КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СЕРВІСУ ТАКСІ

Виконав студент 4-го курсу
Віктор СВИНАР _____
(підпис)

Науковий керівник:
доцент
Євген ІВАНОВ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засідання кафедри інтелектуальних програмних систем
«____» _____ 2023 р.,
протокол № ___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 58 сторінок, 41 рисунків, 10 використаних джерел.

ANDROID, JAVA, Firebase Realtime Database, ORACLE, TYPESCRIPT, МОБІЛЬНИЙ ДОДАТОК, ІНФОРМАЦІЙНА СИСТЕМА, СЕРВЕР, САЙТ АДМІНІСТРАТОРА, ФРЕЙМВОРК,.

Об’єктом роботи є інформаційна система для сервісу таксі.

Метою роботи є розробити програмне забезпечення для замовлення авто у невеликому місті, яке б мало перш за все зручну систему адміністрування, аналізу роботи, цікавої гри, що дала б змогу заохочувати більшу кількість клієнтів користуватися даним сервісом.

Предметом роботи є сервер який керує усією бізнес логікою системи, мобільний додаток, що надає відповідний функціонал як для користувача так і для водія, а також сайт для адміністрування та аналізу роботи системи.

Інструменти розроблення: середовища IntelliJ IDEA, Android Studio мови програмування Java, TypeScript, Android SDK, фреймворки Spring, Angular, СУБД Oracle та не реляційна база даних Firebase Realtime Database.

Результати роботи: розроблено програмне забезпечення для сервісу таксі, а саме: сервер, мобільний додаток та сайт адміністратора. Набуті знання у проектуванні інформаційної системи та правильної комунікації різних компонент сервісу.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API - Application Programming Interface, прикладний програмний інтерфейс;

CRUD – Create read update delete, створення, читання, оновлення і вилучення;

DAO – Data access object, об'єкт доступу до даних;

HTTP - HyperText Transfer Protocol, протокол передачі даних;

JVM - Java Virtual Machine, віртуальна машина Джава;

JWT - JSON Web Token, джейсон веб токен;

REST API - Representational State Transfer API, API передачі репрезентативного стану;

URL - Uniform Resource Locator, уніфікований локатор ресурсу;

АОР - Аспектно-орієнтоване програмування;

ООП - Об’єктно-орієнтоване програмування.

ВСТУП

Протягом багатьох років технології змінюють наш світ і повсякденне життя. Вони прокладають шлях до багатофункціональних пристроїв, які і надалі продовжують розвиватися, стаючи ще кращими. Завдяки всім цим революціям, технології зробили наше життя легшим, швидшим і веселішим.

Сьогодні смартфони, а й відповідно мобільні додатки, які створюються для них, стали невід'ємною частиною життя людини. Різні речі, які раніше потребували значних затрат на їх виконання, сьогодні робляться в один клік. Наприклад, операції з банківськими картками, страхування, покупки в інтернеті, медичні послуги, розважальний контент, можливість зробити та обробити фото, спілкування, а також, швидке замовлення різних, корисних для людини послуг у будь-який час. Однією з таких послуг є оренда таксі.

Людям притаманно поспішати, тому часто їм доводиться викликати автомобіль для того, щоб як найшвидше дістатися свого місця призначення. Сучасні технології розвинули даний сервіс до неймовірних масштабів і майже витіснили з ринку своїх попередників. Мобільні додатки надають зручний і швидкий спосіб замовлення таксі, а також багато різних корисних функцій для клієнтів. Існує безліч аналогів подібних сервісів, кожен з яких намагається йти в ногу із сучасними технологіями, а також зберігати власну особливість, щоб бути конкурентоспроможним на ринку.

Світ швидко змінюється, можуть відбуватися події, які кардинально змінюють життя великої кількості людей, через це виникає потреба адаптації у різних сферах життя. Тому, були проаналізовані аналоги сучасних мобільних додатків для сервісу таксі. Результати дослідження спонукнули поставити мету створити програмне забезпечення для замовлення авто у невеликому місті, яке б мало перш за все зручну систему адміністрування, аналізу роботи, цікавої гри, що дала б змогу заохочувати більшу кількість клієнтів користуватися даним сервісом. Також, необхідно створити комфортний як і для замовника, так і для водія, не тільки графічний інтерфейс, а й систему

комунікацій між ними, адже, у малому за розміром місті, часто виникатиме ситуація повторної зустрічі клієнта і таксиста, що наводить на думку створення системи улюблених водіїв. На жаль, зважаючи на сьогоденні реалії, а саме війну, не можна її обійти стороною, тому за мету також ставиться реалізувати відповідні привілеї для захисників, військових, які боронять державу Україну, адже саме вони дають змогу втілювати у життя дану ціль.

Для досягнення вищенаведеного, потрібно було розв’язати кілька завдань. Перша задача - це визначення тенденцій сучасного життя людей, з врахуванням того, що в державі відбувається війна. Друга – це проектування та розробка програмного забезпечення для мобільної платформи. Третім завданням було створення сайту для адміністрування та аналізу роботи системи. Четверта, остання задача – це налагодження та тестування продукту.

У результаті розробки проекту очікується, що він створить невелике, дружнє суспільство з водіїв та клієнтів, які будуть із задоволенням взаємодіяти між собою, що зробить продукт успішним.

1 ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ РОЗРОБКИ

У дипломній роботі поставлена задача розробити проект, а саме: інформаційну систему для адміністрування та аналізу роботи, а також мобільний додаток, для надання користувачам сервісу замовлення автомобілів, а для водії прийому та виконання замовлень.

1.1 Інструменти для розробки серверної частини

Для розробки сервера було обрану таку мову програмування, як Java. Об’єктно орієнтована мова Java була розроблена компанією Sun Mircosystems у 1995 році.

ООП – це методологія програмування, яка базується на функціонуванні програмного продукту як результат взаємодії сукупності об’єктів, кожен з яких є екземпляром конкретного класу.

Об’єкт – це іменована модель реальної сутності, яка володіє конкретними значеннями властивостей і проявляє свою поведінку.

Клас – це модель інформаційної сутності, яка надає універсальний тип даних, що складається з набору полів даних і методів їх обробки.

У застосуванні до об'єктно-орієнтованих мов програмування поняття об'єкта та класу конкретизуються. У всіх мовах класи та об'єкти мають ряд загальних властивостей, таких як інкапсуляція (об'єднання відкритих даних і закритих методів), успадкування (запозичення функціональності базових класів похідними), поліморфізм (можливість використання об'єктів з однаковим інтерфейсом при успадкування)

Java стала популярною завдяки своїй простоті використання, мобільності та широкому застосуванні у різних сферах розробки програмного забезпечення. При створенні Java передбачалась простішою ніж його синтаксичний предок C++, але сьогодні з появою нових версій можливості мови суттєво розширились і багато в чому переважають функціональність C++. Також у Java відсутні покажчики, тому замість них широко

використовуються адреси. Відсутність множинного наслідування легко замінюється на більш зрозумілі конструкції з використанням інтерфейсів.

Системна бібліотека класів мови Java містить класи і пакети, які реалізовують та розширюють базові можливості мови, а також мережеві засоби, взаємодія з базами даних, багатопоточність та багато іншого. Методи, які включені в ці бібліотеки, викликаються JVM під час інтерпретації програми.

У Java всі об’єкти програми розташовані у динамічній пам’яті – купі даних (heap) – і доступні за об’єктними адресами, які зберігаються у стеці (stack). Дане рішення виключило безпосередній доступ до пам’яті, але ускладнило роботу з елементами масивів і зробило її менш ефективною порівняно з програмами на C++. В свою чергу в Java запропонований удосконалений механізм роботи з колекціями, який реалізовує основні динамічні структури даних. Об’єктна адреса мови Java містить інформацію про клас об’єкта, на який вона адресована, тому вона є не тільки адресою об’єкта, а й дескриптором (описом) об’єкта. Наявність дескриптора дозволяє JVM виконувати перевірку на сумісність типів на фазі інтерпретації коду, генеруючи виключення у випадку помилки.

У Java змінена концепція організації динамічного розподілу пам’яті: відсутні способи програмного очищення динамічно виділеної пам’яті за допомогою деструктора, поняття якого виключено з Java. Замість цього реалізована система автоматичного очищення пам’яті - “garbage collector”, яка була виділена за допомогою оператора new.

На відміну від C++, Java не підтримує перевантаження операторів, без знакові цілі, пряме індексування пам’яті. У Java існують конструктори, але немає деструкторів, томущо застосовується автоматичне очищення.

Також, потрібно згадати про безпеку: Java має вбудовану систему безпеки, яка дозволяє запускати програми в безпечному середовищі. Вона має контроль доступу до ресурсів системи і механізми перевірки безпеки, що допомагають запобігти вразливостям і зловмисним діям [1].

В даному проекті для розробки веб сервісу був використаний такий архітектурний стиль як REST API, який базується на принципах HTTP – протоколу. Він дозволяє здійснювати взаємодію між клієнтом за допомогою стандартних HTTP – запитів, таких як GET, POST, PUT та DELETE.

До основних принципів REST API можна віднести [2]:

- клієнт-сервер — клієнти та сервери мають розділяти завдання. Це дозволяє компонентам клієнта та сервера розвиватися незалежно, що, у свою чергу, дозволяє масштабувати систему.
- без стану — зв’язок між клієнтом і сервером має бути без стану. Серверу не потрібно запам'ятовувати стан клієнта. Замість цього клієнти повинні включити всю необхідну інформацію в запит, щоб сервер міг її зрозуміти та обробити.
- багаторівнева система — між клієнтом і сервером може існувати кілька ієрархічних рівнів, таких як шлюзи, брандмауери та проксі. Шари можна прозоро додавати, змінювати, змінювати порядок або видаляти для покращення масштабованості.
- кеш — відповіді від сервера мають бути оголошені такими, що кешуються або не кешуються. Це дозволить клієнту або його компонентам-посередникам кешувати відповіді та повторно використовувати їх для наступних запитів. Це зменшує навантаження на сервер і сприяє підвищенню продуктивності.
- уніфікований інтерфейс — усі взаємодії між клієнтом, сервером і компонентами-посередниками базуються на уніфікованості їхніх інтерфейсів. Це спрощує загальну архітектуру, оскільки компоненти можуть розвиватися незалежно, доки вони виконують узгоджений контракт. Уніфіковане обмеження інтерфейсу далі розбивається на чотири підобмеження: ідентифікація ресурсу, представлення ресурсу, самоописові повідомлення та гіпермедіа, як механізм стану програми або HATEOAS.

- код за запитом — клієнти можуть розширити свої функціональні можливості, завантаживши та виконавши код за запитом.

Програми, які дотримуються цих принципів, вважаються RESTful. Теоретично можна побудувати програму RESTful за допомогою будь-якої мережевої інфраструктури або транспортного протоколу. На практиці використовують функції та можливості Інтернету та використовують HTTP як транспортний протокол.

Для побудови REST API у даному проекті використаний один із найпопулярніших Java фреймворків - Spring. Він надає розширений набір інструментів та бібліотек для розробки високоефективних, масштабованих та підтримуваних програмних додатків. Spring був створений з метою спростити розробку програмного забезпечення та забезпечити високу продуктивність, гнучкість та надійність додатків. Серед багатьох інших речей Spring фреймворк надає модель впровадження залежностей (DI), а також підтримує аспектно-орієнтоване програмування.

Впровадження залежностей, або ж dependency injection – це процес передачі залежностей об’єкту ззовні, замість того, щоб він самостійно їх створював. Це надає кілька переваг. По-перше: dependency injection полегшує тестування, оскілки можна передавати залежності замість реальних об’єктів і контролювати їх поведінку під час тестування. По-друге: він робить код більш гнучким, оскільки залежності можна легко замінити або модифікувати без зміни самого об’єкта.

Аспектно-орієнтоване програмування — це модель програмування, яка реалізує наскрізну логіку або проблеми. Ведення журналів, транзакції, показники та безпека – це деякі приклади проблем, які охоплюють (перетинають) різні частини програми. Ці проблеми не стосуються бізнес-логіки та часто дублюються в програмі. AOP забезпечує стандартизований механізм, який називається аспектом, для інкапсуляції таких проблем в одному місці. Потім аспекти вплітаються в інші об’єкти, щоб наскрізна логіка автоматично застосовувалася до всієї програми. Spring забезпечує реалізацію

AOP на основі Java через свій модуль Spring AOP. Spring AOP не вимагає жодної спеціальної компіляції або змін в ієрархії завантажувача класів. Натомість Spring AOP використовує проксі-сервери для зв’язування аспектів у компоненти Spring під час виконання [2].

Spring дозволяє обирати модулі на основі потреб програми, яка розробляється. Для поставленої задачі було використано такі модулі: Spring Boot, Spring Web MVC, Spring Security та Spring WebSocket.

Spring Boot – це частина модуля Spring Framework, який надає простий спосіб створення самостійних, готових до виконання додатків. Він надає різноманітні функції і зручні інструменти для розробників, що допомагають прискорити процес розробки, спростити конфігурацію та забезпечити високу продуктивність додатків. Він поєднує в собі потужність та гнучкість платформи Spring зі стандартами та конвенціями, що спрощують розробку. Також Spring Boot забезпечує автоматичну конфігурацію багатьох компонентів додатку. Він розпізнає залежності та налаштовує їх за замовчуванням, що зменшує необхідність у власноручній конфігурації. Spring Boot включає вбудований вебсервер, такий як Tomcat, Jetty або Undertow. Він допомагає керувати залежностями проекту, додаючи відповідні конфігураційні файли у збірку Maven або Gradle. Також Spring Boot інтегрується з різними модулями та розширеннями Spring Framework, що дозволяє використовувати широкий спектр функціональності, такої як керування транзакціями, доступ до баз даних, безпека, робота з веб-сервісами та інше.

Spring Web MVC це - частина веб-модуля Spring Framework, яка є популярною технологією для створення веб-додатків. Він заснований на архітектурі модель-подання-контролер і надає багатий набір анотацій і компонентів. Основна ідея Spring Web MVC полягає в розділенні додатку на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Цей підхід дозволяє розмежувати бізнес-логіку, представлення даних та управління потоком додатку. Модель (Model) представляє дані або

стан додатку, представлення (View) відповідає за відображення даних користувачу, а контролер (Controller) обробляє вхідні дані від користувача, взаємодіє з моделлю і відправляє відповідь назад до представлення. Така модель дозволяє легко розподіляти відповідальності між різними компонентами додатку. Spring Web MVC надає широкий набір функціональності для обробки HTTP-запитів, включаючи маршрутизацію, обробку форм, валідацію даних, перехоплення помилок та багато іншого. Він також підтримує різні формати відповідей, такі як HTML, JSON, XML тощо [2].

Spring Security, раніше відома як Acegi Security, є модулем Spring Framework для захисту додатків. Він забезпечує готову інтеграцію з різними системами автентифікації, такими як LDAP, Kerberos, OpenID, OAuth тощо. З мінімальною конфігурацією його можна легко розширити для роботи з будь-якими спеціальними системами автентифікації та авторизації. Фреймворк також реалізує найкращі методи безпеки та має вбудовані функції для захисту від атак, таких як CSRF або Cross Site Request Forgery, а також фіксацію сеансу, та інше [2].

У якості заходів безпеки було обрану систему Json Web Token автентифікації. JSON Web Token - це відкритий стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Кожен токен має цифровий підпис, що створюється за допомогою секрету (з алгоритмом HMAC) або пари відкритих чи приватних ключів за допомогою RSA або ECDSA алгоритму [3].

Spring WebSocket – це ще одна частина модуля Spring Framework призначена для розробки додатків, які використовують двосторонній зв’язок між сервером і клієнтом за допомогою протоколу WebSocket. WebSocket - це протокол, який дозволяє встановити постійне з'єднання між сервером і клієнтом, що дозволяє обмінюватися повідомленнями у реальному часі. Spring WebSocket надає зручний спосіб реалізації WebSocket-комунікації в додатках. Він забезпечує серверну частину WebSocket за допомогою анотацій, що

дозволяють визначити обробники повідомлень, реагувати на події з'єднання та відключення, а також передавати повідомлення між сервером і клієнтом. Spring WebSocket також інтегрується з іншими модулями Spring, такими як Spring MVC, що дозволяє поєднувати WebSocket-комунікацію з традиційним веб-інтерфейсом [2].

1.2 Інструменти для розробки мобільного додатка

Java є найпопулярнішою мовою, яка використовується для розробки програм Android. Пристрої Android не здатні запускати файли .class і .jar. Натомість, щоб підвищити швидкість і продуктивність акумулятора, Android девайси використовують власні оптимізовані формати для скомпільованого коду. Це означає, шо потрібно спеціальні інструменти для перетворення коду у формат Android, щоб розгорнути його на пристрої Android і налагоджувати програму після її запуску. Усе це є частиною такого інструменту як Android SDK.

Android Software Development Kit містить бібліотеки та інструменти, які потрібні для розробки Android додатків:

- SDK платформу (окремо для кожної версії Android);
- SDK інструменти (інструменти для налагодження та тестування, а також інші корисні утиліти. Він також містить набір залежних від платформи інструментів);
- прості приклади проектів;
- Google Play Billing (дозволяє інтегрувати сервіс оплати у додаток);
- додаткові API, недоступні на стандартній платформі;
- найновішу версію документації, яка доступна без доступу до мережі.

Також для розробки було використано одну із спеціальних версії IntelliJ IDEA, а саме: Android Studio. Дане середовище розробки включає в собі Android SDK, а також додаткові інструменти графічного інтерфейсу.

Android Studio дає змогу легко керувати різними конфігураціями додатків, наприклад, різними версіями Android, екранами пристроїв тощо. У

середовищі є підтримка контролю версій додатка за допомогою Git, а також підтримка мультиплатформеного розроблення. Android Studio надає можливість генерувати APK, що є упакованим файлом додатку готовим для розгортання на пристроях або публікації в магазині Google Play. На кінець, середовище постійно оновлюється, що забезпечує доступ до нових функцій, покращень продуктивності та виправлень помилок [4].

У додатку було використано три додаткових API: Maps SDK для Android, Places API, а також Distance Matrix API. Всі ці набори інструментів, надані Google, але кожен з них дозволяє розробникам інтегрувати різну функціональність.

Maps SDK для Android надає доступ до широкого спектру функцій, які дозволяють відображати, взаємодіяти та використовувати карти Google у мобільних додатках. До основних можливостей API можна віднести: відображення карт, переміщення по карті, позначки та маркери, пошук місць, маршрутизація, геолокація, персоналізація карт, візуалізація даних, інтеграція з іншими сервісами.

Places API надає доступ до географічних даних про різні місця та розташування у всьому світі. Цей API дозволяє отримувати інформацію про підприємства, пам’ятні об’єкти, географічні області, адреси та інші місцевості. До можливостей інструменту можна віднести:

- пошук місць (дозволяє здійснювати пошук локацій за ключовими словами, категоріями, радіусом від заданої точки, а також іншими параметрами);
- деталі місця (дозволяє отримувати детальну інформацію про конкретне місце, таку як адреса, координати, години роботи, рейтинг, відгуки користувачів тощо);
- автозаповнення адрес (може надати підказки під час вводу адреси, пропонуючи можливі варіанти, що спрощує процес введення адреси користувачем);

- перегляд зображень (API дозволяє отримувати фотографії, пов’язані з місцями, що допомагає створити більш візуально привабливі додатки та сервіси).

Distance Matrix API – це сервіс, який дозволяє розраховувати відстань та час подорожі між різними місцями на основі географічних координат. API надає розширені можливості для визначення оптимальних маршрутів для різних видів транспорту, таких як автомобілі, велосипеди, пішки та громадський транспорт. За допомогою Distance Matrix API можна вирішувати такі задачі, як визначення найкоротшого маршруту між двома точками, оцінка часу подорожі, враховуючи трафік, а також визначення відстані між багатьма точками одночасно. API повертає результати у зручному форматі, такому як JSON (JavaScript Object Notation), що спрощує обробку та використання отриманої інформації. API широко використовується у різних галузях, таких як транспортна логістика, маршрутизація доставки, туризм та геопросторовий аналіз [5].

1.3 Інструменти для розробки сайту адміністратора

Для створення сайту адміністратора у проекті обрана така мова програмування як TypeScript. Вона зосереджена на тому, щоб розробка програм JavaScript масштабувалася до багатьох тисяч рядків коду. Також мова вирішує масштабну проблему програмування JavaScript, пропонуючи кращі інструменти під час проектування, перевірку під час компіляції та динамічне завантаження модулів під час виконання. Мова TypeScript – це типізований над набір JavaScript, який скомпільовано у звичайний JavaScript. Це робить програми, написані на TypeScript, портативними, оскільки вони можуть працювати майже на будь-якій машині — у веб-браузерах, на веб-серверах і навіть у власних програмах операційних систем, які надають JavaScript API.

Мовні функції TypeScript можна розділити на три категорії залежно від їх зв’язку з JavaScript. Перші два набори пов’язані з версіями специфікації мови ECMA-262 ECMAScript, яка є офіційною специфікацією для JavaScript.

ECMAScipt 5 є основою TypeScript і надає найбільшу кількість функцій мови. Подальші версії специфікації ECMAScript об’єднуються у випуски TypeScript, часто як попередні версії, що містять компіляцію нижнього рівня до старіших версій специфікації. Третій і останній набір мовних функцій включає елементи, які не планують стати частиною стандарту ECMAScript, такі як узагальнення та анотації типів.

Усі додаткові функції TypeScript можуть бути виведені на низку широко підтримуваних версій JavaScript. Оскільки JavaScript має C-подібний синтаксис, він виглядає знайомим для більшості програмістів. Це одна з ключових сильних сторін JavaScript, але вона також є причиною багатьох сюрпризів, особливо в таких областях: прототипове успадкування, управління модулями, область застосування, відсутність типів. Typescript вирішує або полегшує ці проблеми [6].

Також у проекті використано Angular, що є фреймворком такої мови програмування як JavaScript з відкритим кодом, який підтримує Google. Це повний перепис попередника Angular JS. Angular є фреймворком на основі компонентів, а будь-яка програма Angular – це дерево компонентів (продумані представлення) Кожне представлення є екземпляром класів компонентів. Програма Angular має один кореневий компонент, який може мати дочірні компоненти. Кожен дочірній компонент може мати своїх власних дочірніх компонентів і так далі. Фреймворк Angular чудово підходить для розробки односторінкових додатків, де вся сторінка браузера не оновлюється, і лише певна частина сторінки може замінювати іншу, коли користувач переміщується програмою.

Також було використано Angular CLI, що є інструмент для керування проектам Angular протягом усього життєвого циклу програми. Він слугує генератором коду, який значно полегшує процес створення нового проекту, а також процес створення нових компонентів, служб і маршрутів у існуючій програмі [7].

Для відображення карти на сайті адміністратора було використано Maps JavaScript API. Функціонал даного інструменту є схожим на той, що надає Maps SDK для Android.

1.4 База даних

У якості інструменту для зберігання даних у проекті використано об’єктно-реляційну систему керування базами даних Oracle, хмарну базу даних NoSQL Firebase Realtime Database, а також мову програмування SQL для здійснення запитів та внесення змін до бази даних.

База даних Oracle — це сукупність даних, які розглядаються як єдине ціле. Сервер баз даних є ключем до вирішення проблем управління інформацією. Загалом, сервер надійно керує великою кількістю даних у багатокористувацькому середовищі, щоб багато користувачів могли одночасно отримувати доступ до тих самих даних. Все це досягається при високій продуктивності.

Сервер бази даних також запобігає несанкціонованому доступу та забезпечує ефективні рішення для відновлення після збою.

Oracle Database — це перша база даних, розроблена для корпоративних мережевих обчислень, найбільш гнучкий і економічно ефективний спосіб керування інформацією та програмами. Грід-обчислення підприємства створюють великі пули стандартних модульних сховищ і серверів. З такою архітектурою кожна нова система може бути швидко підготовлена з пулу компонентів. Немає потреби в пікових навантаженнях, оскільки потужність можна легко додати або перерозподілити з пулів ресурсів за потреби.

База даних має логічні структури та фізичні. Оскільки фізична та логічна структури розділені, фізичним зберіганням даних можна керувати, не впливаючи на доступ до логічних структур зберігання [8].

База даних Firebase Realtime — це база даних, розміщена в хмарі. Дані зберігаються як JSON і синхронізуються в реальному часі з кожним підключеним клієнтом. База даних Firebase у реальному часі дає змогу

створювати багатофункціональні програми для спільної роботи, надаючи безпечний доступ до бази даних безпосередньо з коду на стороні клієнта. Дані зберігаються локально, і навіть у режимі офлайн події в реальному часі продовжують запускатися, надаючи кінцевому користувачеві оперативне реагування. Коли пристрій відновлює з'єднання, база даних реального часу синхронізує локальні зміни даних із віддаленими оновленнями, які відбулися, коли клієнт був офлайн, автоматично об'єднуючи будь-які конфлікти.

База даних у реальному часі надає гнучку мову правил на основі виразів, яка називається правилами безпеки бази даних у реальному часі Firebase. База даних у реальному часі — це база даних NoSQL, і тому вона має інші оптимізації та функціональність порівняно з реляційною базою даних.

API бази даних у реальному часі розроблено для того, щоб дозволяти лише операції, які можна виконати швидко. Це дає змогу створювати чудовий досвід у реальному часі, який може обслуговувати мільйони користувачів без шкоди для швидкості реагування [9].

SQL — це інструмент для організації, керування та отримання даних, що зберігаються в комп'ютерній базі даних. Назва «SQL» є абревіатурою від Structured Query Language. З історичних причин SQL зазвичай вимовляється як "сиквел", але альтернативна вимова "S.Q.L." також використовується. Як випливає з назви, SQL — це комп'ютерна мова, яка використовується для взаємодії з базою даних. Насправді SQL працює з одним конкретним типом бази даних, яка називається реляційною базою даних. Для отримання даних з бази даних використовується мова SQL, щоб зробити запит. СУБД обробляє запит SQL, отримує запитані дані та повертає їх. Цей процес запиту даних із бази даних і отримання результатів називається запитом до бази даних — звідси й назва Structured Query Language. SQL використовується для керування всіма функціями, які СУБД надає своїм користувачам, зокрема:

- визначення даних (SQL дозволяє користувачеві визначати структуру та організацію збережених даних і зв'язки між збереженими елементами даних);

- отримання даних (SQL дозволяє користувачеві або прикладній програмі отримувати збережені дані з бази даних і використовувати їх);
- маніпулювання даними (SQL дозволяє користувачеві або прикладній програмі оновлювати базу даних, додаючи нові дані, видаляючи старі дані та змінюючи раніше збережені дані);
- управління доступом (SQL можна використовувати для обмеження можливості користувача отримувати, додавати та змінювати дані, захищаючи збережені дані від несанкціонованого доступу);
- обмін даними (SQL використовується для координації спільного використання даних одночасними користувачами, гарантуючи, що вони не заважають один одному);
- цілісність даних (SQL визначає обмеження цілісності в базі даних, захищаючи її від пошкодження через неузгоджені оновлення або системні збої).

Також SQL не є особливо структурованою мовою, особливо в порівнянні з високоструктурованими мовами, такими як C, Pascal або Java. Натомість оператори SQL нагадують речення англійською мовою, доповнені «шумовими словами», які не додають значення оператору, але роблять його більш природним.

У мові SQL є чимало неузгодженостей, а також є деякі спеціальні правила, які запобігають створенню операторів SQL, які виглядають абсолютно законними, але не мають сенсу. Незважаючи на неточності, SQL став стандартною мовою для використання реляційних баз даних. SQL є одночасно потужною мовою та відносно легкою для вивчення [10].

2 РОЗРОБКА СИСТЕМИ ДЛЯ СЕРВІСУ ТАКСІ

2.1 Загальний концепт системи

Проект було розділено на три модуля:

- сервер;
- сайт адміністратора;
- мобільний додаток.

Сервер слугує для зберігання та надання даних для користувачів, або адміністратора. За допомогою сайту відбувається керування всією системою, а також він дає можливість аналізу її роботи. Мобільний додаток надає відповідний функціонал як і для користувачів, так і для водіїв. Уся комунікація між модулями відбувається за допомогою HTTP запитів.

Інформацію у системі було розділено на документи та показники. Документи – це ті дані, які надає сама система. Показники – це інформація, яка отримується в результаті роботи системи. Вони потрібні для аналізу продукту, щоб покращувати роботу системи, або виправляти наявні проблеми.

База даних проекту містить такі таблиці з документами:

- дані про автомобілі (первинний ключ, назва марки, моделі, кількость місць, ідентифікатор класу автомобіля, чи закріплений за ним водій, чи перебуває у замовленні, а також дата реєстрації у системі);
- дані про клас автомобіля (первинний ключ, назва класу, мінімальний досвід водія, який потрібен для того, щоб мати змогу використовувати автомобіль даного класу, мінімальна кількість кілометрів, мінімальний середній рейтинг водія та ціна за кілометр);
- дані про ціну автомобіля за кілометр відносно періоду дня (первинний ключ, ідентифікатор класу автомобіля, ціна зранку, вдень, увечері, вночі);
- дані про водіїв (первинний ключ, ідентифікатор автомобіля, ідентифікатор прізвища, імені, по батькові, номер телефону водія,

досвід роботи, зарплата, дата реєстрації в системі, статут резюме, а також ім’я користувача);

- дані про характеристики кожного звання (первинний ключ, назва звання, мінімальна кількість замовлень, яка потрібна для досягнення даного рангу, мінімальна кількість коментарів, протягом якого періоду доступна знижка на поїздку, значення знижки і чи є елітним дане звання);
- дані про характеристики елітного звання (первинний ключ, ідентифікатор базового звання, ідентифікатор класу автомобіля, протягом якого періоду доступна безкоштовна поїздка та їх кількість);
- дані про користувачів (первинний ключ, у якості унікального ідентифікатора слугує ім’я користувача, пароль, роль користувача, а також ідентифікатор звання).

Також база даних містить таблиці з показниками, зокрема:

- дані про адреси (первинний ключ, назва, кількість разів). Даний показник надає змогу аналізувати найпопулярніші адреси і таким чином, наприклад, адміністратор може давати вказівки для водіїв, щоб вони більше часу знаходились біля даних адрес;
- показник середньої оцінки водія за день (первинний ключ, ідентифікатор водія, дата та оцінка). Дає змогу робити висновки про якість роботи того чи іншого водія;
- показник рекомендацій автомобілів для водіїв (первинний ключ, дата коли водій досяг нового класу автомобіля, ідентифікатор водія, ідентифікатор автомобіля, статус чи прийняв водій нове авто чи ні). Даний показник дозволяє водіям мати кар’єрний ріст;
- показник улюблених адрес (первинний ключ, ідентифікатор користувача, назва адреси). Система надає користувачеві обирати свої улюблені адреси, щоб полегшити їх пошук під час замовлення;

- показник улюблених водіїв (первинний ключ, ідентифікатор водіїв, ідентифікатор користувача). Система надає користувачеві обирати своїх улюблених водіїв, щоб вони могли замовити автомобіль у конкретного водія;
- показник доходу кожного автомобіля за день (первинний ключ, ідентифікатор автомобіля, дата, витрати на паливо, заробіток);
- показник витрат кожного автомобіля за день (первинний ключ, дата, тип витрат, значення витрат, ідентифікатор автомобіля);
- показник кількості подоланих кілометрів водієм за день (первинний ключ, дата, кількість кілометрів, ідентифікатор водія);
- показник завантаженості кожного водія у різний період дня (первинний ключ, ідентифікатор водія, дата, кількість поїздок зранку, вдень, увечері, вночі);
- дані про замовлення (первинний ключ, ідентифікатор водія, адреса відправлення, адреса доставки, ціна, дата, оцінка роботи водія, кількість кілометрів, ім’я користувача, ідентифікатор користувача, коментар користувача, чи використав користувач знижку, чи використав користувач безкоштовну поїздку);
- інформацію про досягнення користувачем того чи іншого звання (первинний ключ, дата, ідентифікатор користувача, ідентифікатор звання, чи використано поїздку із знижкою, дедлайн знижки);
- інформацію про досягнення користувачем елітного звання (первинний ключ, дата, ідентифікатор користувача, ідентифікатор елітного звання, ідентифікатор класу автомобіля, кількість доступних безкоштовних поїздок для окремого класу автомобіля, дедлайн безкоштовних поїздок);
- інформацію про користувачів, які подали заяву на отриманні знижки для військових (первинний ключ, дата, ідентифікатор користувача, фото документа, статус, значення знижки).

Так як у якості заходів безпеки було обрано токен автентифікацію, то у базі дані також наявна таблиця де зберігаються токени користувачів, для того, щоб контролювати їх дійсність. Також проект надає різний функціонал в залежності від ролі користувача. Діаграма загального концепту проекту та схеми бази даних наведена у Додатку А.

2.2 Опис розробки серверної частини

Сервер має трирівневу архітектуру, яка передбачає наявність наступних компонент програми: клієнтський інтерфейс, який підключений до сервера застосунків, який в свою чергу підключений до серверу бази даних.

Для проектування засобу доступу до бази даних використовувався такий патерн проектування як DAO. Він допомагає приховати від програми всю складність виконання операцій CRUD у базовому механізмі зберігання. Це дозволяє усім шарам розвиватися окремо, нічого не знаючи один про одного. Для його реалізації було створено загальний інтерфейс, який імплементували усі сутності, що використовувались як контейнер для зберігання інформації з бази даних. Також був реалізований клас для ініціалізації з’єднання з базою даних і клас, де зберігаються запити до неї.

За допомогою СУБД Oracle та мови PL/SQL були реалізовані функції та процедури. Обидва на вході приймають параметри, які потрібні для обчислення, але різниця між ними полягає в тому, що функція повертає значення - результат її роботи, а процедура цього не виконує. Розроблена база даних містить такі процедури:

- SETADDRESS (приймає у якості вхідних даних назву адреси і використовується для того, щоб у випадку, якщо у таблиці адреси уже вона наявна, то збільшити лічильник на одиницю);
- SETWORKLOADDRIVERS (приймає у якості вхідних даних ідентифікатор водія і поточну дату, а використовується для того, щоб у таблиці завантаженість водіїв додати першу виконану поїздку у певний період дня);

- UPDATEDRIVERRATING (приймає ідентифікатор водія, поточну дату, та нову оцінку і використовується для того, щоб оновити середню оцінку водія за день);
- UPDATEWORKLOADDRIVERS (приймає у якості вхідних даних ідентифікатор водія, а також поточну дату і використовується для оновлення кількості виконаних поїздок у певний період дня у таблиці завантаженості).

Також база даних має такі функції:

- GETCARCLASSFORDRIVER (приймає на вхід значення досвіду водія та повертає ідентифікатор рекомендованого класу автомобіля);
- GETCARFORDRIVER (приймає на вхід значення досвіду водія та використовуючи функцію GETCARCLASSFORDRIVER повертає ідентифікатор рекомендованого автомобіля);
- GETCARID (приймає ідентифікатор водія та повертає ідентифікатор автомобіля, який йому належить);
- GETMAINTANCECOSTS (приймає ідентифікатор водія і поточну дату, а повертає витрати автомобіля);
- GETNEWRATING (приймає оцінку водія, його ідентифікатор, а також поточну дату і в результаті повертає нову середню оцінку водія за день);
- ISDRIVERALREADYEXISTS (приймає ідентифікатор водія, потону дату та повертає кількість виконаних замовлень даним водієм у поточний день).

Також було використано такий інструмент як тригери. Вони дозволяють запрограмувати перед виконанням запиту , або після, автоматичне виконання будь-якої функції чи процедури. У системі наявні такі тригери:

- DRIVER_INSERT_TRIGGER (спрацьовує перед додаванням нового водія у таблицю для того, щоб обрати рекомендований автомобіль);

- MAINTENANCE_COSTS_INSERT_TRIGGER (спрацьовує перед додаванням нових витрат для автомобіля для того щоб оновити стовбець витрати у таблиці доходу);
- ORDERS_INSERT_TRIGGER (спрацьовує перед додавання у базу даних нового замовлення для того, щоб оновити усі показники в системі).

У якості автоматичних генераторів ідентифікаторів було використано такий інструмент як послідовності (sequences), які дають змогу налаштовувати мінімальне значення від якого починається послідовність, крок збільшення, максимальне значення та інше).

Для розділення логіки між доступом до бази даних та клієнтським інтерфейсом було створено відповідні сервіси, які за допомогою впровадження залежності ін’єктували DAO класи.

Також було реалізовано сервіс для обрахунку досвіду водія, автентифікації, визначення періоду дня за часом, рекомендацій автомобілів для водіїв, визначення зарплатні, генерації JWT токенів, виходу із системи, керування системою бонусів, а також web socket сервіс для комунікації водія з користувачем.

Сервіс обрахунку досвіду водія використовує дату реєстрації та поточну і таким чином після кожного виконаного замовлення оновлює його. Сервіс визначення періоду дня за часом отримує на вході час та повертає період дня за такою логікою: від 0 до 6 період має назву – ніч, від 6 до 12 – ранок, від 12 до 18 день, від 18 до 24 - вечір.

Сервіс зарплатні під час реєстрації нового водія отримує його досвід і в залежності від значення, якщо менше одного року, то водій отримує мінімальну зарплату, у іншому випадку досвід множиться на мінімальну зарплатню та коефіцієнт, який визначає адміністратор.

Сервіс рекомендації нового автомобіля для водія працює таким чином: після кожного виконаного замовлення сервіс перевіряє показники водія і,

якщо вони задовольняють наступний клас, сервер надсилає повідомлення водію, щоб він підтвердив, або відхилив пропозицію.

Як вже зазначалось, у якості заходів безпеки було використано Spring Security та Jason Web Token автентифікацію. Спочатку для цього було створено відповідні конфігураційні класи: ApplicationConfig, SecurityConfiguration.

ApplicationConfig використовується для зберігання даних користувачів під час їхньої автентифікації.

SecurityConfiguration слугує основним конфігураційним класом. В ньому було налаштовано доступ до даних у залежності від ролі користувача, який робить запит. Користувач із роллю адміністратора можуть отримувати інформацію про всі документи та показники. Звичайні користувачі та водії мають тільки доступ до власних даних. Також було налаштовано, що кожен хто намагається отримати інформацію від сервісу повинні бути обов’язково авторизовані.

У Spring Security кожен запит проходить через так званий фільтр. Для JWT автентифікації було створено JwtAuthenticationFilter, який перевіряє кожен HTTP – запит на наявність заголовка Authorization і дійсність наданого токена. Якщо токен не дійсний, або його період дії скінчився, у відповідь сервер надсилає інформацію про відповідну помилку.

Сервіс автентифікації надає функцію реєстрації як і користувачем так і водієм, функцію перевірки дійсності токена доступу, функцію зміни пароля, або імені, а також функцію оновлення токена, якщо він закінчився. Токен доступу, або ж access token має період дії 24 години, а токен оновлення (refresh token) дійсний протягом одного місяця, що було вказано у конфігураційному файлі application.yml. Під час реєстрації сервер повертає користувачеві новий access token та refresh token, а при автентифікації сервер перед тим як видати новий токен доступу у таблиці токенів, позначає усі токени користувача недійсними у якості заходів безпеки. Коли користувач вирішує вийти із свого

облікового запису, усі його токени також позначаються недійсними. Паролі користувачів шифруються за допомогою bcrypto алгоритму.

Розроблений проект надає своєрідну систему бонусів, яка б заохочувала користувачів користуватися даним сервісом. Для цього було введено так звану систему звань, аналогічну до козацьких. Самі звання розділені на базові та елітні. Базові дають можливість користувачам замовити автомобіль із знижкою і у залежності від звання, знижка має різний розмір, а також період використання. Елітні звання дозволяють користувачам використати безкоштовну поїздку. У залежності від звання, кількість поїздок автомобілем різного класу мають різні розміри та період використання. Адміністратор має можливість змінювати будь-які характеристики кожного рангу за допомогою сайту.

Сервіс, який відповідає за звання працює таким чином: після кожного замовлення перевіряє показники користувача і у разі, якщо досягнуто базовий, або елітний ранг у таблицю додається інформація про це. Якщо період дії знижки або безкоштовної поїздки досяг дедлайну, то система автоматично оновлює нову дату за допомогою інструменту, який надає Spring, а саме Scheduling. Він дає змогу встановлювати заплановані функції на будь-який час. При проектуванні системи було вирішено, що оновлення даних про досягнення користувачем того чи іншого звання відбуватиметься о 00:00:00, кожного дня.

Зважаючи на сьогоденні події, у системі наявний бонус для військових у вигляді знижки на всі поїздки. Користувач повинен надіслати документ, який підтверджує, що людина дійсно є військовим. Адміністратор переглядає інформацію та приймає рішення про зарахування бонусу чи ні.

Комунікація між водієм та користувачем відбувається за допомогою протоколу web socket. Spring дає можливість реалізувати дану технологію завдяки Spring WebSocket. Для цього спочатку було створено конфігураційний клас WebSocketConfig. У ньому було встановлено префікс доступу до сервера, користувача, а також кінцеві точки для Stomp.

Web Socket сервіс містить методи, які слугують для перенаправлення повідомлення від користувача до водія, або навпаки. Для того, щоб сервер зміг прийняти повідомлення використовують анотацію, а саме @MessageMapping і у якості аргументу встановлюють шлях по якому можна звертатися до нього.

Останнім, найвищим шаром у архітектурі сервера є контролери, у яких за допомогою інструменту Spring MVC, за допомогою анотацій встановлюються шлях до відповідних даних, які отримуються завдяки HTTP – запитам.

Так як сервер побудований по архітектурі REST, шляхи доступу до даних були спроектовані таким чином:

- доступ до документів відбувається по url – “/api/v1/documents/… ”,
- доступ до показників по url – “/api/v1/indicators/…”,
- доступ до даних про бонуси по url – “/api/v1/bonuses/…”,
- доступ до даних користувача, які потрібні для додатку по url – “/api/v1/user-app/…”,
- доступ до даних водія, які потрібні для додатку по url – “/api/v1/driver-app/…”.

Даний шар відділяє всю логіку сервісів від обробки запитів, тобто при зміні якогось із сервісів не потрібно змінювати логіку у контроллерах. Деякі фрагменти коду наведені у додатку Б

2.3 Опис розробки мобільного додатка

Першим кроком при створенні додатку було налаштування головного конфігураційного файлу, який є обов’язковим для всіх Android проектів та має назву Manifest. Файл має формат xml. Спочатку у ньому було задекларовано усі потрібні дозволи користувача за допомогою тегу “user-permission” та атрибуту “android:name”:

- ACCESS_COARSE_LOCATION (Дозволяє отримувати доступ до приблизного місцезнаходження);

- ACCESS_FINE_LOCATION (Дозволяє отримувати доступ до точного місцезнаходження);
- INTERNET (Дозволяє додатку відкривати мережеві сокети);
- ACCESS_LOCATION_EXTRA_COMMANDS (Дозволяє отримувати доступ до додаткових команд постачальника місцезнаходження);
- ACCESS_NETWORK_STATE (Дозволяє отримувати доступ до інформації про мережі);
- READ_EXTERNAL_STORAGE (Дозволяє додатку читати із зовнішньої пам'яті).

Далі налаштовувався такий тег як "application". У ньому було задекларовано основні параметри додатка: іконку додатка, назву - Energy Taxi, орієнтацію екрана – портретна, загальну візуальну тему додатку, чи програма має намір використовувати чистий текстовий мережевий трафік.

Також у даний тег були задекларовані усі діяльності, які використовує додаток. Діяльність, або ж Activity – це єдина цілеспрямована річ, яку може зробити користувач. Майже всі дії взаємодіють з користувачем, тому клас Activity піклується про створення вікна, у якому можна розмістити інтерфейс користувача за допомогою setContentView(View).

Activity реалізують два методи: onCreate(Bundle) та onPause(). У onCreate(Bundle) ініціалізується діяльність. Тут зазвичай викликається setContentView(int) ресурс макета, який визначає інтерфейс користувача та використовує findViewById(int) для отримання віджетів у цьому інтерфейсі користувача, з якими потрібно взаємодіяти програмним шляхом. onPause() призупиняє активну взаємодію з діяльністю. У цьому стані діяльність усе ще видно на екрані.

Другим кроком було створення усіх необхідних діяльностей. Під час проектування додаток умовно було розділено на дві частини: одна, яка надає функціонал для користувача, інша - для водія.

Для користувача додаток надає такі функції:

- реєстрації та авторизації;

- надсилання резюме;
- перегляду даних власного профілю і зміни пароля та імені;
- замовлення автомобіля;
- перегляд історії поїздок;
- перегляд списку улюблених водіїв та адрес;
- перегляд інформації про систему бонусів та статистики користувача;
- перегляд контактів для того, щоб звернутися до адміністратора у разі виникнення питань чи проблем.

Для водія додаток надає такі функції:

- реєстрації та авторизації;
- підтвердження або відхилення замовлення;
- перехід із статусу у мережі у статус не в мережі;
- перегляд історії поїздок;
- перегляду даних власного профілю і зміни пароля та імені;
- підтвердження або відхилення нового рекомендованого автомобіля.

Макет для усіх Activity створювався за допомогою xml. У якості тегів слугують елементи графічного інтерфейсу кожен з яких містить свої власні параметри для налаштування.

Так як у додатку часто потрібно надавати список з даними, то були спроектовані спеціальні класи адаптери, які дають можливість у зручний спосіб представляти ту чи іншу інформацію. Для кожного адаптера створювався відповідний макет та клас holder, який ініціалізовував усі елементи. У якості слухача події натискання на елемент списку використовувався інтерфейс SelectListener з відповідними методами для імплементації.

Першою Activity під час запуску додатку є WelcomeActivity (рисунок 2.1). У ній перевіряється дійсність токена доступу, наявність увімкненого GPS та чи додаток має доступ до геоданих смартфону.

Перевірка токена доступу відбувається за допомогою HTTP – запиту до сервера. Для цього використано таку бібліотеку як Retrofit. У проекті було

створено клас RetrofitService, який відповідає за ініціалізацію об’єкта Retrofit, передаючи ім’я хоста сервера та тип даних, у якому передаватимуться дані, а саме json. Також було реалізовано інтерфейс API, де задекларовано усі методи для здійснення HTTP – запиту до серверу. Якщо період дії токена скінчився, то додаток автоматично робить запит на його оновлення і у разі, якщо токен оновлення також не дійсний, відбувається перехід на сторінку авторизації.

Перевірка на наявність увімкненого GPS відбувається завдяки класу LocationManager, який включений у Android SDK. За допомогою класу ContextCompat та його статичному методу checkSelfPermission відбувається перевірка на те чи додаток має доступ до геоданих. Для зберігання токенів у додатку використовується сховище SharedPreferences, яке дає змогу зберігати у вигляді ключ – значення будь-яку інформацію. Якщо у WelcomeActivity при перевірці виявляється, що сховище не містить даних, то користувача автоматично перекидає на сторінку авторизації.

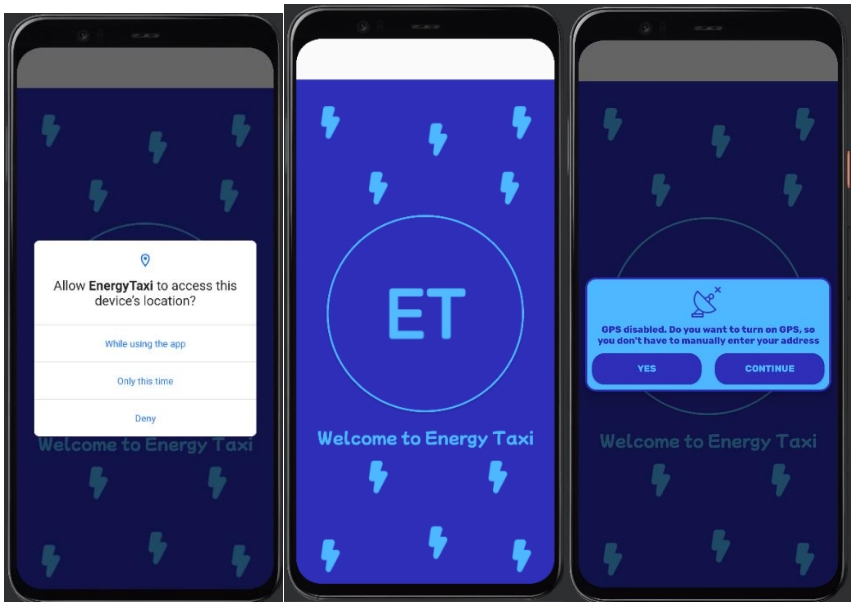


Рисунок 2.1 – Графічний інтерфейс WelcomeActivity

Для реєстрації користувача було створено діяльність UserRegistryActivity (рисунок 2.2 а). У ній наявні два поля типу EditText для імені користувача та пароля. Також реалізовано кнопку типу Button для реєстрації і два TextView, які дають змогу користувачу переходити на сторінку авторизації, або подання резюме. Ім’я користувача повинно мати не менше

чотирьох букв та не більше шістнадцяти. Пароль не повинен бути менше восьми букв та не більше двадцяти. Після успішної проходження реєстрації сервер повертає токени, ідентифікатор користувача і у сховище записуються всі ці дані разом із ідентифікатором звання та відбувається перехід до діяльності під назвою MainActivity.

Activity авторизації має назву UserLoginActivity (рисунок 2.2 б). Її функціонал є схожим до UserRegistryActivity, а відмінності полягають у тому, що сторінка авторизації має елемент типу CheckBox, який дозволяє входити у систему як водієм так і користувачем. Також Activity має відповідний TextView для переходу до UserRegistryActivity. Для подачі резюме на посаду водія було створено DriverResumeActivity (рисунок 2.2 с).

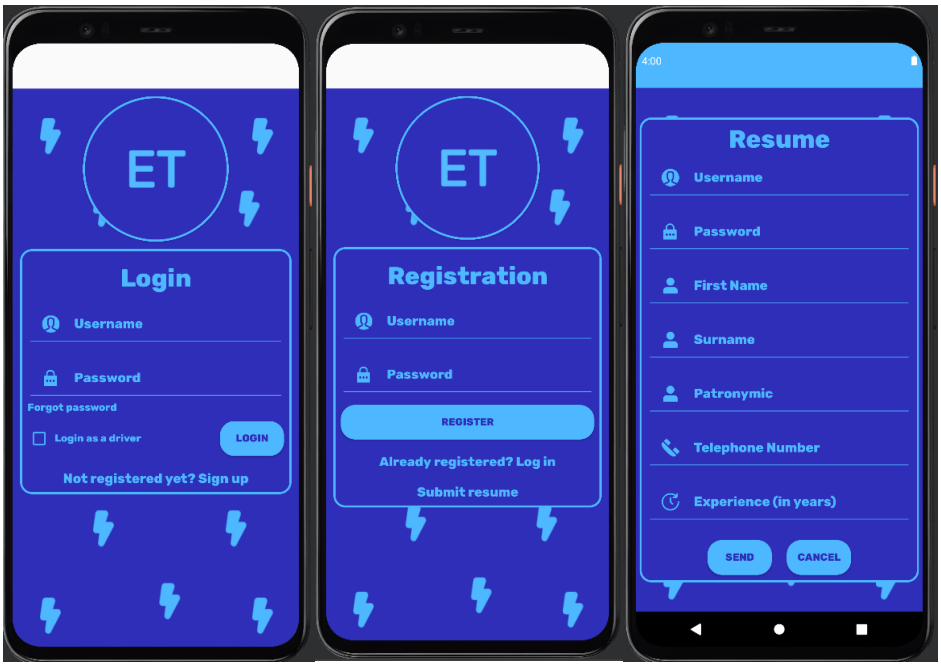


Рисунок 2.2 – Графічний інтерфейс: а – UserLoginActivity, б – UserRegistryActivity, с - DriverResumeActivity

Головне меню додатка розташоване у MainActivity. Воно складається з навігаційної панелі за допомогою якої можна переходити до різних Activity, google map, двох власно створених Fragment, SearchAddressesFragment та SearchAddressOnMapFragment, один з яких слугує для обрання та відображення обраних адрес, а також улюбленого водія, інший слугує для

вибору адреси за допомогою карти, а також кнопка для пошуку локації користувача, яка працює, якщо увімкнений GPS.

Для впровадження google карти було використано Maps SDK для Android. Завдяки зовнішньому ресурсу було створено власний стиль для карти. Для її відображення використовувався такий елемент як Fragment. Fragment представляє багаторазову частину інтерфейсу користувача. Він визначає та керує своїм власним макетом, має власний життєвий цикл і може обробляти власні події. SearchAddressesFragment побудований за допомогою CardView, який у свою чергу має RelativeLayout, що містить такі елементи:

- поле типу TextInputLayout для введення адреси відправлення (якщо у смартфоні увімкнений GPS додаток автоматично встановлює його поточну адресу де він знаходиться);
- поле типу TextInputLayout для введення адреси доставки
- дві кнопки типу ImageButton для обрання адреси із списку улюблених;
- дві кнопки типу ImageButton для переходу до іншого Fragment, щоб обрати потрібну адресу за допомогою карти;
- поле типу TextInputLayout (слугує підказкою для користувача, що він може обрати свого улюбленого водія і у разі, якщо користувач обрав, у полі відображається ім’я та прізвище водія).

Під час переходу до MainActivity додаток робить запити до сервера для того, щоб отримати список улюблених адрес та водіїв відповідного користувача. Адреси потрібні для того, щоб користувач мав змогу швидко обрати адресу не вписуючи її вручну. Також адреси можна обирати за допомогою Places API, який, за допомогою функції autocomplete надає користувачеві підказки (рисунок 2.3).

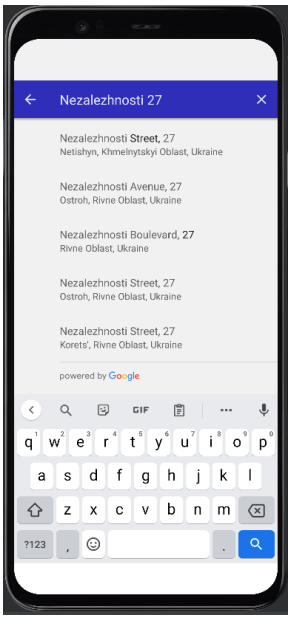


Рисунок 2.3 – Інструмент Place API для введення адреси вручну

Дані про улюблених водіїв використовуються для того, щоб користувач міг замовити автомобіль у відповідного водія. При ініціалізації карти додаток використовуючи Firebase Realtime Database збирає інформацію про водіїв, а саме їхні координати і статус, який може бути трьох типів: у мережі (Online), не в мережі (Offline), у процесі виконання замовлення (In order). У залежності від статусу на карті кожен водій буде зображуватися різними маркерами. Ті, хто не у мережі на карті не відображаються. Маркер улюбленого водія має додаткову позначку для того, щоб користувач мав змогу натиснувши на нього, обрати його для замовлення.

Для переходу до сторінки замовлення авто, користувачу потрібно обрати обидві адреси, після чого додаток за допомого Distance Matrix API перевіряє відстань між ними і якщо вона менше одного кілометра, або більше тридцяти додаток не дозволить продовжити далі. Також у MainActivity наявна функція у вигляді ImageButton для виходу із облікового запису.

SearchAddressOnMapFragment побудований так само як SearchAddressesFragment, але RelativeLayout містить інші елементи:

- поле типу TextView (слугує підказкою для користувача, що він може обрати улюблену адресу натиснувши на відповідне місце на карті);
- кнопка типу Button для повернення до SearchAddressesFragment;

- кнопка типу Button для підтвердження обраної адреси;
- кнопка типу LikeButton для того, що користувач міг додати відповідну адресу до своїх улюблених.

Обрання адреси за допомогою карти виглядає такими чином: при натисканні на карту з'являється маркер, який показує точну локацію, яку обрав користувач і назва адреси відображається у відповідному полі, щоб користувач міг бути впевненим у своєму виборі (рисунок 2.4).

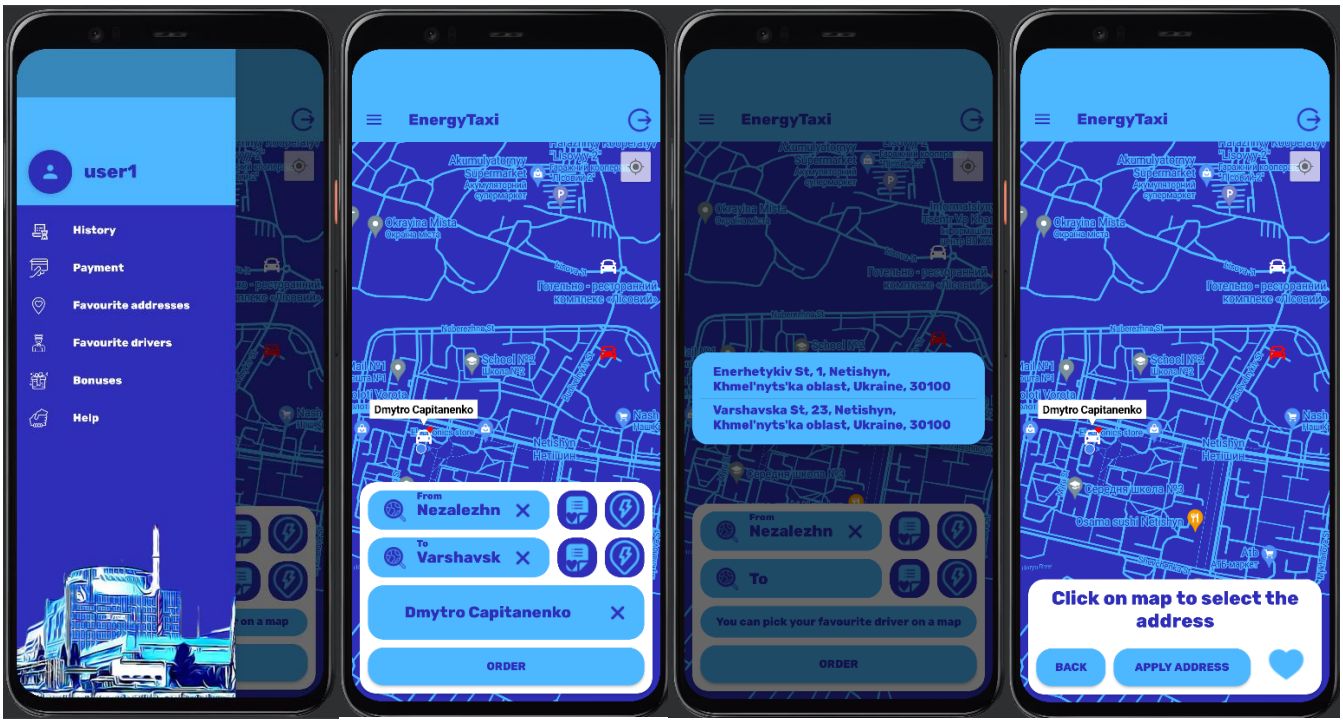


Рисунок 2.4 – Графічний інтерфейс MainActivity

MakeOrderActivity - це наступна Activity, до якої переходить користувач після MainActivity для того, щоб обрати клас автомобіля і бонуси, якщо вони наявні, після чого здійснити пошук водія (рисунок 2.5). При ініціалізації MakeOrderActivity додаток робить запити до сервера. Один для того, щоб надати користувачеві ціни на відповідні класи автомобілів, інший, щоб перевірити наявність бонусів і у тому випадку, якщо вони є, можливість ними скористатися. Якщо до цього було обрано улюбленого водія, то користувачеві не треба нічого вибирати, лише бонуси. Перед пошуком водія потрібно обрати один із класів автомобілів за допомогою такого елемента як RadioButton, після чого ввести своє ім'я та номер телефону у відповідні поля, щоб водій у разі,

якихось проблеми міг зв’язатися із користувачем. При натисканні на кнопку замовлення додаток здійснює пошук усіх доступних водіїв та обирає найближчого до користувача за допомогою координат та формули гаверсинуса для відстані між двома точками на сфері:

$$d = 2rsin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right), \text{де } r$$

$$- \text{радіус сфери,} \qquad \phi_1 \text{ і } \phi_2 - \text{широти двох точок, а } \lambda_1 \text{ і } \lambda_2$$

$$- \text{довготи}$$

Якщо додаток знайшов водія, але він не прийняв замовлення, то під час наступного пошуку система не буде його враховувати, а вибере іншого найближчого водія.

Так як для комунікації водія з користувачем було вирішено використовувати протокол web socket, то для додатку встановлено бібліотеку spring-websocket та spring-messaging. Для реалізації вищенаведеного було створено конфігураційний клас WebSocketClient, у якому реалізовано метод для з’єднання з сокетом. Під час натискання на кнопку замовлення, користувач підписується на відповідний mapping для, того, щоб отримати відповідь від водія, а також посилає свої дані для нього ж. Коли додаток знаходить найближчого водія у користувача з’являється діалогове вікно яке повідомляє, що очікується відповідь. Після того як її отримано, якщо вона позитивна, то відбувається перехід до наступної сторінки, а саме SetRatingActivity і відписка від mapping, якщо ні, користувачу потрібно повторити пошук.

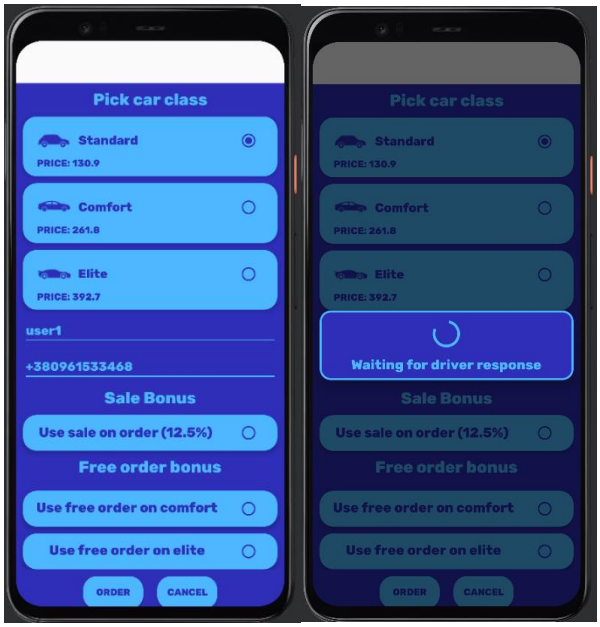


Рисунок 2.5 – Графічний інтерфейс MakeOrderActivity

SetRatingActivity слугує для того, щоб користувач міг залишити коментар та оцінку роботи водія (рисунок 2.6). Але цю дію він може виконати лише коли водій підтвердить те, що замовлення виконане. Під час ініціалізації SetRatingActivity додаток робить запит до серверу, щоб отримати дані про те чи є водій улюбленим, та підписується на інший mapping, щоб очікувати підтвердження від водія про виконане замовлення, адже саме після цього стає доступна функція оцінки роботи. Замовлення рахується завершеною як тільки користувач поставив бал.

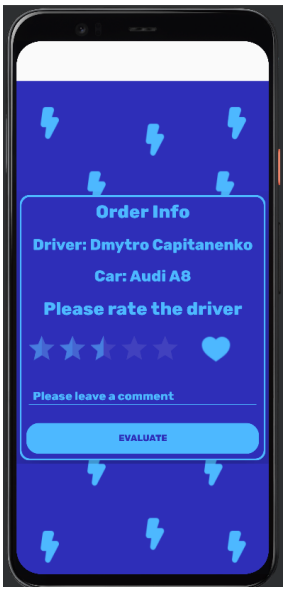


Рисунок 2.6 – Графічний інтерфейс SetRatingActivity

Для перегляду історій замовлення користувача було створено UserOrderHistoryActivity (рисунок 2.7 а). При переході до даної сторінки додаток робить PUT запит до сервера передаючи у якості параметрів ідентифікатор користувача. Після отримання інформації, вона передається адаптеру UserOrderHistoryAdapter, який у свою чергу передається класу RecyclerView, що слугує інструментом для відображення списку даних.

Функція перегляду улюблених водіїв реалізована у FavouriteDriversActivity (рисунок 2.7 б). Під час переходу до даної сторінки, додаток також робить PUT запит до сервера, передаючи у якості параметрів ідентифікатор користувача. Після отримання інформації, вона передається адаптеру FavouriteDriversAdapter, який у свою чергу передається класу RecyclerView. Адаптер містить додаткову функцію видалення водія, яка відправляє відповідний DELETE запит до сервера передаючи у якості параметрів ідентифікатор водія і користувача. Функція перегляду улюблених адрес реалізовано у FavouriteAddressesActivity (рисунок 2.7 с) і відрізняється від FavouriteDriversActivity лише тим, що замість водіїв у якості даних слугують адреси.

Рисунок 2.7 – Графічний інтерфейс: а - UserOrderHistoryActivity, б - FavouriteAddressesActivity, с - FavouriteDriversActivity

Для того, щоб користувач міг переглянути свою статистику та детальну інформацію про систему звань, було створено RanksInfoActivity (рисунок 2.8 а). Після того як користувач перейшов до даної Activity, додаток по черзі робить три GET запита для того, щоб отримати статистику користувача, характеристики кожного звання, інформацію про кількість безкоштовних поїздок автомобілем різного класу та їх дедлайн, а також про наявність знижки і її період дії.

Для відображення інформації про звання використовувався такий елемент як CardView та RelativeLayout, який дозволяє робити адаптивні Activity. HorizontalScrollView був задіяний для того, щоб зробити CardView з статистикою про користувача адаптивним. Також у даній Activity є функція для завантаження документа для того, щоб отримати знижку на всі наступні поїздки. Для цього був використаний такий клас як ActivityResultLauncher, який дозволяє користувачеві відкрити зовнішнє сховище смартфону і обрати будь-який файл після чого завантажити його на сервер, де він буде перевірений адміністратором. Якщо документ користувача був прийнятий, то кнопка типу Button буде виконувати функції не завантаження, а видалення.

Сторінка профілю користувача була реалізована у UserProfileActivity (рисунок 2.8 б). У ній наявний такий функціонал як зміна ім’я та пароля. Для зміни пароля у діалоговому вікні потрібно ввести спочатку старий, а потім новий, після чого додаток відправить запит на зміну пароля, для редагування імені потрібно лише вписати нове і все. Також у UserProfileActivity відображена інформація про поточне звання, яке має користувач.

Усі контактні дані, за допомогою яких користувач може зв’язатися з сервісом, задекларовані у такій Activity як HelpActivity (рисунок 2.8 с). З функціоналу у ній наявна кнопка типу Button для повернення до MainActivity.

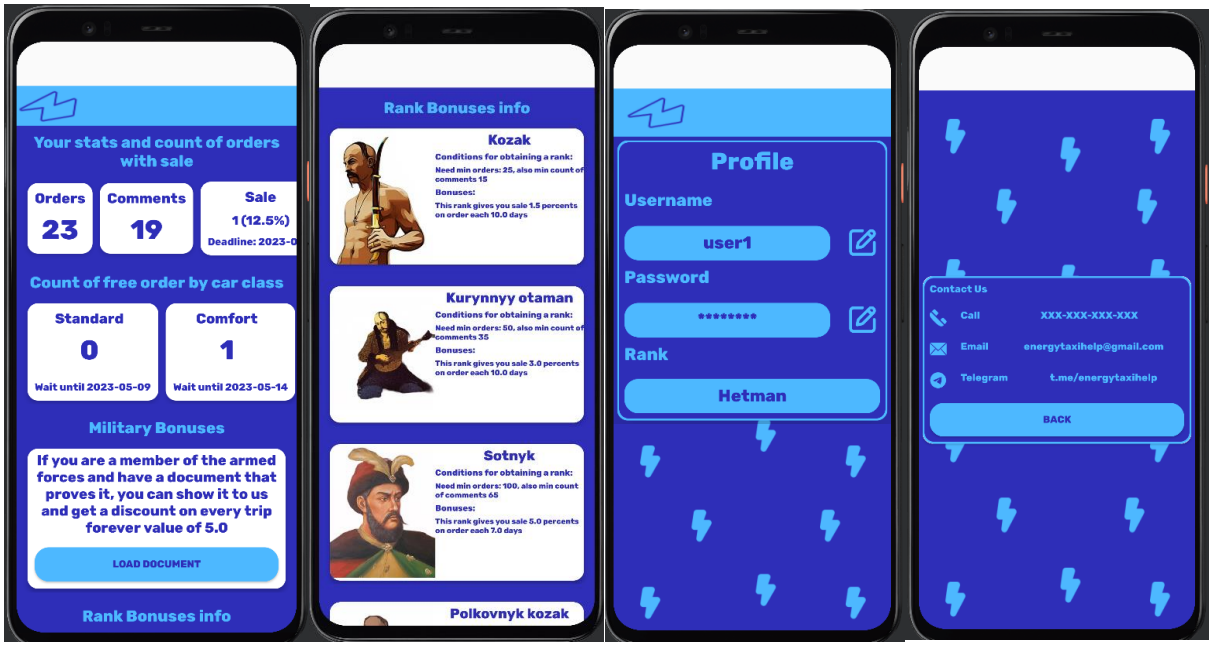


Рисунок 2.8 – Графічний інтерфейс: а - RanksInfoActivity, б – UserProfileActivity, с - HelpActivity

Головне меню для водія було реалізовано у DriverMenuActivity (рисунок 2.9). Під час його ініціалізації додаток відправляє запит до сервера для того, щоб отримати історію поїздок водія і показити її йому. При натисканні на замовлення у списку додаток переходить до OrderInfoActivity. Також програма з’єднується з Firebase Realtime Database для того, щоб передати поточні координати водія. Для прослуховування зміни локації було створено сервіс DriverLocationService, який імплементував інтерфейс LocationListener, а саме метод onLocationChanged. Також у DriverMenuActivity наявний елемент типу Switch, який слугує для того, щоб водій міг повідомляти системі чи він у мережі чи ні. Інші функції, які доступні в DriverMenuActivity це перехід до сторінки профілю водія – DriverProfileActivity, яка по функціональності не відрізняється від UserProfileActivity, виходу із системи, а також функція переходу до DriverCarRecommendationsActivity. Кожного разу, коли водій вмикає статус у мережі, він підписується до mapping, щоб очікувати нове замовлення. Коли приходить запит на замовлення, то відображається діалогове вікно, де водій може переглянути всю інформацію та прийняти, або

відхилити пропозицію. Якщо водій приймає замовлення, додаток переходить до CurrentOrderActivity.

Рисунок 2.9 – Графічний інтерфейс DriverMenuActivity

OrderInfoActivity відображає усю детальну інформацію про конкретне замовлення. При ініціалізації він дістає інформацію із сховища Intent да відображає її (рисунок 2.10 а).

DriverCarRecommendationsActivity було створено для того, щоб водій міг перевіряти чи не надійшло йому пропозиції щодо зміни автомобіля кращого класу. Activity містить два Fragment: CarRecFragment та EmptyCarRecFragment. Якщо у водія немає пропозицій, то відображається EmptyCarRecFragment у іншому випадку CarRecFragment, який містить дві функції: прийняти автомобіль або відхилити. (рисунок 2.10 б)

CurrentOrderActivity слугує для відображення інформації про поточне замовлення та має одну функцію – підтвердити, що замовлення виконане (рисунок 2.10 с). Деякі фрагменти коду наведені у додатку В.

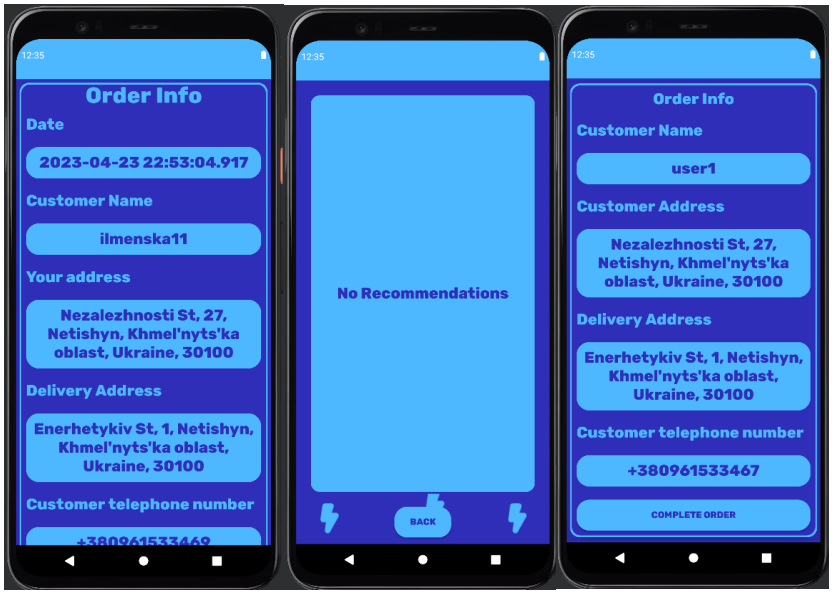


Рисунок 2.10 – Графічний інтерфейс: а - OrderInfoActivity, б - DriverCarRecommendationsActivity, с – CurrentOrderActivity

2.4 Опис розробки сайту адміністратора

Першим кроком при розробці сайту, було створення сторінки авторизації (рисунок 2.11). Для цього за допомогою Angular CLI згенеровано два компонента login і logout та два модуля маршрутизації для входу та виходу із системи. Сторінка авторизації містить два поля для введення логін та пароля і кнопку входу.

Рисунок 2.11 – Сторінка авторизації

Для автентифікації користувача реалізовано AuthenticationService, який проводить валідацію даних і у разі їх дійсності записує токени, роль та ім’я у локальне сховище. Також було створено клас AuthGuardService, який

перевіряє чи користувач авторизований у системі і чи у токена доступу не скінчився термін дії.

Другим кроком була реалізація головної сторінки сайту (рисунок 2.12). Для цього створено компонент та модуль маршрутизації manager. Сторінка відображає карту за допомогою Maps JavaScript API, на якій адміністратор може відслідковувати пересування водіїв. Також верхня панель сайту містить два списки: перший для перегляду та редагування інформації про документи, інший для перегляду показників. Праворуч від списків було розташовано кнопку повідомлень. Вона використовується для того, щоб адміністратор мав змогу розглядати нові резюме водіїв, або запити на отримання бонусу для військових. Під час кожного переходу на головну сторінку, сайт робить запит до сервера для перевірки чи не з'явились нові резюме або запити. Також відбувається запит до Firebase Realtime Database для того, щоб отримати поточні координати водіїв і відобразити їх на карті. Усі запити до серверу виконуються за допомогою одного сервісу. Також для сайту використовувались стилі, які надає такий інструмент як Bootstrap.

Рисунок 2.12 – Головна сторінка сайту

При переході до будь-якого документа адміністратору надається меню, яке містить дві функції: перехід до таблиці та перехід до форми (рисунок 2.13). Для кожного документа за допомогою Angular CLI було згенеровано чотири компоненти: меню, список даних, форма додання нової інформації та модуль маршрутизації. Кожен документ у таблиці має додатковий стовпчик, що дає

змогу адміністратору редагувати дані. Форма для додання нової інформації містить перевірку на правильність введення даних .

Під час переходу до будь-якого показника адміністратору також надається меню, яке обов'язково містить функцію для відображення таблиці. У деяких показниках, меню додатково надає можливість переходу до форми для оформлення звіту, щоб аналізувати роботу системи. Форма містить два поля для введення періоду між якими датами потрібно зробити звіт. Для кожного показника за допомогою Angular CLI було згенеровано чотири компоненти: меню, список даних, дані звіту та модуль маршрутизації. Щоб адміністратор міг сортувати інформацію за будь-яким стовпчиком, було створено директиву та клас NgbdSortableHeader. Для пошуку та фільтрування за стовбцями таблиці, кожному показнику реалізовано відповідний сервіс. Деякі фрагменти коду на інтерфейс сайту наведені у додатку Г.

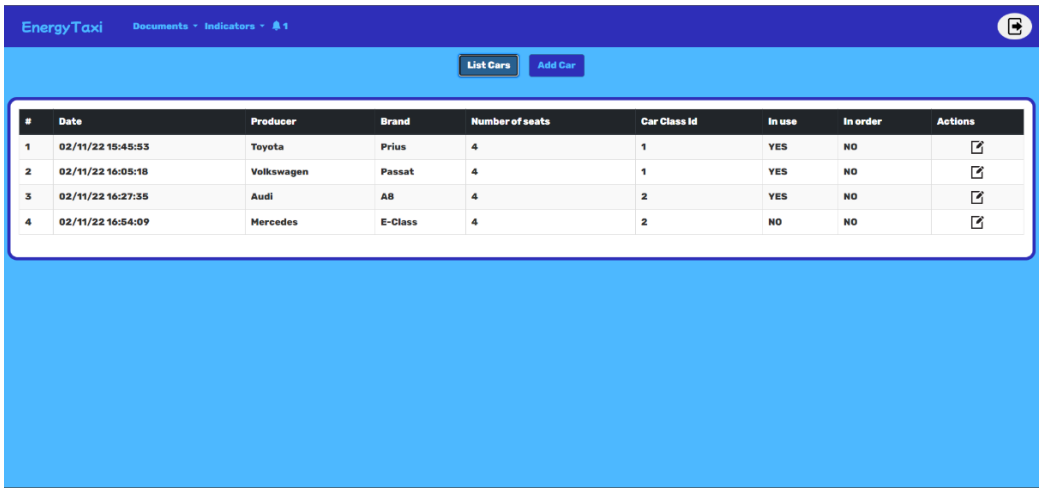


Рисунок 2.13 – Таблиця з даними про автомобілів

ВИСНОВОК

Отже, було розроблено програмне забезпечення для сервісу таксі. За результатами роботи були набуті знання у проектуванні інформаційної системи та правильної комунікації різних компонент сервісу і їх розширення. Застосовані попередні навички для створення серверної частини сервісу, використовуючи таку мову програмування як Java та його фреймворк Spring, що забезпечило швидку та надійну роботу та дозволило зручно керувати бізнес-логікою, забезпечити безпеку даних. Також вивчено відповідний програмний інструмент для реалізації Android додатків, а саме Android SDK та всі його додаткові доповнення та зовнішні ресурси, що дозволило створити зручний інтерфейс для клієнтів. З використанням Android SDK була забезпечена взаємодія з серверною частиною, отримання та відображення необхідної інформації, а також інші функціональні можливості, які допомогли покращити користувацький досвід.

Набуті знання у створені веб сайту за допомогою мови TypeScript та фреймворку Angular, що дозволило створити потужний та ефективний інтерфейс для керування системою таксі. Angular забезпечив зручний спосіб взаємодії з серверними даними, створення стилізованого та привабливого інтерфейсу. Проектування системи надало відповідний досвід для створення самостійних проектів.

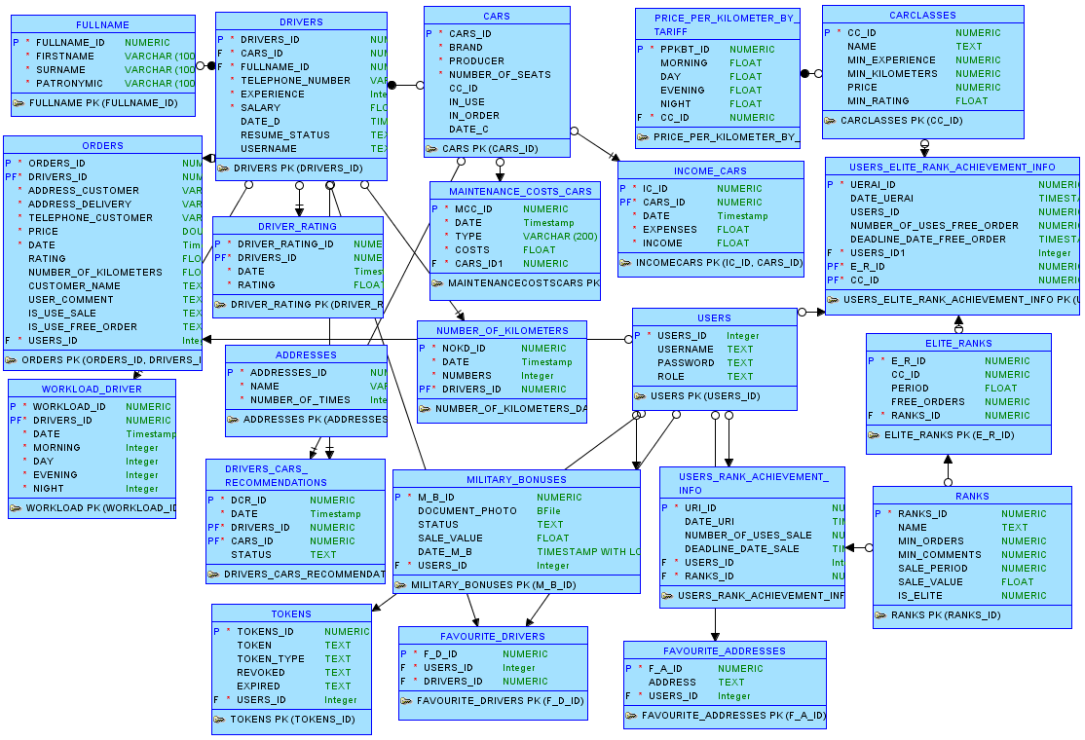
У майбутньому планується підтримання даного проекту. Система отримає нові оновлення, а саме: інші види бонусів, більше функцій для додатку користувача і водія, нові можливості аналізу роботи системи для адміністратора. Планується викладення додатка на таку платформу як “Play Market”, де ним зможуть користуватися користувачі, які проживають у відповідному місці для якого розроблявся сервіс і залишати свої коментарі, які будуть корисними для розробника, адже, таким чином будуть знаходитись та вирішуватись різні проблеми не тільки додатка, а й всієї системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

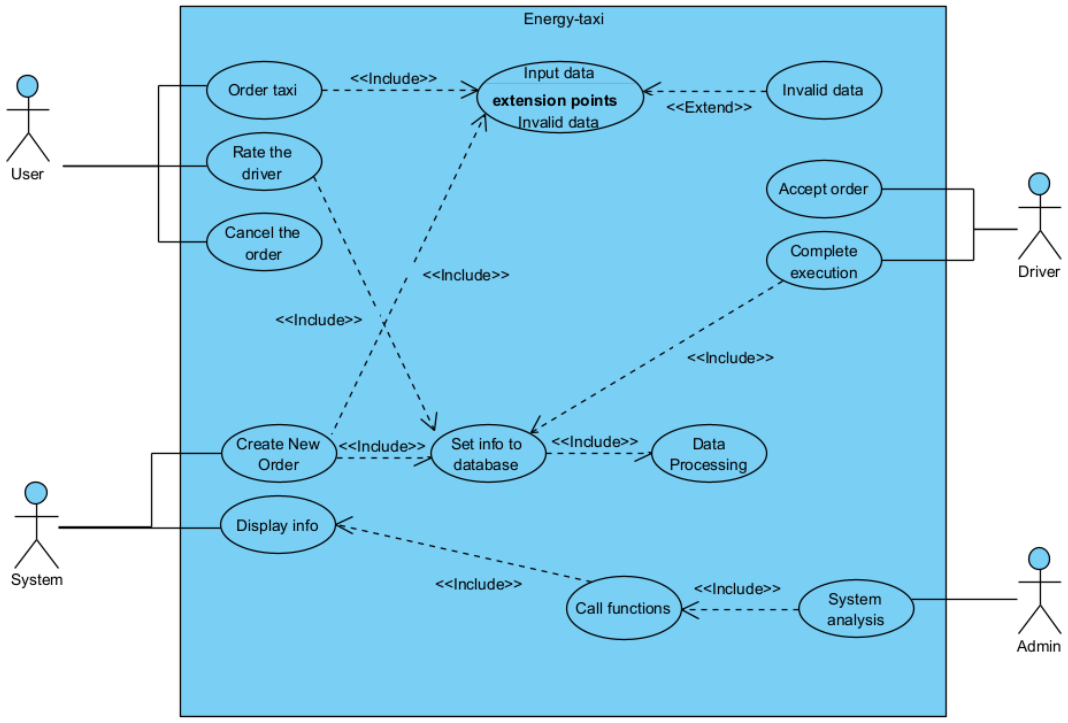
1. Java from EPAM: навч.-метод. посібник / І. М. Блінов, В. С. Романчик, 2020.
2. Spring REST Balaji Varanasi and Sudha Belida.
3. https://jwt.io/introduction.
4. Head First Android Development by Dawn Griffiths and David Griffiths.
5. https://developers.google.com.
6. Pro TypeScript: Application-Scale JavaScript Development by Steve Fenton.
7. Angular Development with Typescript Second Edition by Yakov Fain and Anton Moiseev.
8. Oracle Database Concepts, 10g Release 1 by Michele Cyran and Paul Lane.
9. https://firebase.google.com/docs/database.
10. SQL: The Complete Reference by James R. Groff and Paul N. Weinberg.

ДОДАТОК А

Схема бази даних



Діаграма прецедентів системи



ДОДАТОК Б

Загальний інтерфейс, який можуть реалізовувати сутності для доступу до бази даних.

```
26 implementations  Acer
public interface DAO <Entity>{
    26 implementations  Acer
    boolean add(Entity entity);
    26 implementations  Acer
    List<Entity> getAll();
    26 implementations  Acer
    Entity getOne(int ID);
    26 implementations  Acer
    boolean update(Entity entity);
    26 implementations  Acer
    boolean delete(int ID);
}
```

Клас для з’єднання з базою даних

```
Acer
public class DatabaseConnection {
    Acer
    public static Connection initializeDatabase()
            throws SQLException, ClassNotFoundException
    {
        String DRIVER = "oracle.jdbc.driver.OracleDriver";
        String dbURL = "jdbc:oracle:thin:@host.docker.internal:1521:xe";
        String dbUsername = "System";
        String dbPassword = "karamba614";
        Class.forName(DRIVER);
        Connection con = DriverManager.getConnection(dbURL, dbUsername, dbPassword);
        return con;
    }
}
```

Процедура SETADDRESS для додання нової адреси

```
create or replace NONEDITIONABLE PROCEDURE SETADDRESS
(
  ADDRESS_NAME IN VARCHAR2
) AS
BEGIN
    DECLARE
        isExists NUMBER:=0;
    BEGIN
        SELECT COUNT(*) INTO isExists FROM ADDRESSES WHERE NAME=ADDRESS_NAME;
        IF isExists=0 THEN
        INSERT INTO ADDRESSES VALUES(addresses_sequence.nextval, ADDRESS_NAME, 1);
        ELSE
        UPDATE ADDRESSES SET NUMBER_OF_TIMES=NUMBER_OF_TIMES+1 WHERE NAME=ADDRESS_NAME;
    END IF;
    END;
END SETADDRESS;
```

Функція для знаходження рекомендованого автомобіля для водія

```
create or replace NONEDITIONABLE FUNCTION GETCARFORDRIVER(EXPERIENCE IN NUMBER)
RETURN NUMBER IS
    car_id NUMBER:=0;
BEGIN
    DECLARE
        c_id CARS.CARS_ID%type;
        carClass number;
    BEGIN
        carclass:=GETCARCLASSFORDRIVER(EXPERIENCE);
        FOR c IN (SELECT CARS_ID INTO c_id
        FROM CARS
        WHERE IN_USE='NO' AND IN_ORDER='NO' AND CC_ID=carClass)
        LOOP
        car_id:=c.CARS_ID;
        END LOOP;
        UPDATE CARS SET IN_USE='YES', IN_ORDER='NO' WHERE CARS_ID=car_id;
    END;
  RETURN car_id;
END GETCARFORDRIVER;
```

Тригер, який спрацьовує при доданні нового замовлення

```
create or replace NONEDITIONABLE TRIGGER ORDERS_INSERT_TRIGGER
BEFORE INSERT ON ORDERS
FOR EACH ROW
BEGIN
  SETADDRESS(:NEW.ADDRESS_CUSTOMER);
  IF ISDRIVERALREADYEXISTS(:NEW.DATE_O, :NEW.DRIVERS_ID)=0 THEN
        INSERT INTO DRIVER_RATING
        VALUES(driver_rating_sequence.nextval, :NEW.DRIVERS_ID, :NEW.DATE_O,
                :NEW.RATING);
        INSERT INTO INCOME_CARS
        VALUES(in_come_car_sequence.nextval, GETCARID(:NEW.DRIVERS_ID),:NEW.DATE_O,
                GETMAINTANCECOSTS(:NEW.DRIVERS_ID, :NEW.DATE_O), :NEW.PRICE);
        INSERT INTO NUMBER_OF_KILOMETERS
        VALUES(number_of_kilometers_sequence.nextval, :NEW.DATE_O,
                :NEW.NUMBER_OF_KILOMETERS, :NEW.DRIVERS_ID);
        UPDATE CARS SET IN_ORDER='NO' WHERE CARS_ID=GETCARID(:NEW.DRIVERS_ID);
        SETWORKLOADDRIVERS(:NEW.DRIVERS_ID, :NEW.DATE_O);
   ELSE
        UPDATEDRIVERRATING(:NEW.DRIVERS_ID, :NEW.DATE_O, :NEW.RATING);
        UPDATE INCOME_CARS SET EXPENSES=EXPENSES+GETMAINTANCECOSTS(:NEW.DRIVERS_ID, :NEW.DATE_O),
        INCOME=INCOME + :NEW.PRICE
        WHERE CARS_ID=GETCARID(:NEW.DRIVERS_ID) AND TRUNC(DATE_IC)=TRUNC(:NEW.DATE_O);
        UPDATE NUMBER_OF_KILOMETERS SET NUMBERS=NUMBERS+:NEW.NUMBER_OF_KILOMETERS
        WHERE DRIVERS_ID=:NEW.DRIVERS_ID AND TRUNC(DATE_NOK)=TRUNC(:NEW.DATE_O);
        UPDATE CARS SET IN_ORDER='NO' WHERE CARS_ID=GETCARID(:NEW.DRIVERS_ID);
        UPDATEWORKLOADDRIVERS(:NEW.DRIVERS_ID, :NEW.DATE_O);
   END IF;
END;
```

Сервіс для обрахунку оновлення досвіду водія

```
1 usage  Acer
public class ExperienceService {
    2 usages  Acer
    public static float getUpdateExperience(Timestamp driver, Timestamp date){
        final long MILLIS_PER_DAY = 1000*60*60*24;
        long time_difference = driver.getTime() - date.getTime();
        long days_difference = (time_difference / MILLIS_PER_DAY) % 365;
        DecimalFormat df = new DecimalFormat( pattern: "#.##");
        df.setRoundingMode(RoundingMode.CEILING);
        Float result = (float)days_difference * (-100) / 36500;
        return Float.parseFloat(df.format(result).replace( oldChar: ',', newChar: '.'));
    }
}
```

Головний метод у конфігураційному класі SecurityConfiguration

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.cors();
    httpSecurity.csrf().disable();
    httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    httpSecurity.authorizeHttpRequests().requestMatchers("/api/v1/auth/**").permitAll();
    httpSecurity.authorizeHttpRequests().requestMatchers("/ws/**").permitAll();
    httpSecurity.authorizeHttpRequests().requestMatchers("/api/v1/documents/**")
            .hasAnyAuthority(ROLE.SUPER_ADMIN.name(), ROLE.ADMIN.name());
    httpSecurity.authorizeHttpRequests().requestMatchers("/api/v1/indicators/**")
            .hasAnyAuthority(ROLE.SUPER_ADMIN.name(), ROLE.ADMIN.name());
    httpSecurity.authorizeHttpRequests().requestMatchers("/api/v1/user-app/**")
            .hasAnyAuthority(ROLE.SUPER_ADMIN.name(), ROLE.ADMIN.name(), ROLE.USER.name());
    httpSecurity.authorizeHttpRequests().requestMatchers("/api/v1/driver-app/**")
            .hasAnyAuthority(ROLE.SUPER_ADMIN.name(), ROLE.ADMIN.name(), ROLE.DRIVER.name());
    httpSecurity.authorizeHttpRequests().requestMatchers("/api/v1/bonuses/**")
            .hasAnyAuthority(ROLE.SUPER_ADMIN.name(), ROLE.ADMIN.name(), ROLE.USER.name());
    httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
    httpSecurity.authenticationProvider(authenticationProvider);
    httpSecurity.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
    httpSecurity.logout().logoutUrl("/api/v1/auth/logout").addLogoutHandler(logoutService)
            .logoutSuccessHandler(((request, response, authentication) -> {
                SecurityContextHolder.clearContext();
            }));
    return httpSecurity.build();
}
```

Головний метод у конфігураційному класі ScheduledRankUpdate

```
@Scheduled(cron = "0 1 0 * * ?")
public void scheduleFixedRateTaskAsync() {
    Timestamp currentDate = new Timestamp(System.currentTimeMillis());
    userRankAchievementInfoDAO = new UserRankAchievementInfoDAOImpl();
    userEliteRankAchievementInfoDAO = new UserEliteRankAchievementInfoDAOImpl();
    eliteRankDAO = new EliteRankDAOImpl();
    rankDAO = new RankDAOImpl();
    List<EliteRank> eliteRankList = eliteRankDAO.getAll();
    List<Rank> ranks = rankDAO.getAll();
    for(Rank rank: ranks) {
        Calendar cal = Calendar.getInstance();
        cal.setTime(currentDate);
        cal.add(Calendar.DATE, (int) rank.getSalePeriod());
        Timestamp newSaleDeadLine = new Timestamp(cal.getTime().getTime());
        userRankAchievementInfoDAO.updateByDeadline(currentDate, newSaleDeadLine, rank.getRankId());
    }
    for (EliteRank eliteRank: eliteRankList){
        Calendar cal = Calendar.getInstance();
        cal.setTime(currentDate);
        cal.add(Calendar.DATE, (int) eliteRank.getPeriod());
        Timestamp newFreeOrderDeadLine = new Timestamp(cal.getTime().getTime());
        userEliteRankAchievementInfoDAO.updateByDeadline(eliteRank.getFreeOrders(), currentDate, newFreeOrderDeadLine,
                eliteRank.getEliteRankId());
    }
}
```

Клас контроллер CarController

```java
Acer
@RestController
@CrossOrigin
public class CarController {

    14 usages
    private CarsDAOImpl carsDAO;
    Acer
    @GetMapping("/api/v1/documents/cars")
    public ResponseEntity getCars(){
        try {
            carsDAO = new CarsDAOImpl();
            System.out.println(carsDAO.getAll());
            return ResponseEntity.ok(carsDAO.getAll());
        }
        catch (Exception e){
            return ResponseEntity.badRequest().body("Error to get cars");
        }
    }


    Acer
    @PostMapping("/api/v1/documents/cars")
    public void save(@RequestBody Car car){
        carsDAO = new CarsDAOImpl();
        car.setDate(new Timestamp(new Date().getTime()));
        System.out.println(car);
        carsDAO.add(car);
    }
```

Метод для реєстрації користувача

```java
public RegisterResponse registerUser(UserRequest request){
    User user = User.builder()
            .userName(request.getUserName())
            .password(passwordEncoder.encode(request.getPassword()))
            .role(ROLE.USER)
            .rankId(1)
            .build();
    if(userDAO.add(user)){
        String jwtToken = jwtService.generateToken(user);
        String refreshToken = jwtService.generateRefreshToken(user);
        saveUserToken(userDAO.getOneByUserName(user.getUsername()), jwtToken);
        return RegisterResponse.builder()
                .accessToken(jwtToken)
                .refreshToken(refreshToken)
                .userId(userDAO.getOneByUserName(user.getUsername()).getUserId())
                .build();
    } else {
        return RegisterResponse.builder()
                .accessToken("error")
                .refreshToken("error")
                .userId(0)
                .build();
    }
}
```

ДОДАТОК В

Метод onCreate у UserLoginActivity

```
Acer
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_user_login);
    userName = findViewById(R.id.userNameLoginEditText);
    password = findViewById(R.id.passwordLoginEditText);
    checkBox = findViewById(R.id.driverUserCheckBox);
    loginRegisterTextView = findViewById(R.id.loginRegisterTextView);
    loginForgotPasswordTextView = findViewById(R.id.loginForgotPasswordTextView);
    loginRegisterTextView.setOnClickListener(view -> goRegistryAsUser());
    loginForgotPasswordTextView.setOnClickListener(view -> resetPassword());
    Button login = findViewById(R.id.loginUserLoginButton);
    login.setOnClickListener(view -> login());
    UserInfoService.init( context: UserLoginActivity.this);
    DriverInfoService.init( context: UserLoginActivity.this);
}
```

Метод, який робить HTTP – запит, щоб отримати список улюблених водіїв

```
1 usage   Acer
public void getFavouritesDrivers(){
    RetrofitService retrofitService = new RetrofitService();
    MiniTaxiApi favouriteDriversApi = retrofitService.getRetrofit().create(MiniTaxiApi.class);
    String userId = UserInfoService.getProperty("userId");
    favouriteDriversApi.getFavouriteDriverUserInfo( header: "Bearer " + UserInfoService.getProperty("access_token"),
                Integer.valueOf(userId))
            Acer
            .enqueue(new Callback<List<FavouriteDriverUserInfo>>() {
                Acer
                @Override
                public void onResponse(Call<List<FavouriteDriverUserInfo>> call, Response<List<FavouriteDriverUserInfo>> response)
                    if(response.body()!=null){
                        try {
                            if(response.errorBody() != null){
                                if(response.errorBody().string().contains("Token expired")){
                                    refreshToken();
                                }
                            }
                            else{
                                populateListView(response.body());
                            }
                        } catch (IOException e) {
                            throw new RuntimeException(e);
                        }
                    }
                }
```

Метод, що робить запит до Firebase Realtime Database, щоб отримати координати водіїв

```
1 usage  Acer
private void getDriverLocation(){
    DatabaseReference uidRef = databaseReference.child( pathString: "drivers-info");
    Query query1 = uidRef.orderByChild( path: "status").equalTo( value: "IN_ORDER");
    Query query2 = uidRef.orderByChild( path: "status").equalTo( value: "ONLINE");
     Acer
    ValueEventListener valueEventListener = new ValueEventListener() {
         2 usages  Acer
        @Override
        public void onDataChange(@NonNull @NotNull DataSnapshot snapshot) {
            for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                DriverInfo driverInfo = dataSnapshot.getValue(DriverInfo.class);
                setDriverOnMap(driverInfo);
            }
        }
         Acer
        @Override
        public void onCancelled(@NonNull @NotNull DatabaseError error) {

        }
    };
    query1.addListenerForSingleValueEvent(valueEventListener);
    query2.addListenerForSingleValueEvent(valueEventListener);
}
```

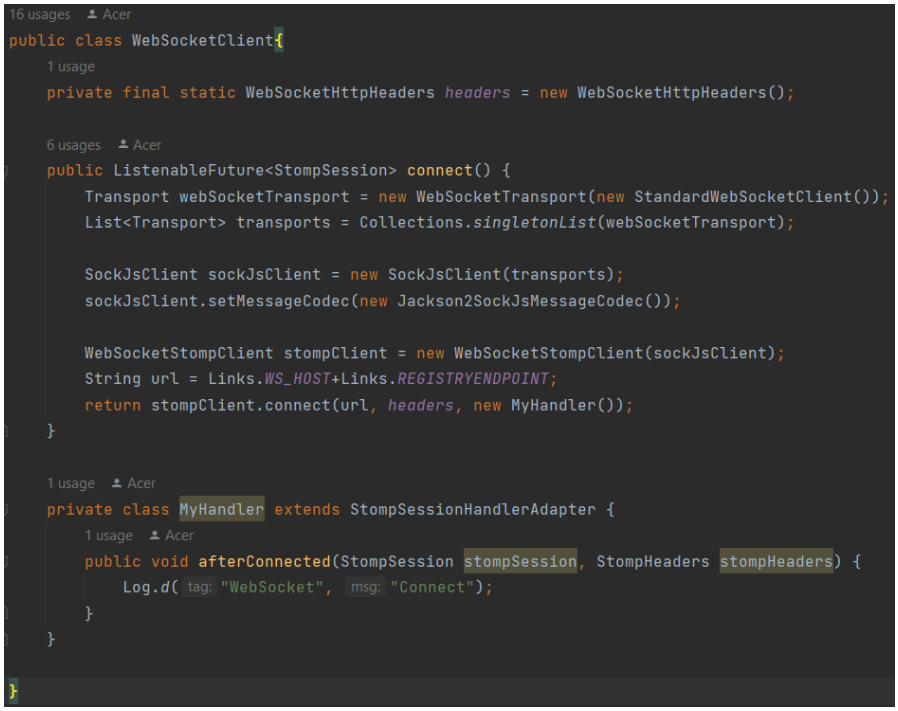
Частина методу, що ініціалізує GoogleMap.

```
public void onMapReady(@NonNull GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMyLocationEnabled(true);
    mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(
            clientContext: this, R.raw.map_style_json));
    mMap.setLatLngBoundsForCameraTarget(NETISHYN_BOUNDS);
    mMap.setMinZoomPreference(10);
    mMap.getUiSettings().setMyLocationButtonEnabled(true);
    locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
    userLocationService = new UserLocationService();
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission( context: this,
            Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {

        return;
    }
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  minTimeMs: 1000,  minDistanceM: 15,
            userLocationService);
    boolean isGPSEnable = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
    if(isGPSEnable){
        locationManager.getCurrentLocation(
                LocationManager.GPS_PROVIDER,
                cancellationSignal: null,
                getMainExecutor(),
```

Два головних метода у адаптерах

```
 Acer
@NonNull
@Override
public UserOrderHistoryHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
    View view = LayoutInflater.from(viewGroup.getContext())
            .inflate(R.layout.user_order_history_item, viewGroup,  attachToRoot: false);
    return new UserOrderHistoryHolder(view);
}

 Acer
@Override
public void onBindViewHolder(@NonNull UserOrderHistoryHolder userOrderHistoryHolder,
                             @SuppressLint("RecyclerView") int i) {
    UserOrderInfo userOrderInfo = userOrderInfoList.get(i);
    userOrderHistoryHolder.driverName.setText(userOrderInfo.getDriverName());
    userOrderHistoryHolder.carName.setText(userOrderInfo.getCarName());
    userOrderHistoryHolder.addressCustomer.setText(userOrderInfo.getAddressCustomer());
    userOrderHistoryHolder.addressDelivery.setText(userOrderInfo.getAddressDelivery());
    userOrderHistoryHolder.ratingOrderBar.setRating(userOrderInfo.getRating());
    userOrderHistoryHolder.price.setText(String.valueOf(userOrderInfo.getPrice()));
    userOrderHistoryHolder.date.setText(String.valu
                                        Fetching Documentation...
     Acer
    userOrderHistoryHolder.constraintLayout.setOnClickListener(new View.OnClickListener() {
         Acer
        @Override
        public void onClick(View view) { selectListener.onItemClicked(userOrderInfoList.get(i)); }
    });
}
```

Клас для ініціалізації WebSocket



```
16 usages  Acer
public class WebSocketClient{
    1 usage
    private final static WebSocketHttpHeaders headers = new WebSocketHttpHeaders();

    6 usages  Acer
    public ListenableFuture<StompSession> connect() {
        Transport webSocketTransport = new WebSocketTransport(new StandardWebSocketClient());
        List<Transport> transports = Collections.singletonList(webSocketTransport);

        SockJsClient sockJsClient = new SockJsClient(transports);
        sockJsClient.setMessageCodec(new Jackson2SockJsMessageCodec());

        WebSocketStompClient stompClient = new WebSocketStompClient(sockJsClient);
        String url = Links.WS_HOST+Links.REGISTRYENDPOINT;
        return stompClient.connect(url, headers, new MyHandler());
    }

    1 usage  Acer
    private class MyHandler extends StompSessionHandlerAdapter {
        1 usage  Acer
        public void afterConnected(StompSession stompSession, StompHeaders stompHeaders) {
            Log.d( tag: "WebSocket",  msg: "Connect");
        }
    }

}
```

Сервіс для прослуховування зміни локації користувача

```
7 usages  Acer
public class UserLocationService implements LocationListener {

    3 usages
    private LatLng currentLocation;
    4 usages
    public final static LatLng DEFAULT_LOCATION = new LatLng( latitude: 50.333798206435915,  longitude: 26.65000580334267);

    4 usages  Acer
    public LatLng getCurrentLocation() { return currentLocation; }

    2 usages  Acer
    public void setCurrentLocation(LatLng currentLocation) { this.currentLocation = currentLocation; }

    4 usages  Acer
    @Override
    public void onLocationChanged(@NonNull Location location) {
        Log.d( tag: "mylog",  msg: "Got User Location: " + location.getLatitude() + ", " + location.getLongitude());
        currentLocation = new LatLng(location.getLatitude(), location.getLongitude());
    }
}
```

ДОДАТОК Г

Головний маршрутизатор

```
const routes: Routes = [

  { path: '', redirectTo: 'manager', pathMatch:'full'},
  { path: 'manager', loadChildren:() => import('./manager/manager.module').then(m => m.ManagerModule),
    canActivate:[AuthGuardService]},
  { path: 'login', loadChildren:() => import('./login/login-module/login-module.module').then(m => m.LoginModuleModule)},
];

2 usages
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Сервіс AuthGuardService

```
5+ usages
@Injectable({
  providedIn: 'root'
})
export class AuthGuardService {

  no usages
  constructor(
    private router:Router,
    private authService: AuthenticationService
  ) { }

  no usages
  canActivate(route: ActivatedRouteSnapshot, state:RouterStateSnapshot) :boolean {
    if(this.authService.isUserLoggedIn()){
      console.log("in guard true")
      return true;
    }
    else {
      console.log("in guard false")
      this.router.navigate( commands: ['login']);
      return false;
    }
  }
}
```

Методи із сервісу для здійснення HTTP-запитів до сервера

```
5+ usages
public findAll() :Observable<T[]> {
  let httpHeaders :HttpHeaders  = new HttpHeaders()
    .set('Content-type','application/Json')
    .set('Authorization','Bearer ' + localStorage.getItem( key: "accessToken"));
  let options :{headers: HttpHeaders} ={
    headers:httpHeaders
  };
  return this.http.get<T[]>(this.url, options);
}
6+ usages
public save(object: T) :Observable<T>  {
  let httpHeaders :HttpHeaders  = new HttpHeaders()
    .set('Content-type','application/Json')
    .set('Authorization','Bearer ' + localStorage.getItem( key: "accessToken"));
  let options :{headers: HttpHeaders} ={
    headers:httpHeaders
  };
  return this.http.post<T>(this.url, object, options);
}
```

Метод ngOnInit, що слугує для отримання даних з бази даних та відображення їх у таблиці

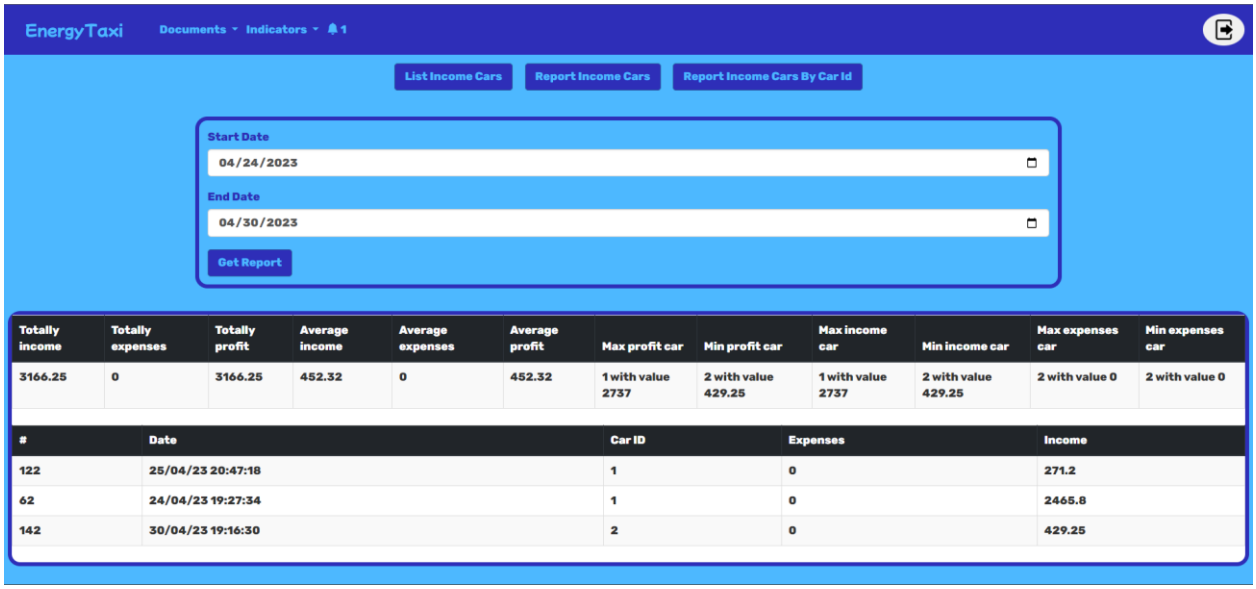
```
ngOnInit() :void {
  this.service.findAll().pipe(
    catchError( selector: error => {
      console.log("token: " + error.error.error_message === 'Token expired');
      if (error.error.error_message === 'Token expired'){
        this.service.refreshToken().subscribe( next: data :RegisterResponse => {
          console.log("regresp: " + data);
          if (data.accessToken === 'Token expired') {
            localStorage.clear();
            this.router.navigate( commands: ['login'])
          } else {
            localStorage.setItem("accessToken", data.accessToken)
            this.service.setMap("documents/cars");
            this.service.findAll().subscribe( next: data :Car[] => {
              this.cars = data;
              this.isEdit?.fill( value: false,  start: 0, this.cars?.length);
              console.log("get" );
              console.log(data);
            });
          }
        });
      }
      else {
        localStorage.clear();
        this.router.navigate( commands: ['login'])
      }
      return of( value: []);
    })
  ).subscribe( next: data :Car[] | never[] => {
    this.cars = data;
```

Метод завдяки йому здійснюється пошук по стовпцю

```
1 usage
private _search(): Observable<SearchResult> {
  const { sortColumn :SortColumn , sortDirection :SortDirection , pageSize :number , page :number , searchTermDate :string , searchTermDriverId :st

  // 1. sort

  let driverRating :DriverRating[]  = sort(this.DRIVER_RATING, sortColumn, sortDirection);

  // 2. filter
  driverRating = driverRating.filter((data :DriverRating ) => matches(data, searchTermDate, searchTermDriverId,
    searchTermRating, this.pipe));
  const total :number  = driverRating.length;

  // 3. paginate
  driverRating = driverRating.slice((page - 1) * pageSize, (page - 1) * pageSize + pageSize);
  return of( value: { driverRating, total });
}
```

Форма для додання нових даних

Форма для отримання звіту



Список повідомлень

Форма для підтвердження або відхилення звіту