обслуговування транспортного засобу.

Об’єктом дослідження є методи та засоби обробки інформації та взаємодія з даними автомобіля для програмного засобу ведення обліку обслуговування автомобіля.

Предметом дослідження є процес та методи створення інформаційної системи у вигляді кросплатформного додатка.

Мета дослідження полягає у реалізації програмного продукту який матиме змогу при потребі легко масштабуватись та матиме зручний функціонал для кінцевого користувача.

Щоб досягнути поставленої цілі потрібно вирішити такі **задачі**:

— провести аналіз аналогів;

— вивчити особливості роботи з React Native;

— вивчити особливості обслуговування автомобілів;

— запропонувати ведення обліку;

— створити лабораторний зразок програмного засобу із певним набором функціоналу;

— виконати економічні розрахунки по доведенню доцільності розробки нового програмного продукту;

— зробити висновок за результатами розробки.

Наукова новизна полягає у вдосконаленні обробки даних з автомобіля для автоматизації процесу обслуговування транспортного засобу та об’єднання багатьох функцій в одному програмному засобі, що дозволило зменшити енергозатрати мобільного пристрою при використані даного програмного засобу.

Практичне значення одержаних результатів наукового дослідження полягає у створенні платформи для загального користування та зручного її масштабування до комерційних розмірів.

Апробація - результати магістерської роботи направлені на Всеукраїнську науково-практичну Інтернет-конференцію «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» (Вінниця, 2023 р.) під назвою «Програмний засіб ведення обліку обслуговування автомобілів та його тестування».

Особистий внесок здобувача - основні положення й результати магістерської роботи отримані автором самостійно та пов’язані з його професійною діяльністю.

1 АНАЛІЗ СПОСОБІВ ПОБУДОВИ ПРОГРАМНИХ ЗАСОБІВ

На сьогоднішній день транспорт є дуже важливою складовою повсякденного життя. В цьому розділі наведено дослідження актуальних методів взаємодії з програмною частиною транспортних засобів, побудови програмного засобу за кращими практиками та визначення актуальності створюваного програмного продукту.

1.1 Огляд технології для створення мобільного додатку

Поняття інформаційне суспільство з’явилось відносно недавно, це поняття визначає таке суспільство, в якому більшість робочих займаються виробництвом, обробкою, зберіганням та переробкою і реалізацією інформації в тому числі і найбільш цінною інформацією – знаннями. Таке суспільство вже дійшло до такої стадії розвитку що для нього значно збільшилось значення інформації, знань та інформаційних технологій в житті кожної людини. Тут збільшилась кількість людей які займаються інформаційною сферою, та сферою обслуговування галузей інформаційного глобального простору. Це позитивно впливає на інформаційну взаємодію людей, з’являється електронна демократія, інформаційна економіка, цифрові ринки, електронне управління та поширення соціальних мереж.

Таким чином суспільство розвивається і постійно створює нові технології, які з кожним разом перевершують їх попередників. Також позитивно це впливає і на формування інформаційного соціуму. Таким чином створюються не лише нові продукти, але і інструменти для їх розробки, внаслідок чого розробка стає більш доступною для середньостатистичної людини, розробникии це вже не тільки такі люди які володіють якимись рідкісними знаннями і щоб їх здобути потрібно приділити з десяток років науці, це вже звичайні особи яким просто цікава ця галузь, вони можуть і не мати профільної освіти в цьому напрямку, зараз такі знання можна здобути самоосвітою. А інструменти розробника стають все більш інтуїтивними та приємними для використання.

Відносно недавно мова JavaScript могла використовуватись тільки для накладання більшого функціоналу сайтам в браузері, тепер ця мова охоплює дуже багато інших галузей, вже не можна сказати що ця мова придумана тільки для того щоб «оживити» сайт. На цій мові побудовано дуже багато інших інструментів якими розробники користуються щодня. Більшість додатків які є в кожного на телефоні написані за допомогою JavaScript, це і додатки магазинів, банків, інтернет магазинів, страхових компаній, програми соцмереж, програми для слідкуванням за комунальними послугами і так далі. І всі вони написані на JavaScript або на її фреймворках.

Близько семи років назад, компанія Facebook запропонувала спільноті роробників новий фреймворк для розробки на JavaScript – React Native.

React Native — це крос-платформний фреймворк з відкритим вихідним кодом для розробки нативних мобільних та настільних програм на JavaScript та TypeScript, створений Facebook, Inc. Він побудований на базі ReactJS і працює на базі WebView, тому немає DOM API. React Native підтримує такі платформи, як Android, Android TV, iOS, MacOS, Apple tvOS, Web, Windows та UWP. Фреймворк дозволяє розробникам використовувати можливості бібліотеки React поза браузером для створення нативних програм, що мають повний доступ до системних API платформ. React Native не має HTML і CSS, але є деякі компоненти платформи в JSX та CSS-подібних поліфіліях[2].

Весь додаток будується із так званих компонентів, це такі елементи які призначені для користувацького інтерфейсу, вони завжди мають свою структуру, за потреби мають свій внутрішній стан який може пов’язуватись із гобальним станом додатка виконуючи певний функціонал який потрібен розробнику.

React Native має деяке япі над деякими нативними компонентами, саме через це код написаний розробником має зв’язок із функціями смартфону та наприклад такими речами як доступ до камери та пам’яті смартфону. Зв’язок між збілдженим бандлом та власне смартфоном здійснюється через bridge. Більш детально ознайомитись з архітектурою React Native можна на рисунку 1.1.

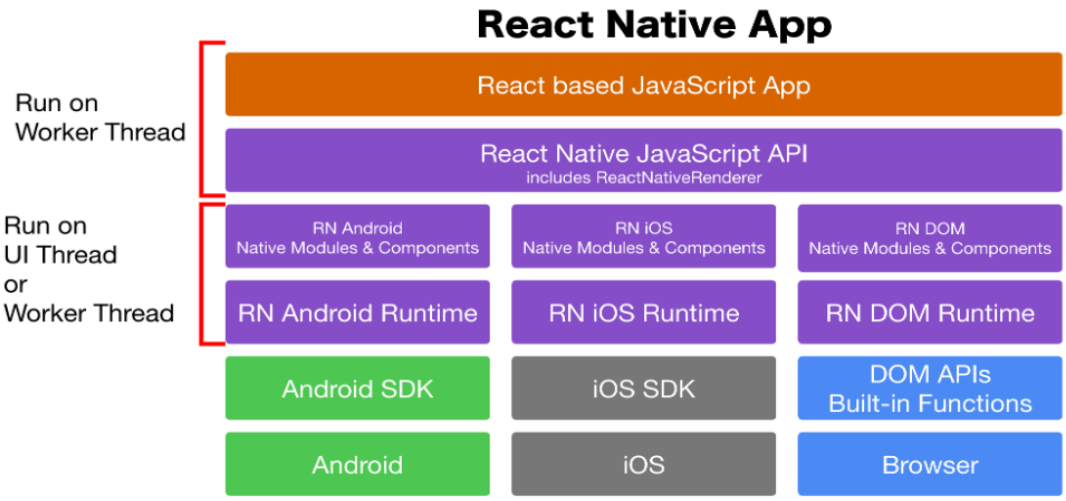

Рисунок 1.1 — Архітектура React Native

Плюси React Native:

— кросплатформність, зазвичай це стає ключовим фактором при виборі технології;

— вартість кінцевого продукту, виходячи з того що розробників і часу на розробку двох додатків на двох мовах потрібно більше, то дана технологія виглядає як швидке та недороге рішення;

— велике ком'юніті, це дуже важливо для розробника при розробці якогось дійсно серйозного функціоналу;

— великий вибір готових бібліотек, завжди можна написати щось своє, але добре коли більшість питань вже має готові перевірені рішення;

— додаток можна кастомізувати під кожну систему окремо, при специфічних вимогах замовника це досить корисно;

Підбір спеціалістів для розробки також не є проблемою, навіть при дефіциті React Native розробників можна знайти React розробників які з легкістю можуть мігрувати між однією мовою та іншою. Перехід між ними максимально простий, порібно тільки вивчити базові компоненти React Native та відповідності веб компонентів до мобільних компонентів.

Є і особливості які притаманні як React Native так і React, наприклад те що невід'ємною частиною кодової бази додатка є клас App, він потрібен для

коректної обробки помилок. Розробники цих мов постійно обіцяють це виправити і замінити всі на функціональні компоненти, але поки що це не видається можливим, хоча останні кілька років над цим ведеться активна робота.

Також в кожного компонента є свій життєвий цикл, це його ренер, оновлення та видалення. Раніше це як і обробка помилок відбувалось через класові компоненти, зараз це вже застарілі методи і зараз вважається хорошою практикою коли все побудовано на функціональних компонентах, в них розробниками передбачено абсолютно все окрім того єдиного класу який відповідає за помилки. На кожному етапі життєвого циклу компонента ми можемо визначити певні дії які будуть відбуватись з компонентом, як будуть отримуватись дані, як будуть вони оброблятись, яким чином їх потрібно відмалювати користувачу і таке інше. Більше подробиць можна отримати на рисунку 1.2.

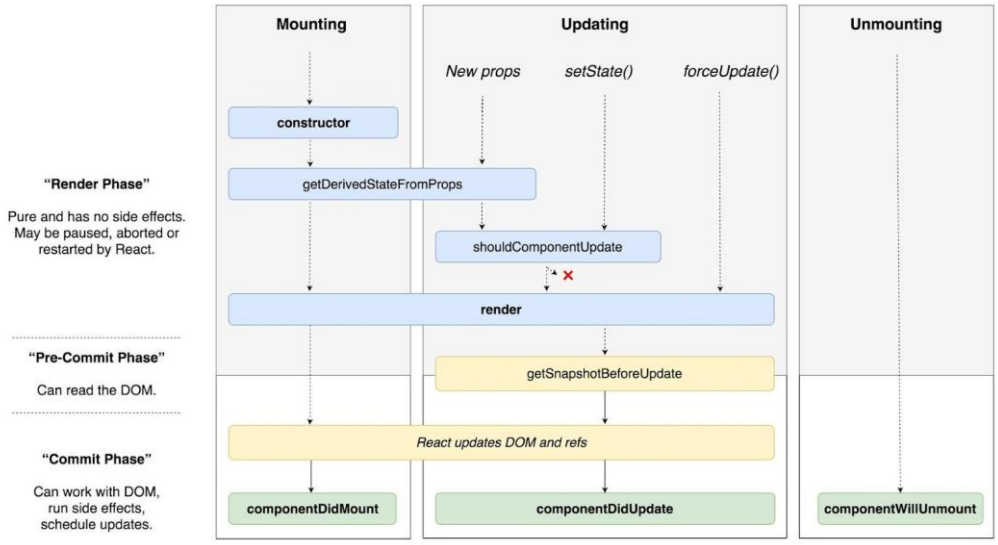

Рисунок 1.2 — Методи життєвого циклу React

Життєвий цикл будь-якого компонента завершується видаленням компонента із загального дерева компонентів сторінки. На такому етапі зазвичай виконують від’єднання від різних лістнерів, сокетів, відключення різних таймерів та подібного. Якщо не робити цього то відбувається витік пам’яті, таким чином додаток може використовувати набагато більше пам’яті ніж повинен від чого може почати глючити та працювати не дуже приємно для

користувача, такі помилки перевіряються зборщиком або середовищем в якому відбувається компіляція написаного коду і при допусканні таких помилок в консолі будемо спостерігати помилки в яких буде написано що допущені помилки і відбувається витік пам’яті.

Також передбачено ще досить багато помилок які розробники допускають доволі часто, це такі помилки як зацикленність якихось станів, невірна обробка даних, неправильний рендеринг даних, вказання неправильних стилів, неправильне використання методів життєвого циклу і багато чого ще. Це зроблено розробниками React Native для полегшення роботи програмістів, насамперед тих в кого ще не так багато досвіду щоб одразу зрозуміти в чому проблема, і щоб програміст не витрачав години на те щоб знайти де він допустив помилку, інструмент сам вкаже де помилка та якого вона характеру, якщо це зовсім типова помилка то навіть напише підказку як це виправити, для менш типових помилок передбачені посилання на документацію де теоретично може бути відповідь на майбутні питання програміста.

Але не тільки для недосвідчених розробників зроблені такі зручності, дуже часто при довгій розробці програміст втрачає пильність та може допустити банальні опечатки, або просто зробити помилку по неуважності, адже на даний момент розробниками є люди, а як всім нам відомо людський фактор завжди існує.

1.2 Сучасні архітектурні рішення

З кожним роком кодова база та стиль коду який вважається «найкращою практикою» змінюється. Постійно одна технологія змінює іншу, певні стандарти змінюються, виходять все нові і нові кодові рішення які оптимізованіші ніж ті які були до цього і так далі.

Таким чином деякі бібліотеки, які вважаються класикою у світі React та React Native, змінюються структурно майже повністю, і це нормально, адже світ не стоїть на місці і постійно вдосконалюється. Прикладом «класики» в React вважається менеджер станів додатка Redux. Це основа великих додатків, так

зване сховище в якому розташовуються всі дані для роботи додатка, всі стани сторінок та компонентів які потрібні для відображення потрібної інформації користувачу. Розглянемо цю технологію більш детально.

В останні роки Redux поділяється на «класичний» Redux та Redux Toolkit. Зараз при розробці з нуля завжди використовується Redux Toolkit, оскільки вже весь світ перейшов на функціональні компоненти, хуки та інші зручні рішення, а класичний Redux використовується більше для старих проектів які потрібно підтримувати або які вже настільки масштабні, що переписати їх на новий лад не являється можливим. Новий Redux Toolkit пропонує готові спрощені рішення для роботи із запитами на сервер, готові рішення для шаблонних даних, таких як таблиці, списки, безкінечні списки із підвантаженням, заготовки для пагінації і так далі. Такі рішення значно спрощують життя програмісту, оскільки при використанні старого Redux він повинен був все це написати самостійно, а тут вже все працює «з коробки».

Розглянемо API які пропонує Redux Toolkit.

ConfigureStore() — обгортка для createStore(), що спрощує налаштування сховища за замовчуванням. Дозволяє автоматично комбінувати окремі редуктори (slice reducers), додавати проміжні шари або посередників (middlewares), за замовчуванням включає redux-thunk (перетворювача), дозволяє використовувати розширення Redux DevTools (інструменти розробника Redux);

CreateReducer() — дозволяє використовувати таблицю пошуку (lookup table) операцій для редукторів випадку (case reducers) замість інструкцій switch. У цьому API використовується бібліотека immer, що дозволяє безпосередньо змінювати іммутабельний код;

CreateAction() — генерує творця операції (action creator) для переданого типу операції. Функція має перевизначений метод toString(), що дозволяє використовувати її замість константи типу;

CreateSlice() — приймає об'єкт, що містить редуктор, назву частини стану (state slice), початкове значення стану, та автоматично генерує частковий редуктор з відповідними творцями та типами операції;

CreateAsyncThunk() — приймає тип операції та функцію, що повертає проміс, і генерує thunk, що відправляє типи операції pending/fulfilled/rejected на основі промісу;

CreateEntityAdapter() — генерує набір редукторів і селекторів, що перевикористовуються, для управління нормалізованими даними в сховищі[7];

Сховище Redux нічого не знає про асинхронну логіку. Воно знає лише про те, як синхронно відправляти операції, оновлювати стан за допомогою виклику кореневого редуктора та повідомляти UI про зміни. Будь-які асинхронні операції мають виконуватися поза сховища.

Найчастішим випадком використання посередників є забезпечення взаємодії асинхронної логіки зі сховищем. Це дозволяє писати код, що відправляє операції та перевіряє сховище, зберігаючи цю логіку незалежної від UI.

Існує кілька посередників для реалізації асинхронності у Redux. Рекомендовано Redux Thunk. Він чудово підходить для більшості випадків, а використання синтаксису async/await робить його ще кращим. configureStore() встановлює thunk автоматично.

Також на сьогодні є хорошою практикою використання TypeScript. TypeScript або TS – це мова з відкритим вихідним кодом, розроблена та підтримується Microsoft. У сучасному світі, TypeScript це вже більше необхідність ніж рідкість, його використовують майже в усіх серйозних проектах. Він використовується для полегшення розробки та знаходження помилок ще на стадії розробки:

— це багатопарадигмальна мова (як JavaScript).;

— це альтернатива JavaScript;

— він дозволяє використовувати статичні типи;

— він має додаткові функції (генерики, інтерфейси, кортежі тощо);

— це дозволяє поступове впровадження (тобто ви можете перетворити існуючий проект на проект TS, змінюючи файли один за одним, це не велика зміна);

— Ви можете використовувати його для зовнішньої та внутрішньої розробки (як JS);

Браузер не розуміє код TS. Він повинен бути транскомпільований у JS. JS має значення динамічного відображення типів, а TS має статичні типи, які менш схильні до помилок.

У React відбувається транскомпіляція JS за допомогою Babel, тому потреба транскомпілювати код сьогодні не є додатковою незручністю.

TypeScript, Java та багато інших мов мають статичну типізацію, яка визначатиме тип, пов'язаний зі змінною. Тип буде перевірено під час компіляції. Після того, як ви визначили щось як рядок або логічне значення, ви не можете змінити його тип.

JavaScript, з іншого боку, має динамічну типізацію. Це означає, що ви можете призначити рядок змінній, а потім перетворити її на логічне значення, число чи будь-що інше. Тип буде динамічно призначено під час виконання.

З тих самих причин програмісти використовують IDE, розширення IDE, підсвічування синтаксису та лінтери замість програми Блокнот. TypeScript може вписатися в ці допоміжні засоби.

Також TS допомагає при написанні коду підказками в середовищі розробника, при потребі написання якогось аргументу функції середовище саме підкаже якого типу цей аргумент повинен бути, а якщо якась компонента приймає параметр невірного типу то середовище підкреслить його червоним та напише який тип ви прокинули і який тип потрібно прокинути. Також це працює при створенні типізованих об'єктів.

Щодо файлової структури в React спільноті також є певні прийняті правила яких всі намагаються дотримуватись. Групування за ознаками або маршрутами, одним із поширених способів структурування проектів є розміщення CSS, JS і тестів разом у папках, згрупованих за функціями чи маршрутами. Визначення «функції» не є універсальним, і ви самі вибираєте деталізацію. Якщо ви не можете створити список папок верхнього рівня, ви можете запитати

користувачів вашого продукту, з яких основних частин він складається, і використати їх розумову модель як план.

Групування за типом файлу це ще один популярний спосіб структурування проектів — групувати схожі файли разом. Деякі люди також вважають за краще піти далі та розділити компоненти в різні папки залежно від їх ролі в програмі. Наприклад, Atomic Design — це методологія проектування, побудована на цьому принципі. Слід пам’ятати, що часто більш продуктивно розглядати такі методології як корисні приклади, а не суворі правила, яких слід дотримуватися.

Є багато проблемних моментів, пов’язаних із глибоким вкладенням каталогів у проекти JavaScript. Стає важче записувати відносні імпорти між ними або оновлювати ці імпорти під час переміщення файлів. Якщо у вас немає вагомої причини використовувати глибоку структуру папок, подумайте про те, щоб обмежитися максимум трьома або чотирма вкладеними папками в одному проекті. Звичайно, це лише рекомендація, і вона може не стосуватися вашого проекту. Їх слід дотримуватися.

1.3 Огляд аналогів розроблюваного програмного засобу

Розглядаючи ринок можна знайти продукти які є аналогічними до того що розроблюється в даному проекті. На приклад програма CarLogger, вона також дозволяє записувати та отримувати дані з автомобіля, виводити їх на екран смартфону користувача та записувати їх відповідно. Із цікавих інструментів які пропонує дана програма є виведення різних даних автомобіля у графіках (рис. 1.3). Це буває корисно при виявленні різних неявних поломок в машині, або для слідкування за роботою окремих модулів машини таких як наприклад турбіна чи двигун. Вся ця інформація отримується через OBD роз’єм машини, зазвичай зчитується через провод, блютуз чи вай фай модуль який підключається до телефону безпосередньо без проводів. Такі модулі є доволі популярними і знайти такий звичайному користувачу не стане проблемою.

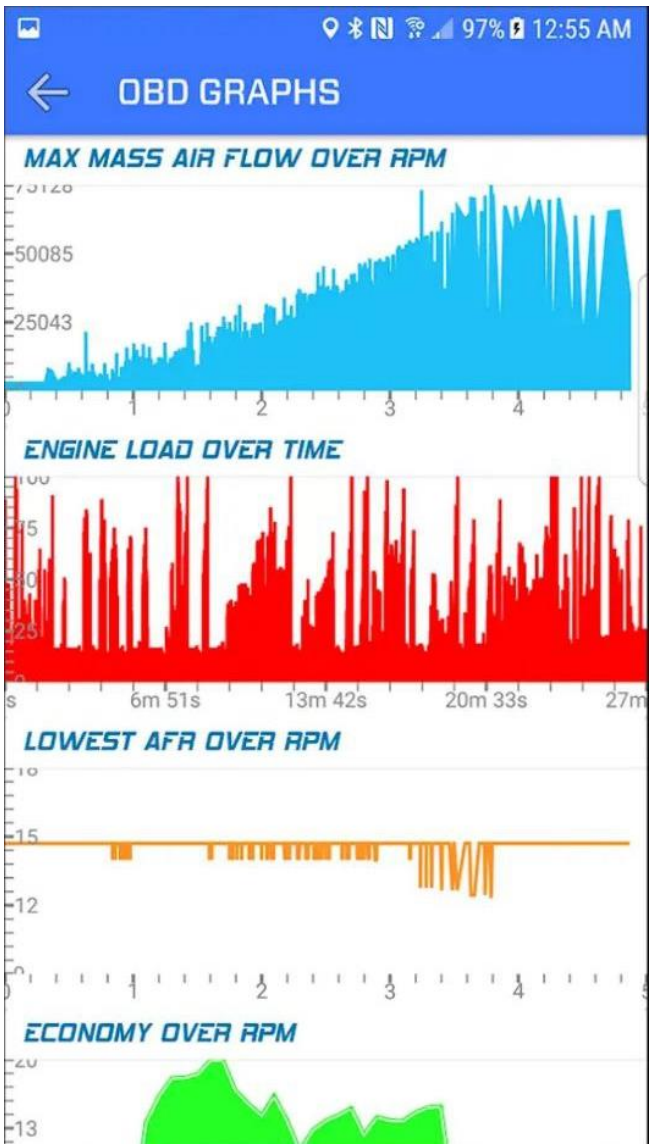

Рисунок 1.3 — Графіки в програмі аналозі

Також із функціоналу в даному додатку є розрахунок вартості місяця користування автомобілем відносно палива яким заправлене авто. Тобто при кожній заправці користувач вводить ціну пального, кількість якою він поповнив бак транспортного засобу, та відстань яку він проїхав після останнього відвідування заправки. Ці дані також відображаються в якості графіка де є шкала ціни та шкала місяців, таким чином люди які мають великі пробіги можуть відслідковувати скільки в середньому за кожен місяць вони тратять коштів на пальне. Далі ці дані при тривалому використанні додатка записуються протягом великого часу і стає доступна статистика за кожен рік користування (рис. 1.4).

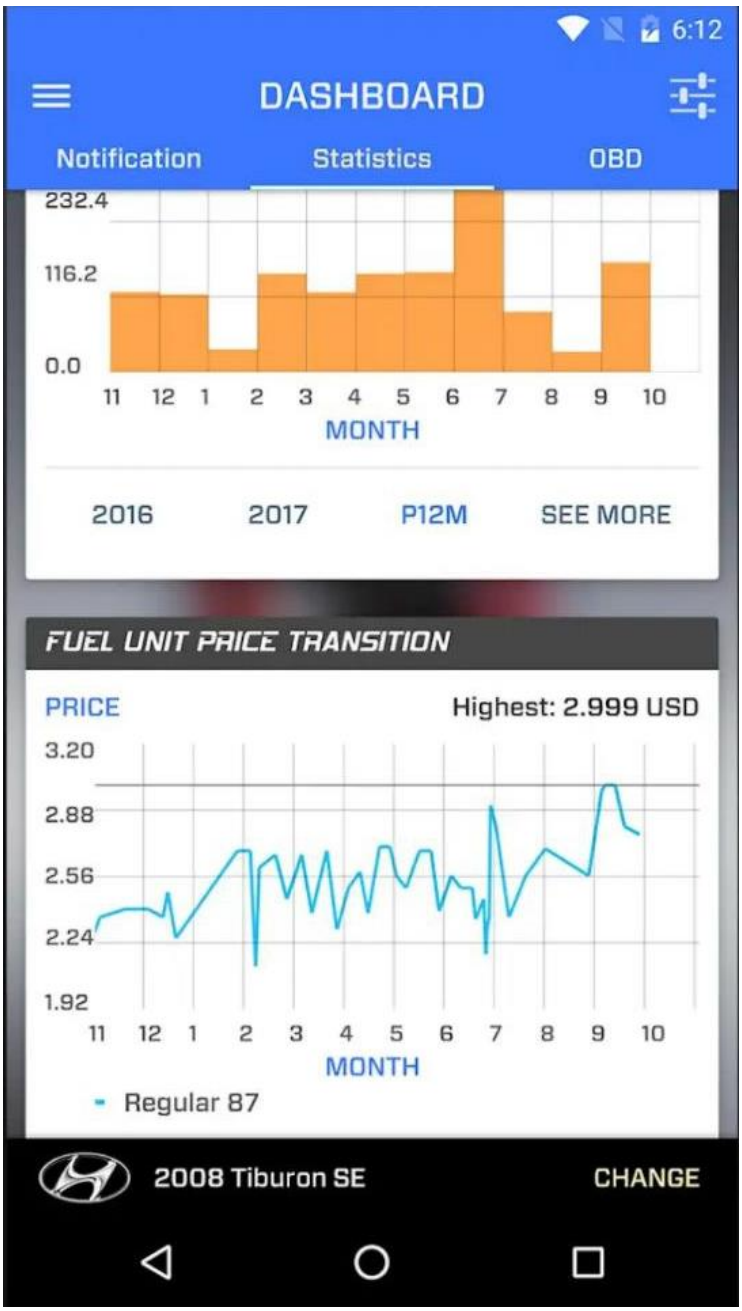

Рисунок 1.4 — Статистика

Також є в додатку сторінка із загальними даними про авто, такими як вартість, середня ціна милі яку ви проїхали на цьому авто, місткість бака, розхід і багато інших загальних даних. Під цим блоком є більш цікавий блок із даними про розхідники, при натисканні на який можна отримати список розхідників, вартість за яку вони були замінені та дату заміни. Також тут є досить багато інформації яка відноситься до стану авто, така як поточна температура охолоджуючої рідини, температура масла, та інше. І взагалі додаток

перенасичений даними про стан авто, тобто вести ним історію обслуговування досить незручно адже він здебільшого розрахований на спостереження за поточним станом авто в режимі реального часу.

Також мінусом слід відмітити відсутність підтримки IOS, тобто додаток одноплатформний, що по-перше зменшує потенційну кількість його користувачів, а по-друге користувачі які змінили телефон на телефон з іншою операційною системою автоматично перестануть користуватись даним додатком. Слід відзначити і обмежений функціонал та велику кількість реклами на безкоштовній версії додатку.

Ще одним прикладом аналогу є сервіс DriverNotes.net, в цього сервісу є як і веб версія так і мобільна версія на обидві мобільні платформи. Цей додаток на відміну від попереднього має більш сучасний інтерфейс та дизайн. Тут також можна вести записи щодо обслуговування авто, записати можна фактично майже кожну витрату, як заміну масла чи заправку, так і поїздку на мийку самообслуговування, додаток достатньо сильно розвинений в цьому напрямку (рис. 1.5).

Мінусом який одразу потрапляє в поле зору користувача є реклама, її тут досить багато і якщо в попереднього аналогу є можливість її відключити за гроші то в цьому додатку такої можливості немає. Слід відмітити погану підтримку додатка, в маркеті є відгуки яким вже по кілька місяців в яких написано що додаток видаляється після оновлення операційної системи і більше його встановлення не є можливим. На такі відгуки немає ніякої відповіді від розробників, тобто рішення цієї проблеми не те що немає, нею навіть не займались. Тому довіра до такого сервісу одразу зникає.

Проаналізувавши аналогічні продукти можна прийти до висновку що в кожного із них є недоліки, які при розробці власного продукту можна виправити. Також є багато моментів яких немає в цих додатках але вони були б дуже зручні для кінцевого користувача. Це такі функції як додавання подій за календарем,

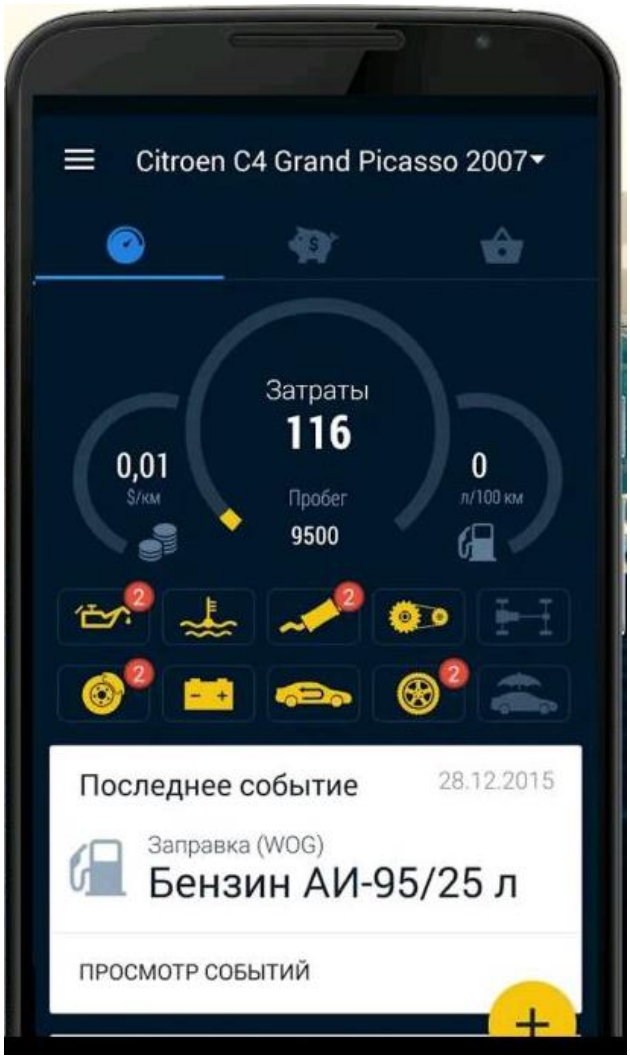

Рисунок 1.5 — Головна сторінка додатка DriverNotes

карта найближчих СТО, можливість запису до цих СТО, запис та ведення статистики по заміні розхідників і наявність користувацьких нагадувань які б індивідуально підлаштовувались під кожного хто користується додатком.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

Одною із найголовніших частин створення програмного забезпечення є вибір технологій та проектування архітектури. Саме на цьому етапі визначається наскільки технічно правильним та підтримуваним буде розроблюване програмне забезпечення. В даному розділі буде проведено аналіз принципів проектування мобільних додатків такого типу та вивчено найкращі рішення які зараз є в сфері програмування.

2.1 База даних Firebase як серверна частина проекту

Firebase — це база даних від компанії Google, за допомогою якої користувачі можуть зберігати і отримувати інформацію, зі сторони розробника вона надає зручні та прості рішення для взаємодії із даними. Також вона є досить гнучкою в плані продуктивності і при масштабуванні проекту її завжди можна розширити.

Firebase дозволяє зберігати текстові дані в JSON форматі, при потребі можна зберігати і медіафайли, такі як фото, відео та різні звукові записи[4]. Все працює бездоганно, це рішення вже не перший рік на ринку тому сумніватись у надійності даного сервісу не приходиться. Всі методі для взаємодії із даними, зчитування, зміна, їх збереження, все описане в подробицях в документації і працює що називається «з коробки». Для демо проектів надається безкоштовне місце на хмарі розміром у 5 Гігабайт пам’яті, цього достатньо для розробки, тестів та початкового старку проекта, але при масштабуванні завжди є можливість цеп сховище розширити.

Звісно такий відточений до дрібниць продукт не може бути повністю безкоштовним, велика кількість найкращого функціоналу залишається тільки для тих хто може заплатити за нього, але основні функції які зазвичай потрібні для більшості розробників безкоштовні і у відкритому доступі завжди. Це такі функції як асторизація та реєстрація, логін через гугл акаунт, зберігання та редагування тексту та мультимедіа і так далі.

Ця база даних має значний плюс перед конкурентами – це гнучкість та швидкість роботи і деплоя в проект. Сам по собі JavaScript гнучкий та швидкий при розробці програмного забезпечення, і дуже не хотілось би втрачати ці плюси при відключенні до проекта сторонніх сервісів. Firebase дозволяє зберегти це. API вже реалізоване розробниками Google, все відтестоване і готове до встановлення окремою бібліотекою. Фактично вся серверна частина лягає на цей сервіс, можна випустити додаток без власного фізичного серверу і знання мови програмування для написання свого власного серверу. Це значно пришвичує розробку, вартість розробки та можливі помилки які можуть виникати в процесі розробки.

Завдяки такому пропрацюваному інструменті відносно швидко можна реалізувати функціонал який потрібно замовнику і показати демо версію, а потім вже допрацювати за правками які замовник вкаже.

При додаванні Firebase до проекту, спочатку потрібно зареєструватись, якщо акаунт гугл вже є то можна просто авторизуватись через нього.

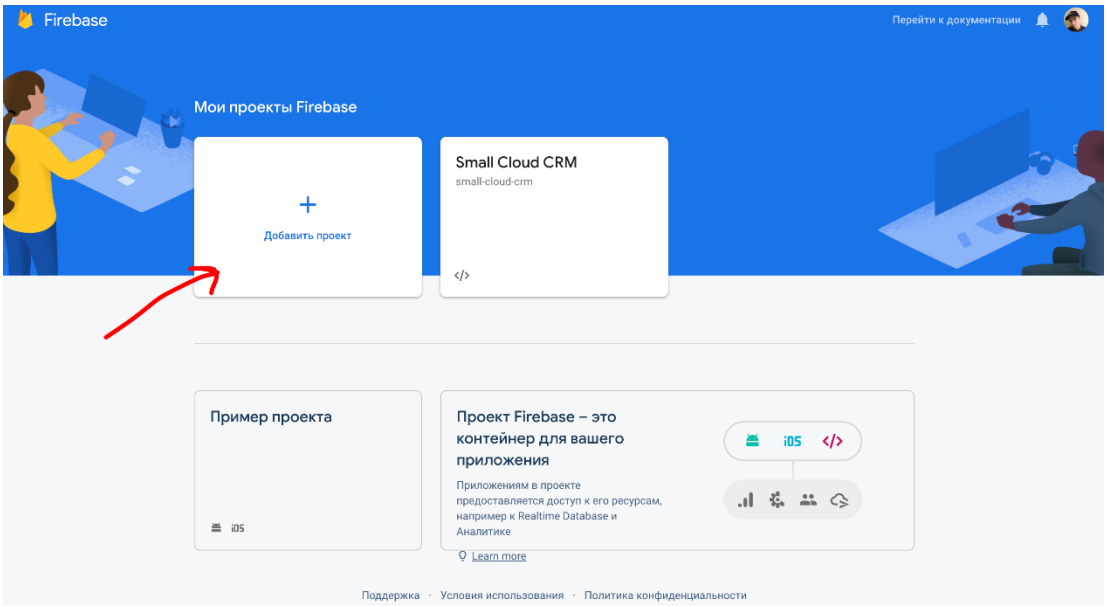

Рисунок 2.1 — Панель управління Firebase

Після авторизації потрібно створити новий проект, для цього потрібно натиснути на велику кнопку на якій написано «Додати проект» як показано на рисунку 2.1, після чого відкриється сторінка консолі розробника.

При потраплянні в консоль, для цього проекту потрібні три речі: база даних, аутентифікація та налаштування проекту. Для кожного пункту є окрема вкладка на сторінці консолі, де можна провести налаштування будь якого із цих пунктів.

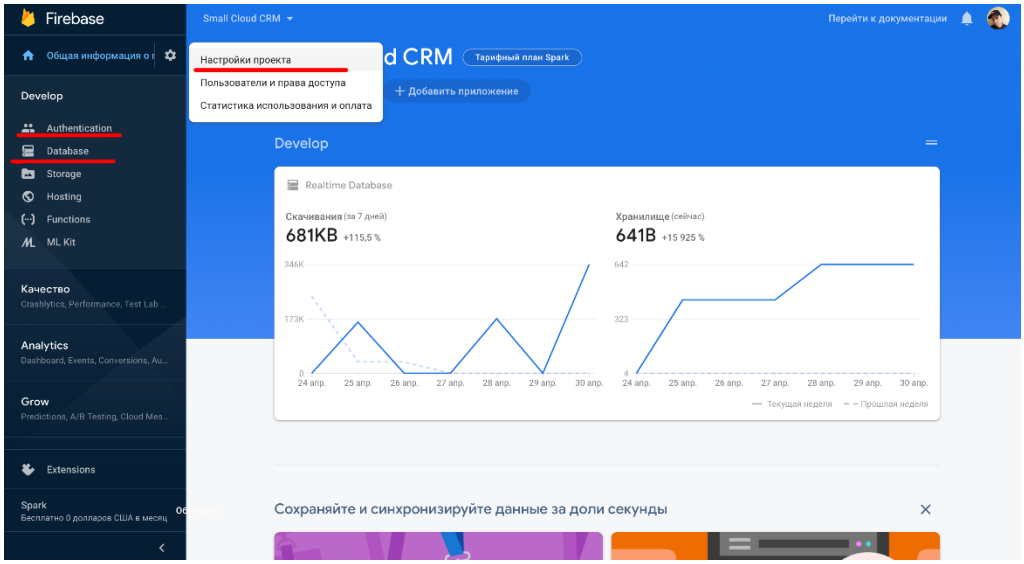

Рисунок 2.2 — Консоль проекту Firebase

Після того як всі параметри налаштовані, потрібно здійснити наступний крок — підключити Firebase до проекту.

2.2 Файлова структура та організація проекта в середовищі розробки

Існують певні неписані правила за якими повинна будуватись структура файлів у кожному веб проекті. Тобто вона може бути інакшою, але зазвичай всі проекти структурно майже однакові, це зроблено для того, щоб будь-який програміст якому прийдеться працювати з кодовою базою проекта міг відкрити його і з перших хвилин вже розумів де і що знаходиться.

Для цього в корені проекта є папка src, в якій знаходиться майже увесь основний код програми. В цій папці створюються певна кількість папок кожна з яких призначена для того щоб містити в собі схожі за семантичним навантаженням речі(рис. 2.3).

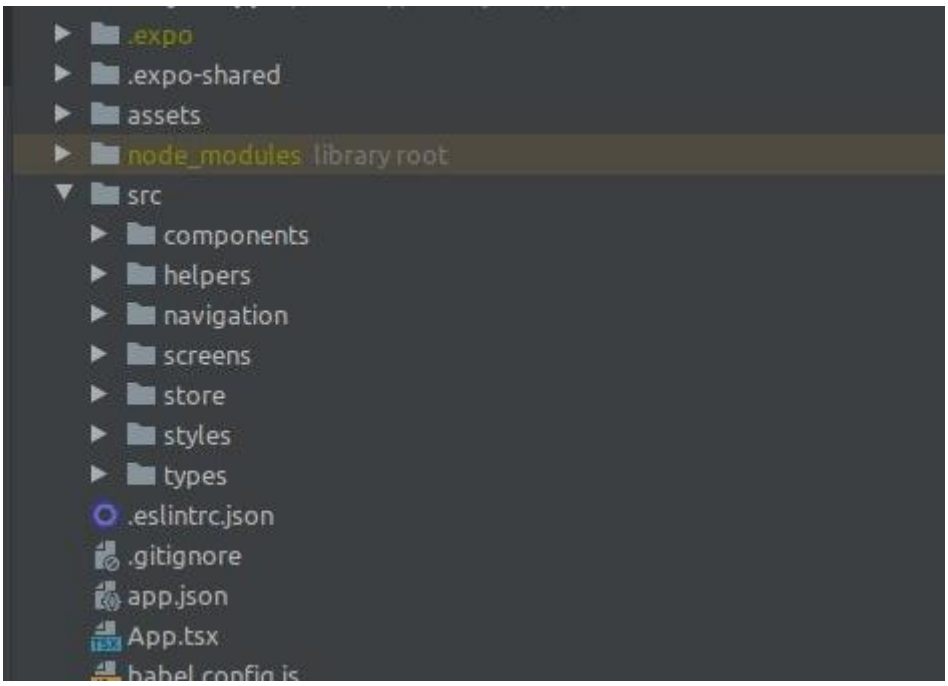

Рисунок 2.3 — Організація структури папок

В будь якому веб проекті завжди буде папка components або ui, тут, як можна зрозуміти із назви містяться всі компоненти з яких будується інтерфейс, зроблено це для того щоб уникати дублікації коду, код який туди виноситься зазвичай написаний таким чином щоб при потребі його можна було підлаштувати під різні умови.

Є папка helpers або її ще також називають utils, тут будуть знаходитись різні допоміжні функції для обчислень, форматування об’єктів, масивів чи чогось іншого, функції для форматування чисел при виводі на екран, також різні частини коду які будуть повторюватись також будуть виноситись сюди.

Navigation — папка для конфігурацій нафігації, тут буде лежати опис всіх сторінок на проекті, умови переключення, обробки неіснуючих сторінок, сайдбари, слайдбари, навігаційні панелі і так далі.

Папка screens, її також часто називають pages, оскільки екрани і на мобільному пристрої фактично і є сторінками сайту який потім компілюється в додаток. Тут містяться функціональні компоненти які власне і є сторінками додатка.

Папка store використовується для опису redux, власне сховища інформації додатку, правельніше сказати менеджера станів. Тут будуть міститись папки для екшенів, всі запити на сервер які є в додатку, редюсери, тобто окремі гілки дерева даних програмного засобу і також файл із конфігурацією самого сховища.

В styles будуть виноситись окремі стилі, знову ж таки все для того щоб уникнути зайвого повторення коду, стилі які будуть тут знахдитись буде зручно імпортувати в будь яку частину проекту завдяки їх розташуванню в корені папки src [9].

Якщо проект написаний із використанням TypeScript то для окремих інтерфейсів типізації завжди є папка types, тут винесені загальні типи які використовуються при написанні програми, вони тут не всі, тільки ті які потрібно в багатьох місцях, знову ж таким все в угоду уникання повторення коду. Також хорошою практивою є створення в папці типів підпапок із назвами сторінок, щоб типізацію кожного модуля програми можна було винести в окрему папку.

2.3 Вибір мови розробки додатку

Сама JavaScript з самого початку була створена для того щоб «оживити» веб сторінки в браузері. Виконуваний код написаний на цій мові називається скриптом і може вбудовуватись в розмітку веб сторінки і виконувати різні дії автоматично, як тільки завантажиться сторінка.

Самі по собі скрипти поширюються як звичайний текст, завдяки браузеру їм не потрібен ніякий окремий компілятор чи якась підготовка перед тим як їх можна буде запустити. Це виділяє JavaScript від інших мов програмування які зараз зазвичай використовуються.

На сьогодні код написаний на JavaScript може виконуватись не лише в браузері, його можна запускати фактично будь де, куди є можливість додати будь який із існуючих «движків» JavaScript.

В браузері завжди є вбудований JavaScript движок, який виконує роль компілятора і збірника, його також називають віртуальною машиною JavaScript.

Фактично це ще один рівень абстракції це відбувається перетворення коду який пише людина у код який розуміє машина на якій цей код потрібно запустити.

У різних браузерів є різни движки, в кожного є своє кодове ім'я, наприклад:

— V8 працює у таких популярних браузерах як Chrome і Opera;

— SpiderMonkey підтримує працездатність FireFox;

— Trident і Chakra використовується для різних моделей Internet Explorer;

— для Microsoft Edge був розроблений Nitro і так далі.

Ці назви досить часто використовуються в різній літературі та статтях, тому їх досить корисно знати на пам'ять, оскільки у вас не буде розуміння про який браузер йдеться мова. Наприклад можна прочитати що певний функціонал працює з движком V8, отже одразу дізнаємось що він буде працювати в браузерах Chrome і Opera, відповідно саме ці браузери можна вже не перевіряти на помилки[10].

Самі по собі, движки це дуже складні програмні комплекси, але кожен поважаючий себе програміст повинен знати їх основи:

— сам движок вбудований у браузер, він читає в обробляє текст скрипта;

— саме він перетворює скрипт в той вигляд який сприймає машина – в машинний код;

— після цієї процедури машинний код працює і запускається досить швидко.

Двигун не тільки компілює код в машинний, а й оптимізує його на кожному етапі збірки. Під час роботи скрипта двигун переглядає його скомпільований код, аналізує як проходять і обробляються дані, застосовує оптимізації щодо машинного коду покладаючись на отримані дані. Сьогодні двигуни дуже досконалі і пропрацьовані і скрипти в результаті працюють дуже швидко.

Сучасний JavaScript є «безпечною» мовою програмування. Він не забезпечує низькорівневий доступ до пам’яті чи ЦП, оскільки спочатку був розроблений для браузерів, яким він не потрібен.

Можливості JavaScript сильно залежать від середовища, в якому він працює. Наприклад, Node.JS підтримує функції читання/запису довільних файлів, виконання мережевих запитів тощо.

JavaScript в браузері призначений для роботи з веб-сторінками, взаємодії з користувачем і веб-сервером.

Наприклад, у браузері JavaScript може:

— додавати нові HTML теги на сторінку сайта, змінювати вміст сторінки та модифікувати зовнішній вигляд;

— опрацьовувати дії користувача та виконувати відповідні до них дії такі як клік мишки, натискання клавіш на клавіатурі тощо;

— створювати запити на сервер, отримувати звідти дані, відкправляти дані і так далі;

— працювати з куками, інтерактивно працювати з користувачем, відправляти сповіщення, задавати питання і т. д.;

— записувати дані користувача в сховищі браузера.

JavaScript — єдина технологія для браузера, що підтримується всіма браузерами, код написаний на JavaScript, точно запуститься на кожному браузері, це дуже вагомий аргумент при виборі технології на якій буде створюватись програмне забезпечення[11]. Ось чому це найпоширеніше рішення для створення користувацьких інтерфейсів в браузері.

2.4 Вибір середовища розробки

Для виконання завдання дипломної роботи було обрано програмне середовище JetBrains WebStorm, це досить популярне інтегроване середовище розробки яке має всі необхідні інструменти для веб розробника які оформлені приємним та інтуїтивно зрозумілим інтерфейсом. Головною перевагою цього середовища є зручний редактор для JavaScript, HTML і CSS, який також

підтримує CoffeeScript, TypeScript, Dart, Sass, Less і Stylus і такі фреймворки, як наприклад Angular, React і Vue.js(рис.2.4).

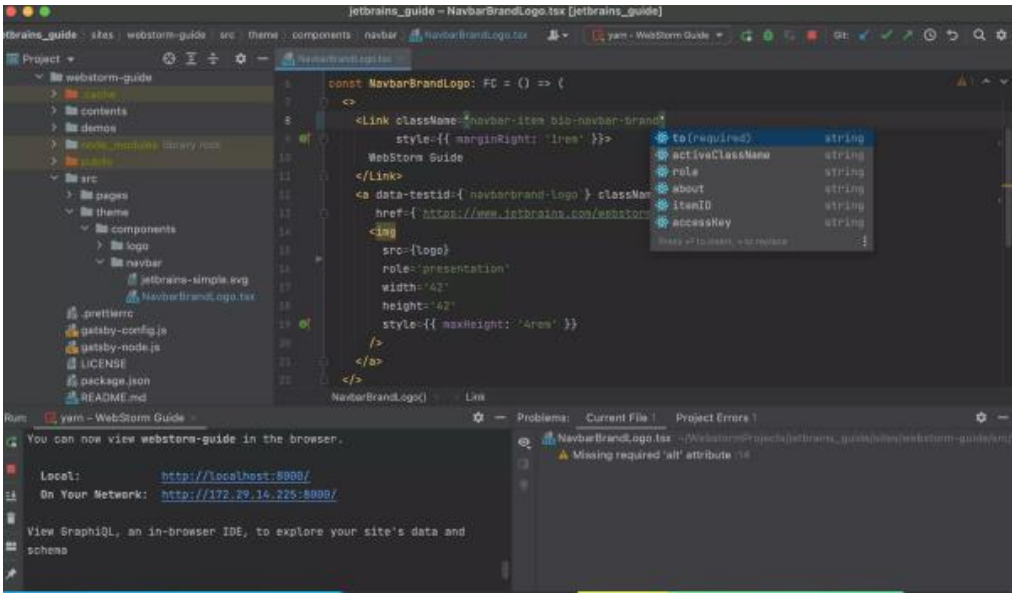

Рисунок 2.4 — Інтерфейс інтегрованого середовища розробки

WebStorm добре розуміє структуру проектів та допомагає з будь-якими аспектами написання коду. Автодоповнення коду, безпечний рефакторинг, постійний пошук потенційних проблем та підказки щодо їх виправлення завжди під рукою.

Ще один з головних плюсів IDE у тому, що тут поєднуються усі необхідні інструменти. WebStorm можна використовувати для налагодження та тестування клієнтського коду та програм на Node.js, а також для роботи з системою контролю версій. Також тут є інтегровані лінтери, інструменти збирання, термінал та HTTP-клієнт (рис. 2.5).

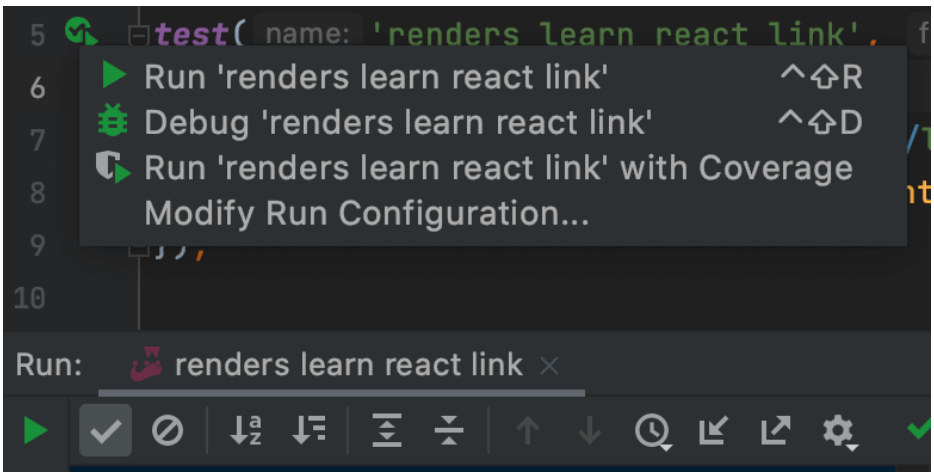

Рисунок 2.5 — Діалогове вікно інструментів

Тут можна швидко рухатись за кодом незалежно від розмірів проекту. Шукайти файли, класи або символи, щоб переглянути всі результати в одному місці. Якщо перейти до визначення функцій, методів, змінних, компонентів або класів, то знайти їх використання можна буквально в пару кліків.

Програма займає весь екран але має досить хороший і зрозумілий інтерфейс. Існує швидкий доступ до всіх інструментів, які найчастіше використовуються, і для зручності завжди є гарячі клавіші, які забезпечують швидкий доступ до корисних функцій, якими програміст може користуватися сотні разів на день. Крім того, гарячі клавіші також можна налаштувати індивідуально.

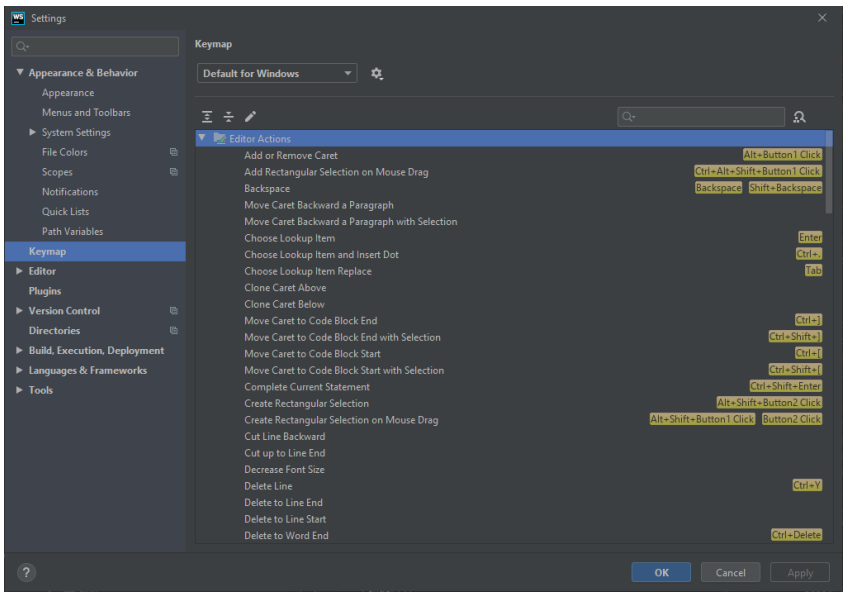

Рисунок 2.6 — Вікно налаштувань гарячих клавіш

А беручи до уваги що робота програміста це однотипна робота де людина постійно пише код, то і потреби зазвичй в цієї людини однотипні, тому без гарячих клавш не обійтись, вони будуть економити дуже багато часу.

Крім того, одним із найважливіших інструментів є GIT, і він глибоко інтегрований у це середовище розробки. GIT — одна з найпопулярніших систем керування версіями з відкритим кодом, на яку покладаються мільйони проектів (включно з комерційними та безкоштовними) по всьому світу. GIT — це абсолютно безкоштовне програмне забезпечення, яке підтримує найпопулярніші операційні системи в світі, таких як Mac, Linux, Windows і Solaris.

Для зручності роботи із системою контроля версій передбачена окрема панель із власним терміналом та історією змін, де можна подивитись кожну зміну версії, файли які в ній було змінено, порівняти вибрану версію із поточною і так далі. При роботі із великими проектами це просто необхідність. Також при роботі в команді це полегшує розуміння проекту та хто яку частину кода писав.

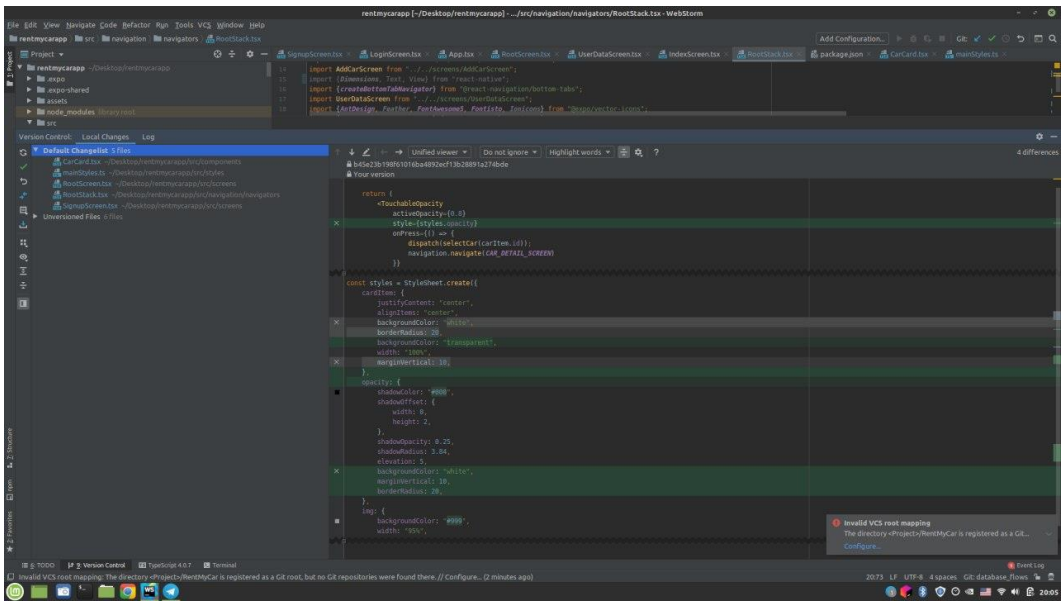

Рисунок 2.6 — Панель системи контроля версій

У цьому меню також можна побачити час і дату конкретного комміту або злиття гілок. Також ідентифікується користувач, який виконав цю дію, та гілка, в якій відбулася зміна. Самі гілки позначені різними кольорами та банерами з написами. Зелені прапорці — це гілки, створені локально та не завантажені на

віддалений сервер. Гілки, які вже були перенесені на віддалений сервер, пофарбовані у фіолетовий колір, а помаранчевий прапор вказує на поточне розташування розробника, це місце, звідки представлений код у середовищі розробки.

Також для зручності розробника папки і файли можна виділяти. Таким чином, програмісти можуть бачити, які файли були змінені, які файли не були включені в репозиторій проекту, також будуть підсвічуватись новостворені файли, через що програміст ніколи не забуде додати їх до коміту. Також завдяки підсвічуваню буде завжди видно що перенесено в хмарне сховище або навпаки, додано до списку ігнорування. Файли з помилками будуть виділені червоним, щоб знайти та виправити помилки.

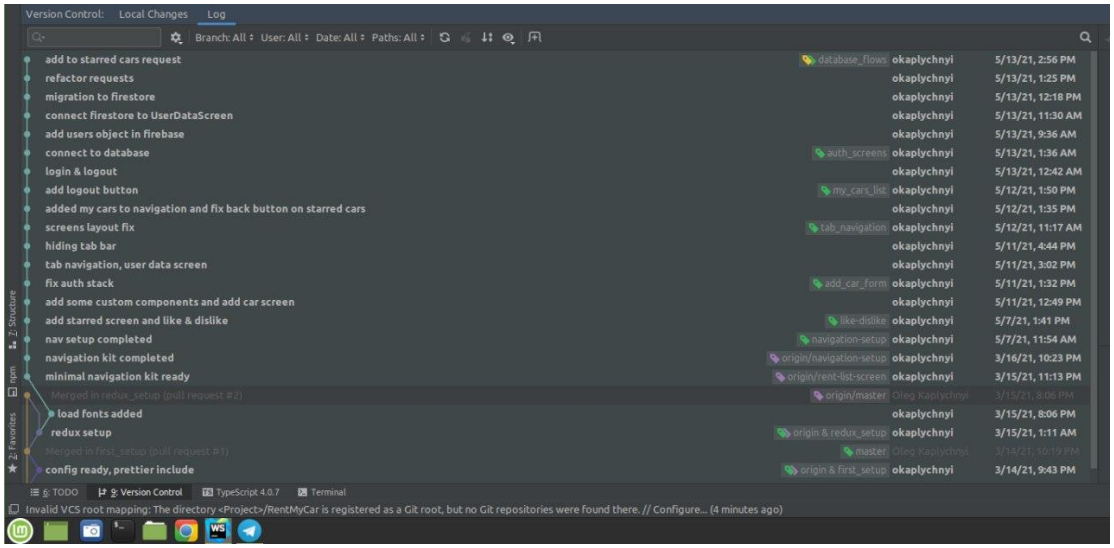

Рисунок 2.7 — Меню дерева версій

Також є меню, яке показує лише локальні та віддалені гілки. Це меню також доступне на головному екрані IDE. Це важливо для досягнення швидкого переходу між версіями та детального розуміння середовища проекту.

За допомогою цього меню можна створити нову гілку, для цього передбачено клавішу «new branch». Нижче наведено список гілок, які не було перенесено на віддалений сервер, і ви можете отримати доступ до кожної гілки, клікнувши по ній та вибравши відповідний пункт зі спадного списку, що з’явиться.

2.5 Засоби типізації кодової бази

TypeScript — це строго типізований JavaScript, тобто він додає деякі синтаксичні переваги мові, але все ще дозволяє писати звичайний JavaScript, якщо це необхідно. Він сприяє більш декларативному стилю програмування через такі речі, як інтерфейси та статичний тип, пропонує модулі та класи, і, що найважливіше, досить добре інтегрується з популярними бібліотеками та кодом JavaScript. Про нього також можна сказати що це стилістична статична надбудова над кодом JavaScript, який має функції які покращують роботу розробника.

TypeScript привернув особливу увагу кілька років тому, оскільки його було обрано для повної підтримки Angular 2 і пізніших його версій. Його також розробила Microsoft, що означає підтримку двох найбільших технологічних компаній. Відтоді TypeScript отримав статус основного над JavaScript.

TypeScript справді схожий на сучасний JavaScript. На базовому рівні він представляє парадигму статичної типізації JavaScript. Тому замість простих об'яв нетипізованих змінних при їх оголошенні розробник одразу прописує тип даних, який буде наданий змінною.

На перший погляд здається, що програмістам потрібно писати більше коду без необхідності. Але завдяки тому, що він дає системі більше інформації про програму, а це, у свою чергу, означає, що вона виправить помилки, які потенційно можуть бути в майбутньому.

Мутація змінної та зміна її типу, ймовірно, призведе до помилки та зупинки роботи програми, особливо у дійсно великому продукті, тому добре, якщо компілятор може показати цей момент до того, як програміст завантажить програму у браузер і витратить багато часу, щоб знайти та виправити це. Відповідно, це робить програму безпечнішою та надійнішою.

У TypeScript є інтерфейси, вони використовуються для визначення структури об'єктів (і лише об'єктів)[12]. Створюючи інтерфейс та описуючи через нього змінну, програміст вказує, що ця змінна має бути об'єктом, що

містить певні поля з певними типами даних. Фактично, програміст створює власний тип даних для об'єкта.

Це потрібно для того, щоб компілятор, а також розробник, який працюватиме над ним надалі, знав, який тип даних очікувати. Рекомендується писати інтерфейси у верхній частині файлів TypeScript, оскільки вони забезпечують представлення даних, з якими працює програма. Тоді якщо програміст напише щось таке, що не співпадає із інтерфейсом який в нього є, TypeScript видасть помилку з підказкою, де одразу буде зрозуміло де саме розробник допустив помилку (рис. 2.7).

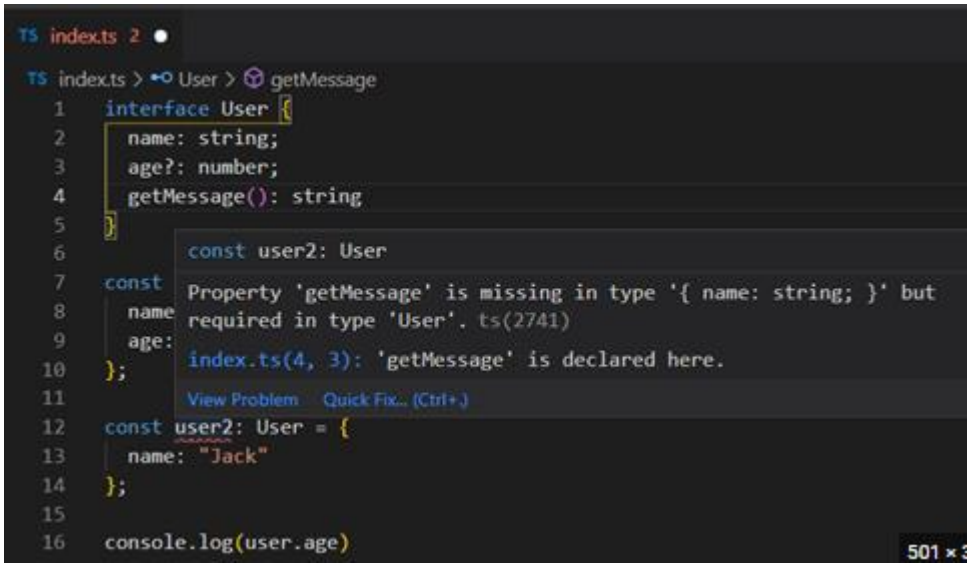

Рисунок 2.7 — Підказка TypeScript

Поговоримо про Generic Types. Візьмемо для прикладу масив. Масив - це тип-контейнер, який зберігає в собі значення будь-якого зазначеного типу. Логіка роботи масиву залежить від типу даних, що зберігаються всередині. Таке визначення автоматично свідчить, що маємо справу з узагальненим типом.

Щоб працювати з таким типом, потрібно конкретизувати внутрішній тип у той момент, коли ми хочемо розпочати роботу з даними цього типу:

```
const numbers: Array<number> = [];
numbers.push(1);
const strings: Array<string> = [];
```

numbers.push('hexlet');

Тип, який вказується усередині кутових дужок, називається параметром типу. Така назва вибрана невипадково вона вказує на параметр виклику функції.

Припустимо, що ми хочемо визначити свою власну колекцію, яка працює як масив, але з додатковими можливостями. Такі колекції часто роблять ORM для роботи з даними, завантаженими з бази. Опишемо спочатку конкретну версію цього типу, що працює тільки з числами та парою стандартних методів:

```
type MyColl = {
  data: Array<number>;
  forEach(callback: (value: number) => void, index: number, array: Array<number>): void;
  at(index: number): number | undefined;
}
```

Тут бачимо, що ці колекції зберігаються у числовому масиві. При цьому в типі визначено два методи, один з яких (forEach) передає елементи колекції колбек, а інший (at) повертає елементи колекції за вказаним індексом.

Для того щоб узагальнити цей тип, тобто зробити з нього дженерик, потрібно зробити одну просту річ: замість number скрізь написати T (або будь-яке інше ім'я, що починається з великої літери) і додати T як параметр типу до визначення.

На таке визначення типу можна дивитись як на своєрідне визначення функції. Коли вказується конкретний тип, наприклад так: MyColl<string>, то у цій ситуації замінюється на string всередині визначення типу. Причому якщо всередині типу використовуються інші дженерики, то вони викликають тип далі. Тобто все це працює як вкладені виклики функцій.

2.6 Керування пам’яттю

Управління пам'яттю JavaScript виконується автоматично і непомітно. Програміст створює примітиви, об'єкти, функції. Усе це пам'ять.

Але що відбувається, коли щось більше не потрібне? Як JavaScript дізнається, коли час очистити пам'ять?

Основною концепцією управління пам'яттю JavaScript є принцип досяжності[16]. Простіше кажучи, «досяжні» значення це ті, які доступні або використовуються. Вони гарантовано залишаться у пам'яті.

Є дані, значення які записані в коді, які не можуть бути видалені ні при яких обставинах, наприклад:

— локальні змінні і параметри функції що виконується;

— змінні і параметри функцій в ланцюжку вкладених викликів що виконується;

— глобальні змінні;

— деякі інші внутрішні значення.

Будь-яке інше значення вважається досяжним, якщо воно встановлено з кореня за посиланням або ланцюжком посилань.

Наприклад, якщо в локальній змінній є об'єкт, і він має властивість, що зберігає посилання на інший об'єкт, то цей об'єкт вважається досяжним. І ті, на які він зсилається, також можна досягти. Далі ви познайомитеся з докладними прикладами цієї теми.

Інтерпретатор JavaScript має фоновий процес, який називається збирачем сміття. Він відстежує всі об'єкти та видаляє ті, що стали недоступними. Сам збирач сміття інтегрований в двигун браузера, в нього є свої алгоритми і правила. Він працює самостійно і автоматично, але програміст повинен слідкувати за своїм кодом, адже якщо код буде поганий то збирач сміття не зможе бути максимально ефективним і програма буде використовувати більше пам’яті ніж потрібно і відповідно працювати не так добре як могла б працювати при правильно написаному коді.

Наприклад є посилання на об’єкт (рис. 2.8):

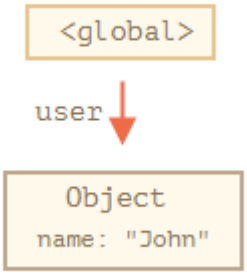

Рисунок 2.8 — Посилання на об’єкт

Тут стрілка вказує посилання на об'єкт. Глобальна змінна user посилається на об'єкт {ім'я: "Джон"} (далі просто "Джон"). Властивість name об'єкта John зберігає примітив. Якщо ви перезапишете значення користувача, посилання перестане працювати(рис. 2.9).

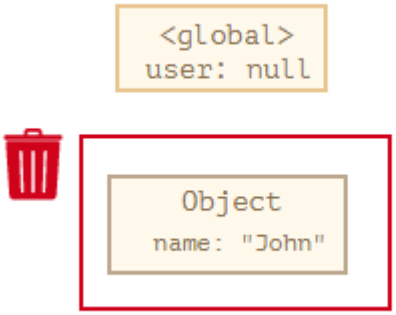

Рисунок 2.9 — Об’єкт без посилання який потрібно видалити

Тепер об'єкт John стає недосяжним. До нього немає доступу, на нього немає посилань. Складальник сміття видалить ці дані і звільнить пам'ять.

Якщо скопіювати посилання з user в admin то ситуація кардинально змінюється.

Тепер, якщо ми видалимо посилання на користувача, об'єкт John, як і раніше, буде доступний через глобальну змінну admin, тому він знаходиться в пам'яті. Якщо ви також перепишете змінну адміністратора, Джон буде видалено.

Основним алгоритмом складання сміття є алгоритм «помітки та очищення». За цим алгоритмом збирач сміття регулярно виконує такі кроки:

— збирач сміття "помічає" (запам'ятовує) усі кореневі об'єкти;

— далі переходить за їхніми посиланнями та зазначає всі знайдені об'єкти;

— далі переходить за посиланнями зазначених об'єктів і зазначає об'єкти, на які від них є посилання, всі об'єкти запам'ятовуються, щоб надалі ви не відвідували один і той самий об'єкт двічі.

І так далі, поки не будуть відвідані всі посилання (досяжні від коренів). Всі непомічені об'єкти видаляються.

Наприклад маємо наступну структуру даних:

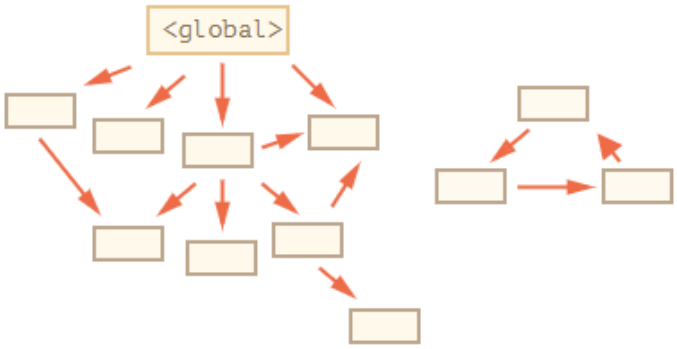

Рисунок 2.10 — Структура даних

Справа виразно видно «недосяжний острів». "Алгоритм маркування" збирача сміття буде проходити за всіма посиланнями, доступними з глобальної області видимості, позначаючи їх. Це буде відбуватися доти, доки це можливо, наприкінці проходження всі недоступні об'єкти будуть видалені збирачем сміття.

Інтерпретатори JavaScript застосовують безліч оптимізації, щоб складання сміття працювало швидше і не впливало на продуктивність. Ось деякі з оптимізації:

— поколінська колекція, предмети діляться на "нові" і "старі", багато предметів з'являються, виконують своє завдання і швидко вмирають, їх можна видаляти агресивніше, а ті, що живуть довго, стають "старими" і перевіряються менш часто;

— інкрементальна колекція (Incremental collection), якщо об'єктів багато, то обхід всіх посилань і позначка досяжних об'єктів може зайняти багато часу і призвести до видимих затримок виконання скрипту, тому інтерпретатор намагається організувати збір поетапно, етапи відбуваються окремо один за одним, що у свою чергу вимагає додаткового обліку для відстеження змін між етапами, але тепер у нас замість однієї затримки багато крихітних;

— збирання у вільний час (Idle-time collection) для зменшення можливого впливу на продуктивність збирач сміття намагається працювати тільки під час простою процесора.

Є й інші способи оптимізації та різні типи алгоритмів складання сміття. Інтерпретатори JavaScript використовують різні прийоми та прийоми, тому існує безліч типів алгоритмів. І що важливіше, все змінюється у міру розвитку інтерпретаторів, тож заглиблюватися в цю тему заздалегідь, без реальної необхідності, не варто.

2.8 Розробка структури програми

Основним із важливих етапів розробки будь якого програмного забезпечення є його проектування. На цьому етапі потрібно визначитись із кількістю сторінок, розміщенням елементів на них, на цьому етапі вирішується майбутній користувацький досвід від готового в майбутньому продукту.

Якщо створювати загальну схематичну структуру будь якого веб проекту то для якого б проекта вона не створювалась, схема буде зазвичай майже однакова, кожен із таких проектів має головну сторінку, на якій є хедер, якийсь контент і футер. Також окрім головної сторінки завджи є іншу сторінки в яких також міститься певний контент, окремою частиною додатка є сховище та сторінка помилки щоб показувати її у випадку якогось непередбачуваного моменту або коли не підвантажились якісь дані. Із загальною схематичною структурою проекта можна ознайомитись на рисунку 2.11.

Рисунок 2.11 — Схематична структура проекту

Фактично кожен із елементів схеми несе в собі окремий функціонал який може бути як зв’язаний із іншими сторінками так і не зв’язаний з ними.

Сама сторінка помилки несе окреме смислове навантаження, вона лежить окремо і на неї можна потрапити з будь якої сторінки тільки при виникненні помилки. Власноруч прямо перейти на цю сторінку не можна вона не повинна бути прив’язаною до навігаціїї.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ

В даному розділі описана розробка та реалізація функціоналу розроблюваного програмного забезпечення. В цьому розділі було враховано всі висновки які були зроблені в попередніх розділах та вся інформація яку було здобуто. Створений прототип продукту можна буде випускати в загальне користування.

3.1 Створення проекту програмного засобу

При дослідженні програмних рішень у галузі транспортних засобів був отриманий висновок, що аналогічні рішення сумарно не мають такого функціоналу який закладається в дану розробку. Фактично є рішення які отримують дані через спеціальний роз'єм автомобіля і отримують звідти дані, є рішення які ведуть облік обслуговування авто, але додатків які поєднують ці дві речі знайдено не було.

Також в додатках де є облік обслуговування немає зв'язки їх із СТО на які можна записатись при потребі заміни якогось розхідника. Це досить зручно зі сторони користувача, коли в одному місці можна переглянути всю актуальну інформацію та отримати карту з місцями рішення питань які можуть виникнути.

Наявність карти з СТО досить перспективне рішення для подальшого розширення даного програмного продукту. Якщо такий продукт набере достатню кількість користувачів то можна буде зробити інтергацію сервісів по всьому місту і тоді кожна проблема яка виникне у власника авто може бути вирішена автоматично, тобто модемо уявити ситуацію, коли в машини в системі виникає помилка, її зчитує датчик який стоїть в роз'ємі, передає інформацію на телефон власника, той в свою чергу аналізує інформацію і створює запис на визначені користувачем СТО, призначає час на коли має бути ремонт, і видає сповіщення користувачу. Той в свою чергу або підтверджує запис, або відхиляє в разі не співпадіння з графіком. Але сама ідея автоматизованого вирішення питань по авто на думку автора є дуже зручною та в перспективі може перерости у велику мережу яка поєднує в собі всі СТО в місті.

Отже розроблюване програмне рішшення повністю актуальне і має великий потенціал в подальшій розробці та підтримці.

Перед початком створення проекта потрібно встановити певні пакети, без них робота над додатком буде неможливою. Це такі пакети як Node.js, expo-cli та yarn.

Yarn це новий менеджер пакетів, спільно створений Facebook, Google, Exponent та Tilde. Як можна прочитати в офіційній документації, його метою є вирішення низки проблем, з якими зіткнулися розробники при використанні npm, а саме:

— установка пакетів була досить швидкої і послідовної;

— існували проблеми з безпекою, тому що npm дозволяє пакетам запускати код під час встановлення.

Це не є спробою повністю замінити npm. Yarn це новий клієнт командного рядка, що завантажує модулі з реєстру npm. У самому реєстрі нічого не змінюється — ви можете завантажувати та публікувати модулі так само, як і раніше.

Після встановлення yarn необхідно встановити expo-cli. Для цього в терміналі виконується наступна команда: yarn install expo-cli-global.

Також потрібно скачати та встановити Node.js, адже без нього збірка та запуск проекта буде неможливою. Node.js – це середовище виконання JavaScript, побудоване на механізмі JavaScript V8 Chrome.

Такі пакети краще встановлювати глобально, адже необхідність в них ж в усіх проектах такого типу і при розробці наступного потрібно буде двічі робити одну і ту ж саму роботу.

Далі необхідно створити порожній шаблон проекту, що робиться виконанням команди ерхо init у терміналі. Після встановлення Expo відкривається інтерфейс командного рядка Expo, у якому можна вибрати ім'я проекту та шаблон. Для початку розробки потрібно вибрати пусту сторінку на TypeScript. Після створення перейдіть до папки створеного проекту cd InfoCar. Після цього запускаємо щойно створений проект командою expo start. У браузері

відкриється вікно з інструментами розробника, за допомогою яких ви зможете запускати цю програму на емуляторі або симуляторі, публікувати оновлення, перемикатися в робочий режим та переглядати логи в консолі. Також важливим елементом є QR-код, відсканувавши його за допомогою якогось пристрою Android або IOS з встановленим додатком Expo, ви зможете відкрити його на цьому пристрої та протестувати без складання та встановлення програми. Всі ці режими підтримують гаряче перезавантаження, тому технологія буде комфортною у будь-якому випадку.

Цей проект використовуватиме TypeScript. TypeScript - це мова з відкритим вихідним кодом, заснований на JavaScript, одному з найбільш широко використовуваних інструментів у світі, з додаванням статичних типів визначень.

Типи надають спосіб опису форми об'єкта, дозволяючи TypeScript перевіряти, чи правильно працює код, який він пише. Типи можуть бути необов'язковими у TypeScript.

Відразу потрібно встановити Redux, це менеджер станів програмного продукту, що розробляється, встановлюється за допомогою команди npm install --save react-redux. Він буде налаштований пізніше.

Одним із перших кроків є встановлення та налаштування навігації. Це робиться за допомогою команди yarn add react-nativation. Оскільки цей додаток буде досить об’ємним, навігація та екрани, які будуть створені, будуть розміщені в окремих папках проекту. Таким чином, потрібно створити папку навігації, у яку можна винести всю конфігурацію навігації, а потім імпортувати все це в App.tsx (місце, де програма починає працювати).

Пакети встановлюються за допомогою команди yarn add package_name.

Основою навігації будуть два стеки навігації, перемикання між якими відбуватиметься після того, як тернарний оператор перевірить змінну, яка відповідатиме за авторизацію користувача.

Конфігурація навігації переміщується в окрему папку, а сама навігація буде перенесена в окремий файл - IndexScreen. Це робиться для того, щоб після завершення всіх налаштувань файл App.tsx виглядав так, як показано на рис. 3.1.

Рисунок 3.1 — Кореневий файл проекту

Шрифти можна завантажити за допомогою спеціального пакета expo-font, це потрібно робити в хуку useEffect за допомогою спеціального синтаксису асинхронного запиту. У список залежностей цього хука потрібно вписати порожній масив, щоб функція, яка буде передана всередині хука, викликалася лише один раз.

Після того, як додали завантаження шрифту, виникає проблема, що при завантаженні програми смикається інтерфейс. Це пов'язано з тим, що вміст сторінки вже був змальований, потім були завантажені шрифти і весь контент відрендерен знову, що негативно впливає на UX. Цю проблему можна вирішити за допомогою додаткової перевірки та використання спеціального пакету expo-app-loading.

Цей пакет дозволяє відображати екран, з якого завантажується програма. Тобто, створивши змінну, яка буде відповідати за завершення завантаження шрифту і зробити перевірку, щоб при завершенні всього відображався вміст, але ще не відображався екран завантаження, можна позбутися цього неприємного моменту відкриття програми. Реалізація цього процесу представлена на рис. 3.2.

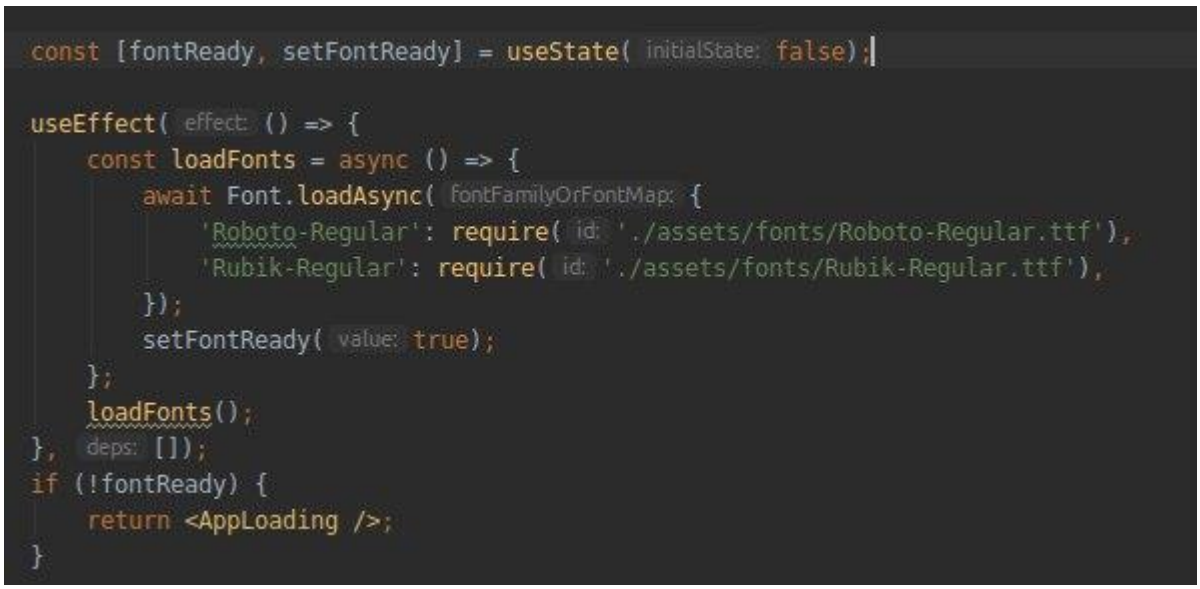

Рисунок 3.2 — Завантаження шрифтів

Для продовження роботи над програмою необхідно налаштувати так зване «сховище». Для початку потрібно в кореневій папці створити папку store, в ній створити дві папки — actions і reducers і два файли типу .tsx і route.tsx. при подальшому збільшенні програмного коду можна додатково створити папку з файлами, в яких будуть описані типи та структура даних, які будуть перебувати в «сховищі» програми, що надсилаються до бази даних та з неї. Файли з діями будуть створюватися в папці дій, тут будуть виконуватися всі запити до сервера та обробка даних, що надходять. У папці редукторів будуть створюватися редуктори, це деякі "гілки" стану програми. Оскільки використовується TypeScript, є файл types.tsx, в якому будуть описані типи даних та типи полів об'єкта, з якими вестиметься робота. У файлі route.tsx будуть постійні терміни визначення типу екшенів.

Початкову структуру конфігурації сховища можна побачити на рисунку 3.3. Де INITIAL_STATE — початковий стан програми, rootReducer — основний редьюсер, який створюється з усіх доступних редюсерів, store — власне сховище, яке буде передано як параметр обгортці Provider (рис. 3.3) для подальшого використання в додатку, а проміжне ПЗ буде використовуватися для обробки проміжних станів виконання будь-якої дії в коді Якщо вам потрібно, щоб програма запускалася з певними вихідними даними, їх слід занести в об'єкт

INITIAL_STATE відповідно до розробленої структури зберігання. Оскільки у цьому проекті використовується TypeScript, необхідно описати типи вхідних даних, якщо об'єкт перетворюється на вихідний стан, необхідно описати як типи даних, а й його структуру.

```
import {createStore, combineReducers, applyMiddleware} from "redux";
import thunk from "redux-thunk";

//reducers
import auth from "./reducers/auth";

const INITIAL_STATE = {};

const middleware = [thunk];

const rootReducer = combineReducers( reducers: {
    auth
});
export type RootState = ReturnType<typeof rootReducer>;

const store = createStore(
    rootReducer,
    INITIAL_STATE,
    applyMiddleware(...middleware),
);
export default store;
```

Рисунок 3.3 — Початкова конфігурація сховища

Всі дії в сховищі потрібно описувати в додатковому файлі з типами даних щоб відділити логіку та сам опис типів даних. Приклад можна перег-лянути на рисунку 3.4.

Вся навігація в програмі повинна бути обгорнута тегом NavigationContainer, щоб програма розуміла, яка частина відповідає за навігацію[13]. У даному випадку цей контейнер містить тернарний оператор, який повертає або стек екранів аутентифікації, або основний стек навігації. Змінна, над якою буде працювати цей тернарний оператор, зберігається в редюсері аутентифікації, по суті це поле об'єкта, яке можна отримати в будь-якій частині програми за допомогою хука useSelector.

Рисунок 3.4 — Опис типів для модуля аутентифікації

Хорошою практикою також створити хук useAppSelector, для того щоб отримувати типізований об’єкт сховища, таким чином в процесі звернення до стора середовище розробки буде висвічувати підказки про поля які воно мітить що значно спростить роботу із даними які там лежать.

3.2 Створення інтерфейсу користувача

При вході і успішній авторизації користувачу буде показуватись екран додавання його автомобіля. На екрані будуть поля для введення основної інформації про його авто. Під основною мається на увазі марка, модель, рік виробництва, та тип палива. Цей екран можна спостерігати нижче на рис. 3.5.

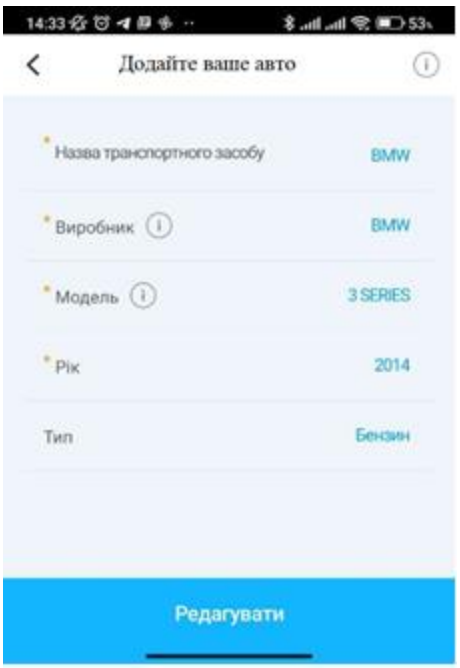

Рисунок 3.5 — Екран додавання автомобіля

Для введення типу палива та року випуску моделі авто було створено спеціальне модальне вікно, в яке просто прокидається назва параметру та власне самі параметри які можна обрати. Цей компонент було винесено в окрему папку та перевикористано у всіх випадках де користувачу потрібно обирати один параметр зі списку який йому пропонують. Із зовнішнім виглядом даного модального вікна можна ознайомитись на рисунку 3.6.

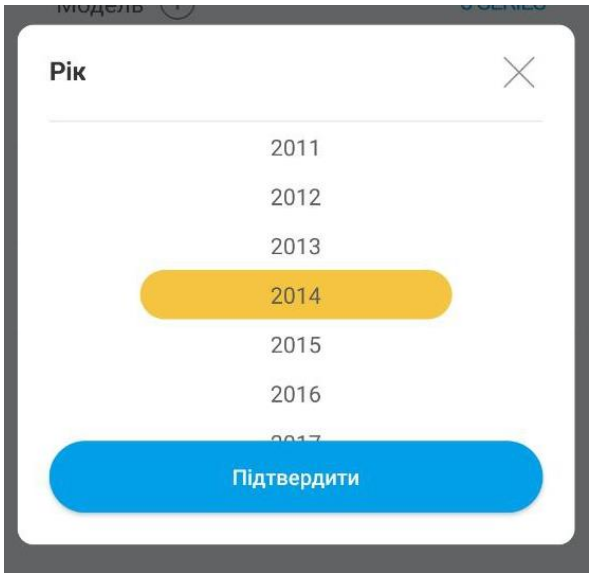

Рисунок 3.6 — Модальне меню з вибором опції

Після додавання автомобіля користувач буде потрапляти на головний екран. Тут буде розташовуватись мінімальна інформація про авто, щоб не навантажувати користувача зайвою інформацією як це було зроблено в аналогах які були досліджені в розділі вище. Із інформації яку користувач буде бачити на головному екрані:

— марка та модель авто;

— середній пробіг за поїздку;

— середній час поїздки;

— останній день діагностування;

— плитки із переходом на подробиці по певних параметрах, таких як журнал обслуговування, паливо, діагностика автомобіля, управління витратами.

Знизу головного екрану буде панель із індикатором підключення плати яка отримує інформацію від машини. В індикаторі є три поділки які загораються при:

— додаванні автомобіля в додаток;

— підключення додатка до пристрою для зчитування через порт OBD2;

— отримання актуальних даних з машини.

Для реалізації функціоналу під’єднання машини телефону до пристрою зчитування інформації з машини було використано сторонню бібліотеку, оскільки це доволі складний модуль який крім того що реалізувати потрібно ще відтестувати, тому такі речі найкраще брати перевіреними і готовими до інтеграції що власне і відбулось при реалізації даного програмного забезпечення.

При натисканні на кнопку під назвою «Діагностика автомобіля» відкривається сторінка із переходом на підключення до автомобіля. На цій сторінці прописана загальна інформація по кодах помилок і кнопка переходу до карти СТО. Внизу сторінки знаходиться велика кнопка «Діагноз» по якій відбувається власне саме підключення мобільного додатку до автомобіля.

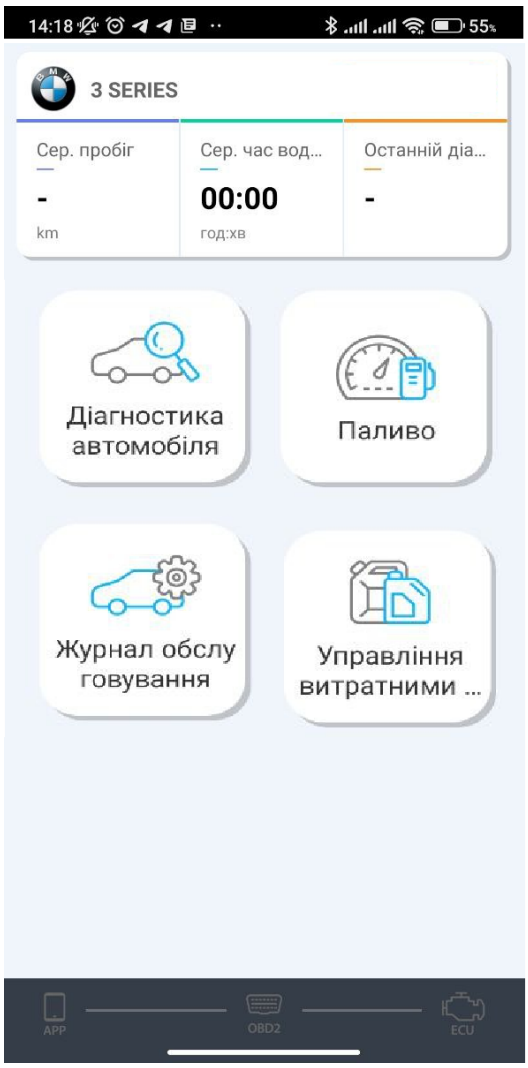

Рисунок 3.7 — Головний екран додатка

При натисканні на кнопку «Ремонтна майстерня» відкривається Google карти із введеним в них словом «сервіс» для миттєвого знаходження сервісних центрів, Google надає рекомендації опираючись на попередні дії користувача телефону, тобто якщо користувач раніше теллефонував в якийсь сервіс, або його шукав, Google це запам’ятає і подасть списком сервісів в які можна звернутись. Далі користувач натискає на сервіс і відкривається телефон для того щоб можна було подзвонити на обрану СТО як показано на рисунку нижче.

Якщо перейти на сторінку управління витратами то відкриється список розхідників та пробіг який накопичився після останньої заміни. В кожного розхідника є свій рекомендований пробіг на якому його варто замінити. На цій сторінці можна переглянути наскільки давно було замінено мастило, тормозні

колодки, охолоджуюча рідина, фільтри і так далі. Окремою шкалою на кожен розхідник виведено пробіг який залишився до заміни розхідника.

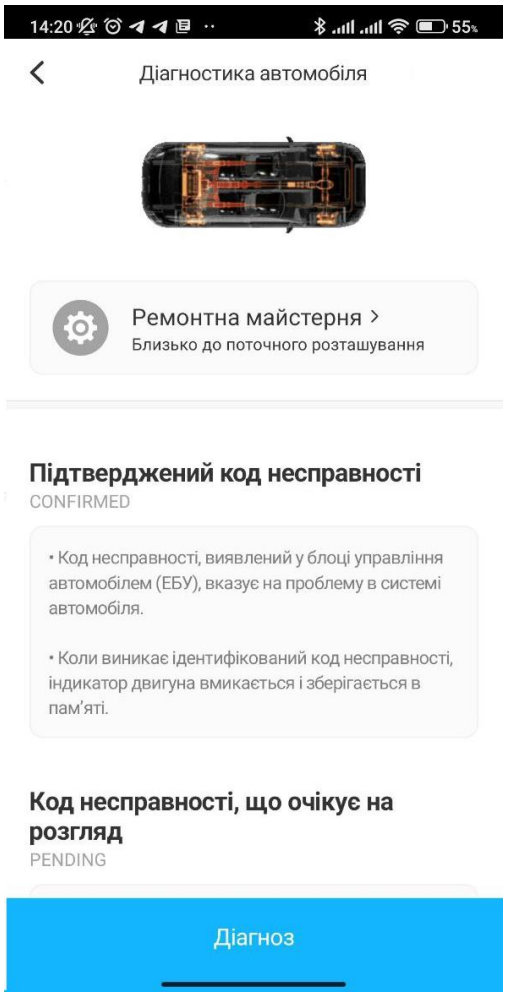

Рисунок 3.8 — Сторінка діагностики автомобіля

Для кожного розхідника є підказка із рекомендованими пробігами заміни, тобто рекомендації які дає виробник при продажі авто. Кожен із цих пробігів збивається при введенні користувачем в програму інформацію про його заміну. Нажаль цей список поки що не можна редагувати, але в подальшому розвитку цього проекту це можна допрацювати і дати користувачу вибір самому створювати список деталей та розхідників які йому потрібно замінити.

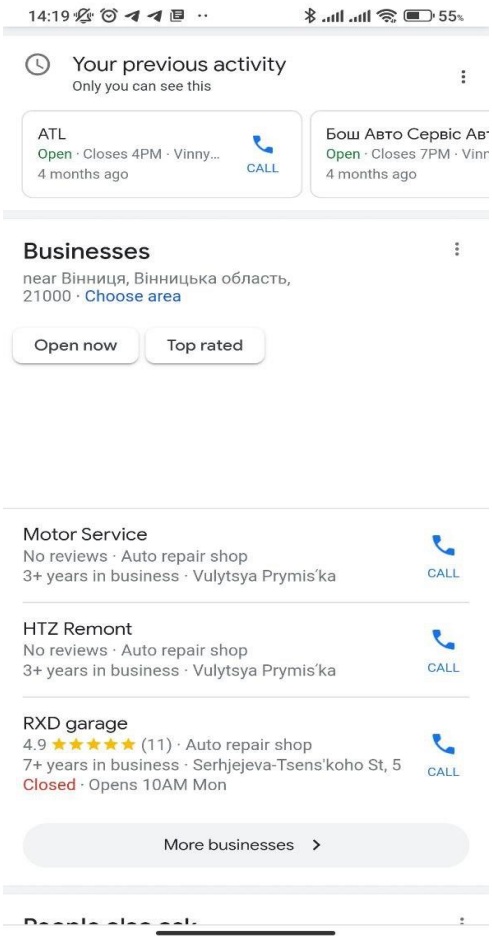

Рисунок 3.9 — Рекомендації користувачу від сервісу Google

Також було б зручно мати можливість ставити свої діапазони заміни розхідників, оскільки кожен власник авто проводить заміни тоді, коли вважає за потрібним і дуже часто це не ті пробіги які вказує виробник.

У цьому дипломному проекті стилі, які використовуються багато разів, були винесені в окремий файл, щоб не писати їх повторно, тому частина стилів підключена ззовні, а частина знаходиться у файлі програми. В інтерфейсі використовуються компоненти, що настроюються, які використовуються в різних частинах програми. Кольори, що використовуються, також є постійними значеннями, імпортованими з файлу стилю. Це зроблено для того, щоб при необхідності змінити колір певної частини програми, це можна було б зробити, змінивши лише одну константу.

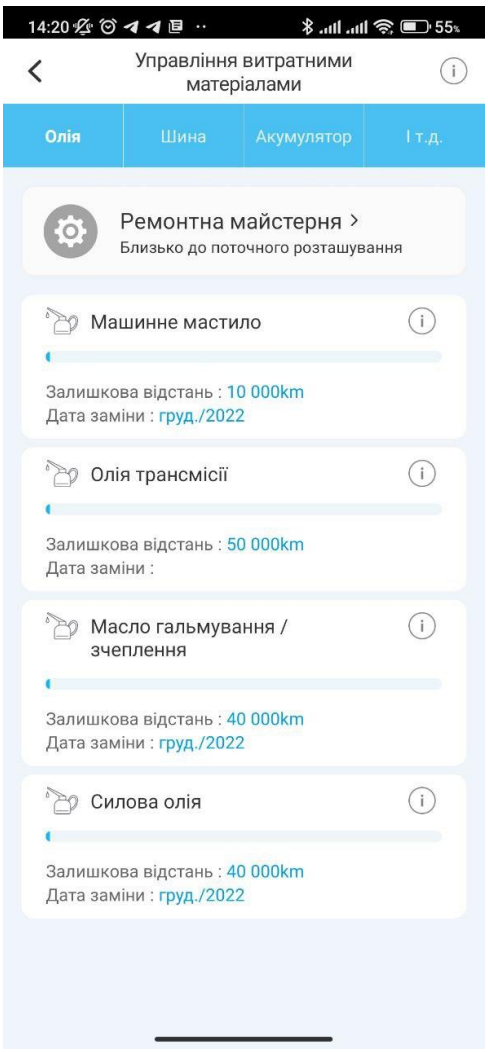

Рисунок 3.10 — Сторінка управління витратними матеріалам

3.3 Інтеграція та оптимізація роботи Bluetooth

Bluetooth — технологія бездротової передачі. Вона забезпечує обмін інформацією між такими пристроями як персональні комп'ютери (настільні, кишенькові, ноутбуки), мобільні телефони, принтери, цифрові фотоапарати, мишки, клавіатури, джойстики, навушники, гарнітури на надійній, недорогій, доступній радіочастоті для ближнього зв'язку[12].

Bluetooth дозволяє цим пристроям спілкуватися, на відстані від 1 до 100 метрів один від одного (дальність залежить від перешкод), навіть у різних приміщеннях.

Bluetooth, як уже було сказано вище, сучасна технологія бездротової передачі даних, що дозволяє з'єднувати один з одним практично будь-які

пристрої. Можна з'єднати все, що має вбудований мікрочіп Bluetooth. Технологія стандартизована, отже, проблеми несумісності пристроїв від конкуруючих фірм не повинно бути.

Bluetooth — це маленький чіп, що є високочастотним (2.4 - 2.48 ГГц) приймачем[14].

Оскільки розроблюване програмне забезпечення напряму працює з залізом по технології Bluetooth, то потрібно багато моментів передбачити і опрацювати зі сторони front-end. Це такі речі як перевірка на увімкнений Bluetooth, перевірка на зникаючий сигнал з датчика, перевірка роботи технології у фоні, перевірка енергоефективності роботи із датчиком і так далі. Сама робота Bluetooth із датчиком та отримання з нього даних відбувається через сторонню бібліотеку, але вона передбачена тільки для отримання форматованих даних із датчика, тому деякі речі потрібно дописувати власноруч.

Перевірка на те чи увімкнений Bluetooth в коді має вигляд як показано на рисунку 3.11. На рисунку можемо бачити код певного нативного модулю який буде піідключатис до проекту окремо і фактично являється Java кодом, але так як підтримує вставки на цій мові, а в мережі інтернет є перевірені приклади, то використовуватись будуть саме вони.

```
@ReactMethod
public void checkEnabledBluetooth(Promise promise) {
    BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    try {
        promise.resolve(mBluetoothAdapter.isEnabled())
    } catch (Exception e) {
        promise.reject(null, "Error checking the enabled bluetooth");
    }
}
```

Рисунок 3.11 — Перевірка на увімкнений Bluetooth

Фактично отримуємо картину що є певний функціонал на Java, який виглядає як окремий клас та обернений в окремий модуль, далі йому надається доступ до JavaScript де його вже можна використовувати.

Такі модулі зазвичай додаються за шляхом android/src/main/java/com/app-name/. Далі в MainApplication.java потрібно знайти виклик функції getPackages() і потім додати власний написаний пакет packages.add(new ReactOnePackage()). І далі цей модуль можна імпотрувати за допомогою класу нативних модулів який буде імпортуватись в свою чергу з пакету "react-native". Виглядатиме це наступним чином:

```
import { NativeModules } from 'react-native';
const {ReactOneCustomMethod} = NativeModules;
```

де ReactOneCustomMethod це назва кастомного модуля.

Таким чином можно створити безліч модулів і використовувати їх у власному проекті при потребі. В даній магістерській роботі це виконано для створення власного модуля для роботи з Bluetooth. В кастому модулі додано функції для включення Bluetooth, отримання списку доступних пристроїв, вибору пристрою, підключення до пристрою і так далі.

Оскільки робота з Bluetooth в даному додатку є ключовим функціоналом, то її потрібно приділити досить багато часу і уваги. Так як дана технологія постійно працює у фоні, то потрібно звернути увагу на енергоефективність її використання, тому що при неправильному використанні вона може використовувати зайву енергію і надмірно розряджати телефон. Для цього буде використана бібліотека react-native-ble-plx. Після налаштування даного пакету на обох платформах потрібно прописати запити на дозволи користуванням Bluetooth на прострої на якому запускатиметься розроблюване програмне забезпечення.

Очікуваними результатами по проведених роботах із налаштуванням роботи Bluetooth є стабільна і передбачувана робота додатка із даною технологією та уникнення помилок при непередбачуваних раніше обставинах. Встановленням та налаштуванням останнього пакету було оптимізовано енергоспоживання самого процесу роботи із безпровідною технологією в результаті чого смартфон користувача буде набагато менше енергії витрачати у фоновій роботі.

Всі написані модулі та робота з ними проходить в основному на екрані підключення смартфона до автомобіля (рис. 3.12).

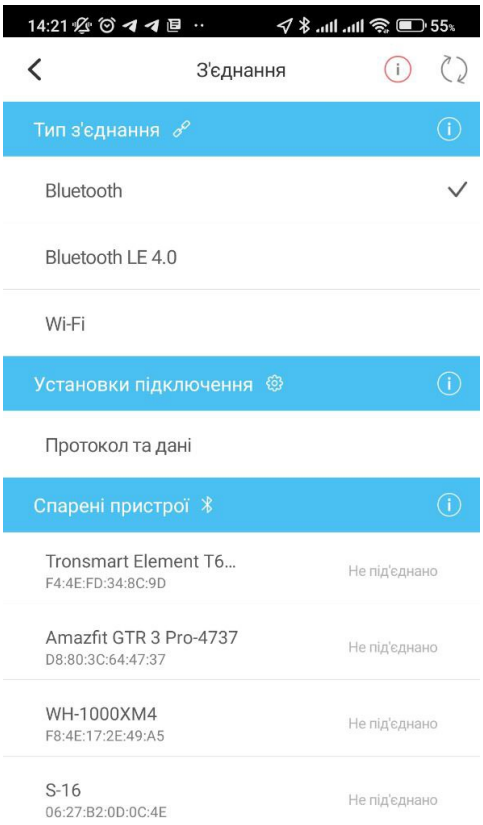

Рисунок 3.12 — Екран роботи з Bluetooth

4 ТЕСТУВАННЯ СТВОРЕНОГО ПРОГРАМНОГО ЗАСОБУ

4.1 Тестування працездатності Bluetooth

Тестування важливий заключний етап розробки програмного забезпечення, від цього етапу залежить наскільки якісний продукт вийде в ринок та наскільки першим користувачам сподобається користувацький досвід. Від цього залежать результуюче враження від продукту та компанії яка його випускає вцілому.

Щоб перевірити налаштування роботи Bluetooth, потрібно зробити найелементарніші можливі речі та просто сканувати пристрої Bluetooth. Для цього потрібно додати швидкий одноразовий код до компонента головної сторінки, адже вона відкривається одразу при запуску програми, даний код потрібен тільки для тесту та повинен бути видалений перед кінцевою збіркою прототипу додатка. Для отримання класу менеджера Bluetooth потрібно під імпортами написати const manager = new BleManager(), таком чином створиться змінна в яку за допомогою понструктора класу BleManager буде присвоєний клас для роботи із налаштованим вище функціоналом.

Після цього на сторінці головного екрану потрібно додати наступний код:

```
const scanForPeripherals = () => {
manager.startDeviceScan(null, null, (error, scannedDevice) => {
console.log(scannedDevice)
})
}
```

Цей кож потрібен для пошуку активних пристроїв до яких є змога підключитись, в даному випадку вони просто виводяться у вигляді списку до терміналу розробника. Цей функціонал додається тільки для тесту та повинен бути видаленим на стадії релізу.

Після чого потрібно створити кнопку та у її властивість onPress прокинути наступний код:

```
onPress={() => {
  dispatch(bluetoothPeripheralsFound(['AA:DD:CC:DD']));
```

```
  scanForPeripherals()
}}
```

Це дозволить прив’язати створену вище функцію до функції яка буде виконуватись при натисканні на дану тестову кнопку.

Після натискання на створену кнопку в терміналі розробника можна бачити результат роботи (рис. 3.13). Можемо чітко бачити що сканування пристроїв відбулось і в термінал розробника вивівся результат сканування, а саме список пристроїв із їхніми даними які ми можемо використовувати в своїх власних цілях.

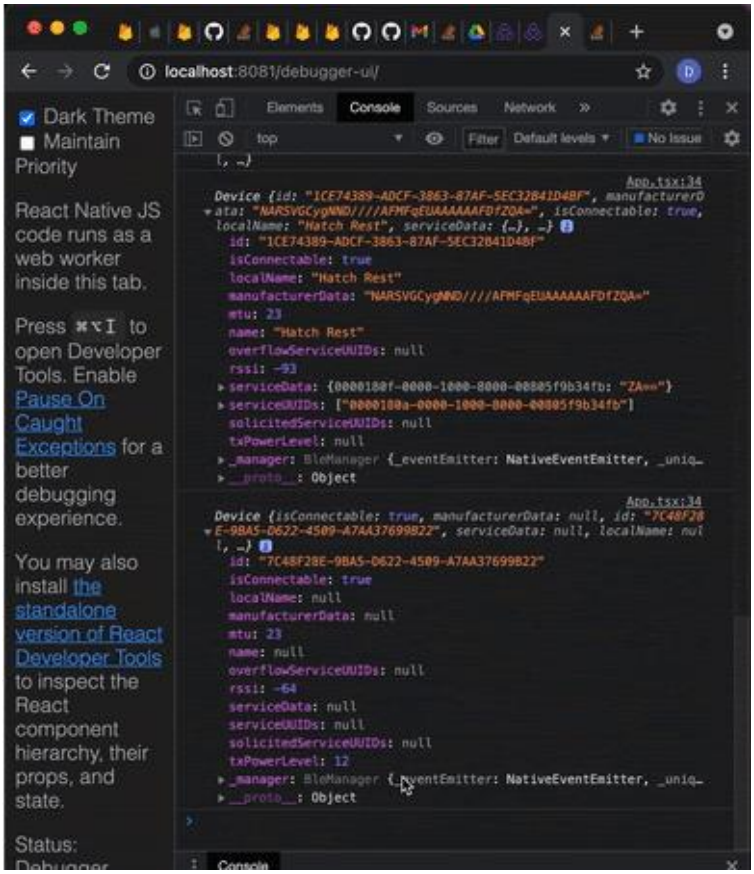

Рисунок 4.1 — Інформація про пристрої

4.2 Тестування енергоефективності

При розробці даного програмного засобу були використані певні міри оптимізації які описані вище. Попередньо дана оптимізація повинна мати прямий вплив на використання енергії смартфоном. Для перевірки ефективності даної доробки потрібно провести ряд тестів. Потрібно виміряти на скільки відсотків смартфон розряджається за ніч без запущеного додатка який працює у фоні, із

запущеною старою версією додатка та із запущеною оптимізованою версією додатка. Також для оцінки конкурентноспроможності даної розробки проти аналогічних рішень, які є у вільному доступі потрібно провести ще один дослід зі встановленим аналогічним додатком.

Для чистоти експерименту, телефон потрібно зарядити до ста відсотків і починати відлік від конкретної години та закінчувати конкретною годиною. Під час проведення дослідження встановлені програми та їх кількість не повинні змінюватись.

Сам смартфон протягом всього часу перебування в стані спокою не має вмикатись та повинен перебувати в режимі польоту, оскільки неоднорідність сповіщень та кількість даних які можуть знадобитись телефону за цей період може бути не однакова. Також перед проведенням кожного із дослідів, потрібно увімкнути режим без обмежень системи, оскільки операційна система при використанні якоюсь програмою надмірної на її думку енергії, може накладати певні обмеження на її роботу, або й навіть вимкнути її взагалі. Для цього потрібно перейти в налаштування, обрати програму з якої потрібно знати обмеження, зайти в налаштування «Зберігача батереї», та обрати пункт «без обмежень» (рис. 4.2).

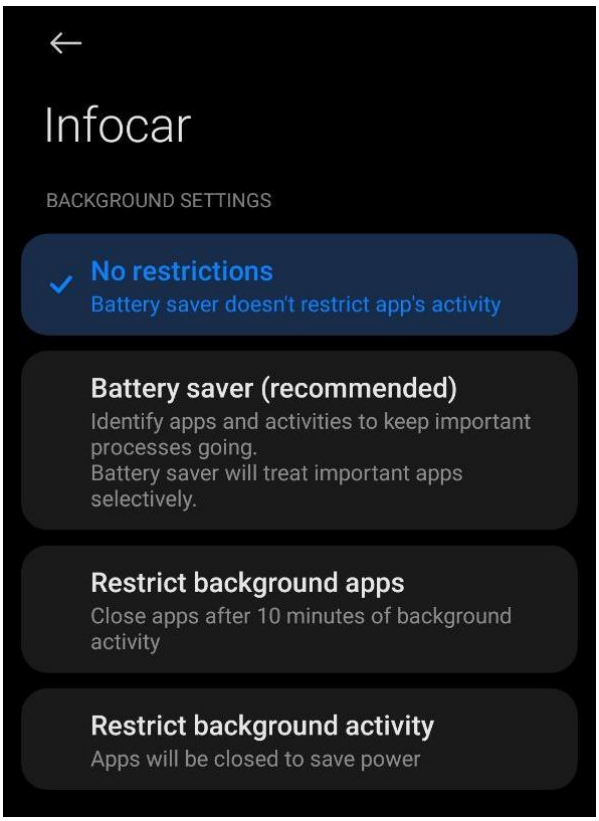

Рисунок 4.2 — Налаштування режиму «зберігача батареї»

Дане налаштування потрібне для максимальної свободи використання ресурсів смартфону встановленим додатком. Таким чином, система не буде оптимізувати його роботу та економити на цьому заряд батареї.

Після трьох днів проведення досліду маємо наступні результати, які можна спостерігати в таблиці 4.1.

Таблиця 4.1 – Результати досліду

Умови досліду	Використаний процент батареї, %
Без встановленого додатку	3%
Із старою версією додатку	8%
Із оптимізованою версією додатку	6%
Із встановленим аналогом	7%

Різниця в споживанні енергії завдяки оптимізації створеного програмного забезпечення, склала 25%, отже додаток, який працює у фоні почав використовувати на чверть менше ресурсів енергоносія смартфона.

Різниця в споживанні енергії в порівнянні з аналогом склала 15%, що є позитивним аргументом в сторону вибору саме цієї розробки замість аналогічного рішення, яке представлено на ринку.

Дане дослідження не є коректним для всіх його аналогів, оскільки не всі аналоги можуть здійснювати безпровідне підключення до машини і по суті є записниками зі глибоко зміненим функціоналом, що не сходиться з самою ідеєю створюваного додатка. Оже для дослідження було обрано саме додаток із схожим функціоналом та ідеєю. В якості аналога було обрано програму CarLogger.

За результатами проведеного досліду, можемо прийти до висновку, що дана оптимізація із модулем Bluetooth має сенс та значно зменшує споживання енергії смартфоном що позитивно відбивається на користувацькому досвіду від розробленого програмного засобу.

5 РОЗРАХУНОК ЕКОНОМІЧНОЇ ДОЦІЛЬНОСТІ СТВОРЕННЯ ПРОГРАМИ ВЕДЕННЯ ОБЛІКУ ОБСЛУГОВУВАННЯ АВТОМОБІЛІВ

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного засобу ведення обліку обслуговування автомобілів. Особливістю програми є те, що дана технологія полегшує ведення обліку по обслуговуванню авто таким чином щоб користувач завжди мав актуальну інформацію про стан розхідників його авто. Крім того, програмний засіб об’єднав в собі весь необхідний функціонал, який є у аналогів, та різні програмні рішення, якиз в конкурентів немає.

Аналогом може бути додаток CarLogger, його ціна близько 1440 грн/рік.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	3	4
Наявність аналогів на ринку	3	3	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	3	3	4
Сума	42	41	44
Середньоарифметична сума балів	(42+41+44) / 3 = 42,33		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт полегшує ведення обліку по обслуговуванню авто таким чином щоб користувач завжди мав актуальну інформацію про стан розхідників його авто. Крім того, програмний засіб об’єднав в собі весь необхідний функціонал.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \,, \tag{5.1}$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ — число робочих днів в місяці, 22 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.3.

Таблиця 5.3 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	52000	2363,64	43	101636,364
Програміст	45000	2045,45	43	87954,545
Всього				189590,91

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Додаткова заробітна плата розробників, які приймали участь в розробці обладнання.

Додаткова заробітна плата прийнято розраховувати як 15 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 15\ \% / 100\ \%\ , \qquad (5.2)$$

$$З_д = (189590{,}91 \cdot 15\ \% / 100\ \%\ ) = 28438{,}64\ \ (грн.)$$

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (189590{,}91 + 28438{,}64) \cdot 22\ \% / 100\ \% = 47966{,}50\ (грн.)$$

Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в

спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ — термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 32000 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його використання — 1,95 міс.

$$A_{обл} = \frac{32000}{2} \times \frac{1{,}95}{12} = 2660{,}06 \; грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.4.

Але, так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн (WebStorm, Android Studio), то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $B_{нем.ак.}$ = 600 грн.

Таблиця 5.4 — Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія (Mi Gaming Laptop)	32000	2	1,95	2606,061
Офісне обладнення (меблі)	25000	4	1,95	1017,992

Приміщення	1200000	20	1,95	9772,727
Всього				13396,78

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 43 \cdot 6,2 = 575,856 \text{ (грн.)}$$

Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$І_в = 189590,91 * 95\% / 100\% = 180111,4 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 189590,91 * 145\ \% / 100\ \% = 274907 \text{ (грн.)}$$

Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = 189590,91+28438,64+47966,50+13396,78+600+575,86+180111,4+ +274907 = 735586,86 \text{ грн.}$$

Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються *ЗВ*, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

ЗВ = 735586,86 / 0,5 = 1471174 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

а) вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

б) зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

в) кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

г) визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.10)$$

де $\pm \Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій ціні 350 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *50* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 50000 шт., протягом другого року — на 75000 шт., протягом третього року на 100000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*50 + (350 + 50)*50000)* 0{,}8333* 0{,}33) * (1 - 0{,}18) = 3946249{,}842 \text{ грн.}$$

$$\Delta П_2 = (0*50 + (350 + 50)*(50000+75000)* 0{,}8333* 0{,}33) * (1 - 0{,}18) = 11274999{,}549 \text{ грн.}$$

$$\Delta П_3 = (0*50 + (350 + 50)*(50000+75000+100000)* 0{,}8333* 0{,}33) * (1 - 0{,}18) = 20294999{,}18 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 35516248,58 грн.

Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

Де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}05 \ldots 0{,}15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

ПП= (3946249,842/(1+0,1)[1])+(11274999,549/(1+0,1)[2])+(20294999,188/
/(1+0,1)[3]) = 3587499,86 + 9318181,445 + 15247933,27 = 28153614,58 грн.

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \qquad (5.12)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку

приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 1471174 = 2942347{,}45 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{5.13}$$

$$E_{абс} = 28153614{,}58 - 2942347{,}45 = 25211267{,}12 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.14}$$

$T_{ж}$ – життєвий цикл наукової розробки, роки.

$$E_{в} = \sqrt[3]{(1 + 25211267,12/2942347,45} - 1 = 1,123$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.15}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_{в}}, \tag{5.16}$$

$$T_{ок} = 1 / 1,123 = 0,89 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,89 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 1471174 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для

інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,89 роки.

ВИСНОВКИ

Оглядаючись на сучасний розвиток інформаційних технологій та обчислювальної техніки, нове програмне забезпечення створюється буквально щомиті. Багато з нього є дуже досконалим та перевищує інше в одних чи інших показниках. Дана магістерська кваліфікаційна робота є програмним засобом для ведення обліку обслуговування автомобіля.

При виконанні даної роботи було розроблено працюючий прототип програми для ведення обліку обслуговування автомобілів, було проведено дослідження у сфері web галузі, на основі яких можна створити власний додаток, який не уступатиме аналогам на ринку. Також створено користувацький інтерфейс для збирання необхідних даних користувача та описано всі етапи розробки.

У першому розділі магістерської роботи проведений огляд особливостей роботи систем по веденню обліку обслуговування автомобілів, виконано огляд технології для створення мобільного додатку та огляд аналогів розроблюваного програмного засобу.

У другому розділі магістерської роботи була розроблена структура програмного продукту, вибрані засоби для її створення та середовище для наповнення додатку.

У третьому розділі магістерської роботи описаний процес створення, для формування якого було використано такі технології: TypeScript та React Native для написання фронтенду, Expo як основа створюваного додатку та система для відлагоджування розроблюваного продукту, Firebase як повноцінна платформа з власним API та базою даних, мова розмітки CSS та менеджер станів Redux. Було розроблено користувацький інтерфейс, який адаптується під операційну систему та фізичні розміри екрану. Розробка відбувалась у інтегрованому середовищі розробки WebStorm.

У четвертому розділі магістерської роботи описано перевірку працездатності розробленого програмного продукту та проведено тестування й експериментальне дослідження його роботи.

У п’ятому розділі магістерської роботи виконані економічні розрахунки із обґрунтування доцільності виконання нової роботи для ведення обліку обслуговування автомобілів, обчислені фінансові затрати на створення програмного продукту та визначено економічні переваги від впровадження запропонованого наукового рішення у вигляді завершеного програмного продукту.

Розроблений кросплатформний додаток відповідає всім вимогам, що були поставлені при його проектуванні. Дане програмне рішення було протестоване, програмних збоїв та помилок зафіксовано не було.

Даний програмний продукт є актуальним на ринку та повністю готовий до запуску у відкрите користування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. React Native [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/React_Native.
2. Пасічник О. Г. Основи веб-дизайну / О. Г. Пасічник, І. В. Стеценко., 2009. – 336 с.
3. Українськи блог [Електронний ресурс] – Режим доступу до ресурсу: https://hackit-ukraine.com.
4. Жизненный цикл компонента [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/web/react/2.6.php
5. Эволюция в вашем кармане: как развивались мобильные приложения [Електронний ресурс] – Режим доступу до ресурсу: https://apptractor.ru/info/articles/evolyutsiya-v-vashem-karmane-kak-razvivalis-mobilnyie-prilozheniya.html
6. Капличний О. С. Інформаційна система доставки контенту про оренду транспортних засобів [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2021/paper/view/13185.
7. Что такое Firebase и почему стоит с этим познакомиться [Електронний ресурс] – Режим доступу до ресурсу: https://kolmogorov.pro/what-is-firebase-chto-takoe
8. Virtual DOM and Internals[Електронний ресурс] – Режим доступу до ресурсу:https://uk.reactjs.org/docs/faq-internals.html
9. React Native для самых маленьких. Опыт мобильной разработки [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/oleg-bunin/blog/499636/
10. Мотивация [Електронний ресурс] – Режим доступу до ресурсу: https://rajdee.gitbooks.io/redux-in-russian/content/docs/introduction/Motivation.html
11. Створення сайтів [Електронний ресурс] – Режим доступу до ресурсу: https://webstudio2u.net.
12. Web технології та web дизайн [Електронний ресурс] – Режим

доступу до ресурсу:
https://sites.google.com/site/webtehnologiietawebdizajn/mova-javascriptta-ieie-mozlivosti.

13. Класифікація веб-сайтів [Електронний ресурс]: Конструктор сайтів – Режим
доступу: http://www.webkonstruktor.com/українською/типи-сайтів/.

14. Засоби створення Web-сайтів [Електронний ресурс]:
Навчальні матеріали онлайн – Режим доступу:
http://pidruchniki.com/1970070547797/informatika/zasobi_stvorennya_web-saytiv.

15. Архитектура Web-сайта [Електронний ресурс]: Основы и секреты современной front-end разработки – Режим доступу:
http://www.xiper.net/learn/bonus-books/programming-the-mobile-web/architectureand-design/website-architecture.html.

ДОДАТОК А

Міністерство освіти та науки України

Вінницький національний технічний університет

Інститут інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри _ОТ_

д.т.н., професор Азаров О. Д._____

(наук. ст., вч. зв., ініц. та прізв.) (підпис)

"____" _____20__р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Програмний засіб ведення обліку обслуговування автомобілів

08—23.МКР.021.00.000.ПЗ

Науковий керівник:

_____ к.т.н., доцент Колесник І. С.

(підпис) ме

студент групи _2КІ-21м_

_____ Капличний О.С.

(підпис)

Вінниця 2022

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Важливим є актуальність розробленого програмного рішення яке б дозволяло полегшити ведення обліку обслуговування авто. Враховуючи важливість мобільних програмних рішень, було прийнято рішення створювати саме кросплатформну мобільну систему.

1.2 Наказ про затвердження теми дипломної роботи.

2 Мета і призначення МКР

2.1 Мета полягає у створенні актуального та конкурентноспроможного програмного рішення у сфері ведення обліку обслуговування авто.

2.2 Призначення розробки полягає у створенні платформи для загального користування та зручного її масштабування до комерційних розмірів.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих технологій створення програмних рішень ведення обліку обслуговування автомобілів.

3.2 Розробки структури проекту, вибір техлологій та проектування архітектури створюваної платформи.

3.3 Розробка програмного забезпечення на основі отриманих при дослідженні даних.

3.4 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору

4 Вимоги до виконання МКР

Вимоги до виконання МКР:

— 4.1 Провести аналіз аналогічних рішень на ринку.

— 4.2 Вивчити найкращі практики у сфері web програмування;

— 4.3 Запропонувати власне рішення ведення обліку обслуговування авто;

— 4.4 Створити користувацький інтерфейс програми;

— 4. 5 Розробити прототип розроблюваного рішення.

— 5 Етапи МКР та очікувані результати

Етапи МКР описані в таблиці А.1

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз способів програмних засобів	01.09.22р.	04.10.22р.	Розділ 1
2	Проектування програмного засобу	05.10.22р.	18.10.22р.	Розділ 2
3	Розробка програмного засобу	19.10.22р.	01.11.22р.	Розділ 3
4	Економічна частина	02.11.22р.	15.11.22р.	Розділ 4

6 Матеріали, що подаються до захисту МКР

Пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Единa система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Лістинг програми

CarDetailsScreen.tsx

```
import { useNavigation } from "@react-navigation/native";
import React, {FC} from "react";
import {View, Text, StyleSheet, Button, ScrollView, ActivityIndicator, Image} from
"react-native";
import { AntDesign } from '@expo/vector-icons';
import MainStyles from "../styles/mainStyles"
import HeaderButton from "../components/HeaderButton";
import {useSelector} from "react-redux";
import {RootState} from "../store/store";
import {CarsState} from "../store/types/cars";
import LabeledInput from "../components/LabeledInput";
import CustomField from "../components/CustomField";
import CustomButton from "../components/CustomButton";
const CarDetailsScreen: FC = (): JSX.Element => {
   const navigation = useNavigation();
   const {selectedCar}: CarsState = useSelector((store:RootState) => store.cars);
   return (
      <View style={MainStyles.container}>
         <View style={MainStyles.header}>
            <HeaderButton
               onPress={(): void => navigation.goBack()}
               icon={<AntDesign name="left" size={24} color="white" />}
            />
            <View>
               <Text style={MainStyles.headerText}>Подробиці</Text>
```

```
            <Text style={MainStyles.subHeaderText}>{`${selectedCar?.carBrand}
${selectedCar?.model}`}</Text>
          </View>
        </View>
        <ScrollView contentContainerStyle={MainStyles.content}>
          {
            selectedCar ? (
              <View style={[styles.content]}>
                <View style={{marginBottom: 10}}>
                  <Image style={{height: 200, width: "100%", borderRadius: 12}}
source={{uri: selectedCar.url}}/>
                </View>
                <View style={{paddingHorizontal: 8}}>
                  <View style={styles.modelBlock}>
                    <CustomField
                      label={"Марка авто"}
                      text={selectedCar.carBrand}
                    />
                    <CustomField
                      label={"Модель авто"}
                      text={selectedCar?.model}
                    />
                  </View>
                  <CustomField
                    label={"Місто"}
                    text={selectedCar.city}
                  />
                  <CustomField
                    label={"Тип палива"}
                    text={selectedCar?.fuel}
```

```
    />
    <CustomField
      label={"Розхід палива"}
      text={selectedCar?.fuelConsumption}
    />
    <CustomField
      label={"Коробка передач"}
      text={selectedCar?.transmission}
    />
    <CustomField
      label={"Ціна оренди"}
      text={selectedCar?.price}
    />
    <CustomField
      label={"Опис"}
      text={selectedCar.description}
    />
  </View>
  <View style={styles.buttonBlock}>
    <CustomButton
      buttonText={"Подзвонити"}
      active={true}
      onPress={() => alert("write")}
      style={{width: "40%"}}
    />
    <CustomButton
      buttonText={"Написати"}
      active={true}
      onPress={() => alert("call")}
      style={{width: "40%"}}
```

```
                        />
                    </View>
                </View>
            ) : <ActivityIndicator size={"large"}/>
        }
      </ScrollView>
    </View>
  );
};
const styles = StyleSheet.create({
  modelBlock: {
  },
  content: {
    flexDirection: "column",
    justifyContent: "space-between",
    height: "95%",
  },
  buttonBlock: {
    flexDirection: "row",
    justifyContent: "space-around",
    marginTop: 10
  }
});
export default CarDetailsScreen;
```

AddCarScreen.tsx

```
import React, {FC, useEffect, useState} from "react";
import {View, Text, StyleSheet, KeyboardAvoidingView, ScrollView, Dimensions,
Platform} from "react-native";
import MainStyles from "../styles/mainStyles";
```

```
import HeaderButton from "../components/HeaderButton";
import {AntDesign} from "@expo/vector-icons";
import { useNavigation } from "@react-navigation/native";
import LabeledInput from "../components/LabeledInput";
import CustomButton from "../components/CustomButton";
import {useDispatch, useSelector} from "react-redux";
import {addNewCar} from "../store/actions/cars";
import {ADD_NEW_CAR, CarType, UPDATE_CARS} from "../store/types/cars";
import {RootState} from "../store/store";
import "firebase/firestore"
const AddCarScreen: FC = (): JSX.Element => {
    const navigation = useNavigation();
    const {cars} = useSelector((store: RootState) => store.cars);
    useEffect(() => {
        if (cars.length) {
            setForm({...form, id: cars[cars.length - 1].id + 1})
        }
    }, [cars]);
    const dispatch = useDispatch();
    const [form, setForm] = useState<CarType>({
        id: 0,
        carBrand: "",
        model: "",
        fuel: "",
        fuelConsumption: "",
        price: "",
    });
    const sendForm = () => {
        dispatch(addNewCar(form));
        navigation.goBack();
```

```
};
const isActive = (): boolean => {
    for (let item in form) {
        if (!form[item]) return false;
    }
    return true;
};
return (
    <View style={MainStyles.container}>
        <View style={MainStyles.header}>
            <HeaderButton
                onPress={(): void => navigation.goBack()}
                icon={<AntDesign name="left" size={24} color="white" />}
            />
            <View>
                <Text style={MainStyles.headerText}>Додайте своє авто</Text>
            </View>
        </View>
        <ScrollView contentContainerStyle={[MainStyles.content]}>
            <CustomButton
                buttonText={"Upload"}
                onPress={() => {
                    // uploadImageToStorage()
                }}
            />
            <LabeledInput
                label={"Місто"}
                value={form.city}
                placeholder={"Введіть місто..."}
                onChangeText={(text: string) => setForm({...form, city: text})}
```

```
            />
            <LabeledInput
              label={"Марка авто"}
              value={form.carBrand}
              placeholder={"Введіть марку..."}
              onChangeText={(text: string) => setForm({...form, carBrand: text})}
            />
            <LabeledInput
              label={"Модель"}
              value={form.model}
              placeholder={"Введіть модель..."}
              onChangeText={(text: string) => setForm({...form, model: text})}
            />
            <LabeledInput
              label={"Тип палива"}
              value={form.fuel}
              placeholder={"Бензин/газ/електроенергія"}
              onChangeText={(text: string) => setForm({...form, fuel: text})}
            />
            <LabeledInput
              label={"Розхід палива"}
              value={form.fuelConsumption}
              placeholder={"Введіть розхід"}
              onChangeText={(text: string) => setForm({...form, fuelConsumption:
text})}
            />
            <LabeledInput
              label={"Коробка передач"}
              value={form.transmission}
              placeholder={"Механіка/автомат"}
```

```
            onChangeText={(text: string) => setForm({...form, transmission: text})}
          />
          <LabeledInput
            label={"Ціна оренди"}
            value={form.price}
            placeholder={"Введіть ціну..."}
            keyboardType={"numeric"}
            onChangeText={(text: string) => setForm({...form, price: text})}
          />
          <LabeledInput
            label={"Опис"}
            value={form.description}
            placeholder={"Введіть опис..."}
            multiline={true}
            onChangeText={(text: string) => setForm({...form, description: text})}
          />
          <CustomButton
            style={{marginTop: 25, marginHorizontal: 20}}
            onPress={sendForm}
            buttonText={"Надіслати"}
            active={isActive()}
          />
        </ScrollView>
      </View>
    );
};
const styles = StyleSheet.create({
});
export default AddCarScreen;
```

LoginScreen.tsx

```
import React, {FC, useEffect, useState} from "react";
import {
    AsyncStorage,
    Button,
    NativeSyntheticEvent,
    StyleSheet,
    TextInput,
    TextInputChangeEventData,
    View
} from "react-native";
import Toast from "react-native-tiny-toast";
import {useDispatch} from "react-redux";
import validateEmail from "../helpers/EmailValidator";
import {ISignInData} from "../types";
import { useNavigation } from "@react-navigation/native";
import {SIGNUP_SCREEN} from "../navigation/routes";
import DismissKeyboardView from "../components/DismissKeyboardView";
import MainStyles, {MAIN_COLOR, SHADOW_COLOR} from
"../styles/mainStyles";
import CustomButton from "../components/CustomButton";
import {LOGIN_SUCCESS} from "../store/types/auth";
import * as firebase from "firebase";
import * as SecureStore from "expo-secure-store";
const LoginScreen: FC = () => {
    const dispatch = useDispatch();
    const navigation = useNavigation();
    const [formData, setFormData] = useState<ISignInData>({
        username: "",
        password: ""
```

```
});
const { username, password } = formData;
const onChangeTextHandler = (
    event: NativeSyntheticEvent<TextInputChangeEventData>,
    target: string,
): void => {
    setFormData({
        ...formData,
        [target]: event.nativeEvent.text,
    });
};
const loginHandler = () => {
    firebase
        .auth()
        .signInWithEmailAndPassword(formData.username, formData.password)
        .then(() => {
            SecureStore.setItemAsync("email", formData.username);
            dispatch({type: LOGIN_SUCCESS})
        })
        .catch(error => alert(error))
};
return(
    <View style={[MainStyles.content, {flex: 1, justifyContent: "center"}]}>
        <DismissKeyboardView style={{width: "100%"}}>
            <View style={styles.form}>
                <View style={styles.inputTop}>
                    <TextInput
                        style={styles.inputField}
                        onChange={e => onChangeTextHandler(e, "username")}
                        value={username}
```

```
            textContentType="emailAddress"
            placeholder="Email"
            onEndEditing={({nativeEvent}) => {
              if (!validateEmail(nativeEvent.text)) {
                Toast.show('Email must be valid!');
              }
            }}
          />
        </View>
        <View style={styles.inputBottom}>
          <TextInput
            style={styles.inputField}
            onChange={e => onChangeTextHandler(e, "password")}
            textContentType="password"
            placeholder="Пароль"
            value={password}
            secureTextEntry={true}
            onEndEditing={({nativeEvent}) => {
              if (nativeEvent.text.trim().length < 6) {
                Toast.show(`Password must be 6 characters long`);
              }
            }}
          />
        </View>
      </View>
      <CustomButton
        style={{marginTop: 15}}
        buttonText="Вхід"
        onPress={loginHandler}
        active={true}
```

```
            />
            <CustomButton
              style={{marginVertical: 10}}
              buttonText="Зареєструватися"
              onPress={() => navigation.navigate(SIGNUP_SCREEN)}
              active={true}
            />
          </DismissKeyboardView>
        </View>
    );
};
const styles = StyleSheet.create({
    screen: {
        flex: 1,
        justifyContent: "center",
        alignItems: "center"
    },
    form: {
    },
    inputTop: {
        borderColor: MAIN_COLOR,
        borderWidth: 2,
        borderBottomWidth: 1
    },
    inputBottom: {
        borderColor: MAIN_COLOR,
        borderWidth: 2,
        borderTopWidth: 1
    },
    inputField: {
```

```
        paddingLeft: 15,
        paddingVertical: 8
    }
});
export default LoginScreen;
```

RootScreen.tsx

```
import { useNavigation } from "@react-navigation/native";
import React, {FC, useEffect} from "react";
import {
    View,
    Text,
    Button,
    StyleSheet,
    TouchableOpacity,
    ScrollView,
    ActivityIndicator,
    YellowBox,
    Dimensions, StatusBar, TextInput, Image
} from "react-native";
import {useDispatch, useSelector} from "react-redux";
import {ADD_CAR_SCREEN, CAR_DETAIL_SCREEN,
STARRED_CARS_SCREEN} from "../navigation/routes";
import MainStyles, {MAIN_COLOR, SHADOW_COLOR} from
"../styles/mainStyles";
import {AntDesign, FontAwesome } from "@expo/vector-icons";
import CarCard from "../components/CarCard";
import {CarsState, UPDATE_CARS,} from "../store/types/cars";
import {RootState} from "../store/store";
import * as firebase from "firebase";
```

```
import {getCars} from "../store/actions/cars";

const RootScreen: FC = (): JSX.Element => {
    const {cars}: CarsState = useSelector((store: RootState) => store.cars);
    const navigation = useNavigation();
    const dispatch = useDispatch();
    useEffect(() => {
        dispatch(getCars());
    }, []);
    return (
        <View style={MainStyles.container}>
            <StatusBar barStyle={"light-content"} animated={true} backgroundCol-
or={MAIN_COLOR}/>
            <View style={[MainStyles.header, {justifyContent: "space-between"}]}>
                <View style={{marginLeft: 25}}>
                    {/*<Text style={MainStyles.headerText}>Cars List</Text>*/}
                    <AntDesign name="dingding" size={24} color="white" />
                </View>
                <View style={styles.localHeader}>
                    <TouchableOpacity style={{paddingLeft: 15}} onPress={() => navi-
gation.navigate(ADD_CAR_SCREEN)}>
                        <AntDesign name="plus" size={24} color="white" />
                    </TouchableOpacity>
                </View>
            </View>
            <ScrollView
                contentContainerStyle={[MainStyles.content]}
            >
                {
```

```
                cars.length ? cars.map(car => <CarCard key={car.id} img={<Image
style={{height: 200, width: "100%", borderRadius: 12}} source={{uri: car.url}}/>}
carItem={car}/>)
                    :
                    <View style={{flex: 1, justifyContent: "center", alignItems: "cen-
ter"}}>
                        {/*<ActivityIndicator size={"large"} col-
or={MAIN_COLOR}/>*/}
                    </View>
            }
        </ScrollView>
    </View>
  );
};
const styles = StyleSheet.create({
  cardItem: {
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "grey",
    borderRadius: 20,
    width: "100%"
  },
  localHeader: {
    marginRight: 15,
    flexDirection: "row",
    justifyContent: "space-between",
    width: "70%"
  }
});
export default RootScreen;
```

SignupScreen.tsx

```
import React, {FC, useState} from "react";
import {Button, View, Text, TextInput, StyleSheet} from "react-native";
import DismissKeyboardView from "../components/DismissKeyboardView";
import Toast from "react-native-tiny-toast";
import validateEmail from "../helpers/EmailValidator";
import { useNavigation } from "@react-navigation/native";
import {LOGIN_SCREEN, SIGNUP_SCREEN} from "../navigation/routes";
import MainStyles, {MAIN_COLOR} from "../styles/mainStyles";
import {useDispatch} from "react-redux";
import {SignUpData} from "../store/types/auth";
import {signup} from "../store/actions/auth";
import CustomButton from "../components/CustomButton";
const SignupScreen: FC = (): JSX.Element => {
    const [formData, setFormData] = useState<SignUpData>({
        email: "",
        first_name: "",
        second_name: "",
        password: ""
    });
    const dispatch = useDispatch();
    const navigation = useNavigation();
    const {
        email,
        first_name,
        second_name,
        password
    } = formData;
    const signUpHandler = () => {
```

```
    dispatch(signup(formData))
  };
  return (
    <View style={[MainStyles.content, {flex: 1, justifyContent: "center"}]}>
      <DismissKeyboardView style={{width: "100%"}}>
        <Text style={{ fontSize: 30, alignSelf: 'center', marginBottom: 10 }}>
          Реєстрація
        </Text>
        <View>
          <View style={styles.inputTop}>
            <TextInput
              style={styles.inputField}
              onChange={({nativeEvent}) => setFormData({...formData, email:
nativeEvent.text})}
              value={email}
              textContentType="emailAddress"
              placeholder="email"
              onEndEditing={({nativeEvent}) => {
                validateEmail(nativeEvent.text);
              }}
            />
          </View>
          <View style={styles.inputMiddle}>
            <TextInput
              style={styles.inputField}
              onChange={({nativeEvent}) => setFormData({...formData,
first_name: nativeEvent.text})}
              textContentType="name"
              value={first_name}
              placeholder="Ім'я"
```

```
                onEndEditing={({nativeEvent}) => {
                    if (nativeEvent.text.trim().length < 2) {
                        Toast.show('First name must be not empty!');
                    }
                }}
            />
        </View>
        <View style={[styles.inputMiddle, {borderBottomWidth: 1, borderTopWidth: 0}]}>
            <TextInput
                style={styles.inputField}
                onChange={({nativeEvent}) => setFormData({...formData, second_name: nativeEvent.text})}
                textContentType="name"
                value={second_name}
                placeholder="Прізвище"
                onEndEditing={({nativeEvent}) => {
                    if (nativeEvent.text.trim().length < 2) {
                        Toast.show('Last name must be not empty!');
                    }
                }}
            />
        </View>
        <View style={styles.inputBottom}>
            <TextInput
                style={styles.inputField}
                onChange={({nativeEvent}) => setFormData({...formData, password: nativeEvent.text})}
                textContentType="password"
                value={password}
```

```
            placeholder="Пароль"
            secureTextEntry={true}
            onEndEditing={({nativeEvent}) => {
              if (nativeEvent.text.trim().length < 6) {
                Toast.show(` Password must be 6 characters long`);
              }
            }}
          />
        </View>
        <View style={[styles.inputMiddle, {borderTopWidth: 0}]}>
          <TextInput
            style={styles.inputField}
            value={password}
            placeholder="Номер телефону"
          />
        </View>
      </View>
      <CustomButton
        style={{marginVertical: 10}}
        buttonText="Зареєструватись"
        onPress={signUpHandler}
        active={true}
      />
      <CustomButton
        style={{marginBottom: 10}}
        buttonText="Увійти"
        onPress={() => navigation.navigate(LOGIN_SCREEN)}
        active={true}
      />
    </DismissKeyboardView>
```

```
        </View>
    );
};
const styles = StyleSheet.create({
    screen: {
        flex: 1,
        justifyContent: "center",
        alignItems: "center"
    },
    inputTop: {
        borderColor: MAIN_COLOR,
        borderWidth: 2,
        // borderTopLeftRadius: 20,
        // borderTopRightRadius: 20,
        borderBottomWidth: 1
    },
    inputBottom: {
        borderColor: MAIN_COLOR,
        borderWidth: 2,
        borderTopWidth: 1
    },
    inputMiddle: {
        borderColor: MAIN_COLOR,
        borderWidth: 2,
        borderTopWidth: 1,
    },
    inputField: {
        paddingLeft: 15,
        paddingVertical: 8
    }
```

```
});
export default SignupScreen;
```

CarsScreen.tsx

```
import React, {FC, useEffect} from "react";
import {View, Text, ScrollView, StyleSheet, TouchableOpacity, ActivityIndica-tor,
Image} from "react-native";
import { useNavigation } from "@react-navigation/native";
import MainStyles from "../styles/mainStyles";
import {useDispatch, useSelector} from "react-redux";
import CarCard from "../components/CarCard";
import {CarType} from "../store/types/cars";
import {RootState} from "../store/store";
import {getStarredCars} from "../store/actions/cars";
const StarredCarsScreen: FC = (): JSX.Element => {
    const navigation = useNavigation();
    const dispatch = useDispatch();
    useEffect(() => {
        dispatch(getStarredCars())
    }, []);
    const {starred_cars} = useSelector((store: RootState) => store.cars);
    return (
        <View style={MainStyles.container}>
            <View style={[MainStyles.header]}>
                <View style={{marginLeft: 25}}>
                    <Text style={MainStyles.headerText}>Обрані оголошення</Text>
                </View>
            </View>
            <ScrollView contentContainerStyle={MainStyles.content}>
                {
```

```
                starred_cars ? <View>
                    {
                        // @ts-ignore
                        // starred_cars.map((item: CarType) => <CarCard key={item.id}
img={<Image style={{height: 200, width: "100%", borderRadius: 12}}
source={{uri: starred_cars.url}}/>} carItem={item}/>)
                        starred_cars.length ? starred_cars.map(car => <CarCard
key={car.id} img={<Image style={{height: 200, width: "100%", borderRadius: 12}}
source={{uri: car.url}}/>} carItem={car}/>)
                            : <Text style={{color: "black"}}>List is empty</Text>
                    }
                </View> : <Text style={{color: "black"}}></Text>
            }
        </ScrollView>
    </View>
  )
};
const styles = StyleSheet.create({
});
export default StarredCarsScreen;

UserDataScreen.tsx
import React, {FC, useEffect, useState} from "react";
import {View, StyleSheet, Text, Dimensions, TouchableOpacity, StatusBar} from
"react-native";
import MainStyles, {MAIN_COLOR, SHADOW_COLOR} from
"../styles/mainStyles";
import {useDispatch, useSelector} from "react-redux";
import {RootState} from "../store/store";
import LabeledInput from "../components/LabeledInput";
```

```
import CustomButton from "../components/CustomButton";
import DismissKeyboardView from "../components/DismissKeyboardView";
import {getUser, updateUserData} from "../store/actions/user";
import {UserType} from "../store/types/user";
import {logout} from "../store/actions/auth";
import * as firebase from "firebase";
import "firebase/firestore"
import * as SecureStore from "expo-secure-store";
const UserDataScreen: FC = (): JSX.Element => {
    const {user} = useSelector((store:RootState) => store.user);
    const [profileLocalInfo, setProfileLocalInfo] = useState<UserType>({
        first_name: "",
        second_name: "",
        email: ""
    });
    const dispatch = useDispatch();
    useEffect(() => {
        // StatusBar.setBarStyle("dark-content");
        // StatusBar.setBackgroundColor("#f1f1f1", true);
        dispatch(getUser())
    }, []);
    useEffect(() => {
        if (user) setProfileLocalInfo(user);
    }, [user]);
    const updateProfileData = async () => {
        // dispatch(updateUserData(profileLocalInfo))
        const email = await SecureStore.getItemAsync("email");
        if (email) firebase.firestore()
            .collection("/users")
            .doc(email)
```

```
            .update(profileLocalInfo)
            .then(() => dispatch(updateUserData(profileLocalInfo)))
            .catch(err => alert(err))
    };
    const isActive = (): boolean => {
        for (let item in profileLocalInfo) {
            if (!profileLocalInfo[item]) return false;
        }
        return true
    };
    return (
        <View style={[MainStyles.container]}>
            {/*<StatusBar backgroundColor={"#f1f1f1"} animated={true}
barStyle={"dark-content"}/>*/}
            <DismissKeyboardView>
                <View style={[MainStyles.header, {backgroundColor: "transparent",
justifyContent: "space-between"}]}>
                    <View>
                        <Text style={[MainStyles.headerText, {color: MAIN_COLOR,
marginLeft: 25, marginTop: 20}]}>Дані користувача:</Text>
                    </View>
                    <TouchableOpacity onPress={() => dispatch(logout())}>
                        <Text style={[MainStyles.headerText, {color: SHAD-OW_COLOR,
marginHorizontal: 25, marginTop: 20, fontSize: 20}]}>Вихід</Text>
                    </TouchableOpacity>
                </View>
                <View style={MainStyles.content}>
                    <LabeledInput
                        label={"Ім'я"}
                        value={profileLocalInfo.first_name}
```

```
                onChangeText={(text: string) => setProfileLocalIn-
fo({...profileLocalInfo, first_name: text})}
                placeholder={"Введіть ім'я"}
              />
              <LabeledInput
                label={"Прізвище"}
                value={profileLocalInfo.second_name}
                onChangeText={(text: string) => setProfileLocalIn-
fo({...profileLocalInfo, second_name: text})}
                placeholder={"Введіть прізвище"}
              />
              <LabeledInput
                label={"Email"}
                value={profileLocalInfo.email}
                onChangeText={(text: string) => setProfileLocalIn-
fo({...profileLocalInfo, email: text})}
                placeholder={"Введіть email"}
              />
              />
            </View>
          </DismissKeyboardView>
        </View>
    )
};
const styles = StyleSheet.create({
});
export default UserDataScreen;
```

ДОДАТОК В

Опис даних для реалізації аутентифікації

```
export const LOGIN_SUCCESS = 'AUTH/LOGIN_SUCCESS';
export const LOGIN_FAIL = 'AUTH/LOGIN_FAIL';
export const LOGOUT = 'AUTH/LOGOUT';
export interface SignUpData {
   email: string,
   first_name: string,
   second_name: string,
   password: string
}
interface LogInSuccess {
   type: typeof LOGIN_SUCCESS;
   payload: object
}
interface LoginFailAction {
   type: typeof LOGIN_FAIL;
}
interface LogOut {
   type: typeof LOGOUT;
}
export interface AuthState {
   isAuthenticated: boolean;
   loading: boolean;
}
export type AuthActionTypes = | LogInSuccess | LoginFailAction | LogOut;
```

Структура даних для опису користувача

```
import {CarType} from "./cars";
```

```
export const UPDATE_USER_DATA = 'USER/UPDATE_USER_DATA';
export interface UserType {
    first_name: string;
    second_name: string;
    email: string,
    telephone_number: string
}
export default interface UserStateType {
    user: UserType | null;
    user_cars: CarType[] | []
}
export type UserActionTypes = updateUserData
```

Опис даних для роботи з машинами

```
export const UPDATE_CARS = "UPDATE_CARS";
export const SELECT_CAR = "SELECT_CAR";
export const ADD_NEW_CAR = "ADD_NEW_CAR";
export interface CarType {
    id: number;
    carBrand: string;
    model: string;
    fuel: string
    price: string;
    fuelConsumption: string;
    description: string
}
export interface CarsState {
    cars: CarType[];
    selectedCar: CarType | null;
```

```
    starred_cars: CarType[] | [];
    filters: object
}
interface GetCars {
    type: typeof UPDATE_CARS,
    cars: CarType[]
}
interface UpdateStarredCars {
    type: typeof UPDATE_STARRED_CARS,
    cars: CarType[]
}
interface SelectCar {
    type: typeof SELECT_CAR,
    id: number
}
interface AddToStarred {
    type: typeof ADD_TO_STARRED,
    car: CarType
}
interface RemoveFromStarred {
    type: typeof REMOVE_FROM_STARRED,
    id: number
}
interface AddNewCar {
    type: typeof ADD_NEW_CAR,
    car: CarType
}
export type CarsActionsTypes = | GetCars | SelectCar | AddToStarred |
RemoveFromStarred | AddNewCar | UpdateStarredCars;
```

ДОДАТОК Г

План розподілу полів головної сторінки

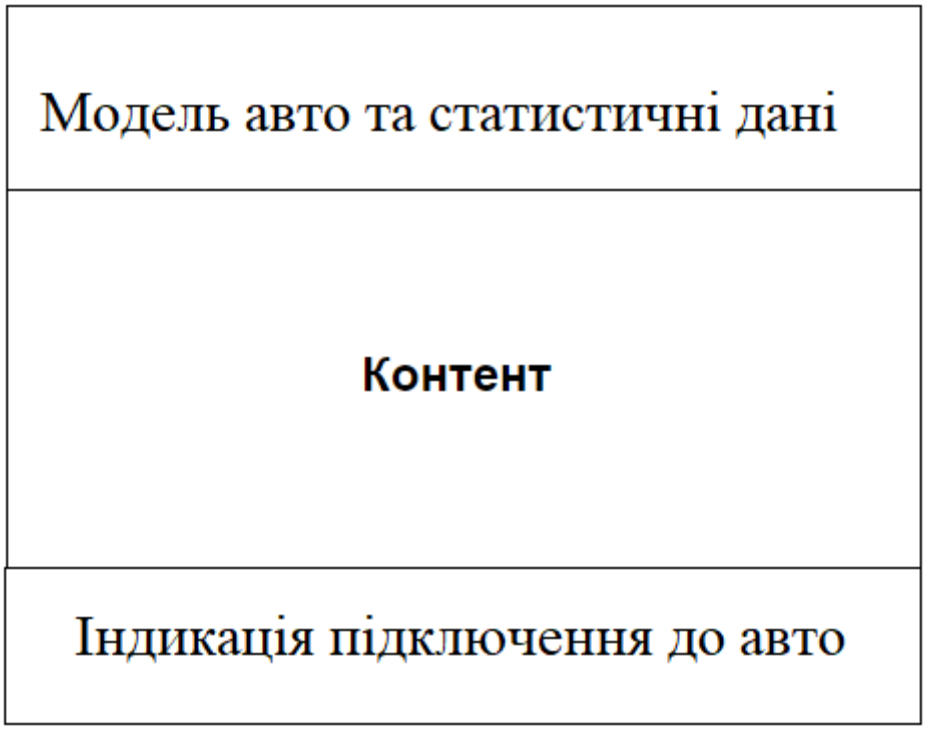

Рисунок Г.1 – План розподілу полів головної сторінки

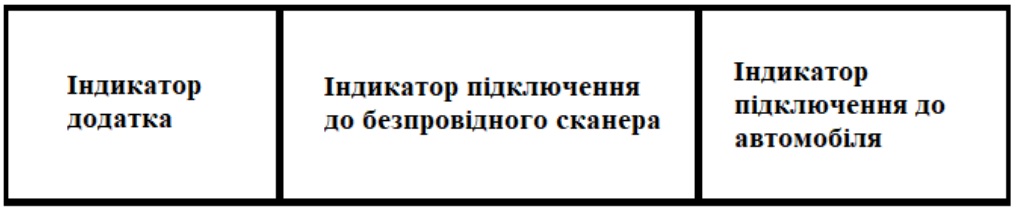

Рисунок Г.2 — Індикаційна панель додатка

ДОДАТОК Д

Загальний вигляд головної сторінки додатка

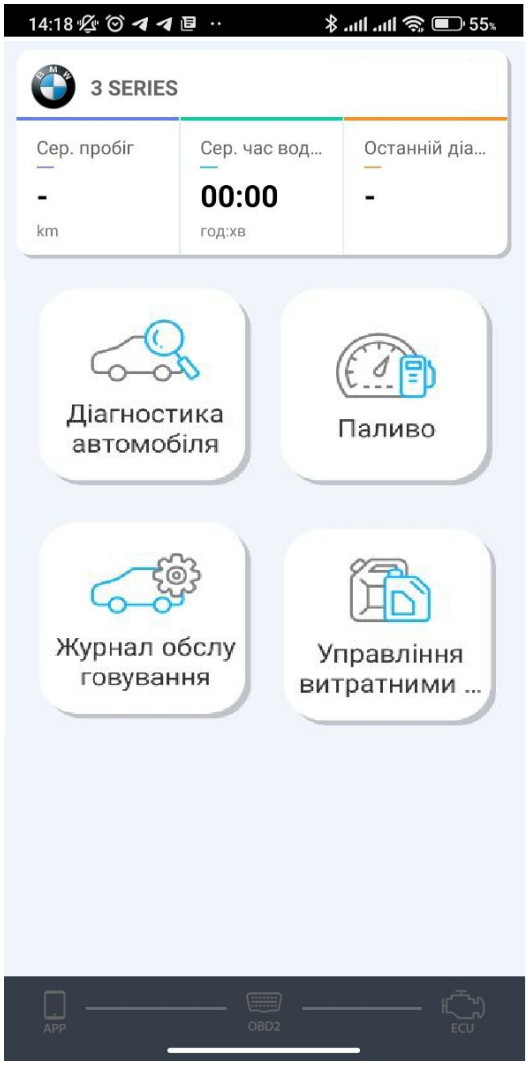

Рисунок Д.1 — Загальний вигляд головної сторінки додатка

ДОДАТОК Е

Структурна схема додатка

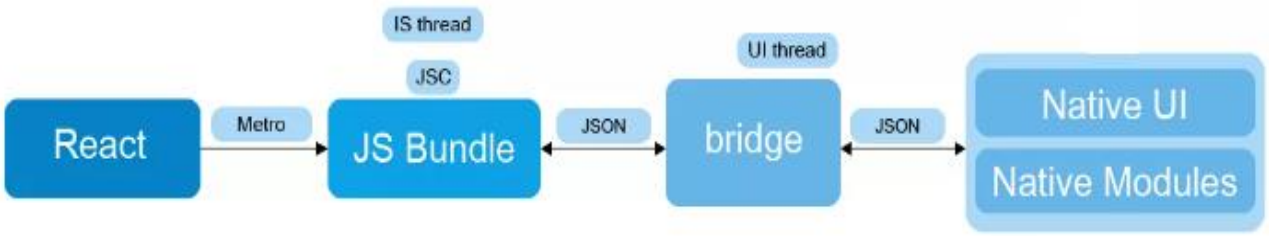

Рисунок Е.1 — Структурна схема додатка

ДОДАТОК Е

Діагостичний сканер-адаптер

Рисунок Е.1 — Діагостичний сканер-адаптер

ДОДАТОК Ж

Сторінка реєстрації авто

Рисунок Ж.1 — Сторінка реєстрації авто

ДОДАТОК З

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний засіб ведення обліку обслуговування автомобілів

Тип роботи: магістерська кваліфікаційна робота
(БДР,МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра,факультет)

Показники звіту подібності Unicheck

Оригінальність 93,9 Схожість 6,1 .

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище,ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Капличний О. С.
(підпис) (прізвище,ініціали)

Керівник роботи _____ Колесник І. С.
(підпис) (прізвище,ініціали)