Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Створення мобільного додатку для станції технічного обслуговування

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Мостецький В.А.
(прізвище та ініціали)

Керівник (підпис) Матійчук Л.П.
(прізвище та ініціали)

Нормоконтроль (підпис) Марценко С.В.
(прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О.
(прізвище та ініціали)

Рецензент (підпис) Жаровський Р.О.
(прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2023р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Мостецький Василь Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Створення мобільного додатку для станції технічного обслуговування

Керівник роботи Матійчук Любомир Павлович, к.е.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 07 » лютого 2023 року № 4/7-133.

2. Термін подання студентом завершеної роботи 12 червня 2023р.

3. Вихідні дані до роботи Наукові публікації, електронні ресурси, підручники , посібники тематики дослідження

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області, проектування додатку станції технічного обслуговування. 1.1 Особливості функціонування огляд і аналіз існуючих аналогів, що реалізують функції предметної області. 1.2 Розроблення архітектури програмної системи 1.3 Проектування інформаційно-програмних модулів системи та бази даних. Розділ 2. Програмна реалізація додатку тестування та дослідна експлуатація для станції технічного обслуговування. 2.1 Особливості програмної реалізації додатків для ОС Android. Обгрунтування вибору технологій Java і SQLite. 2.2 Програмна реалізація бази даних та тестування додатку. 2.3 Процес розгортання та налаштування додатку. Розділ 3. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема. 2. Мета роботи. 3-4. Аналіз існуючих аналогів. 5. Діаграма варіантів використання. 6. Покрокова схема роботи мобільного додатку. 7. Діаграма послідовності формування заявки на ремонт. 8. Діаграма послідовності закриття заявки на ремонт клієнта СТО. 9. Діаграма класів. 10. Діаграма послідовності закриття заявки на ремонт клієнта СТО. 11. Схема бази даних. 12. Тестування додатку. 13. Процес побудови додатку. 14-17. Інструкція користувача. 18. Висновки. 19. Дякую за увагу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я., доцент кафедри МТ		

7. Дата видачі завдання 23 січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	08.05.2023	*Виконано*
2.	Підбір джерел про створення мобільного додатку для станції технічного обслуговування	09.05.2023-11.05.2023	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	12.05.2023-15.05.2023	*Виконано*
4.	Виконання дослідження щодо створення мобільного додатку для станції технічного обслуговування	16.05.2023-22.05.2023	*Виконано*
5.	Оформлення розділу «Аналіз предметної області, проектування додатку станції технічного обслуговування.»	23.05.2023-25.05.2023	*Виконано*
6.	Оформлення розділу «Програмна реалізація додатку тестування та дослідна експлуатація для станції технічного обслуговування»	26.05.2023-28.05.2023	*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	29.05.2023-30.05.2023	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони праці»	31.05.2023-01.06.2023	*Виконано*
9.	Оформлення кваліфікаційної роботи	02.06.2023-06.06.2023	
10.	Нормоконтроль	07.06.2023-08.06.2023	*Виконано*
11.	Перевірка на плагіат	09.06.2023	*Виконано*
12.	Попередній захист кваліфікаційної роботи	12.06.2023	*Виконано*
13.	Захист кваліфікаційної роботи	20.06.2023	

Студент _____ Мостецький В.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Створення мобільного додатку для станції технічного обслуговування // Кваліфікаційна робота освітнього рівня «Бакалавр» // Мостецький Василь Андрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2023 // С. 67, рис. – 43, бібліогр. – 32.

Ключові слова: мобільний додаток, архітектура програмної системи, веб-сервіс, MySQL, база даних, ОС Android.

У першому розділі здійснено опис предметної області, напрями діяльності. Проведено аналіз відомих програмних систем. Здійснено аналіз вимог до програмної системи. Розроблено архітектуру програмного додатку, що дозволить краще зрозуміти функції основних його частин. Створено та описано структурну схему, основними компонентами якої є: рівень клієнта, рівень бізнес-логіки та рівень даних. Описано функціональну структуру системи та її основних елементів – модулів обробки даних.

У другому розділі обґрунтовано технологію, мову програмування та розроблено програмну систему. Обґрунтовано засоби розробки додатку та здійснено опис основних програмних модулів системи.

А також здійснено опис процедур тестування та їхніх результатів, описані тест-вимоги до програмного забезпечення, а також виявлені дефекти. По результатам тестування сформовано підсумок тестування. Також в даному розділі було розкрито питання встановлення та налаштування програмного забезпечення, а також вказані вимоги, дотримання яких необхідно для користування програмою.

ANNOTATION

A Mobile App Development for the Service Station // Qualification work of the educational level "Bachelor" // Vasyl Mostetskyi // Ternopil National Technical University named after Ivan Pulyu, Faculty of Computer Information Systems and Software Engineering, Department of Computer Sciences, group of SN -41 // Ternopil, 2023 // C.67, fig. - 43, bibliography -32

Keywords: mobile application, software system architecture, web service, MySQL, database, Android OS.

In the first section, a description of the subject area, directions of activity was carried out. The analysis of known software systems was carried out. An analysis of the requirements for the software system was carried out. The architecture of the software application has been developed, which will allow a better understanding of the functions of its main parts. A structural diagram has been created and described, the main components of which are: client level, business logic level, and data level. The functional structure of the system and its main elements - data processing modules - are described.

In the second chapter, the technology, the programming language and the software system are developed. The application development tools are justified and the main software modules of the system are described.

Also, a description of the testing procedures and their results, the test requirements for the software, as well as the detected defects were described. Based on the results of the testing, a summary of the testing was formed. Also, in this section, the issue of installing and configuring the software was revealed, as well as the requirements that must be met in order to use the program.

ЗМІСТ

ВСТУП

Актуальність теми. Технічне обслуговування і ремонт легкових автомобілів індивідуального користування організують відповідно до діючого "Положенням про технічне обслуговування і ремонт легкових автомобілів, що належать громадянам".

Автомобілі на станції технічного обслуговування (СТО) приймаються відповідно до встановлених правил.

Значний ріст парку легкових автомобілів, що належать населенню в Україні, вимагає збільшення проектування дорожніх СТО.

Виробничо-технічну базу системи технічного обслуговування автомобілів в основному складають підприємства (структурні одиниці) трьох видів:

- СТО, у тому числі майстерні і пункти ТО і ремонту;
- бази і склади матеріально-технічного постачання;
- гаражі і стоянки автомобілів.

Сучасні СТО здійснюють:

- продаж і передпродажне обслуговування нових автомобілів;
- передпродажне підготовку старих автомобілів;
- продаж запасних частин, експлуатаційних матеріалів і приналежностей до них;
- гарантійний і післягарантійний термін експлуатації;
- відбудовний ремонт автомобілів і агрегатів, у тому числі усунення ушкоджень кузова автомобілів і т.д.

Мета і задачі дослідження. З метою підвищення ефективності функціонування станцій технічного обслуговування в рамках даної кваліфікаційної роботи було вирішено розробити мобільний додаток з можливістю підтримки невеликих станцій технічного обслуговування для мобільної платформи Android. Основними завданнями даної роботи є:

1) проаналізувати існуючі аналоги мобільних додатків для iOS і Android;

2) розглянути інструменти для реалізації проекту;

3) спроектувати мобільний додаток і сервер;

4) реалізувати мобільний додаток для операційної системи Android і серверну частину з використанням MysQL;

5) провести тестування розробленого додатку і сервера.

Практичне значення одержаних результатів. У даній кваліфікаційній роботі реалізований мобільний додаток з можливістю підтримки невеликих станцій технічного обслуговування для мобільної платформи Android.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПРОЕКТУВАННЯ ДОДАТКУ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

1.1 Особливості функціонування огляд і аналіз існуючих аналогів, що реалізують функції предметної області

Станції технічного обслуговування (СТО) – це підприємства, продукція яких складається із послуг для клієнтів і послуг з технічного обслуговування та ремонту легкових, вантажних автомобілів або автобусів. СТО характеризуються наявністю: стоянки для автомобілів клієнтів, гостей та стоянки для автомобілів співробітників, складу нових і держаних автомобілів, зони для демонстрації цих автомобілів; автосалону або зони для продажу автомобілів; приміщень для продажу запчастин і аксесуарів; приміщень для роботи із клієнтами; виробничих зон і приміщень для технічного обслуговування та ремонту автомобілів.

Потужність СТО визначається кількістю робочих постів з ТО і ПР автомобілів. Під робочим постом розуміється робоче місце з обладнанням, технологічним реманентом і автомобілем, на якому виробничий персонал виконує основні роботи з ТО і ПР. Залежно від типу автомобіля і виду робіт площа робочого поста лежить в межах 25 - 40 метрів квадратних.

За кількістю робочих постів СТО поділяються на малі, середні, великі та надвеликі.

До малих СТО належать станції на два робочих пости і чисельністю виробничих робітників від 1 до 4 осіб. Таких сімейних (дворових) СТО налічується 70%. Їх особливістю є обмежений перелік виконуваних робіт.

Середні СТО мають у своєму складі від 3 до 9 робочих постівз чисельністю виробничих робітників5-20 осіб. Питома чисельність цих СТО в Україні становить близько 15%, в Європі 27.-50%.

До великих СТО належать підприємства з кількістю робочих постів 10-19 і чисельністю виробничих робітників 10-40 осіб.

Надвеликі СТО мають більше 20 робочих постів та чисельність виконавців 30-250 осіб.

Розміщення СТО в значній мірі визначає її призначення, спеціалізацію і характер послуг, які вона надає. За особливістю розміщення СТО можуть бути дорожніми та міськими.

Дорожні станції, як правило, є універсальними і виконують технічне обслуговування та ремонт усіх типів автомобілів. Розміщуються вони в основному біля автомобільних доріг та магістралей. Ці станції мають від 1 до 5 робочих постів і призначені для виконання, мийних, змащувальних, кріпильних та регулювальних робіт, усунення дрібних відмов та несправностей, продажу запасних частин та аксесуарів. Ці підприємства розташовують у комплексі з автозаправними станціями.

Міські СТО призначені для обслуговування постійного контингенту клієнтури та парку автомобілів певного регіону міста. За функціональним призначенням і відношенням до джерел фінансування та постачання, які визначають номенклатуру сервісу СТО, вони можуть бути фірмовими (дилерськими) або незалежними. Співвідношення фірмового та незалежного автосервісу у розвинених країнах становить від 40% до 60%, у нашій країні поки що від 5% до 95%[2]. Це зумовлено тим, що фірмовий автосервіс прискорено розвивається у великих містах і слабко розповсюджений на периферії.

Основне призначення фірмових СТО полягає у забезпеченні продажу автомобілів певного виробника на регіональному ринку. Сервіс на цих підприємствах розглядається як умова забезпечення ефективності продаж, що викликає потребу у супутніх послугах: виконанні передпродажної підготовки автомобілів, гарантійному та післягарантійному обслуговуванні та ремонті, продажі оригінальних запчастин. Крім виконання своїх основних функцій

автосервісу, фірмові СТО забезпечують автомобільні заводи правдивою інформацією про якість їх продукції і недоліки певних моделей.

Виробнича структура фірмових СТО включає автосалон, склад автомобілів, стоянку нових автомобілів, склад і магазин продажу оригінальних запчастин і аксесуарів, інфраструктуру і фірмове програмне забезпечення для роботи з клієнтурою, технологічні засоби для обслуговування і ремонту автомобілів із використанням обладнання сертифікованого виробником цих автомобілів, побутові та офісні приміщення.

На території України працюють біля 8500 незалежних станцій технічного обслуговування, але тільки 1360 (16%) з них є комплексними і виконують весь перелік робіт з технічного обслуговування та ремонту автомобілів, маючи у своїй структурі більше 9 робочих постів. Ці станції за своєю потужністю можна віднести до великих.

Всі інші СТО за потужністю є більш дрібними: 1275 станцій (15%) мають від 6 до 8 робочих постів, 5865 СТО (69%) мають від 1 до 5 постів та за обмеженим об’ємом виконуваних робіт є спеціалізованими.

Спеціалізація цих 84% станцій технічного обслуговування має відносні напрямки за видами робіт: 32% із загальної їх чисельності виконують технічне обслуговування та ремонт трансмісії і ходової частини автомобілів, 18% – двигуна, 14% – паливної апаратури, 12% – електрообладнання, 11% – кузова з його риштовкою; 8% – виконують фарбування автомобілів, 5% – тюнінг [2].

Для ефективного створення якісної та потрібної для застосування програмної системи необхідно також провести порівняння вже існуючих аналогів програмних систем для даної області застосування.

Як було відмічено раніше, в даній області вже існує досить багато рішень. Однак з огляду на те, що майже всі вони мають схожу структуру, в цьому розділі будуть розглянуті найбільш відомі з них.

Скріншоти основного функціоналу представлено на рисунку 1.1.

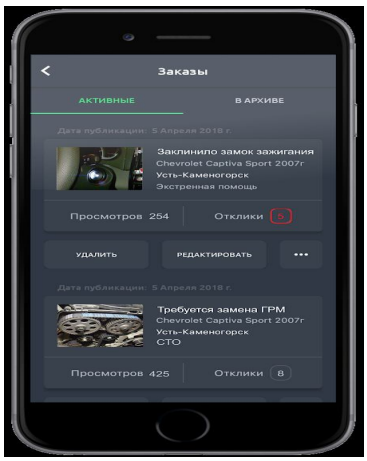


Рисунок 1.1 – Додаток СТО

Наступним цікавим додатком є CarService. Скріншоти основного функціоналу представлено на рисунку 1.2. Додаток для служби технічного обслуговування допомагає вам зареєструвати інформацію про свої транспортні засоби. Ви можете встановити періодичність обслуговування в милях, наприклад 10 000 миль, або в інтервалі часу, як і 6 місяців. Дуже корисно зареєструвати детальну інформацію про вашу запасну частину (назву моделі, код тощо), щоб заощадити час на пошук цієї інформації під час наступного технічного обслуговування. У програмі є попередньо визначений список найпоширеніших запасних частин, але ви також можете створити власні елементи та відстежувати їх.

Служба автомобільної служби може:

- журнал технічного обслуговування декількох транспортних засобів;

- встановити періодичність обслуговування в годинах або місяцях двигуна км;

- додати свої запчастини;

- копіювати VIN-код у буфер обміну у вашому телефоні.

Додаток показує вам чітко, що було змінено у вашому автомобілі та коли.

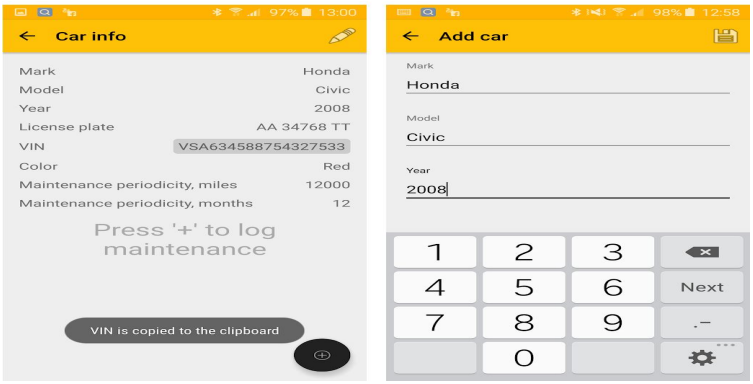


Рисунок 1.2 – Додаток CarService

Таким чином, існуючі рішення не здатні в повній мірі забезпечити повний функціонал невеликих станцій технічного обслуговування, а також є великими комерційними продуктами, тому розробки в рамках даної роботи є бузумовно актуальними.

Специфікація вимог до програмної системи – це специфікація окремого програмного продукту, програми або набору програм, які виконують деякі дії в деякому середовищі. Тобто – це повний опис поведінки системи що розробляється [3].

В загальному випадку специфікація включає наступне:

- глосарій проекту;
- опис варіантів використання.

На рисунку 1.3 представлено діаграму прецендентів взаємодії користувача з програмою. Діаграма варіантів використання дозволяє швидко побачити основні функції, які буде виконувати програма після розробки.

Наступним кроком у специфікації вимог до програмного продукту є розробка діаграми послідовності. Ця діаграма дозволяє виділити деякі об’єкти в системі та відобразити події основного сценарію програмного продукту, у вигляді послідовності повідомлень, якими обмінюються об’єкти. Діаграма послідовності зображена на рисунку 1.4.

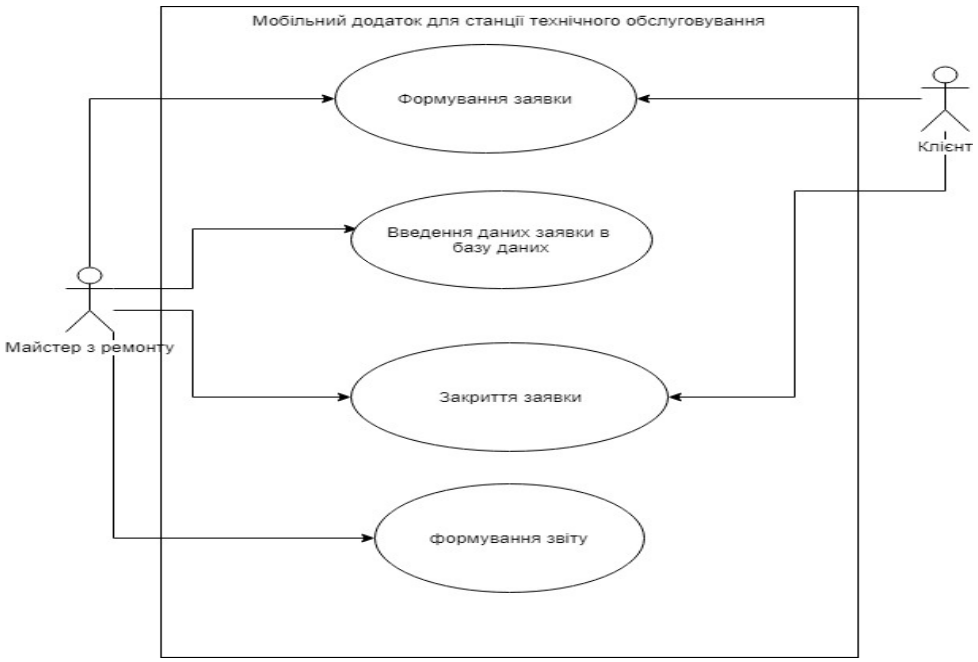


Рисунок 1.3 –Діаграма прецедентів взаємодії користувача з програмою

Як видно з рисунку 1.13 користувач може виконати наступне:

1) реєстрація;
2) вхід в систему;
3) формування заявки;
4) введення даних заявки в базу даних;
5) закриття заявки;
6) формування звіту;
7) вихід з програми.

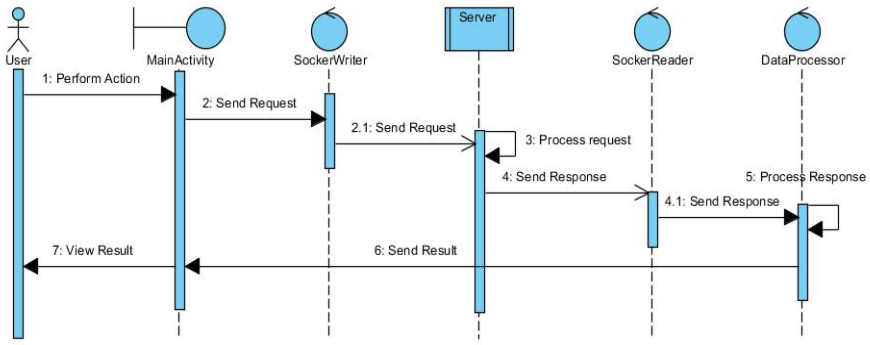


Рисунок 1.4 –Діаграма послідовності

Для створення розкадровки використаємо інструмент Balsamiq Mockups

– зручний програмний засіб для створення ескізів екранних форм. На рисунках 1.5 – 1.8 наведено ескізи основних екранних форм відповідно до сценаріїв використання.

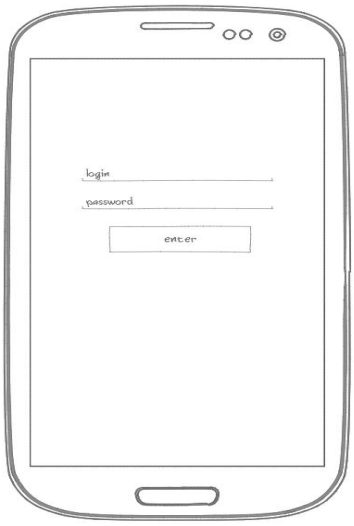


Рисунок 1.5 – Ескіз екранної форми авторизації користувача

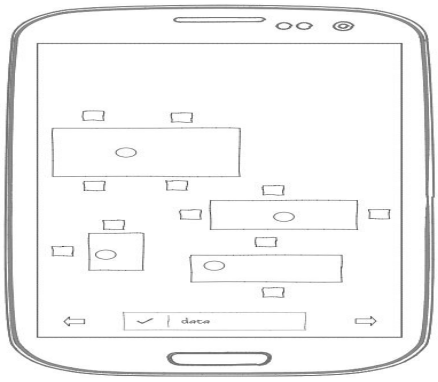


Рисунок 1.6 – Ескіз екранної форми із схемою ремонтних постів

Рисунок 1.7 – Ескіз екранної форми із можливістю замовлення майстра з ремонту автомобілів

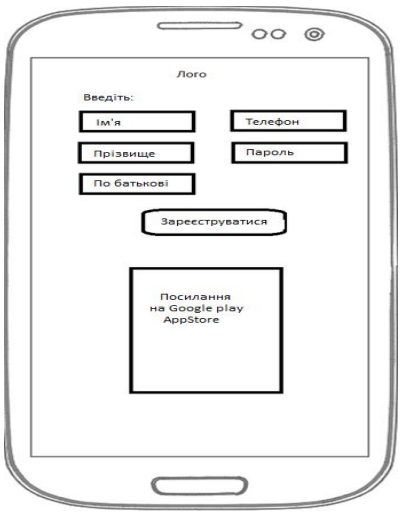


Рисунок 1.8 – Ескіз екранної форми реєстрації клієнтаСТО

Значення нефункціональних вимог:

- час, необхідний для навчання звичайних користувачів – 30хвилин;
- час, необхідний для навчання досвідчених користувачів – 15 хвилин;
- основні вимоги застосовності нової системи відносно інших систем, які знають користувачі – всі функції системи є легкими у виконанні, а структура програми не відрізняється від існуючих аналогів;
- вимоги по відповідності загальним стандартам застосовності та стандартам графічного інтерфейсу користувача – програма повинна працювати в операційній системі Android версії вище 7.0;
- доступність – час, що витрачається на обслуговування системи не повинен перевищувати 3% від загального часу роботи;
- середній час безвідмовної роботи – 3години;
- використання ресурсів – мінімальні системні вимоги:
  - 256 Мб пам’яті;
  - 10Мб вільного дискового простору;
  - Процесор з тактовою частотою 600МГц;

- Android 7.0.

1.2 Розроблення архітектури програмної системи

Програмний додаток для ОС Android складається з набору активностей, кожній з яких відповідає вікно керування. Кожна активність представлена в проекті класом, реалізований на мові Java, що зберігається в однойменному файлі з розширенням .java. Кожній активності відповідає xml файл-опис. В xml-файлі описано у вигляді xml-коду розташування візуалізуються об'єкти. При запуску активності система Android автоматично розпізнає розмір екрану мобільного пристрою і призводить до виведення контенту у відповідність з розміткою, описаною в xml-файлі. Таким чином, одна і та ж активність буде виглядати однаково незалежно від діагоналі використовуваного пристрою. Також, для кожного додатку Android повинен існувати xml-файл, в якому у вигляді xml-коду будуть прописані мінімальні вимоги до системи, а також активність, яка викликається при запуску програми [7].

Додаток працює з реляційної базою даних MySQL. У ролі локальної бази даних використовується SQLite, яка не використовує парадигму клієнт-сервер, тобто движок SQLite не є окремо працюючим процесом, з яким взаємодіє програма, а надає бібліотеку, з якої програма компонується і движок стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. Після формування відповідних замовлень відбувається синхронізація із сервером бази даних MySQL

Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці,

індекси і дані) в єдиному стандартному файлі на тому пристрої, на якому виконується програма [8].

На рисунку 1.9 представлена схема роботи додатку.

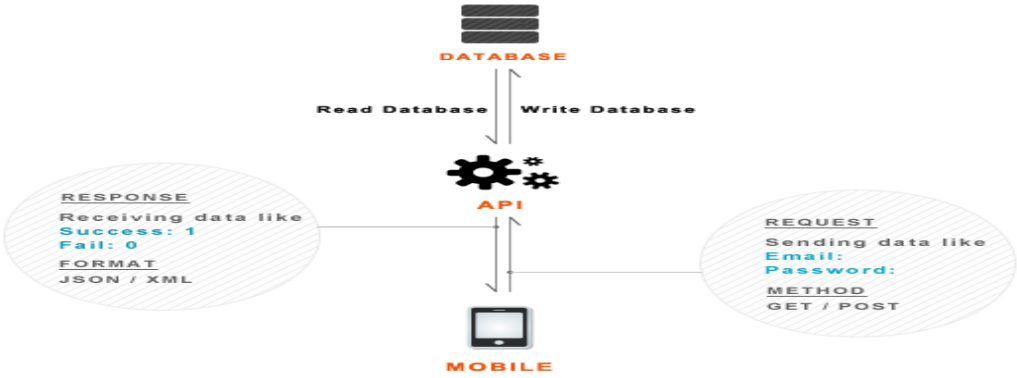


Рисунок 1.9 – Покрокова схема роботи мобільного додатку

Діаграма послідовності являє собою діаграму взаємодії, яка показує, як об’єкти працюють один з одним і в якому порядку. Діаграма послідовності показує взаємодії об’єктів, впорядковані в часі. Вона відображає об’єкти і класи, які беруть участь в сценарії, і послідовність повідомлень, якими обмінюються об’єктами і необхідними для виконання функцій сценарію.

Діаграми послідовності зазвичай пов’язані з реалізацією варіантів послідовності в логічному представленні розроблюваного додатку. Діаграмами послідовності іноді називаються діаграмами подій або сценаріями подій.

На рисунку1.10 зображена діаграма послідовності формування заявки клієнта СТО.

Об’єкт Клієнт звертається до об’єкта інженер з ремонту з проханнямприйняти заявку на обслуговування транспортного засобу.

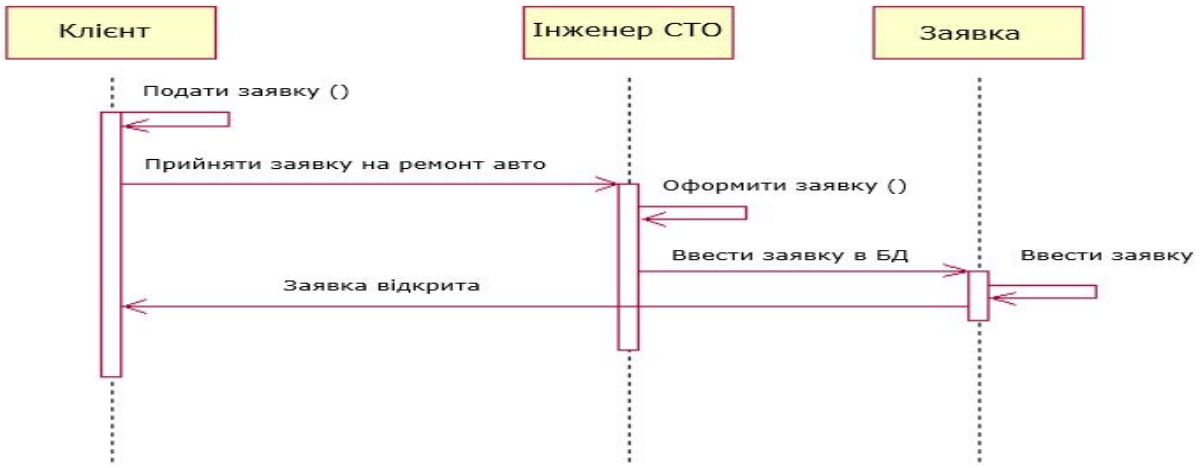


Рисунок 1.10 – Діаграма послідовності формування заявки на ремонт

Об’єкт інженер з ремонту оформляє заявку за заявою клієнта звертається до об’єкта заявка з командою ввести заявку в БД.

Об’єкт заявка забезпечує реєстрацію заявки в БД і повідомляє об’єкту клієнт про відкриття заявки. Процес формування заявки клієнта завершено.

На рисунку1.11 зображена діаграма послідовності закриття заявки клієнта СТО.

Об’єкт інженер з ремонту звертається до об’єкта клієнт з проханням підписати акт виконаних робіт. Об’єкт клієнт підписує акт виконаних робіт і повідомляє про це об’єкту інженер з ремонту.

Об’єкт інженер з ремонту звертається до об’єкта заявка з командою закрити заявку.

Об’єкт заявка забезпечує реєстрацію закриття заявки в БД і повідомляє об’єкт інженер з ремонту.

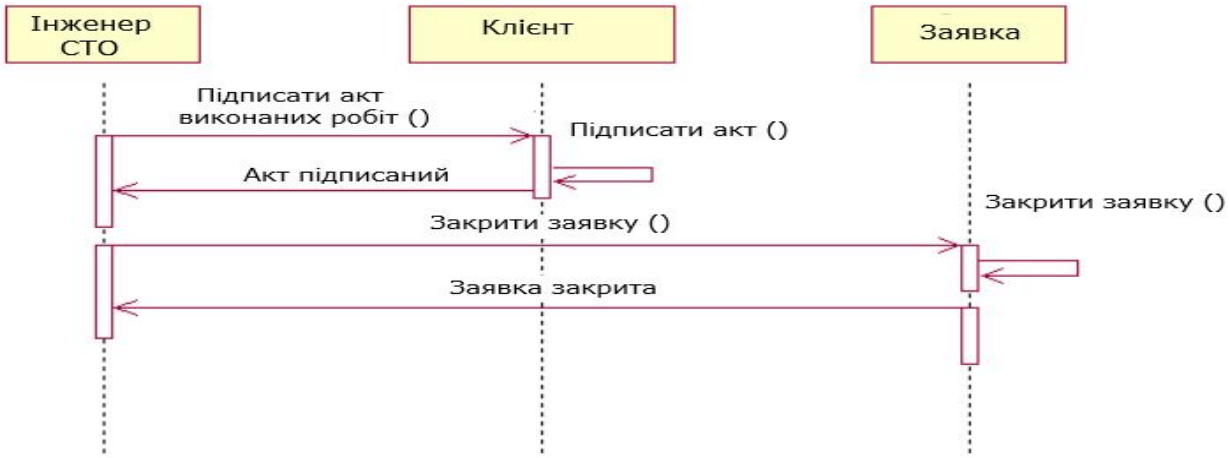


Рисунок 1.11 – Діаграма послідовності закриття заявки на ремонт клієнта СТО

Діаграма класів (Class diagram) служить для опису складу атрибутів класів, а також для відображення взаємозв'язків між класами, які використовується. Ця діаграма відображає статичний аспект системи. На рисунку 1.12 зображена діаграма класів додатку клієнтівСТО.

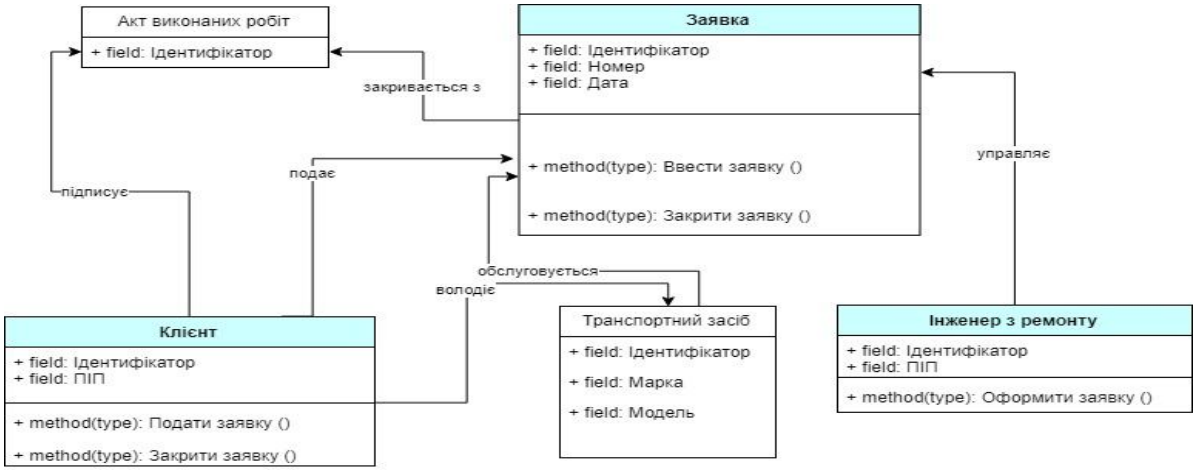


Рисунок 1.12 – Діаграма класів

Специфікація класів:

- Клієнт - клас фізичних або юридичних осіб, які подають заявки на обслуговування транспортних засобів;

- Заявка - клас документів про відкриття замовлення-наряду на обслуговування транспортних засобів клієнтів станції технічного обслуговування;

- інженер з ремонту - клас тих, хто управляє заявками;

- транспортний засіб - клас об’єктів, що обслуговуються;

- акт виконаних робіт - клас документів, що підтверджує завершення робіт по обслуговуванню транспортного засобу.

Представлена діаграма класів є основою для розробки програмного забезпечення і логічної моделі даних мобільного додатку для станції технічного обслуговування.

1.3. Проектування інформаційно-програмних модулів системи та бази даних

Функціонально, додаток складається з наведених нижче модулів (Активностей). Активність є схемою представлення Android-додатків. Кожен екран для користувача інтерфейсу представлений класом Activity і по суті є окремою формою додатка. Android-додаток здатний складатися з декількох активностей і може перемикатися між ними під час виконання програми.

Основна активність містить об'єкт автомобілі і підмінює вибір заявок на ремонт. Вона призначена для визначення користувачем заявки на ремонта також вибору цікавого для нього майстра.

Активність вибору ремонтних робіт містить кілька елементів управління, які дозволяють уточнити область, цікаву для майстра з ремонту. Після вибору цієї області відбувається перехід в наступну активність. В параметрах передається уточнена інформація по виду послуг. Активність використовує шаблон форматування ListView. Робота з базою даних відбувається за допомогою класу SqlDataHelper.

Схема активностей розробленого мною додатку представлена нижче на рисунку 1.13 за допомогою діаграми класів UML.

Активність вибору об'єкта інтересу містить список об'єктів інтересу, задовольняючи обрані раніше критеріям. Для цього активність, отримавши в

параметрах уточнену інформацію про об'єкти інтересу, формує запит до бази даних і на підставі отриманої від бази даних інформації створює динамічний список об'єктів за допомогою адаптера «ArrayAdapter». Робота з базою даних відбувається за допомогою класу SqlDataHelper.

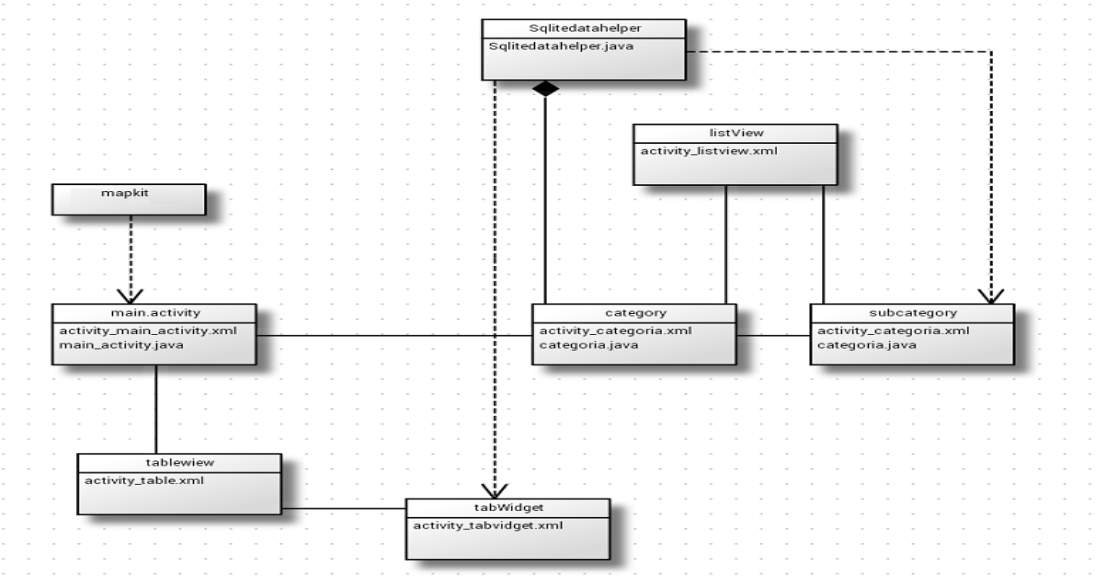


Рисунок 1.13 –Діаграма активностей додатку і зв'язків між ними

Активність TableView містить список об'єктів з виду робіт з вказівками цін на них. Активність посилає запит до бази даних про обрані категорії і на підставі отриманої інформації формує новий SQL-запит для отримання всіх видів із цінами, що цікавлять. При виборі будь-якого об'єкта запускається нова активність TabWidget, як параметрів якої передається ідентифікатор об’єкта.

Активність TabWidget: На підставі ідентифікатора об'єкта формується SQL-запит для отримання розширеної інформації про замовлення, такі як інформація про автомобіль, ціну ремонту, вибраного інженера та майстра, терміни виконання.

Логічні моделі даних допомагають визначити детальну структуру елементів даних в системі і відносини між елементами даних. Вони вдосконалюють елементи даних, введені концептуальною моделлю даних, і складають основу моделі фізичних даних.

Для розробки логічної моделі даних додатку для станції технічного обслуговування застосовуємо метод перетворення UML-діаграми класів системи вреляційну модель її бази даних, створену в методології IDEF1X.

Приведена до нормальної форми Бойса-Кодда з введенимидовідниками логічна модель даних додатку для опрацювання заявок клієнтів СТО зображена у вигляді ER-діаграми на рисунку1.14.

Між сутностями моделі в рамках одного бізнес-процесу встановлені наступні зв’язки:

- клієнт може мати кілька заявок («один до багатьох»);
- інженер з ремонту може оформити кілька заявок («один до багатьох»);
- за заявкою формується тільки одне деталізоване замовлення («один до одного»);
- за заявкою обслуговується тільки один транспортний засіб («один до одного»);
- за одним замовленням можна виконувати кілька видів ремонту («один до багатьох»);
- виконавець може бути задіяний за кількома замовленнями («один до багатьох»);

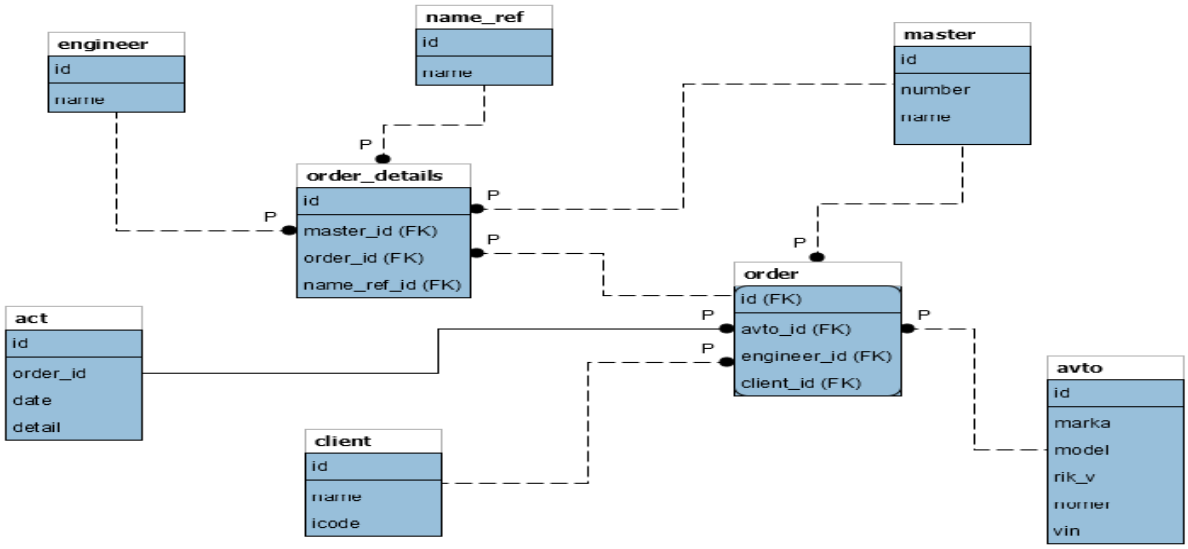


Рисунок 1.14 – Схема бази даних

Представлена логічна модель даних є основою для фізичного проектування бази даних додатку управління заявками клієнтів СТО.

1.4 Висновки до першого розділу

Здійснено опис предметної області, напрями діяльності. Визначено склад функцій, що входять до бізнес-процесу на основі яких розроблено схему управління бізнес-процесом. Проведено аналіз відомих програмних систем. Здійснено аналіз вимог до програмної системи.

Розроблено архітектуру програмного додатку, що дозволить краще зрозуміти функції основних його частин. Створено та описано структурну схему, основними компонентами якої є: рівень клієнта, рівень бізнес-логіки та рівень даних. Описано функціональну структуру системи та її основних елементів – модулів обробки даних.

РОЗДІЛ 2. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ ДЛЯ СТАНЦІЇ ТЕХНІЧНОГО ОБСЛУГОВУВАННЯ

2.1 Особливості програмної реалізації додатків для ОС Android. Обгрунтування вибору технологій Java і SQLite

Android Studio — нове і повністю інтегроване середовище розробки додатків, створене компанією Google для операційної системи Android. Даний продукт покликаний забезпечити розробників новими інстр ументами для створення додатків, а також надати альтернативу Eclipse, що на даний момент є найбільш популярним середовищем розробки.

При створенні нового проекту в Android Studio, буде показана структура проекту з усіма файлами, що містяться в каталозі SDK. Цей перехід до системи управління Gradle надає процесу розробки ще більшу гнучкість.

Android Studio дозволяє побачити будь-які візуальні зміни, які відбуваються в реальному часі у додатку. Також можна побачити, як додаток буде одночасно виглядати на різних пристроях під управлінням Android, з різними настройками і розширенням екрану.

Продукт також володіє новими інструментами для пакування та маркування коду. Це дозволяє швидко зорієнтуватись у проекті, з великою кількістю коду. У програмі також задіяна функція перетягування, завдяки якій можна переміщати компоненти за допомогою інтерфейсу користувача.

Android Studio забезпечує:

- надійне і просте середовище розробки;

- швидку перевірку продуктивності програми на різних типах пристроїв;

- помічники і шаблони для загальних елементів програмування;

- повнофункціональний редактор з безліччю додаткових інструментів, що сприяють прискоренню розробки додатків.

Android SDK — це емулятор мобільної платформи Android з набором бібліотек, яка використовується в мобільних телефонах та планшетних комп'ютерах. Після установки емулятора стає можливим запуск застосунків для телефонів цієї платформи безпосередньо на комп'ютері.

Структура програми.

Додатки можуть бути простими і складними, але їх створення буде однаковим. Є обов'язкові елементи додатків, а є вибіркові, які використовуються в міру необхідності.

Життєвий цикл програми.Життєвий цикл програми в Android жорстко контролюється системою і залежить від потреб користувача, доступних ресурсів тощо. Наприклад, користувач хоче запустити браузер. Рішення про запуск програми приймає система. Вона підпорядковується певним заданим і логічним правилам, що дозволяє визначити, чи можна завантажити, призупинити програму або припинити її роботу. В Android ресурси обмеженіші, тому Android жорсткіше контролює роботу додатків.

Основні методи життєвого циклу програми представлені на рисунку 2.1.

Викликається при створенні Activity. Цей метод приймає один параметр – об'єкт Bundie, що містить попередній стан активності (якщо цей стан було збережено). Система може запускати і зупиняти поточні вікна в залежності від подій, що відбуваються. Android викликає метод onCreate () після запуску чи перезапуску Activity. Усередині цього методу налаштовують статичний інтерфейс Activity. Ініціалізує статичні дані активності, що пов'язують дані зі списками тощо. Пов'язує з необхідними даними і ресурсами. Задає зовнішній вигляд через метод setContentView ().

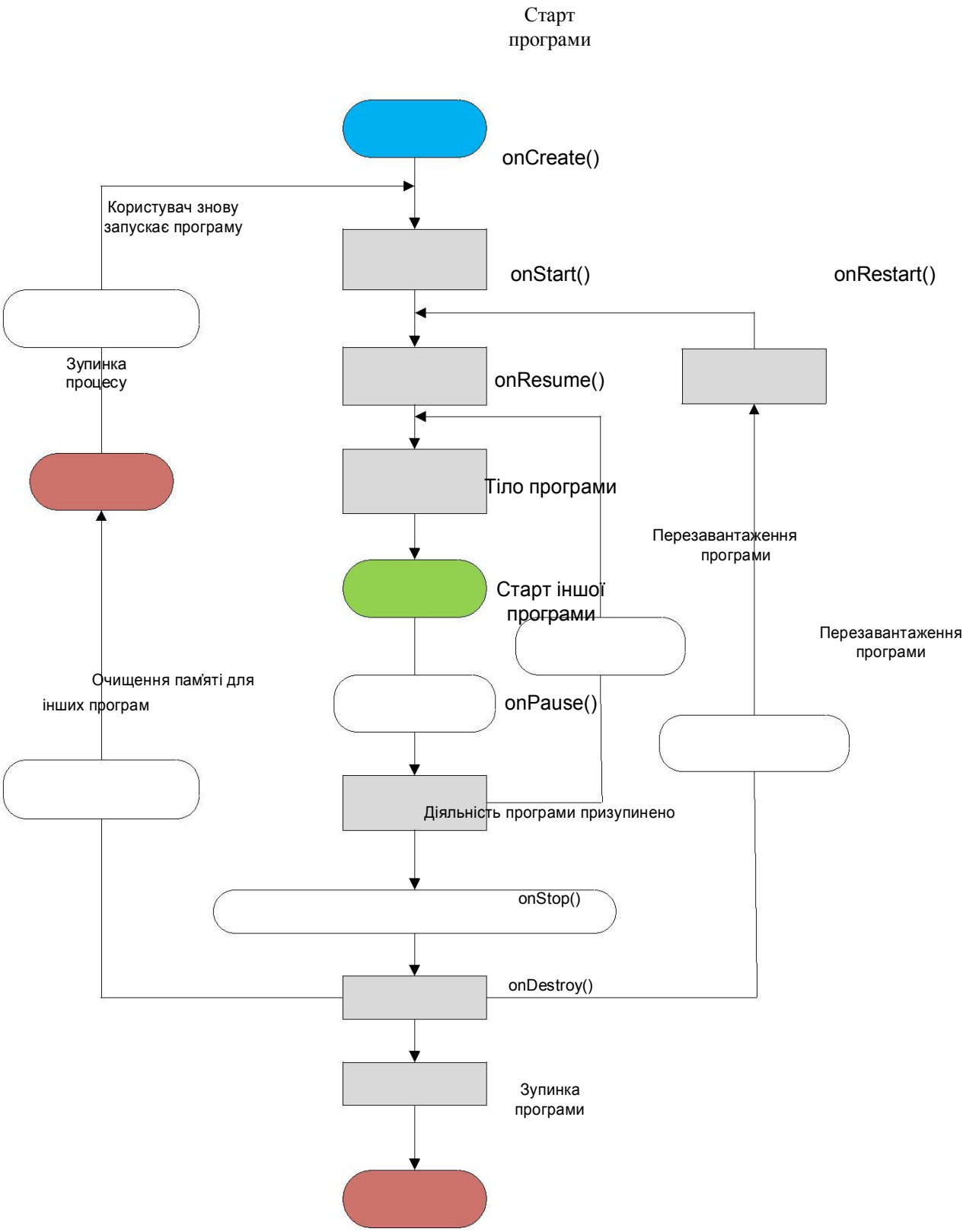


Рисунок 2.1. – Життєвий цикл програми на базі ОСAndroid

onStart. Перед onStart () не обов’язково повинен йти onCreate (), так як onStart () може викликатися і для відновлення роботи припиненого додатку (додаток зупиняється методом onStop ()). При виклику onStart () вікно ще не видно користувачеві. Викликається безпосередньо перед тим, як активність стає видимою користувачеві. Супроводжується викликом методу onResume (), якщо активність стає явною, або викликом методу onStop (), якщо стає прихованою.

onResume. Метод onResume () викликається після onStart (), навіть коли вікно працює в пріоритетному режимі і користувач може його спостерігати. У цей момент користувач взаємодіє із створеним вікном, додаток отримуємонопольні ресурси. Запускає відтворення анімації, аудіо та відео. Також може викликатися після onPause ().

onPause. Коли користувач вирішує перейти до роботи з новим вікном, система викличе для переривання вікна метод onPause (). По суті відбувається згортання активності. Зберігає незафіксовані дані. Деактивує і випускає монопольні ресурси. Зупиняє відтворення відео, аудіо та анімацію. Від onPause () можна перейти до виклику або onResume (), або onStop ().

onStop. Метод onStop () викликається, коли вікно стає невидимим для користувача. Це може статися при його знищенні, або якщо була запущена інша активність (існуюча або нова), яка перекрила вікно поточної активності. Завжди супроводжує будь-який виклик методу onRestart (), якщо активність повертається, щоб взаємодіяти з користувачем, або методу onDestroy (), якщо ця активність знищується.

onRestart. Якщо вікно повертається в пріоритетний режим після виклику onStop (), то в цьому випадку викликається метод onRestart (). Тобто викликається після того, як активність була зупинена і знову була запущена користувачем. Завжди супроводжується викликом методу onStart ().

onDestroy. Метод викликається після закінчення роботи активності, при виклику методу finish () або у випадку, коли система знищує цей

екземпляр активності для звільнення ресурсів. Ці два сценарії знищення можна визначити викликом методу isFinishing (). Викликається перед знищенням активності. Це останній запит, який отримує активність від системи. Якщо певне вікно знаходиться у верхній позиції стеку, але є невидимим для користувача і система вирішує завершити це вікно, викликається методonDestroy (). В цьому випадку метод видаляє всі статичні дані активності. Повертає всі використані ресурси.

У середовищі розробки додатків Android Studio в встановленому плагіні Android SDK є графічний редактор xml-файлів Android Editors (рисунок 2.2), який дозволяє вибрати потрібні об’єкти для розміщення їх на xml-формі. Вона використовує посторінкове групування об’єктів. В редакторі знаходиться набір закладок – Form Widget, Text Field, Composite, Image & Media тощо.

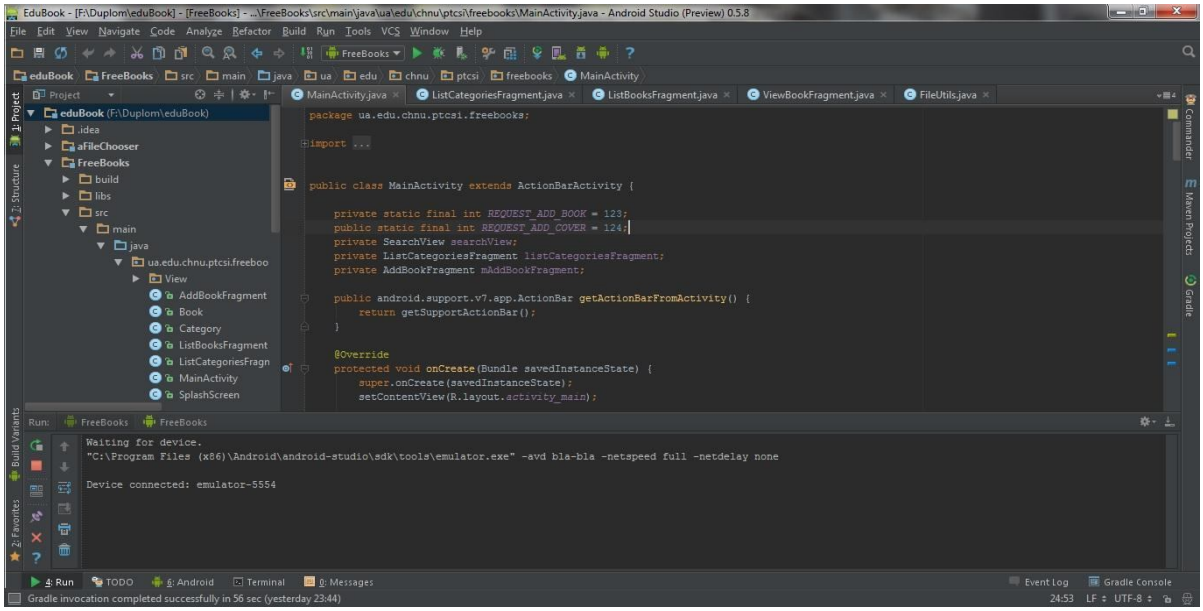


Рисунок 2.2 – Головне вікно середовища Android Studio

За замовчуванням в Android використовується SQLite - популярна і проста в освоєнні реляційна база даних. SQLite підтримує типи TEXT (аналог String в Java), INTEGER (аналог long в Java) і REAL (аналог double в Java).

Решта типів слід конвертувати, перш ніж зберігати в базі даних [11]. Бібліотека Android містить абстрактний клас SQLiteOpenHelper, за допомогою якого можна створювати, відкривати і оновлювати бази даних. Це основний клас, з яким здійснюється робота в проекті. При реалізації цього допоміжного класу ховається логіка, на основі якої приймається рішення про створення або оновлення бази даних перед її відкриттям.

Клас SQLiteOpenHelper містить два абстрактних методи: onCreate () - метод, який викликається при першому створенні бази даних, а також onUpgrade(), який викликається при модифікації бази даних.

У додатку створений власний клас diplom_Data_Base, успадкований від SQLiteOpenHelper. В цьому класі реалізовані методи onCreate () і onUpgrate (). У них описана в них логіка створення і модифікації бази даних.

У фрагменті коду, наведеному в Додатку Б, описано створення бази даних за допомогою методу onCreate, що включає створення трьох таблиць. У методі onCreate () таблиці заповнюються початковими значеннями. При цому метод onCreate викликається тільки один раз при створенні бази даних.

Запит до бази даних виконується за допомогою виклику SQLiteDatabase :: query()

В результаті виконання запитів повертається об'єкт Cursor, що містить таблицю з результатами запиту. Cursor передбачає послідовну роботу з рядками результату. У кожен момент часу активний один рядок, на яку посилається покажчик. Перебираючи записи послідовно, можна отримати доступ до даних.

2.2 Програмна реалізація бази даних та тестування додатку

Для реалізаці бази даних, яка використовується в проектів, використано СУБД MySQL.

На рисунку 2.3 представлено реалізацію відношення act, яке використовується для зберігання інформації про акти виконаних робіт в середовищі MySQL та його відповідну DDL інтерпретацію.

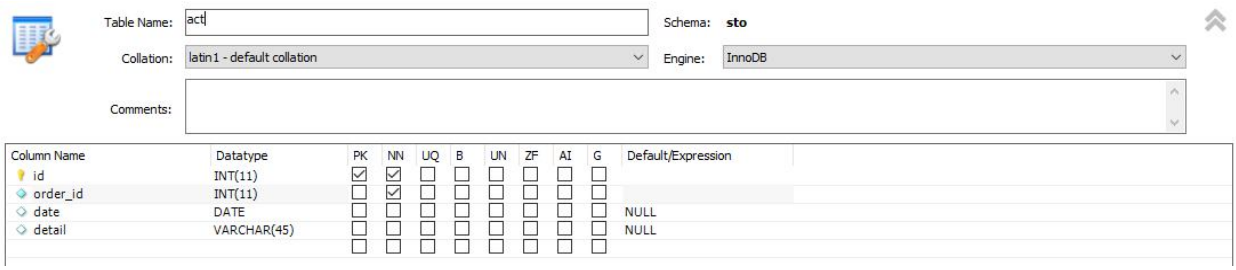


Рисунок 2.3 – Реалізація відношення act в середовищі MySQL

```
CREATE TABLE `act` (
  `id` int(11) NOT NULL,
  `order_id` int(11) DEFAULT NULL,
  `date` date DEFAULT NULL,
  `detail` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `a_idx` (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.4 представлено реалізацію відношення avto в середовищі MySQL та його відповідну DDL інтерпретацію. Це відношення використовується для збереження інформації про транспортні засоби.

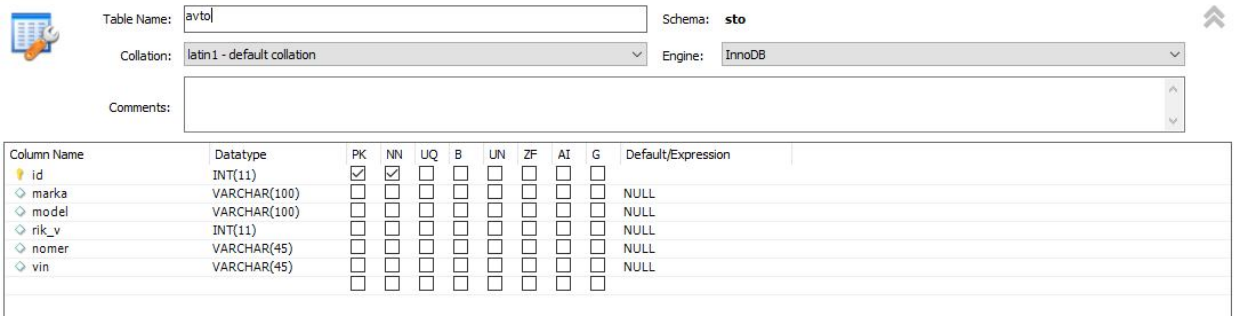


Рисунок 2.4 – Реалізація відношення avto в середовищі MySQL

```
CREATE TABLE `avto` (
  `id` int(11) NOT NULL,
  `marka` varchar(100) DEFAULT NULL,
  `model` varchar(100) DEFAULT NULL,
  `rik_v` int(11) DEFAULT NULL,
  `nomer` varchar(45) DEFAULT NULL,
  `vin` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.5 представлено реалізацію відношення client в середовищі MySQL та його відповідну DDL інтерпретацію.

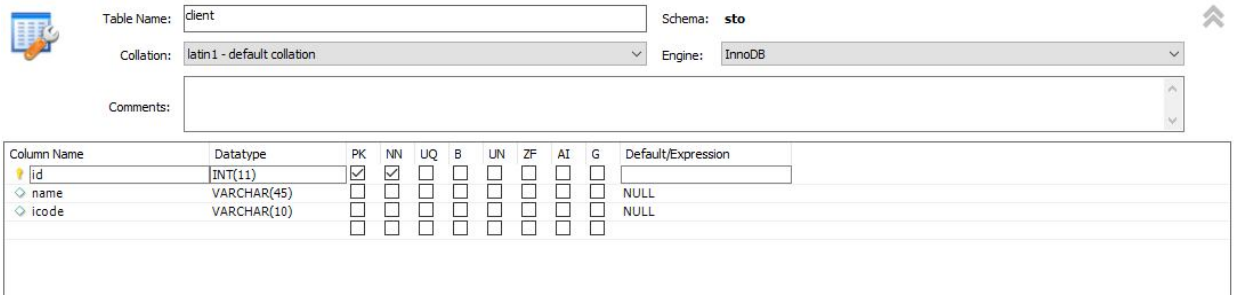


Рисунок 2.5 – Реалізація відношення client в середовищі MySQL

```
CREATE TABLE `client` (
  `id` int(11) NOT NULL,
  `name` varchar(45) DEFAULT NULL,
  `icode` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.6 представлено реалізацію відношення engineer в середовищі MySQL та його відповідну DDL інтерпретацію.

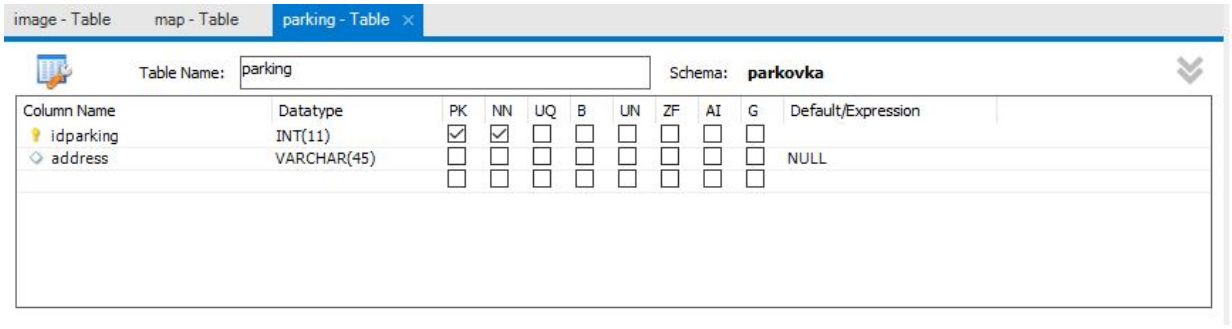
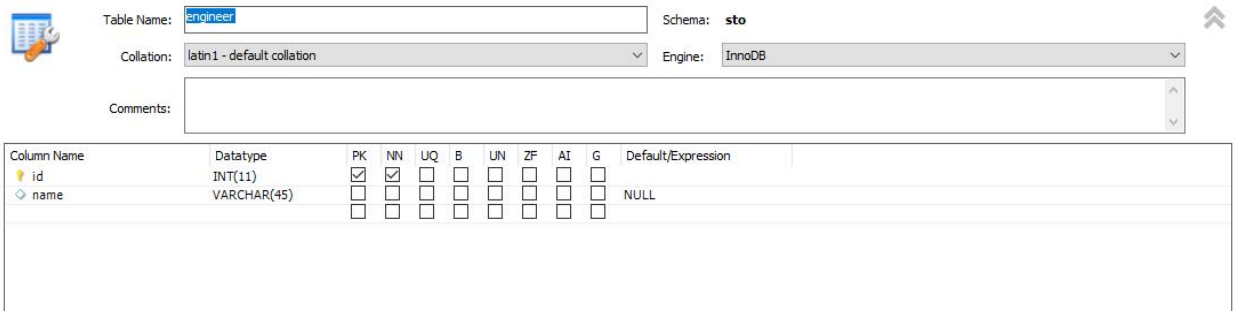


Рисунок 2.6 – Реалізація відношення engineer в середовищі MySQL




```
CREATE TABLE `engineer` (
  `id` int(11) NOT NULL,
  `name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.7 представлено реалізацію відношення master в середовищі MySQL та його відповідну DDL інтерпретацію.

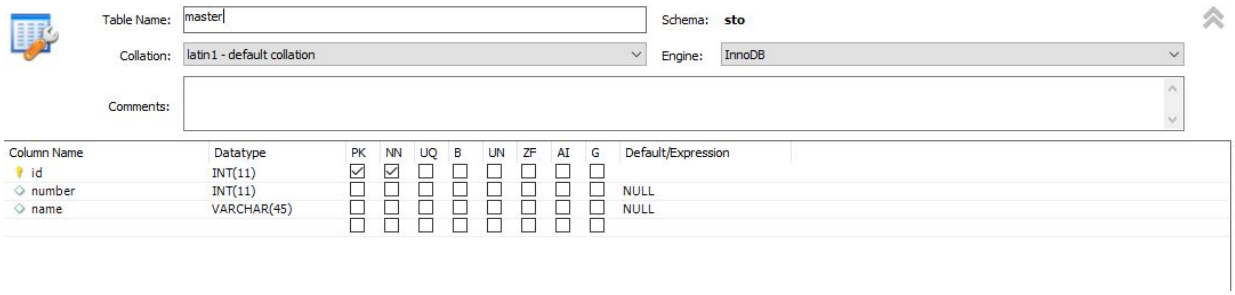


Рисунок 2.7 – Реалізація відношення master в середовищі MySQL

```
CREATE TABLE `engineer` (
  `id` int(11) NOT NULL,
```

```
  `name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.8 представлено реалізацію відношення name_ref в середовищі MySQL та його відповідну DDL інтерпретацію.

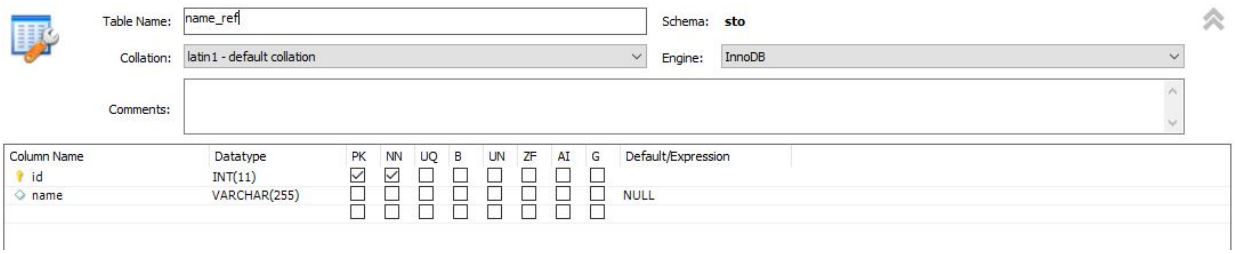


Рисунок 2.8 – Реалізація відношення name_ref в середовищі MySQL

```
CREATE TABLE `name_ref` (
  `id` int(11) NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.9 представлено реалізацію відношення order в середовищі MySQL та його відповідну DDL інтерпретацію.

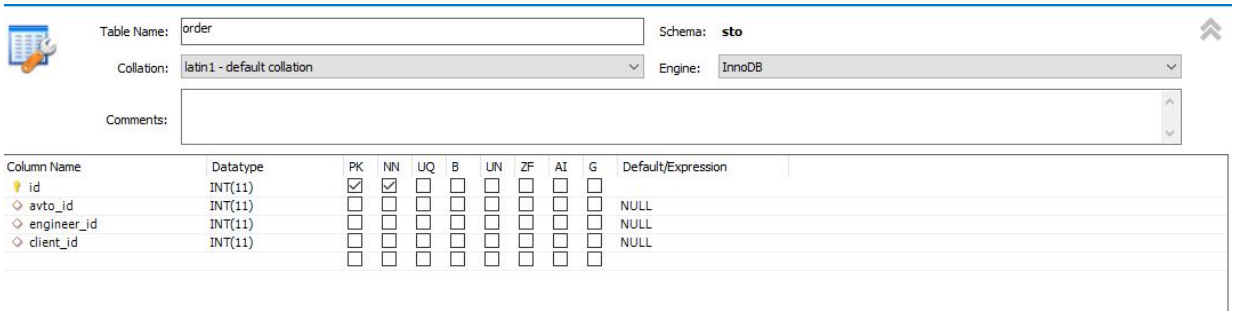


Рисунок 2.9 – Реалізація відношення order в середовищі MySQL

```
CREATE TABLE `order` (
  `id` int(11) NOT NULL,
  `avto_id` int(11) DEFAULT NULL,
```

```
`engineer_id` int(11) DEFAULT NULL,
`client_id` int(11) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `a_idx` (`avto_id`),
KEY `b_idx` (`client_id`),
KEY `c_idx` (`engineer_id`),
CONSTRAINT `a` FOREIGN KEY (`avto_id`) REFERENCES `avto` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT `b` FOREIGN KEY (`client_id`) REFERENCES `client` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT `c` FOREIGN KEY (`engineer_id`) REFERENCES `master` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT `d` FOREIGN KEY (`id`) REFERENCES `act` (`order_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

На рисунку 2.10 представлено реалізацію відношення order_details в середовищі MySQL та його відповідну DDL інтерпретацію.

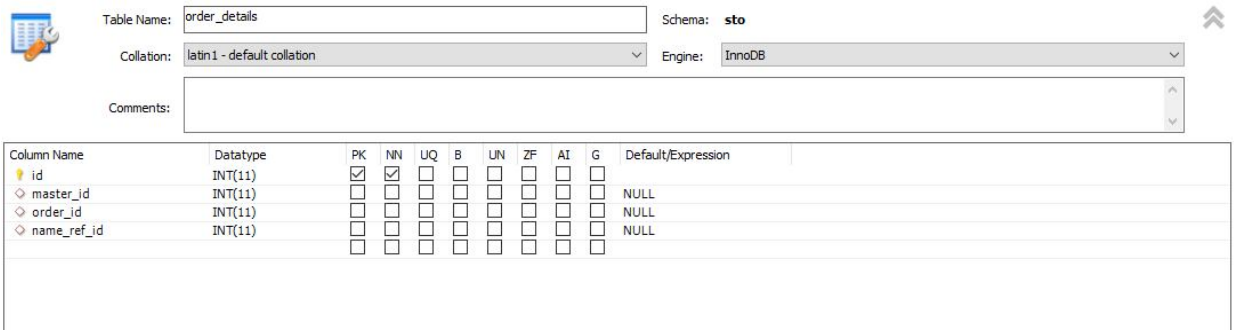


Рисунок 2.10 – Реалізація відношення order_details в середовищі MySQL

```
CREATE TABLE `order_details` (
`id` int(11) NOT NULL,
`master_id` int(11) DEFAULT NULL,
```

```
`order_id` int(11) DEFAULT NULL,
`name_ref_id` int(11) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `a1_idx` (`order_id`),
KEY `a2_idx` (`name_ref_id`),
KEY `a3_idx` (`master_id`),
CONSTRAINT `a1` FOREIGN KEY (`order_id`) REFERENCES `order` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT `a2` FOREIGN KEY (`name_ref_id`) REFERENCES `name_ref` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT `a3` FOREIGN KEY (`master_id`) REFERENCES `master` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT `a4` FOREIGN KEY (`master_id`) REFERENCES `engineer` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

За роботу з БД відповідає спеціальний клас DBhelper (рисунок 2.11), що є підкласом MySQLOpenHelper. Даний клас містить 2 обов'язкових методи:

• onCreate () відповідальний за створення самої БД.

• onUpgrade () використовується для поновлення структури БД у старих користувачів в разі її зміни.

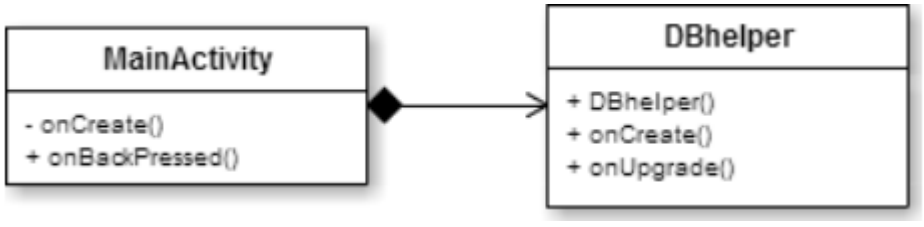


Рисунок 2.11 – Діаграма класів - робота з базою даних

Після створення екземпляра класу в MainActivity викликається метод getReadableDatabase (), який створює нову БД або відкриває існуючу.

На рисунку 2.12 представлена схема БД додатку з відповідними полями та типами, що зберігає інформацію про відповідні об’єкти.

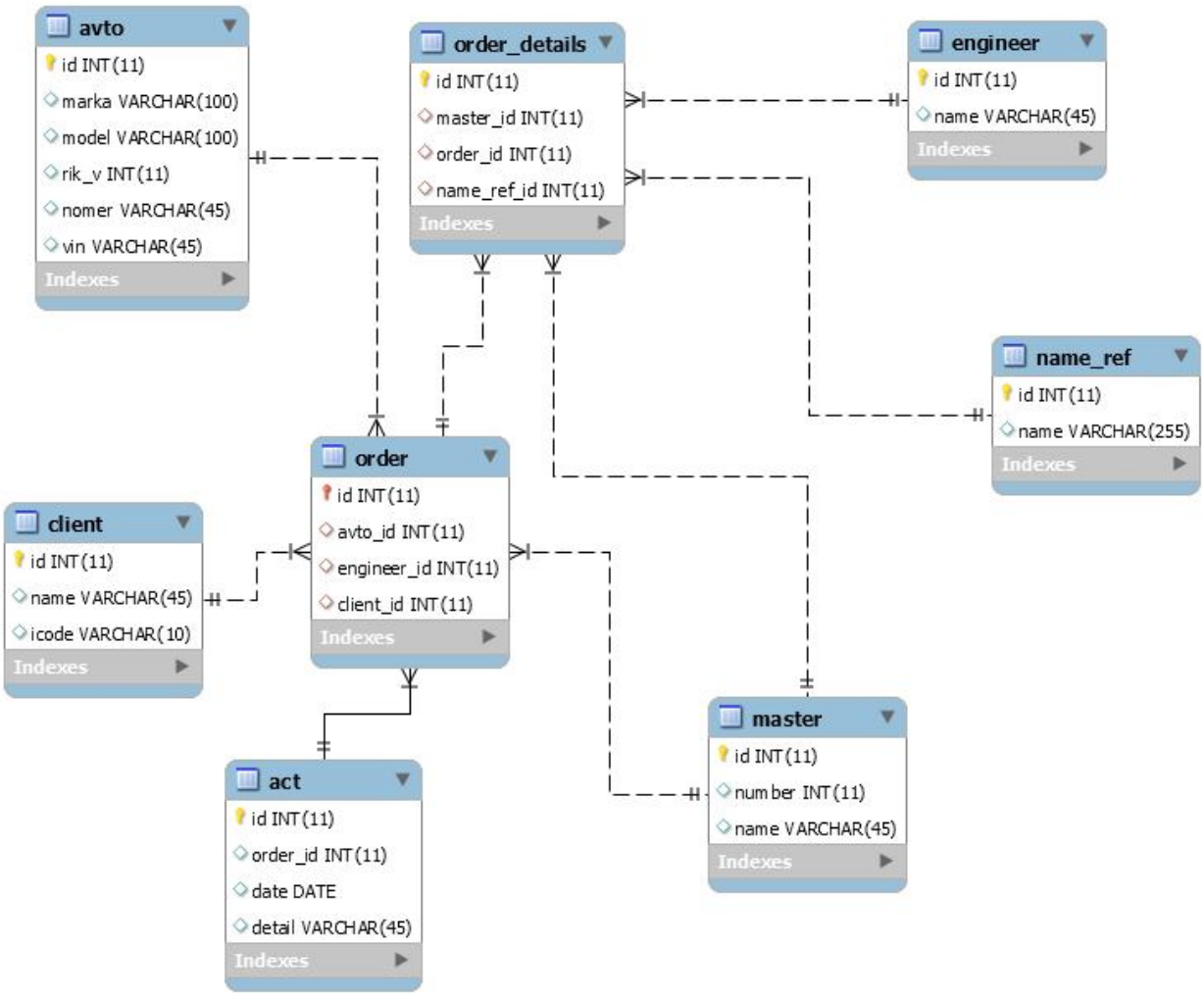


Рисунок 2.12 – Схема бази даних в середовищі MySQL Workbench

У даному озділі обґрунтовано технологію, мову програмування та розроблено програмну систему. Обґрунтовано засоби розробки додатку та здійснено опис основних програмних модулів системи.

У процесі розробки програми проводилося поетапне тестування з метою виявлення програмних помилок і невідповідностей ТЗ (з технічним завданням). Для цього нами були створені емулятори смартфона і планшета з різними діагоналями екрана для різних версій Android.

Тестований програмний продукт послідовно запускався на цих емуляторах, його поведінка аналізувалася, і при необхідності по результатах аналізу вносилися зміни в код [12].

Для тестування окремих модулів роботи з базою даних в текст програми були внесені спеціальними функціями, які аналізувати базу даних і, при підозрі на помилку, виводилися повідомлення в системний журнал. Вони також відомі як юніт-тести. Наприклад, при змінах в базі даних проводилася перевірка цілісності бази даних (перевірка на відповідність ключів - індексам), після чого при необхідності виводилося повідомлення в системний log.

Були проведені наведені нижче тести.

1. Кожна активність була піддана юніт-тестування з метою виявлення помилок, викликаних невідповідністю очікуваних і отриманих параметрів. Для цього для кожної активності був створений спеціальний юніт-клас, який посилає в активність різні вірні і неправильні запити. При аномальній поведінці активності або її збої, мною аналізувалася поведінка і помилка виправлялася.

2. У базу даних навмисно вносилися неприпустимі дані в відповідні поля, які могли бути невірно інтерпретовані програмою. Потім мною аналізувалася поведінка активності у час обробки неприпустимих даних.

3. Додаток було запущено на пристроях, що працюють під керуванням різних версій Android з метою виявлення особливостей роботи програми, запущеного в різних операційних системах.

4. Після завершення циклу розробки, програмний продукт тестувався на реальних пристроях. За результатами тестування була додана віртуальна кнопка «Меню» для пристроїв, які не мають апаратних кнопок.

Тестування є важливою частиною розробки програмного забезпечення. Воно дозволяє виявити помилка та дефекти в роботі програм. Існує безліч

видів тестувань, кожен з яких дозволяє перевірити на працездатність програму з тієї чи іншої сторони[14].

Враховуючи специфіку поставленого завдання, для того щоб максимально точно перевірити функціонування розробленого завдання в різних умовах було вирішено використати наступні види тестування:

- Модульне тестування.
- Функціональне.
- Тестування продуктивності.

Модульне тестування. В першу чергу було проведено модульне тестування. Воно дозволяє протестувати найменші частини додатку, які виконують якусь конкретну функцію. Модульне тестування для кожного модуля проводиться в ізоляції ввід інших частин системи [15].

Для виконання модульного тестування було обрано Java JUnit. JUnit - бібліотека для модульного тестування програмного забезпечення на мові Java. JUnit - простий і в той же час дуже потужний інструмент для написання unit тестів. У даному розділі ми будемо використовувати останню на даний момент версію JUnit, а саме 7.1.0.

Спочатку було протестовано конструктор класу Main_activity. Це проводиться для того, щоб визначити чи встановлюються іконки обраної категорії та чи створюється об'єкт, при вказанні великих розмірів цієї ж іконки.

```
@Test
public void testDrawConstructorS() {
 DrawablePlaceIcon d1 = new DrawablePlaceIcon ("sto", "TestObject", 1000, 50, 500, getResources(), 100);
DrawablePlaceIcon d2 = new DrawablePlaceIcon ("sto", "TestObject2", 1000, 50, 500, getResources(), 500);
DrawablePlaceIcon d3 = new DrawablePlaceIcon ("sto", "TestObject2", 1000, 50, 500, getResources(), 1000);
        assertNotNull(d1);
        assertNotNull(d2);
        assertNotNull(d3);

}
```

Виконання данного тесту показало різні результати на різних

пристроях. На пристроях, з малим об'ємом оперативної пам'яті він був провалений, оскільки не вистарчало пам'яті для відмалювання піктограми. Але на пристроях, де її об'єм був більший 256 Мб, тест виконався успішно.

Наступним тестом буде перевірка конструктора на приймання невалідної інформації, такої як невірний формат картинки, великий розмір файлу та пуста назва об’єкту.

```
@Test(expected = Exception.class)
    public void testDrawConstructorEr() {
        DrawablePlaceIcon d1 = new DrawablePlaceIcon (“sto”, “TestObject”, 1000, 50, 500, getResources(), 100);
          DrawablePlaceIcon d2 = new DrawablePlaceIcon (“”, “TestObject2”, 1000, 50, 500, getResources(), 500);
    DrawablePlaceIcon d3 = new DrawablePlaceIcon (“sto”, “TestObject2”, -50, 50, 500, getResources(), 1000);
    DrawablePlaceIcon d4 = new DrawablePlaceIcon (“sto”, “TestObject2”, 1000, 50, 500, getResources(), -100);
        }
```

Цей тест був провалений, оскільки в класі не було проведено необхідних перевірок на коректність параметрів, і видавалися необроблені виключні ситуації. Для вирішення цієї проблеми, у разі отримання некоректних параметрів, повертається пустий об’єкт.

Також необхідно провести тест методів класу Restoran, щоб дізнатися, чи не повертає він невалідної інформації у разі передачі йому різної якості параметрів.

Для цього також був створений тестовий метод, у якому по черзі були передані некоректні параметри. Код методу представлений нижче.

```
@Test(expected = Exception.class)
    public void testDrawConstructorEr() {
                if(position <-1)
                        fail("Negative value");
                int position2 = Stos.getOnScreenLocation(null, new Sto(500 Mb);
                if(position <-1)
                        fail("Negative value");
```

Цей тест був виконаний успішно, оскільки в класі передбачена можливість вводу некоректних параметрів. В такому разі метод повертає результат -1, який і використаний для перевірки в даному тесті.

Функціональне тестування. Наступним було проведено функціональне тестування. Функціональне тестування дозволяє перевірити чи відповідають функції розробленого ПЗ вимогам, що були зазначені в специфікації вимог [17]. Для цього було створено тестові випадки для кожного з варіантів використання.

За результатами функціонального тестування 3 з 3-х тестових випадків пройшли успішно, отже – тестування можна вважати успішним – 100% тестових випадків пройшли. Це означає, що програма повністю реалізовує всі функції, що зазначені в вимогах до програмного продукту.

Тестування продуктивності. Тестування продуктивності проводиться з метою визначення, як швидко працює програма або її частина під деяким навантаженням та в певних умовах. Тестування продуктивності намагається враховувати продуктивність на стадії.

Важливим показником для розроблюваного додатку є показник використання інтернет трафіку. Адже, чим більше трафіку використає додаток – тим більше потрібно буде заплатити за нього, і тим більше часу буде очікувати користувач на отримання відповіді. Для цього проаналізуємо інтернет з’єднання пристрою під час виконання додатку. Для цього використаємо інструмент «NetworkStatisticTool», що входить до набору інструментів DDMS (DalvikDebugMonitor Service), що входять в пакет ADT (AndroidDevelopmentTools) для Eclipse [19]. Даний інструмент відображає обсяги обміну даних через мережу Інтернет в кожен момент часу. «NetworkStatisticTool» був запущений під час виконання розроблюваного додатка протягом 30 секунд, результати його виконання зображені на рисунку 2.13.

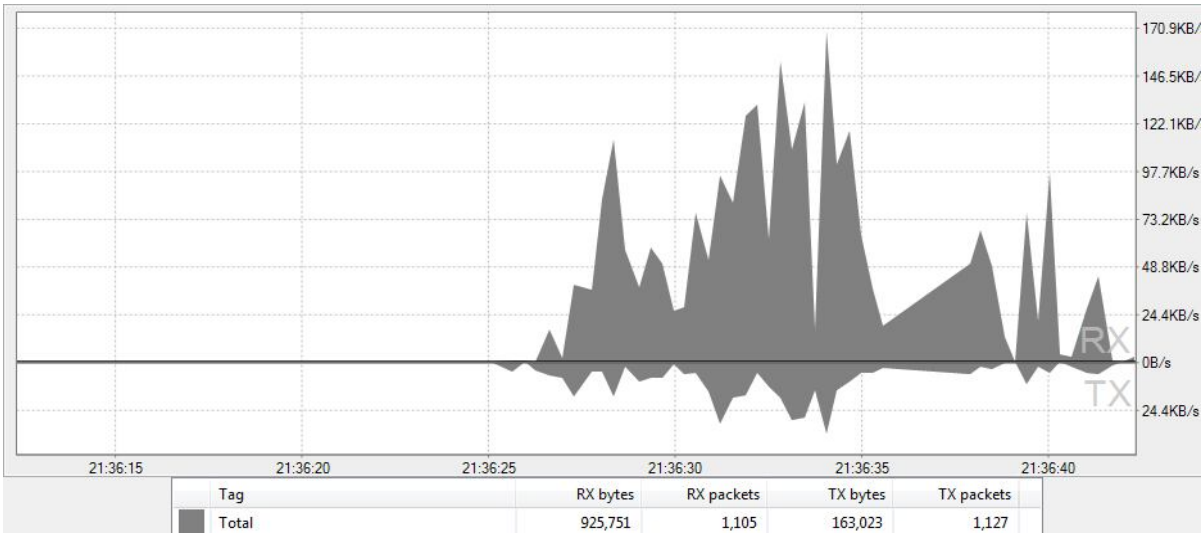


Рисунок 2.13 – Статистика використання інтернет з’єднання

Як бачимо, максимальна кількість використаного трафіка досягає 170 КБ/с. Такий результат тест показує під час отримання інформації про всі об’єкти однієї категорії. Проте в період, коли додаток отримує інформацію про конкретнуСТО, обсяг отриманих даних становить менше 5 кб/с. Враховуючи те, що на даний момент, середня швидкість мобільного інтернет з’єднання дорівнює приблизно 250-300 КБ/с, для даного додатку цього повністю хватає, і не завдаватиме проблем в його роботі.

2.3 Процес розгортання та налаштування додатку

Android Studio будує проект автоматично в процесі його зміни, а не по команді. Під час побудови інструментарій Android бере ваші ресурси, код і файл AndroidManifest.xml (що містить метадані додатку) і перетворює їх в файл .арк. Отриманий файл підписується налагоджувальний ключем, що дозволяє запускати його в емуляторі (рисунок 2.14).

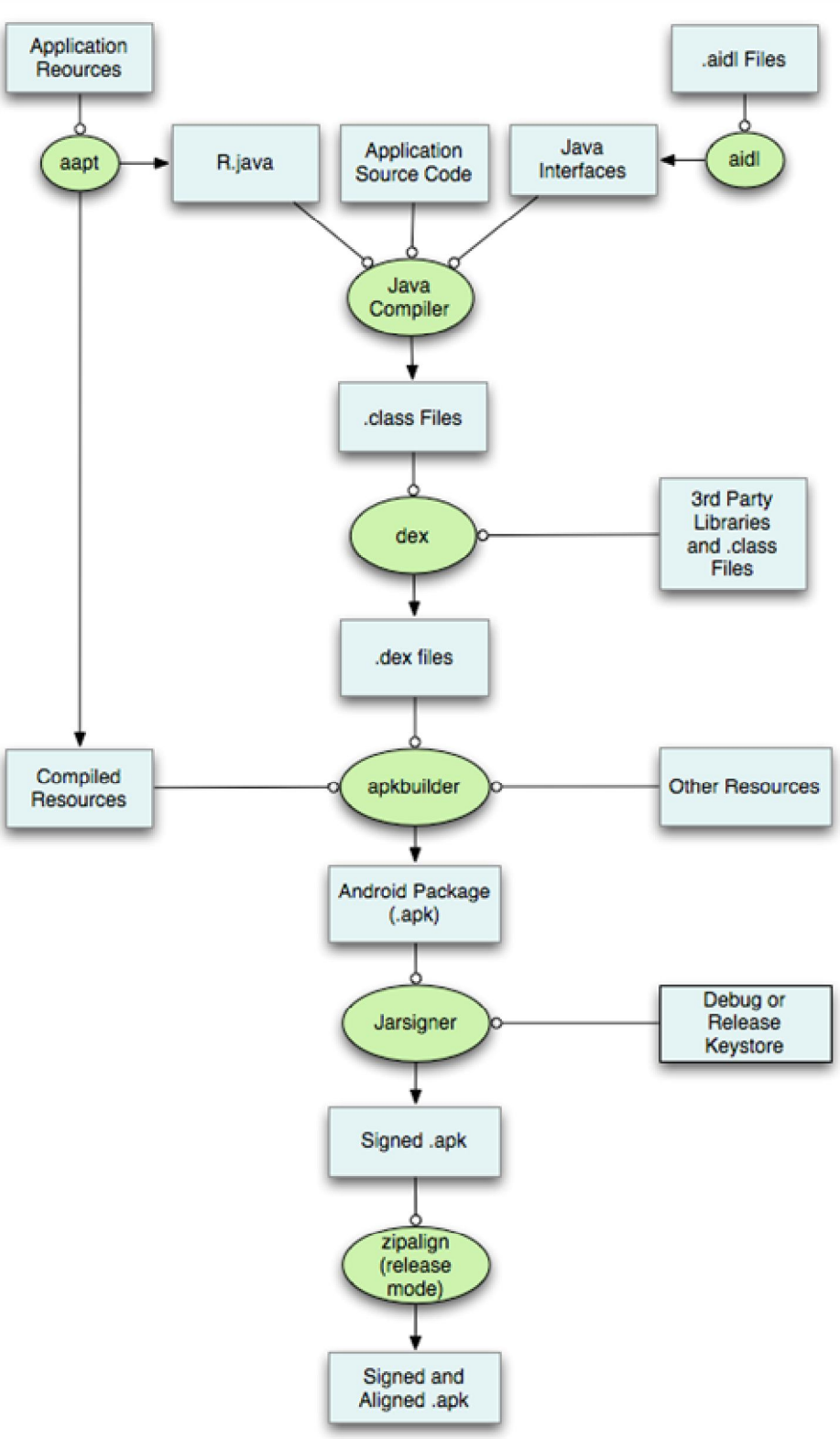


Рисунок 2.14 - Процес побудови додатку

Щоб поширювати файл .apk серед користувачів, необхідно підписати його ключем публікації. В процесі побудови утиліта aapt (Android Asset

Packaging Tool) компілює ресурси файлів макетів в більш компактний формат. Відкомпілювані ресурси упаковуються в файл .apk. Потім, коли метод setContentView (...) викликається в методі onCreate (...) класу QuizActivity. QuizActivity використовує клас Layoutlnflater для створення примірників усіх об'єктів View, визначених у файлі макета.

Компіляція вихідного коду в Android Studio проводиться з допомогою aapt (Android Asset Packaging Tool) - утиліти, яка шукає в проекті компільовані ресурси, такі як AndroidManifest.xml і xml файли з res / і компілює їх в бінарне представлення, а спочатку бінарні ресурси, такі як картинки і файли з / res / raw та / assets не компілюються.

Далі, ця утиліта генерує найважливіший клас R.java для вашого додатку, завдяки якому ви можете звертатися до ресурсів з вашого коду без будь-яких труднощів з читанням файлів, як скажімо с / assets.

До речі, крім компіляції xml-ресурсів, aapt створює файл resources.arsc, який представляє собою таблицю для маппінга ресурсів учас виконання додатку, туди входять всі ресурси з / res /, в тому числі і/ Res / raw /, вміст / assets не включається в таблицю. Утиліта aapt знаходиться в папці platform-tools Android SDK.

У файл classes.dex компілятор записує весь виконуваний код проекту. У підсумку ми отримуємо скомпільовані ресурси - xml-ресурси додатку, скомпільовані в бінарне представлення [8].

Збірка ресурсів робиться для того щоб отримати .apk файл. Цей файл ми зможемо запустити в емуляторі або на пристрої. Для збірки пакетів в файл послідовно використовуються кілька утиліт:

- apkbuilder - утиліта, якій на вхід подають скомпільовані ресурси, classes.dex і інші ресурси, а вона збирає з цього наш жаданий .apk файл. Утиліта лежить в папці tools Android SDK.

- jarsigner - це утиліта Oracle для підписання .jar архівів. Він підписує ваш .apk обраним вами ключем.

- Zipalign - утиліта, яка оптимізує .apk для більш швидкого запуску і меншого споживання ОЗП при роботі програми. Вона вирівнює вміст .apk для більш ефективної розархівації.

Зібраний додаток в файл .apk - це архів, що містить скомпільовані і нескомпільовані ресурси, classes.dex, resources.arsc,META-INF, AndroidManifest.xml і т.д. Формат .apk це надбудова над .jar, a.jar - надбудова над zip, так що, .apk ви можете відкрити zip архіватором.

Крім того, якщо є викладений додаток в Google Play, не можна оновити його, якщо нова версія підписана іншим сертифікатом. Так що потрібно підписати сертифікат, а так само не забути пароль до нього [15].

Debug або Release сховище ключів - сховище, з якого jarsigner візьме ключі для підпису додатка. Якщо ви збираєте Debugверсію (для запуску на емуляторі або підключеному пристрої), то .apk підписується debug ключем, в Windows він знаходиться в папці користувача / .android /.

Тепер запустимо додаток в емуляторі. Для цього в AVD Managerдодаємо пристрій (рисунок 2.15). Налаштовуємо пристрій вказуючи розширення екрана, версію Android, розмір пам’яті на диску.

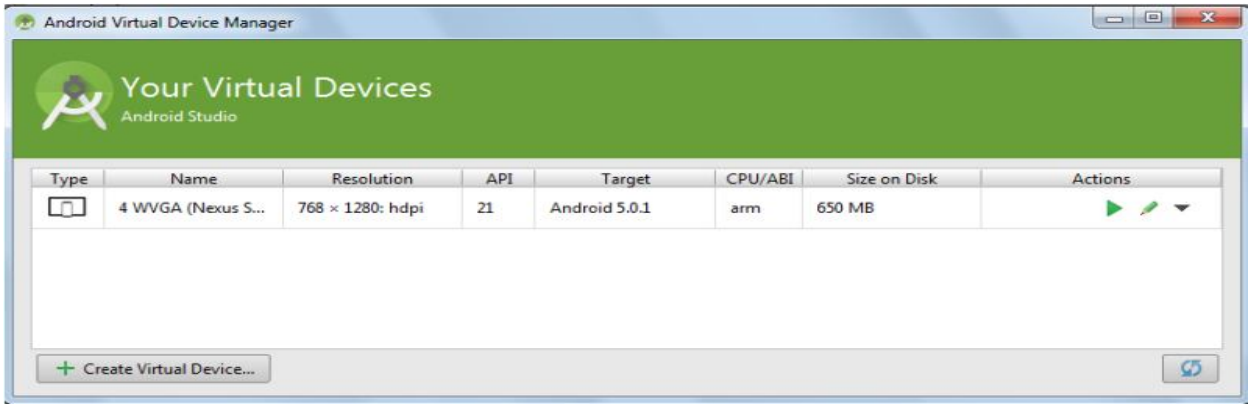


Рисунок 2.15 - Вікно AVD Manager

Після того, як додали пристрій, запускаємо віртуальний пристрій (рисунок 2.16).

Рисунок 2.16 - Віртуальне пристрій AVD Manager

Далі запускаємо Android Studio: Tools → Android і ставимо галочку навпроти рядка «Enable ADB Integration» (ADB - Android Debug Bridge). Після цього потрібно налаштувати Android Studio так, щоб при натисканні на зелену кнопочку «Run» додаток відразу встановлювався і запускався на підключеному смартфоні. Натискаємо Run → Edit Configurations. З'являється наступне вікно (рисунок 2.17):

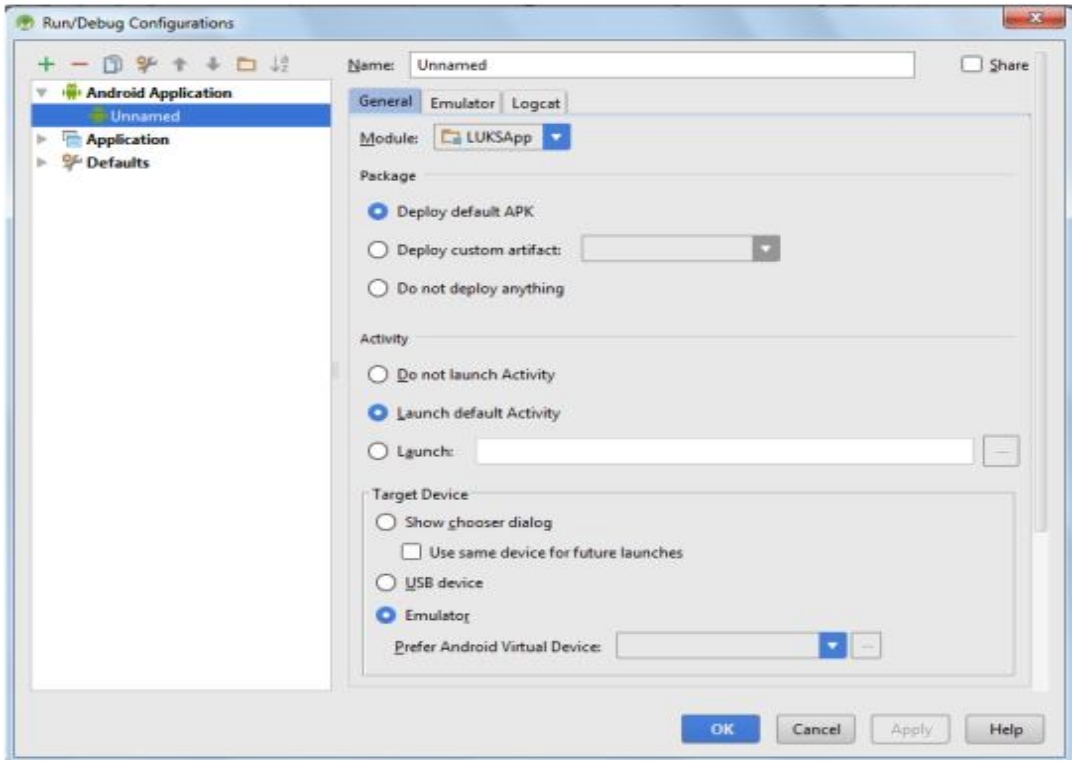


Рисунок 2.17 - Вікно Run / Debug Configuration

У блоці «Target Device» ставимо галочку на пункт «USB Device» і натискаємо OK. Після цього при натисканні на кнопку запуску програми (рисунок 2.18), додаток встановиться і запуститься на підключеному віртуальному пристрої (рисунок 2.19).

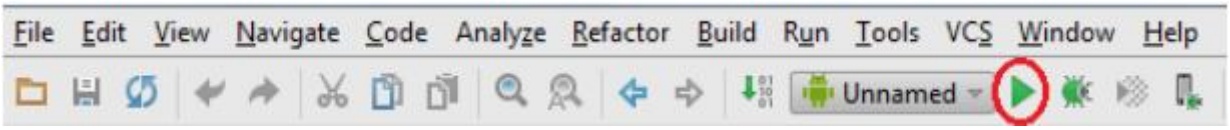


Рисунок 2.18 - ToolBar в Android Studio

Опис процесу запуску налагодження під Windows 7 в Android Studio, по кроках:

а) чи потрібно увімкнути режим налагодження USB. Для цього відкриваємо Налаштування → Параметри розробника → Ставимо галочку «Налагодження USB». Висвітиться попередження, підтверджуючи позитивно - «Так».

Рисунок 2.19 - Запуск додатку в емуляторі

б) треба на комп'ютері встановити драйвер Android ADB Driver. Це можна зробити, якщо завантажити і запустити програму UsbDriverTool-sfx.exe. Після запуску вказуємо папку, куди потрібно розпакувати утиліту,

наприклад c: \ temp, утиліта розпакується в папку C: \ temp \ UsbDriverTool \. В папці C: \ temp \ UsbDriverTool \ AndroidUsb \ знаходитиметься драйвер Android ADB Driver, який нам потрібен.

в) підключаємо смартфон з Android через USB до комп'ютера. На смартфоні повинен визначитися режим «Підключений як камера (PTP)».

Комп'ютер виявить новий пристрій, запуститься майстер установки драйвера.

г) Далі запускаємо Android Studio. Натискаємо Tools → Android і ставимо галочку навпроти рядка «Enable ADB Integration» (ADB - Android Debug Bridge). Після цього потрібно налаштувати Android Studio так, щоб при натисканні на зелену кнопочку «Run» додаток відразу встановлювався і запускався на підключеному смартфоні. Натискаємо Run → Edit Configurations. З'являється наступне вікно (рисунок 2.20):

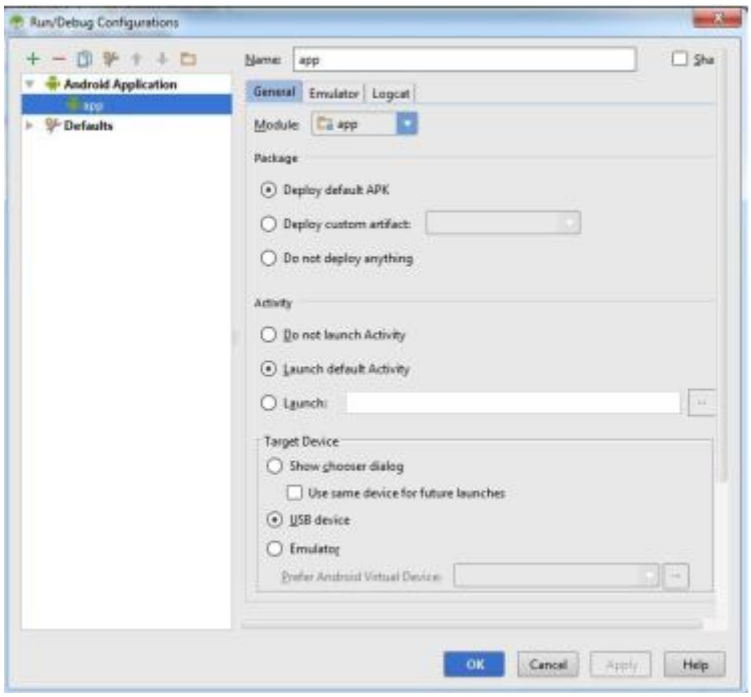


Рисунок 2.20 - Вікно Run / Debug Configuration

У блоці «Target Device» ставимо галочку на пункт «USB Device» і натискаємо OK.Після цього драйвер визначає пристрій, і при натисканні на кнопку запуску програми (рисунок 2.21), додаток встановиться і запуститься

на підключеному пристрої (рисунок 2.22).

Рисунок 2.21 - ToolBar в Android Studio

Після цього залишається тільки взяти в руки підключений апарат і тестувати додаток. Після запуску програми на пристрої, відкриється головне менюдодатку (рисунок 2.22).

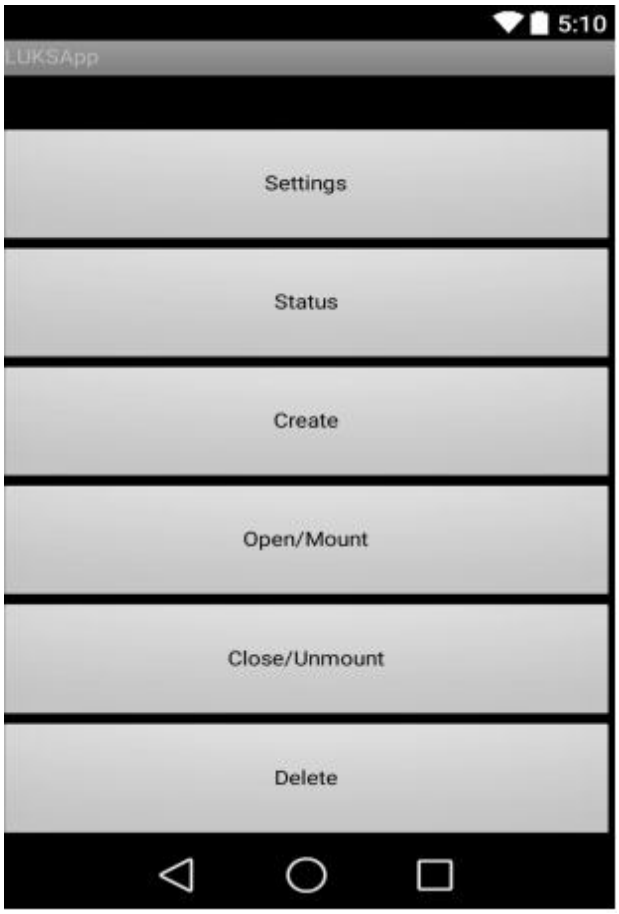


Рисунок 2.22 - Головне меню програми

В результаті виконання бакалаврської роботи був реалізований дуже просте в користуванні додаток, що дозволяє забезпечити зв’язок автосервісів з клієнтами для запису на техобслуговування. Величезний перелік послуг з обслуговування автомобілів. Співробітникові станції технічного обслуговування необхідно лише один раз ввести всі параметри свого СТО, а

в подальшому, за допомогою мобільного додатка в смартфоні, вести прийом записів ремонту автомобілів.

Будь-яких дзвінків співробітнику автосервісу немає необхідності здійснювати, як і самому клієнту, весь процес запису на ремонт проводиться онлайн через мобільний додаток, в разі необхідності можна скористатися чатом, в конкретному запису на ремонт.

Щоб записати автомобіль на ремонт в автосервіс, тепер не потрібно витрачати час на пояснення по телефону про деталі обслуговування, характеристиках автомобіля і погоджувати дату, додаток дозволяє протягом 1 хвилини вирішити всі питання про майбутній ремонт по конкретному клієнтові.

Розглянемо детальніше функціональність розробленого додатку. На рисунку 2.23 представлено сторінка з вибором станції технічного обслуговування.

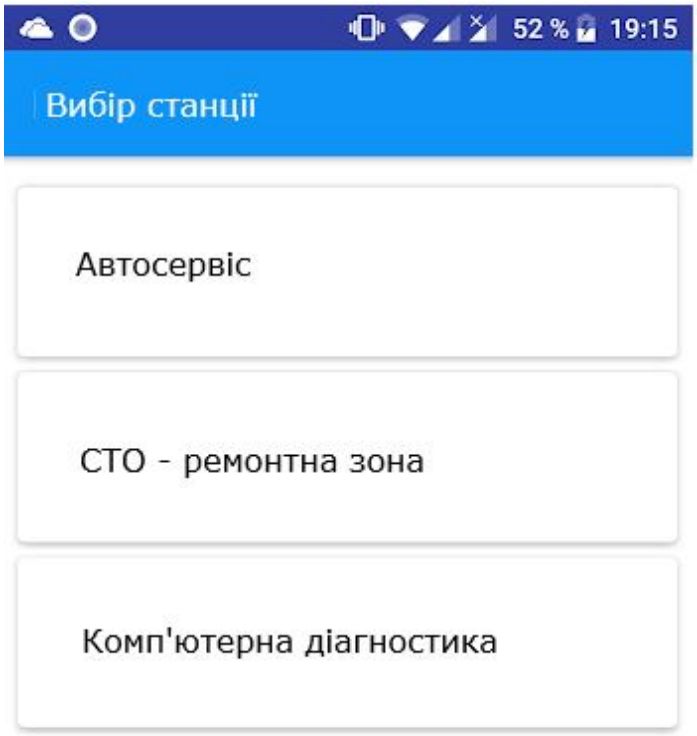


Рисунок 2.23 - Вибір станції технічного обслуговування

Після вибору станції необхідно пройти процедуру реєстрації в системі, яка дозволить в майбутньому формувати заявку на ремонт, а також відстежувати етапи виконання. На рисунку 2.24 представлено форму реєстрації за допомого мобільного додатку.

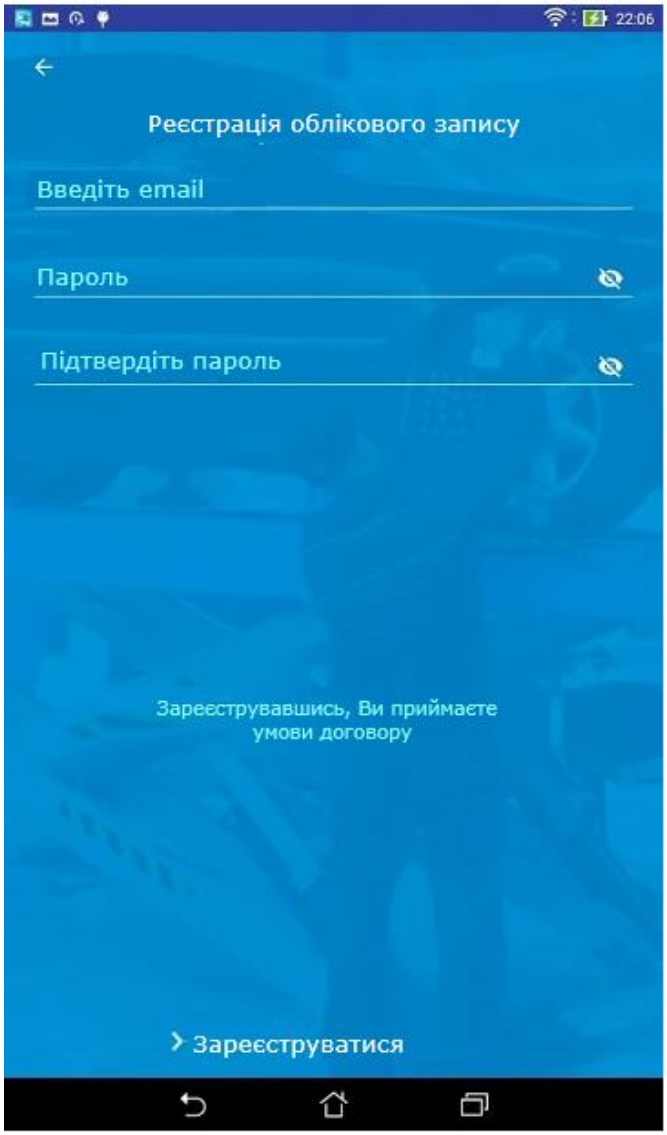


Рисунок 2.24 - Реєстрація за допомогою мобільного додатку

Після реєстрації у користувача зявляється можливість запису на ремонт. Відповідна форма представлена на рисунку 2.25. Ввівши відповідні поля, які стосуються марки автомобіль, року випуску, кузова, також необхідно обрати вид виконуваних робіт, узгодити графік виконання, а також переглянути ціновий діапазон, який може бути виставлений за виконаний на

станції технічного обслуговування за наданий перелік виконани послуг для клієнта.

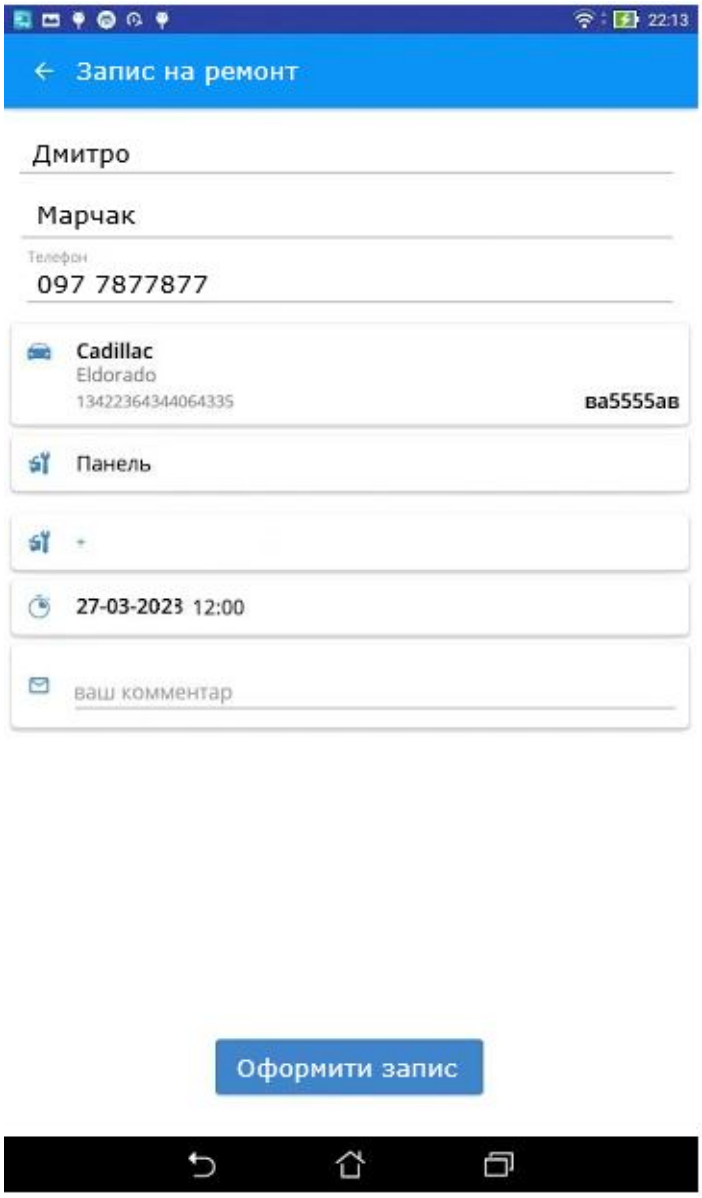


Рисунок 2.25 - Процедура запису на ремонт

Після запису на ремонт, сформована заявка надходить до відповідно автомайстра, який буде виконувати роботи. Майстер може змінювати статус виконання заявки, узгоджувати можливі уточнення або із замовником або із менеджером. На рисунку 2.26 представлено сторінку додатку із деталізацією запису на ремонт, строками виконання, ціною за наданий об’єм послуг в процесі ремонтних робіт.

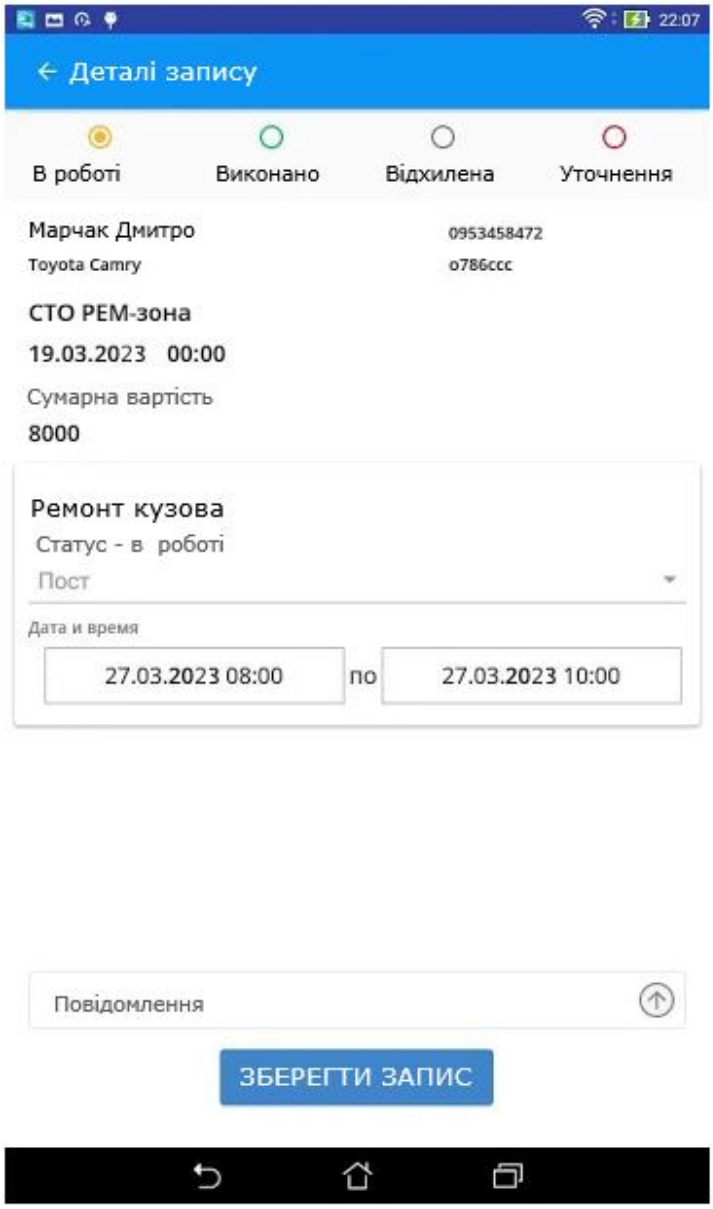


Рисунок 2.26 - Деталізація заявки на ремонт

Після узгодження всіх деталей заявка прикріпляється до ремонтного поста, який і буде далі здійснювати її послідовне опрацювання. На рисунку 2.27 представлено процес опрацювання заявки на пості. Тут необхідно також відзначити можливість зміни статуту виконання заявки, редагування цін за надані послуги. Після успішного виконання усіх робіт заявка переходить у статус виконаної.

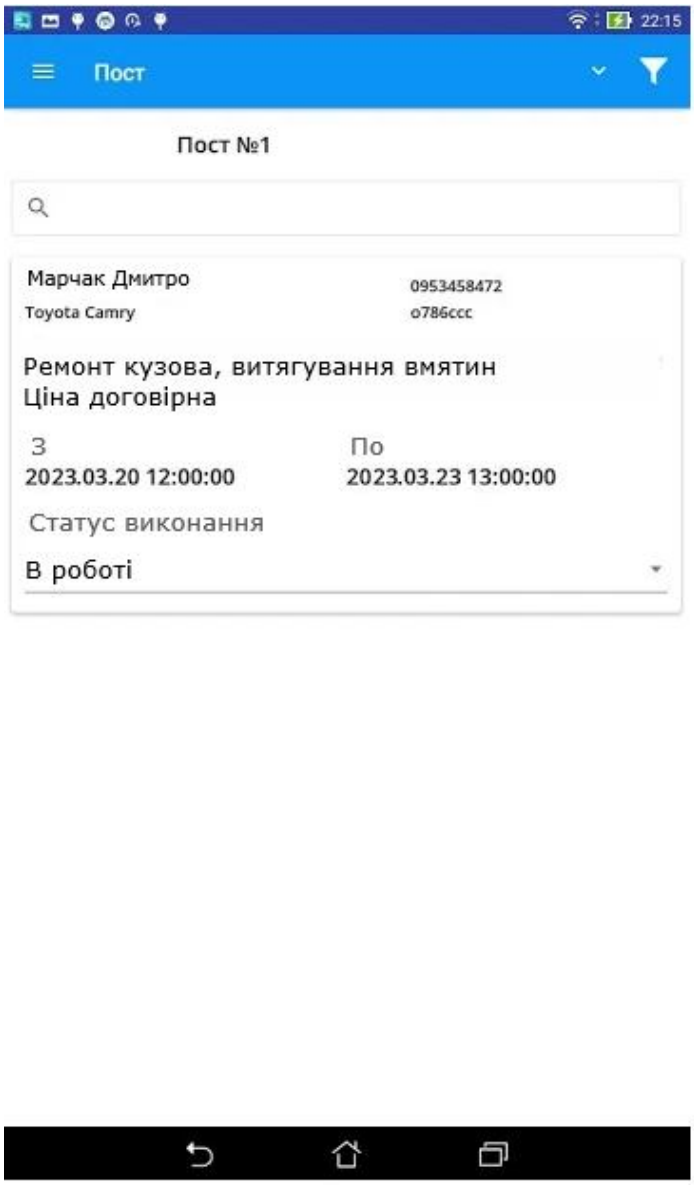


Рисунок 2.27 - Деталізація заявки на конкретному ремонтному пості

В додатку також є можливість оптимізувати роботу і менеджерів і відповідних майстрів. На рисунку 2.28 представлено сторінку, яка дозволяє фільтрувати відповідні ремонтні заявки, регулювати дату формування відповідних звітів. За допомогою відповідних відміток можна переглянути відповідний поточний статус функціонування станції технічного обслуговування, а це підвищує ефективність функціонування та зростання прибутків.

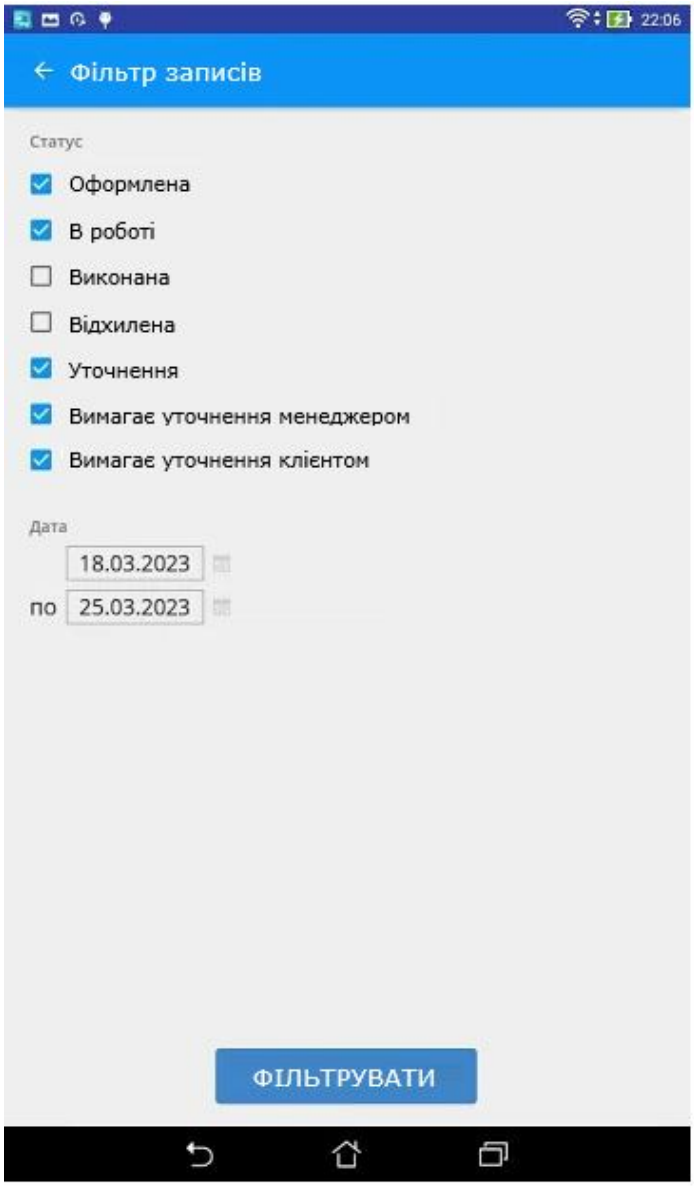


Рисунок 2.28 - Сторінка додатку із можливістю фільтрування заявок

Також необхідно відзначити, що і клієнти і менеджери можуть переглянути графіки роботи відповідних ремонтних постів. Клієнт може обрати пост, який йому найбільше підходить. На рисунку 2.29 відображено сторінку додатку із графіком виконання робіт по постах.

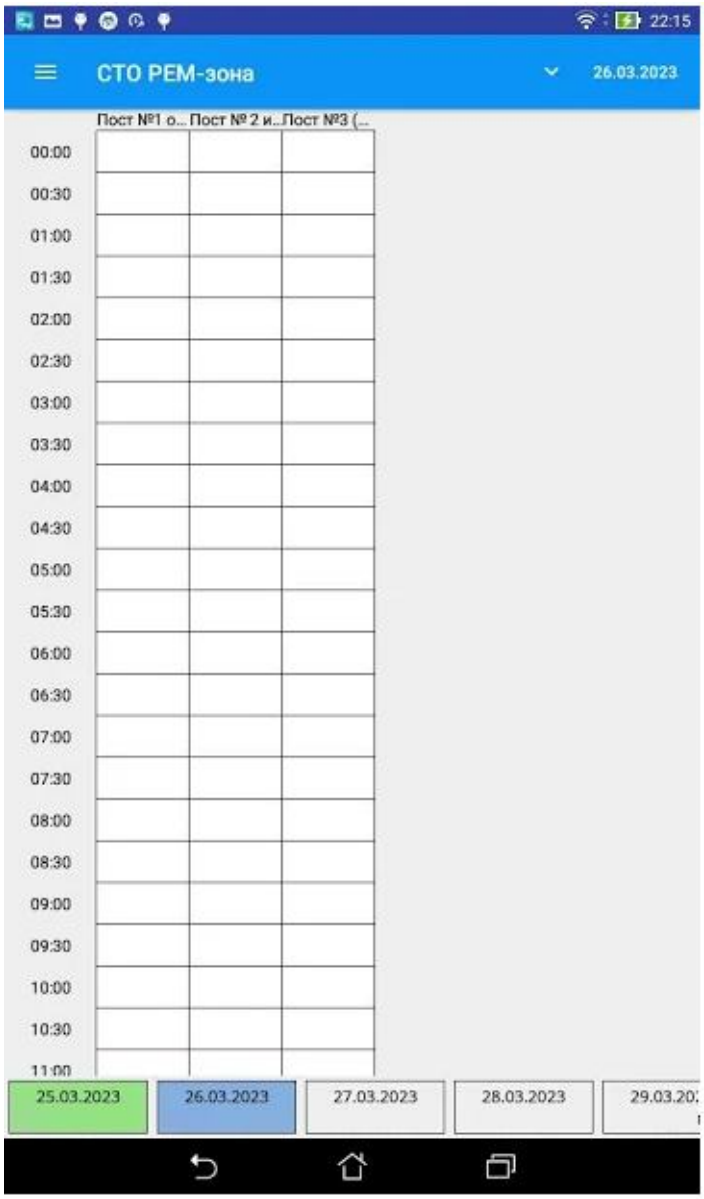


Рисунок 2.29 - Сторінка додатку із графіком роботи ремонтних постів

На закінчення хочеться відзначити, що, як можна зрозуміти з описаного в цьому розділі, розроблений додаток має досить простий і інтуїтивно зрозумілий інтерфейс, а тому під час використання служби користувачами складнощів виникати не повинно.

Розроблений додаток є зручним і практичним у використані, а також дозволяє ефективно працювати, не вимагаючи значних системних ресурсів та швидкісного Інтернету.

2.4 Висновки до другого розділу

У даному розділі обґрунтовано технологію, мову програмування та розроблено програмну систему. Обґрунтовано засоби розробки додатку та здійснено опис основних програмних модулів системи.

А також здійснено опис процедур тестування та їхніх результатів, описані тест-вимоги до програмного забезпечення, а також виявлені дефекти. По результатам тестування сформовано підсумок тестування. Також в даному розділі було розкрито питання встановлення та налаштування програмного забезпечення, а також вказані вимоги, дотримання яких необхідно для користування програмою. У даному розділі також описана інструкція користувача для роботи із додатком.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1. Інженерний захист персоналу об’єкту та населення. Правила застосування.

Функціонування на території нашої країни численних об’єктів підвищеної небезпеки, переважно в зонах з підвищеною концентрацією населення, різко посилює небезпеку великих техногенних катастроф, провокує та збільшує негативну дію особливо небезпечних стихійних явищ. Щороку втрати від таких надзвичайних ситуацій вимірюються тисячами людських життів, мільярдними збитками та невиправною шкодою для природного середовища.

Масштабність і багатогранність завдань щодо протидії сучасним природним і техногенним загрозам вимагають висококваліфікованої, технічно оснащеної, мобільної державної системи цивільного захисту.

Така система визнана складовою національної безпеки, а виконання її завдань - важливим обов’язком органів виконавчої влади всіх рівнів. Відповідно, з метою наближення до світових стандартів, від назви основного інструмента державної політики у сфері протидії наслідкам катастроф - цивільна оборона ми переходимо до назви - цивільний захист. І це не випадково. Сукупність завдань, що стоять перед службами цивільної оборони багатьох країн, більше пов’язані сьогодні з проблемами мирного часу, що дозволяє говорити про цивільний захист населення і територій, а не про цивільну оборону у воєнний час.

До захисних споруд цивільного захисту належать:

1) сховище – герметична споруда для захисту людей, в якій протягом певного часу створюються умови, що виключають вплив на них небезпечних

факторів, які виникають внаслідок надзвичайної ситуації, воєнних (бойових) дій та терористичних актів;

2) протирадіаційне укриття – негерметична споруда для захисту людей, в якій створюються умови, що виключають вплив на них іонізуючого опромінення у разі радіоактивного забруднення місцевості;

3) швидкоспоруджувана захисна споруда цивільного захисту – захисна споруда, що зводиться із спеціальних конструкцій за короткий час для захисту людей від дії засобів ураження в особливий період.

Для захисту людей від деяких факторів небезпеки, що виникають внаслідок надзвичайних ситуацій у мирний час, та дії засобів ураження в особливий період також використовуються споруди подвійного призначення та найпростіші укриття.

Споруда подвійного призначення – це наземна або підземна споруда, що може бути використана за основним функціональним призначенням і для захисту населення.

Найпростіше укриття – це фортифікаційна споруда, цокольне або підвальне приміщення, що знижує комбіноване ураження людей від небезпечних наслідків надзвичайних ситуацій, а також від дії засобів ураження в особливий період.

За місцем розташування сховища можуть бути вбудовані і окремо розташовані. До вбудованих відносяться сховища, які розташовані в підвальних приміщеннях будинків, а до окремо розташованих – сховища, які розташовані за межами будинків і споруд.

Для вирішення питань щодо укриття населення в захисних спорудах цивільного захисту центральні органи виконавчої влади, місцеві державні адміністрації, органи місцевого самоврядування та суб’єкти господарювання завчасно створюють фонд таких споруд.

Порядок створення, утримання фонду захисних споруд цивільного захисту та ведення його обліку визначається Кабінетом Міністрів України.

Проектування, будівництво, пристосування і розміщення захисних споруд та об’єктів подвійного призначення здійснюються згідно з нормами, які розробляються відповідно до Закону України "Про будівельні норми".

Вимоги щодо утримання та експлуатації захисних споруд визначаються центральним органом виконавчої влади, який забезпечує формування та реалізує державну політику у сфері цивільного захисту.

Утримання захисних споруд цивільного захисту у готовності до використання за призначенням здійснюється суб’єктами господарювання, на балансі яких вони перебувають (у тому числі споруд, що не увійшли до їх статутних капіталів у процесі приватизації (корпоратизації), за рахунок власних коштів.

У разі використання однієї захисної споруди кількома суб’єктами господарювання вони беруть участь в утриманні споруди відповідно до укладених між ними договорів.

3.2. Запобігання наслідкам аварії на виробництвах із застосуванням аміаку.

Для отримання низьких температур технологічними схемами компресорного цеху багатьох промислових підприємств харчової та переробної промисловості передбачено застосування токсичної речовини – аміаку.

Потенційна небезпека таких технологічних схем полягає у порушенні герметичності обладнання і трубопроводів, що містять аміак. Найбільшу небезпеку з цієї точки зору являють собою руйнування автоцистерн з рідким аміаком; руйнування напірних трубопроводів компресорів; порушення герметичності відокремлювачів рідини, лінійних та циркуляційних ресиверів, запірної арматури, батарей холодильних камер.

Наслідком таких аварій є виникнення загазованості виробничого приміщення, відкритого майданчика цеху і підприємства в цілому, а також прилеглих житлових районів; утворення вибухонебезпечної суміші аміаку з повітрям в приміщеннях, внаслідок чого можливі вибухи і пожежі.

Джерелами локальних викидів аміаку можуть служити процеси стиснення газоподібного і нагнітання рідкого аміаку, а також зливно-наливні операції.

Аварії (катастрофи) на підприємствах, транспорті та продуктопроводах можуть супроводжуватися викидом (виливом) в атмосферу і на прилеглу територію небезпечних хімічних речовин (НХР), таких як хлор, аміак, синильна кислота, фосген, сірчаний ангідрид та інші. Це являє серйозну небезпеку для населення, заражене повітря уражає органи дихання, а також очі, шкіру та інші органи.

Фактори небезпеки викиду (розливу) хімічно небезпечних речовин: забруднення навколишнього середовища, небезпека для всього живого, що опинилося на забрудненій місцевості (загибель людей, тварин, знищення посівів та ін.), крім того, внаслідок можливого хімічного вибуху виникнення сильних руйнувань на значній території.

Аміак – безбарвний газ з характерним різким запахом і їдким смаком. Він майже у два рази легший від повітря.

За звичайних умов аміак легко зріджується під тиском, а при випаровуванні поглинає тепло – сильно охолоджується. Ця властивість використовується у промислових та побутових холодильниках на м'ясокомбінатах, молокозаводах, овочевих базах, тобто там, де є необхідність в охолодженій продукції. Крім того, він є сировиною багатьох хімічних виробництв. Аміак зберігається і транспортується у зрідженому стані.

Він один з найважливіших продуктів сучасної хімічної промисловості. Головною галуззю його застосування є виробництво нітратної кислоти і азотних добрив. Крім того, аміак використовують для виробництва багатьох

інших хімічних продуктів. Останнім часом зріджений аміак і водний розчин аміаку стали широко застосовувати безпосередньо як азотне добриво.

Як рідина, аміак легший за воду, має меншу густину і при виході на повітря утворює слабкий дим. Вогненебезпечний, створює вибухові суміші з повітрям, отруйний. Особливо небезпечний для очей.

У випадку розливу рідкого аміаку і його концентрованих розчинів не можна доторкатися до розлитої рідини.

Ознаки отруєння аміаком:

- нежить, кашель, важке дихання, задуха;

- підвищене серцебиття, порушена частота пульсу;

- при контакті з рідким аміаком виникає обмороження, можливий опік з пухирями, виразки.

Перша допомога при отруєнні аміаком:

- одягніть протигаз і виведіть ураженого на свіже повітря;

- дайте подихати зволоженим повітрям (теплими водяними парами 10%-ного розчину ментолу в хлороформі);

- дайте йому теплого молока з «Боржомі» або харчовою содою;

- при задусі необхідний кисень;

- при спазмі голосових щілин забезпечте тепло на ділянку шиї, теплі ванночки, інгаляцію;

- при зупинці дихання проведіть серцево-легеневу реанімацію;

- при потраплянні в очі − промийте водою або 0,5-1%-ним розчином квасців, вазеліновою або оливковою олією;

- при уражені шкіри − обмийте чистою водою, зробіть примочки з 5%-ного розчину оцтової, лимонної або соляної кислоти.

3.3 Висновок до третього розділу

Захисні споруди цивільного захисту можуть використовуватися у мирний час для господарських, культурних і побутових потреб у порядку, що визначається Кабінетом Міністрів України. З моменту виключення захисної споруди із фонду споруд цивільного захисту вона втрачає статус захисної споруди цивільного захисту.

Володіння, користування та розпорядження спорудами, які втратили статус захисних споруд цивільного захисту, здійснюється відповідно до закону. Захисні споруди цивільного захисту державної та комунальної власності не підлягають приватизації (відчуженню).

При отруєнні аміаком винести потерпілого із зони зараження, шкіру, рот, ніс промити водою. В очі закапати по дві-три краплі 30% альбуциду, в ніс - оливкове масло. При необхідності відправити потерпілого до медичного закладу.

ВИСНОВКИ

В результаті проведеної роботи було розроблено Android-додаток для станції технічного обслуговування, тобто поставлена мета роботи була виконана.

Крім того, в рамках даної роботи були вивчені інструменти для розробки, а також способи зберігання даних в Android. Підсумком роботи став невеликий і зручний додаток, який підійде тим, хто воліє відмовитися від постійних телефонних дзвінків на станції технічного обслуговування.

В ході виконання бакалаврської роботи були виконані наступні завдання:

- вивчені особливості розробки прикладних програм для операційної системи Android;

- визначені функціональні та не функціональні вимоги до розроблюваного додатку;

- спроектована архітектура додатку, яка включає клієнтську та серверну частину;

- розроблена серверна частина, включаючи базу даних та програмні модулі для реалізації основної бізнес-логіки;

- розроблено мобільний додаток для операційної системи Android, описано основну процедуру йог розгортання на апаратні пристрої;

- виконано повний цикл тестування, починаючи від модульного і завершуючи функціональним тестування.

Тестування програмного забезпечення показало його працездатність і відповідність технічним вимогам.

ПЕРЕЛІКИ ДЖЕРЕЛ

1. Основи ООП. Джефрі Ріхтер, CLR via C#, 4-е видання 2014 by Фримен А. «ASP.NET MVC 4 с примерами на C# 5.0»

2. Onion Architecture In ASP.NET Core MVC. [Електронний ресурс] – Режим доступу:https://www.c-sharpcorner.com/article/onion-architecture-in-aspnetcore-mvc/

3. What is Entity Framework?. [Електронний ресурс] – Режим доступу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx

4. Getting started with ASP.NET MVC 5 [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/enus/aspnet/mvc/overview/gettingstarted/introduction/getting-started

5. Конспект лекцій з дисципліни «Програмування для мобільних пристроїв» для студентів денної форми навчання спеціальності 126 «Інформаційні системи та технології» / Укладачі: Готович В.А., Михайлович Т.В. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2020. – 216 с.

6. ООП в JavaScript [Електронний ресурс] – Режим доступу: https://frontendstuff.com/blog/object-oriented-programming

7. Bootstrap [Електронний ресурс] – Режим доступу: https://etk.lntu.edu.ua/mod/page/view.php?id=4135

8. Рівні ієрархії сучасної АСУТП [Електронний ресурс] – Режим доступу: https://studopedia.com.ua/1_378896_bagatorivneva-arhitektura.html

9. Overview to ASP.NET Core [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/introduction-toaspnetcore?view=aspnetcore-6.0

10. iOS Human Interface. – Електрон. дан. – Режим доступу: https://developer.apple.com/library/ios/documentation/userexperience/conceptual/Mobile HIG/index.html

11. Офіційний сайт SQLite. – Електрон. дан. – Режим доступу: http://www.sqlite.org

12. iOS Technology Overview. – Електрон. дан. – Режим доступу: https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneos techoverview/Introduction/Introduction.html

13. База SQLite и основы SQL. [Електронний ресурс] – Режим доступу: https://maxsite.org/page/sqlite8

14. Розробка додатків засобами мови програмування C#.[Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/354860614_Rozrobka_dodatkiv_zasobami_movi_programuvanna_C

15. ПРОФЕСІЙНА ПРАКТИКА ПРОГРАМНОЇ ІНЖЕНЕРІЇ. Лабораторний практикум / уклад. С. В. Поперешняк – К.: Вид-во «Друк», 2019. – 43-47 с.

16. Best Mobile IDEs for Android [Fall 2019 Update] Intellectsoft: веб-сайт. URL: https://www.intellectsoft.net/blog/5-of-the-best-mobile-ides-for-android

17. Mobile Operating System Market Share Ukraine Oct 2017 - Oct 2019 StatCounter Global Stats: веб-сайт. URL: https://gs.statcounter.com/os-marketshare/mobile/ukraine

18. Комп’ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2013. — 256 с. ISBN 978-617-574-087-3

19. Mobile Android Version Market Share World Oct 2017 - Oct 2019 StatCounter Global Stats: веб-сайт. URL: https://gs.statcounter.com/os-version-marketshare/android/mobile/world

20. Коноваленко І. В. Платформа .NET та мова програмування C# 8.0 : навчальний посібник / І. В. Коноваленко, П. О. Марущак. – Тернопіль : ФОП Паляниця В. А., 2020. – 320 с.

21. Сучасні підходи до розроблення і впровадження інформаційних систем [Електронний ресурс]. Режим доступу: https://pidruchniki.com/1181092047726/informatika/cuchasni_pidhodi_rozroblennya_vprovadzhennya_informatsiynih_sistem. –

22. Кнастер С. Objective-C. Програмування для Mac OS.X та IOS / Скотт Кнастер, Вакар Малик, Марк Далрімпл – Діалектика-Вільямс, 2018. – 304 с.: ил. – ISBN 978-5-8459-1826-0

23. Мови програмування для мобільної розробки[Електронний ресурс]. Режим доступу: https://code.tutsplus.com/uk/articles/mobiledevelopment-languages--cms-29138.

24. Nahavandipoor V. iOS 7 Programming Cookbook / Vandad Nahavandipoor – O'Relly, 2016. – 287 с.: ил. – 978-5-8459-3871-2

25. Петрик М.Р. Моделювання програмного забезпечення : науково методичний посібник / М.Р. Петрик, О.Ю. Петрик – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. – 200 с.

26. Філімончук Т.В., Чепелєв Є.О. Модель мобільного застосунку з використанням фреймворку Flutter. Проблеми інформатизації: Тези доповідей дев’ятої міжнародної науково-технічної конференції. Том 2, секція 4. Черкаси – Харків – Баку – Бельсько-Бяла. 2021. С. 99.

27. App Store Resource Center. – Електрон. дан. – Режим доступу: https://developer.apple.com/appstore/index.html

28. My take on Redux architecture [Електроний ресурс] – Режим доступу: https://krasimirtsonev.com/blog/article/my-take-on-redux-architecture

29. Stripe.js Reference [Електроний ресурс] – Режим доступу: https://stripe.com/docs/js

30. Swagger Codegen Documentation [Електроний ресурс] – Режим доступу: https://swagger.io/docs/open-source-tools/swagger-codegen/

31. Why the UX Is Important for Your Business [Електроний ресурс] – Режим доступу: https://rubygarage.org/blog/why-the-ux-is-important-for-your-business

32. Jest и Puppeteer: автоматизация тестирования веб-интерфейсов [Електроний ресурс] – Режим доступу: https://habr.com/ru/company/ruvds/blog/342578/

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГ ОСНОВНИХ МОДУЛІВ СИСТЕМИ

```
package at.inted.Connection;

import at.inted.utils.Console;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class UDP {

	private final static int PORT = 59126;

	public static void receiveMessage(){
		Thread thread = new Thread(new Runnable() {
			public void run() {

				DatagramSocket socket;
				DatagramPacket in;

				try {

					socket = new DatagramSocket(PORT);

				} catch (SocketException exception) {

					Console.WriteError(exception);
					return;
				}

				while(true){

					byte[] inData = new byte[1024];
					byte[] outData;
					String message;

					in = new DatagramPacket(inData, inData.length);

					try {
						socket.receive(in);

					} catch (IOException exception) {
```

```
                                        Console.WriteError(exception);
                                    }

                                    InetAddress senderIp = in.getAddress();
                                    int sendePort = in.getPort();
                                    message = new String(in.getData(), 0, in.getLength());
                                    Console.WriteInfo("Received Data (UDP):" + message);

                                    if (message.equalsIgnoreCase("car:anyserver")) {

                                        int status = Car.getStatus().getStatus();
                                        String answer;

                                        answer = "is n:Raspberry s:" + status;
                                        Console.WriteInfo("Answered with:" + answer);

                                        outData = answer.getBytes();
                                        DatagramPacket out = new
DatagramPacket(outData, outData.length, senderIp, PORT);

                                        try {
                                            socket.send(out);
                                            socket.close();

                                        } catch (IOException exception) {

                                            Console.WriteError(exception);
                                        }
                                    }

                                    else if (RemoteControl.getIp() == in.getAddress()){

                                        RemoteControl.receivedUDP(message);
                                    }


                                }
                            }
                        });
                        thread.start();

                }
}

package at.inted.commands;

import java.util.ArrayList;
import java.util.List;

public class Command {
```

```
	protected String name = "";
	protected List<String> aliases;

	public void execute(String command, String[] args){

	}

	public Command(){
		aliases = new ArrayList<String>();
	}

	public String getName() {
		return name;
	}

	public List<String> getAliases() {
		return aliases;
	}
}
package at.inted.utils;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Log {

	private static PrintWriter pWriter = null;

	private static void init(){
		try {
			File dir = new File("logs/");
			dir.mkdir();
			dir.setExecutable(true, false);
			dir.setReadable(true, false);
			dir.setWritable(true, false);


			File file = new File("logs/latest.log");
			file.setExecutable(true, false);
			file.setReadable(true, false);
			file.setWritable(true, false);
			if(file.exists()){
				SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd-
HH-mm-ss");
				file.renameTo(new File("logs/"+sdf.format(new Date()) + ".log"));
```

```
                }
                file.createNewFile();

                pWriter = new PrintWriter(new FileWriter(file));
            } catch (IOException ioe) {
                ioe.printStackTrace();
            } finally {
                if (pWriter != null)
                    pWriter.flush();
            }
        }

        public static void writeToLog(Object obj){
            if(pWriter == null){
                init();
            }

            pWriter.println(obj);
            pWriter.flush();
        }

        public static void close(){
            pWriter.close();
        }
}
package at.inted.sensors;

import at.inted.utils.Console;
import com.pi4j.io.gpio.*;
import com.pi4j.io.gpio.event.GpioPinDigitalStateChangeEvent;
import com.pi4j.io.gpio.event.GpioPinListenerDigital;

import java.util.ArrayList;
import java.util.List;

/**
 * User: lukas
 * Date: 17.01.14
 * Time: 16:12
 */
public class DistanceSensor extends Thread{

        public double distance = 0;

        public void run(){
            final GpioController gpio = GpioFactory.getInstance();

            GpioPinDigitalOutput trigger =
gpio.provisionDigitalOutputPin(RaspiPin.GPIO_05, "trigger", PinState.LOW);
            GpioPinDigitalInput echo = gpio.provisionDigitalInputPin(RaspiPin.GPIO_02,
PinPullResistance.PULL_DOWN);
```

```
                final List<Long> times = new ArrayList<Long>();
                echo.addListener(new GpioPinListenerDigital() {
                        @Override
                        public void
handleGpioPinDigitalStateChangeEvent(GpioPinDigitalStateChangeEvent event) {
                                long time = System.nanoTime();
                                times.add(time);
                                if(times.size() >= 2){
                                        double distance =  (0.0000175 * (times.get(times.size()-1) -
times.get(times.size()-2)));
                                        if(distance < 30) {
                                                System.out.println("zu nah" + distance);
                                                }else{
                                                System.out.println("nicht zu nah" + distance);
                                                }
                                        times.remove(0);
                                        times.remove(0);
                                }
                        }

                });



                while(true){
                        try{
                        trigger.high();
                        Thread.sleep(10);
                        trigger.low();
                        Thread.sleep(10);

                        }catch(InterruptedException exception){
                                Console.WriteError(exception.getStackTrace());
                        }
                }



        }
}

package at.inted;

import at.inted.Connection.Car;
import at.inted.Connection.RemoteControl;
import at.inted.Connection.UDP;
import at.inted.commands.CommandHandler;
import at.inted.commands.HelpCommand;
import at.inted.commands.ReloadCommand;
import at.inted.commands.StopCommand;
```

```
import at.inted.sensors.DistanceSensor;
import at.inted.utils.Console;
import at.inted.utils.Log;

public class Main {

	public static boolean closeCar = false;

	public static void main(String[] args) {
		Console.WriteInfo("Starting Car....");
		//TODO init

		CommandHandler ch = new CommandHandler();
		ch.Add(new HelpCommand());
		ch.Add(new StopCommand());
	ch.Add(new ReloadCommand());
		ConsoleInput ci = new ConsoleInput();
		ci.start();

		Car.setStatus(Car.CarStatus.readyForConnection);
		UDP.receiveMessage();
		RemoteControl rc = new RemoteControl();
		rc.start();

		DistanceSensor ds = new DistanceSensor();
		ds.start();

		Console.WriteInfo("Car started!");

		while(!closeCar){
			try{
				Thread.sleep(100);
			}catch(InterruptedException exception){
				Console.WriteError(exception.getStackTrace());
			}
		}


		Console.WriteInfo("Stopping Car...");
		//TODO closing

		Console.WriteInfo("Car stopped!");
		Log.close();



	}
}
package at.inted.Connection;
```

```
import at.inted.utils.Console;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;

public class TCP {

	public static String receiveMessage(Socket socket){

		String nachricht = null;
		try{

			BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
			char[] buffer = new char[200];
			int anzahlZeichen = bufferedReader.read(buffer, 0, 200);
			nachricht = new String(buffer, 0, anzahlZeichen);

		} catch (IOException exception){

			Console.WriteError(exception);
		}

		return nachricht;
	}
}
package at.inted.Connection;

import at.inted.utils.Console;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class UDP {

	private final static int PORT = 59126;

	public static void receiveMessage(){
		Thread thread = new Thread(new Runnable() {
			public void run() {

				DatagramSocket socket;
				DatagramPacket in;

				try {
```

```
                socket = new DatagramSocket(PORT);

            } catch (SocketException exception) {

                Console.WriteError(exception);
                return;
            }

            while(true){

                byte[] inData = new byte[1024];
                byte[] outData;
                String message;

                in = new DatagramPacket(inData, inData.length);

                try {
                    socket.receive(in);

                } catch (IOException exception) {

                    Console.WriteError(exception);
                }

                InetAddress senderIp = in.getAddress();
                int sendePort = in.getPort();
                message = new String(in.getData(), 0, in.getLength());
                Console.WriteInfo("Received Data (UDP):" + message);

                if (message.equalsIgnoreCase("car:anyserver")) {

                    int status = Car.getStatus().getStatus();
                    String answer;

                    answer = "is n:Raspberry s:" + status;
                    Console.WriteInfo("Answered with:" + answer);

                    outData = answer.getBytes();
                    DatagramPacket out = new
DatagramPacket(outData, outData.length, senderIp, PORT);

                    try {
                        socket.send(out);
                        socket.close();

                    } catch (IOException exception) {

                        Console.WriteError(exception);
                    }
                }
```

```
                                    else if (RemoteControl.getIp() == in.getAddress()){

                                        RemoteControl.receivedUDP(message);
                                    }


                                }
                            }
                        });
                        thread.start();

                }
        }
```

ДОДАТОК Б

DDL БАЗИ ДАНИХ

```
-- -----------------------------------------------
-- Schema sto
-- -----------------------------------------------
DROP SCHEMA IF EXISTS `sto` ;

-- -----------------------------------------------
-- Schema sto
-- -----------------------------------------------
CREATE SCHEMA IF NOT EXISTS `sto` DEFAULT CHARACTER SET latin1 ;
SHOW WARNINGS;
USE `sto` ;

-- -----------------------------------------------
-- Table `act`
-- -----------------------------------------------
DROP TABLE IF EXISTS `act` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `act` (
`id` INT(11) NOT NULL,
`order_id` INT(11) NULL DEFAULT NULL,
`date` DATE NULL DEFAULT NULL,
`detail` VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;
CREATE INDEX `a_idx` ON `act` (`order_id` ASC);

SHOW WARNINGS;
```

```
-- -----------------------------------------------
-- Table `avto`
-- -----------------------------------------------
DROP TABLE IF EXISTS `avto` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `avto` (
`id` INT(11) NOT NULL,
`marka` VARCHAR(100) NULL DEFAULT NULL,
`model` VARCHAR(100) NULL DEFAULT NULL,
`rik_v` INT(11) NULL DEFAULT NULL,
`nomer` VARCHAR(45) NULL DEFAULT NULL,
`vin` VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;

-- -----------------------------------------------
-- Table `client`
-- -----------------------------------------------
DROP TABLE IF EXISTS `client` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `client` (
`id` INT(11) NOT NULL,
`name` VARCHAR(45) NULL DEFAULT NULL,
`icode` VARCHAR(10) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;
```

```
-- -----------------------------------------------
-- Table `engineer`
-- -----------------------------------------------
DROP TABLE IF EXISTS `engineer` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `engineer` (
`id` INT(11) NOT NULL,
`name` VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;

-- -----------------------------------------------
-- Table `master`
-- -----------------------------------------------
DROP TABLE IF EXISTS `master` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `master` (
`id` INT(11) NOT NULL,
`number` INT(11) NULL DEFAULT NULL,
`name` VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;

-- -----------------------------------------------
-- Table `name_ref`
-- -----------------------------------------------
DROP TABLE IF EXISTS `name_ref` ;
```

```
SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `name_ref` (
`id` INT(11) NOT NULL,
`name` VARCHAR(255) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;

-- -----------------------------------------------
-- Table `order`
-- -----------------------------------------------
DROP TABLE IF EXISTS `order` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `order` (
`id` INT(11) NOT NULL,
`avto_id` INT(11) NULL DEFAULT NULL,
`engineer_id` INT(11) NULL DEFAULT NULL,
`client_id` INT(11) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;

-- -----------------------------------------------
-- Table `order_details`
-- -----------------------------------------------
DROP TABLE IF EXISTS `order_details` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `order_details` (
```

```
`id` INT(11) NOT NULL,
`master_id` INT(11) NULL DEFAULT NULL,
`order_id` INT(11) NULL DEFAULT NULL,
`name_ref_id` INT(11) NULL DEFAULT NULL,
PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SHOW WARNINGS;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```