МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Симулятор системи швидкого реагування диспетчерської служби патрульної поліції»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-81 Силенко Едуард Володимирович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2019 р.

Науковий керівник _____ к.т.н., доц. Шендрик В.В.
(підпис)

Голова комісії _____ Шифрін Д.М.
(підпис)

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

Студент _____
(підпис)

Суми-2019

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«__» _____ 2019 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Силенко Едуард Володимирович

(прізвище, ім'я, по батькові)

1 Тема проекту *Симулятор системи швидкого реагування диспетчерської служби патрульної поліції*

затверджена наказом по університету від «19» листопада 2019 р. №2305-III

2 Термін здачі студентом закінченого проекту «_10__» ___грудня___ 2019 р.

3 Вхідні дані до проекту дані про виклик патрульної поліції

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Аналіз предметної області симуляторів швидкого реагування, постановка задачі та методи дослідження моделювання симулятора системи швидкого реагування диспетчерської служби патрульної поліції, розробка симулятора системи швидкого реагування диспетчерської служби патрульної поліції

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність, мета і задачі, дослідження аналогів, вимоги до інформаційної системи, функціональне моделювання системи, діаграма варіантів використання, математична модель, архітектура інформаційної системи, засоби реалізації, проектування бази даних, демонстрація роботи програмного продукту, висновки.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	**Завдання прийняв**
Аналіз предметної області	*Шендрик В.В.*		
Постановка задачі та мети дослідження	*Шендрик В.В.*		
Проектування інформаційної системи	*Шендрик В.В.*		
Розробка інформаційної системи	*Шендрик В.В.*		

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Аналіз предметної області симуляторів швидкого реагування	25.09.19-09.10.19	
2	Постановка мети та вибір методів дослідження	10.10.19-14.10.19	
3	Формування вимог до симулятора	15.10.19-23.10.19	
4	Вибір алгоритму пошуку найкоротшого маршруту	24.10.19-05.11.19	
5	Розробка бази даних	09.10.19-24.10.19	
7	Розробка програмного додатку	25.10.19-09.12.19	
8	Формування документації	04.12.19-09.12.19	

Магістрант _____ Силенко Е.В.

Керівник роботи _____ к.т.н., доц. Шендрик В.В.

РЕФЕРАТ

до дипломного проекту

Силенко Едуарда Володимировича на тему: «Симулятор системи швидкого реагування диспетчерської служби патрульної поліції»

Дипломний проект присвячений створенню системи для симуляції роботи диспетчерської служби патрульної поліції.

У роботі проведено аналіз області та аналогів, сплановано етапи роботи, ризики та варіанти їх запобігання, проведено моделювання роботи системи та веб-сайту.

Засоби, за допомогою яких було виконано розробку: HTML, CSS, MySQL та JavaScript.

Результатом роботи є система реагування диспетчерської служби патрульної поліції у вигляді веб-сайту.

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списка використаних джерел із 24 найменувань, додатків. Загальний обсяг роботи складає 81 сторінку, в тому числі 60 сторінок основного тексту, 3 сторінки використаних джерел та 21 сторінок додатків.

Загальний обсяг роботи 81 сторінка, 43 рисунки, 12 таблиць, 2 додатки, 24 бібліографічних найменувань.

Ключові слова: симулятор, система, патрульна поліція, веб-сайт, веб-сторінка, найкоротший маршрут.

ЗМІСТ

ВСТУП

Швидке реагування патрульної поліції на виклики громадян є досить важливим фактором у безпеці суспільства. Повідомлення має бути обробленно максимально точно та без помилок. Людина не завжди може зробити це достатньо якісно. Інформаційні технології стають надійним помічником у цьому.

Програмні застосунки уже активно використовуються для досягнення даної мети. Одні з них закриті для людей, а інші створені для їх використання. Прикладом може бути мобільний додаток “My Pol” чи соціальні мережі, за допомогою яких відбувається фіксація правопорушень [1]. У патрульної поліції існує своя програма, за якою відбувається реагування на виклики. Та ніколи не існує чогось такого, що не хотілося б удосконалити.

Саме це стало основою даної науково-дослідної роботи. Використовуючи найсучасніші технології можна реалізувати автоматизоване реагування диспетчерської служби патрульної поліції. Це дало б змогу швидше обробляти виклики та в найкоротший термін відправити на місце найближчий автомобіль. У додаток до цього вихідний продукт дозволить з легкістю відстежувати всі патрулі на карті, вести статистику викликів та слідкувати за маршрутами автомобілів.

Метою даної роботи є реалізація симулятора системи автоматизованого реагування на виклики патрульної поліції. Досягнення цього зумовлює закріплення навичок та знань отриманих у ході навчання.

Для досягнення мети необхідно виконати наступні кроки:

- дослідити наявні системи швидкого реагування;
- побудувати алгоритм вибору найкотшого маршруту;
- змоделювати роботу системи;
- розробити симулятор.

Результатом роботи буде симулятор у вигляді web-ресурсу з підключенням бази даних для зберігання статистики. Розроблений інтерфейс дозволить також

відстежувати патрульні автомобілі на мапі та контролювати роботу програмного продукту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СИМУЛЯТОРІВ ШВИДКОГО РЕАГУВАННЯ

1.1 Дослідження актуальності проблеми

Наразі інформаційні технології використовуються у кожній сфері нашої діяльності. Це насамперед спрощує роботу людини та робить виконання задач більш точним і якісним. Що стосується систем швидкого реагування, то актуальність цих факторів зростає у порівнянні з іншими сферами. Кожна втрачена секунда при виборі маршруту чи автомобіля, який необхідно направити на місце виклику, може бути вирішальною. Тому дуже важливо зробити це за мінімальний термін та максимально точно. А що, як не програмний продукт впорається з цією задачею якнайкраще.

Уже досить давно існують державні диспетчерські системи реагування для швидкої допомоги чи патрульної поліції. У додаток до цього не так давно почали використовувати соціальні мережі для ідентифікації правопорушень. Це реалізовано у вигляді груп, в які громадяни можуть викласти зафіксовані протиправні дії. Патрульна поліція у свою чергу реагує на ці повідомлення.

Також варто зазначити мобільний додаток “My Pol” [1]. Це офіційний програмний продукт. Його можливості не закінчуються швидким викликом поліції за місцем знаходження користувача. Тут можна знайти і новини у сфері правоохоронних органів, і форму для відгуку про поліцейського та інше. У першу чергу варто зазначити, що даний програмний додаток дозволяє одним кліком викликати поліцію на ваше місцезнаходження. Для цього, має бути надано дозвіл програмі до місцезнаходження та на телефоні повинна бути включена функція геолокації (рис. 1.1).

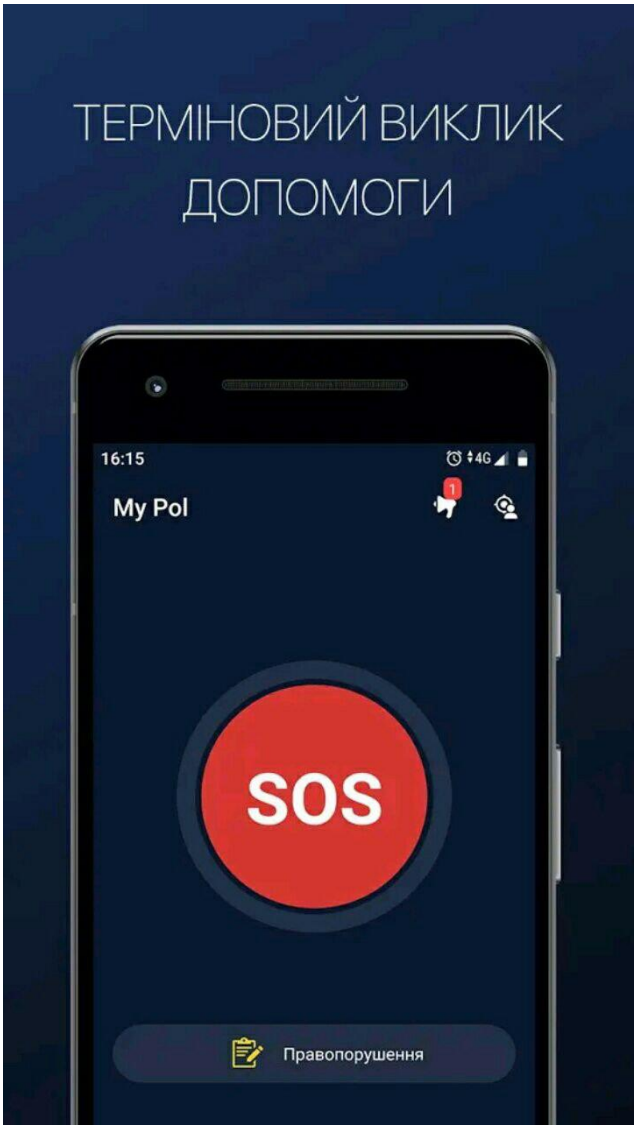


Рисунок 1.1 – Виклик поліції одним кліком

Також доступний вибір виду правопорушення, що дозволяє більш оперативно реагувати патрульній поліції та виконати всі заходи, якщо такі необхідні, перед відправкою на місце виклику (рис. 1.2).

Додатково маємо функцію прикріплення фото з місця правопорушення. Це також дозволяє більш швидко зреагувати на ту чи іншу ситуацію поліцією та передчасно вжити заходи, знову ж таки, якщо такі необхідні (рис. 1.3).

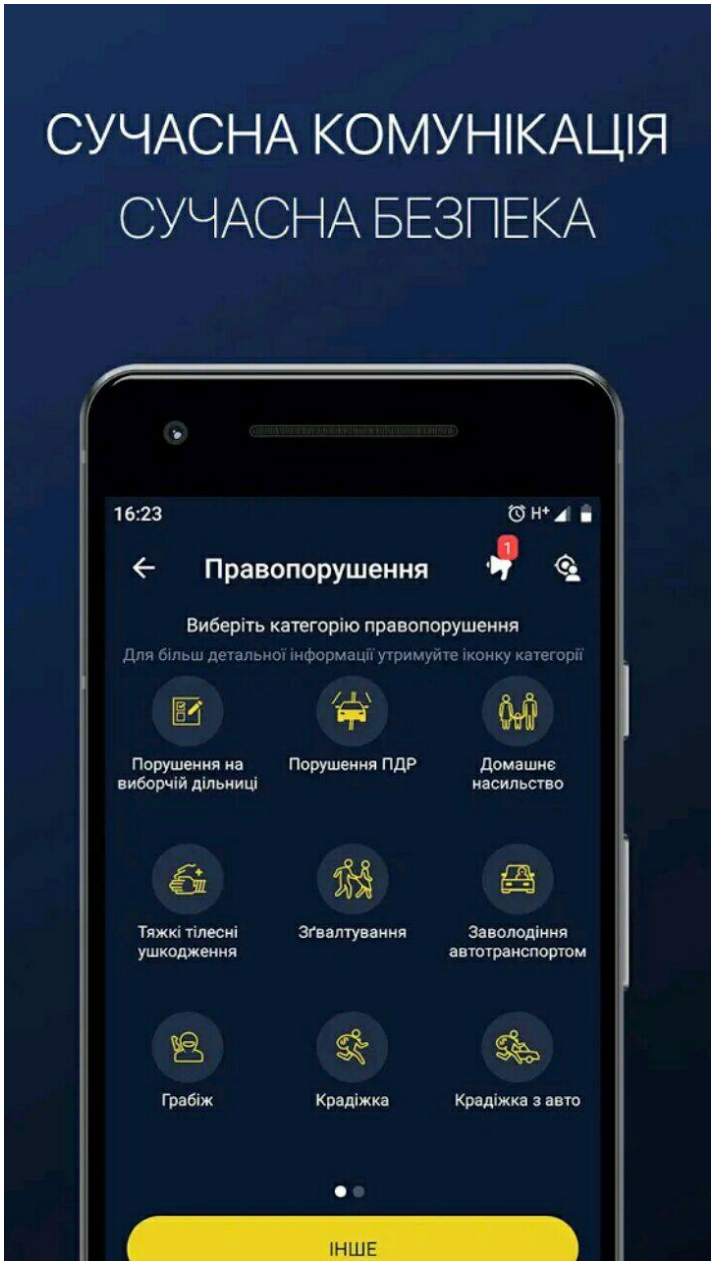


Рисунок 1.2 – Вибір виду правопорушення

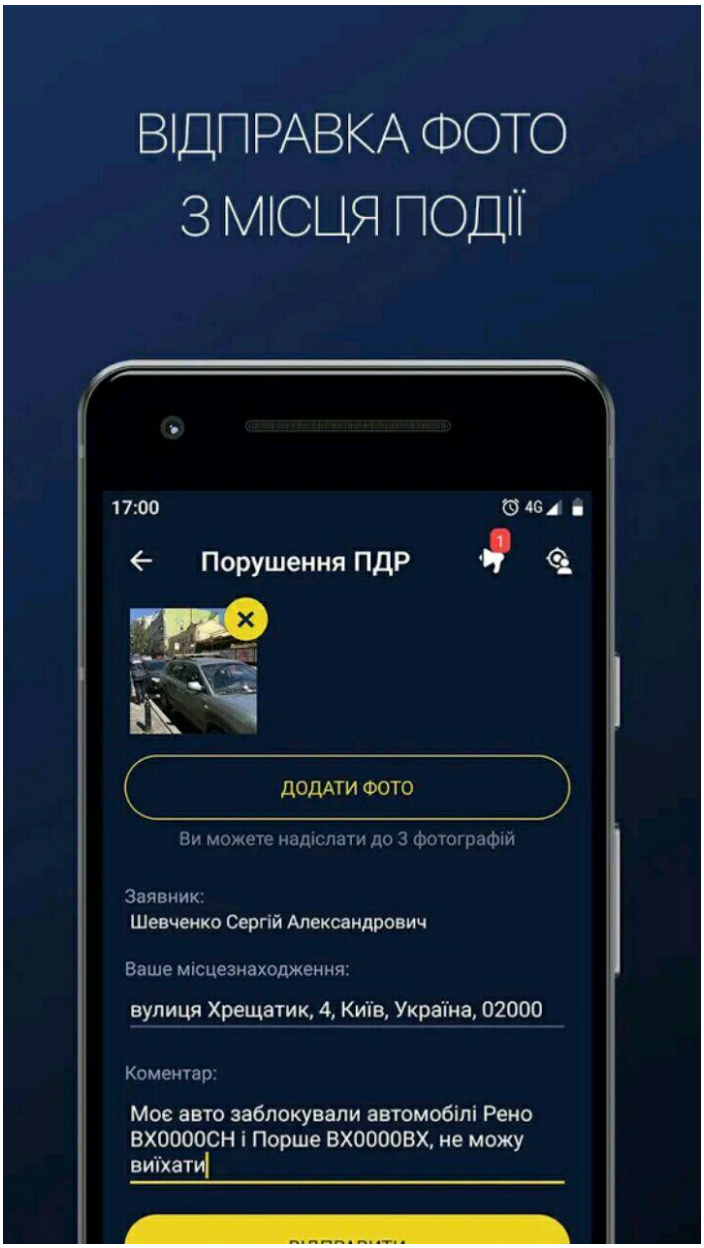


Рисунок 1.3 – Додавання фото правопорушення

На жаль, даний програмний продукт не має зараз широкого використання. Та спроба використати сучасні технології для безпеки суспільства та швидкого реагування на правопорушення є виправданою. У подальшому необхідні допрацювання програмного продукту, щоб він все-таки став поширеним серед населення та насправді приносив користь, а не додаткову непотрібну роботу.

Ще одним прикладом використання сучасних технологій є група в соціальній мережі телеграм «Безпечне місто Суми». Її суть полягає у реагуванні на правопорушення шляхом чату з користувачами (рис. 1.4).

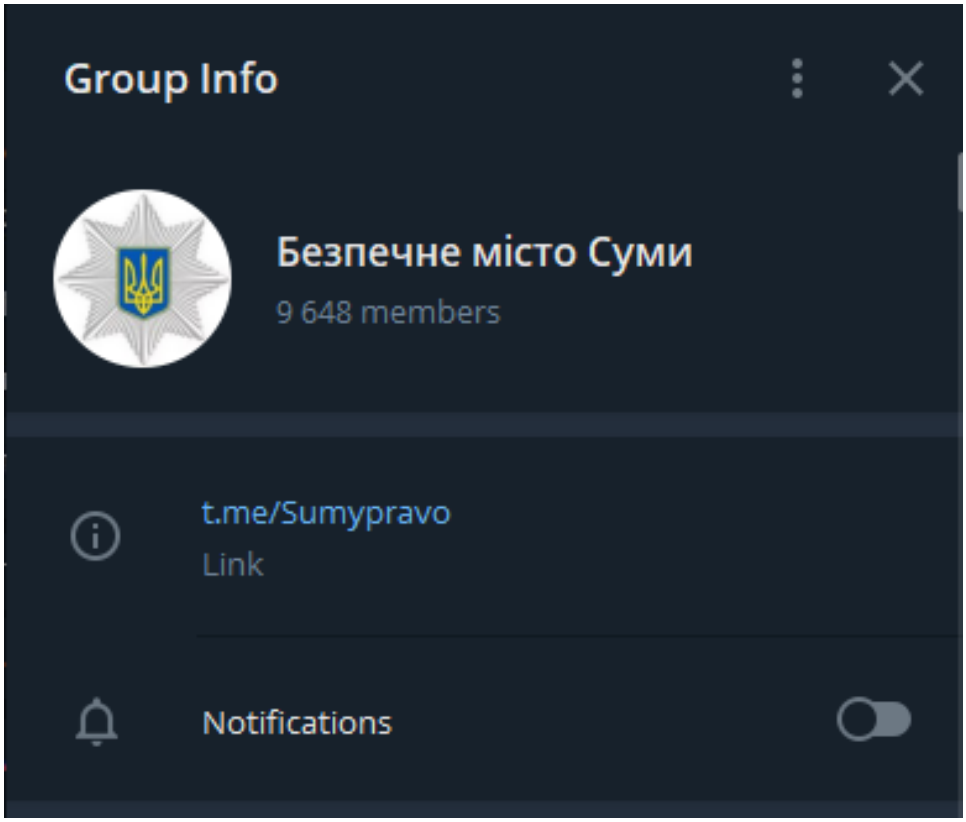


Рисунок 1.4 – Чат в телеграмі

Будь-який учасник групи може викласти повідомлення про правопорушення чи якісь інші дії з боку громадян, що можуть привести до протиправних дій. У свою чергу є адміністор групи, який власне і реагує на такі повідомлення онлайн протягом всього дня та ночі (рис. 1.5).

Здебільшого така інформація надходить як анонімна, та в деяких випадках адміністратор просить надати особисту інформацію про себе. Такий підхід не дає стовідсоткового результату для громадян та повноцінної довіри до прийняття виклику. Але група на це і не розрахована. Її завдання приймати дрібні правопорушення, на які, у випадку відсутності такої групи, можливо, і не було б звернено увагу.

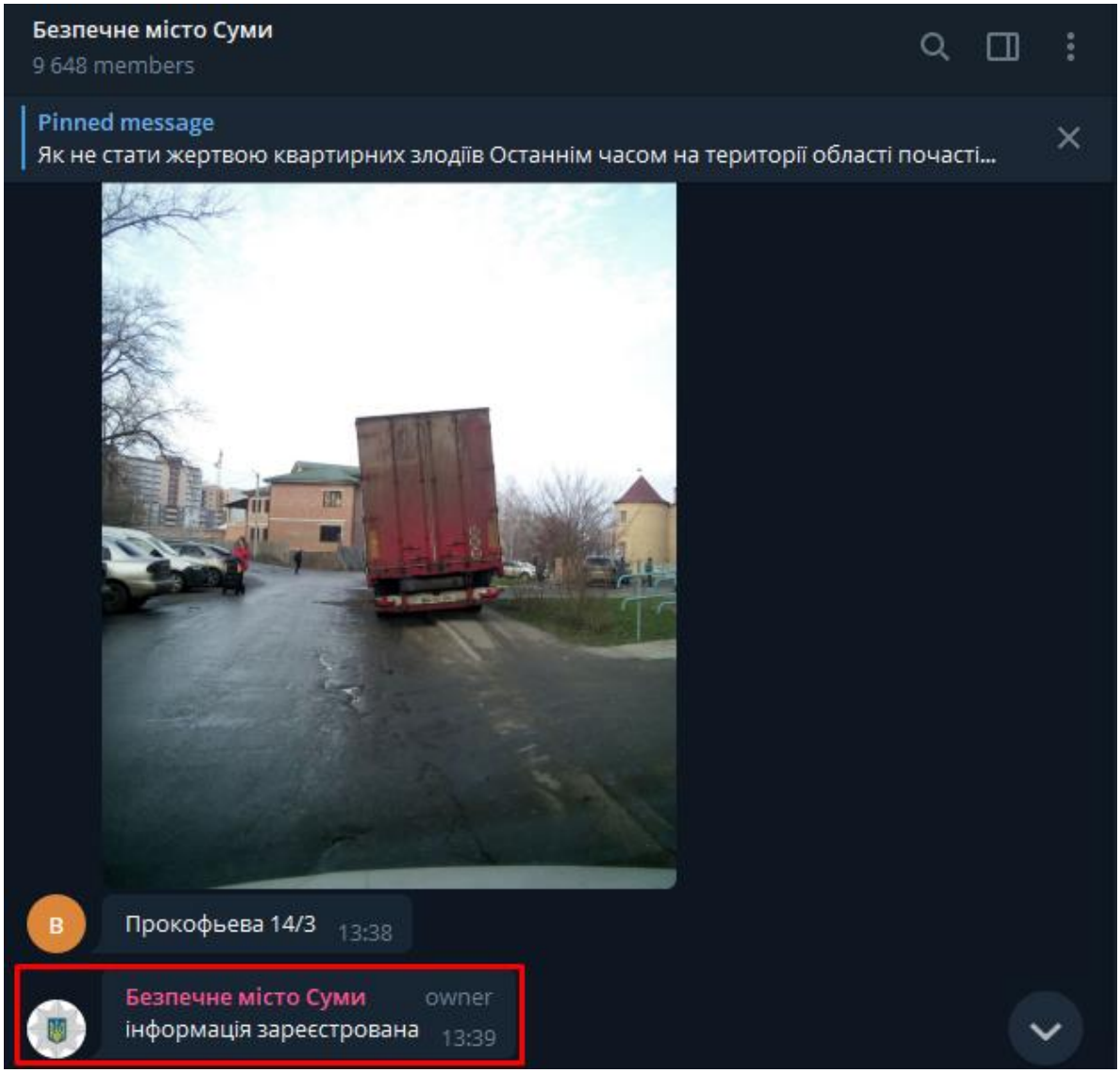


Рисунок 1.4 – Чат в телеграмі

Такі програмні продукти засвідчують важливість інформаційних технологій у даній сфері діяльності. Люди користуються смартфоном постійно, тому на це і має бути акцент у всіх галузях діяльності.

Пошук найближчого автомобіля до точки та побудова маршруту актуальні не тільки для систем диспетчерської служби патрульної поліції. Такі задачі виникають і для служб таксі. Прикладів програмних додатків такого типу можна зайти дуже багато. Це в свою чергу говорить про високу актуальність удосконаленні методів та алгоритмів, які використовуються для цього.

Та, зрештою, важко порівнювати службу таксі з реагуванням на виклик поліції. Різниця в тому наскільки швидко все має бути виконано у такому додатку.

Математична частина повинна працювати без збоїв та показувати найвищі результати. А, отже, модифікація таких систем має більший пріоритет. Разом з цим зростає і актуальність реалізації таких програмних засобів.

1.2 Аналіз аналогів та визначення наявних проблем

Найближчим аналогом для розроблюваного симулятора є сучасна диспетчерська служба патрульної поліції. Але звичайно ж вона недоступна для перегляду. Тому, на жаль, проаналізувати її не вдасться.

Щодо вищезазначених програмних продуктів, то їх не можна назвати аналогами. Мобільний сервіс “My Pol” має дещо іншу аудиторію [1]. Він розроблений для громадян, а дана робота присвячена саме для керування диспетчерською службою патрульної поліції. Групи в соціальних мережах також не відносяться до аналогів даного програмного продукту.

Тому, аби виявити наявні проблеми та сильні сторони схожих додатків, аналіз буде проведено на прикладі служб таксі. У деякому сенсі їх задача дуже схожа з поставленою. Це пошук найближчих автомобілів до точки замовлення. Звичайно наявні значні відмінності, але для виділення основних проблем цього буде достатньо. До того ж такі системи працюють з онлайн-картами, а в даній роботі саме цьому приділено увагу.

Що ж, було обрано 3 програмні продукти для виклику таксі. Розглянемо їх детальніше:

- «Тачка» (далі П1) (рис. 1.5) [2];
- «Uber» (далі П2) (рис. 1.6)[3];
- «OnTaxi» (далі П3) (рис. 1.7) [4].

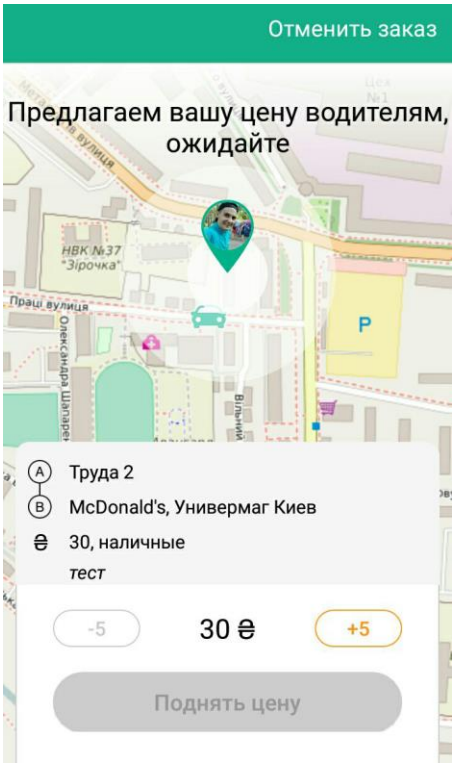


Рисунок 1.5 – Інтерфейс П1

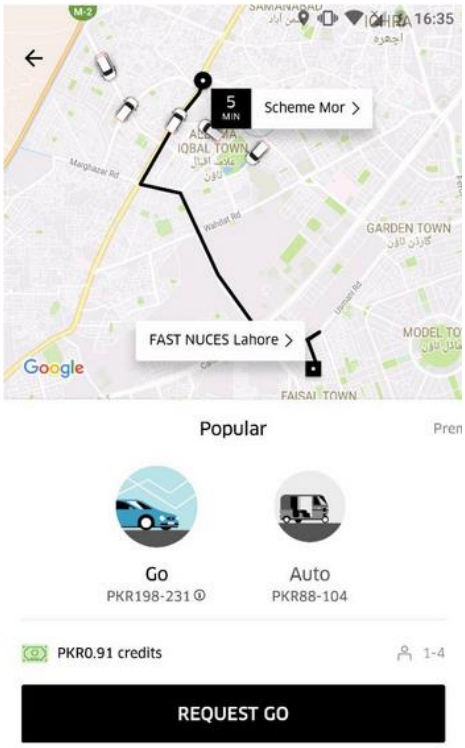


Рисунок 1.6 – Інтерфейс П2

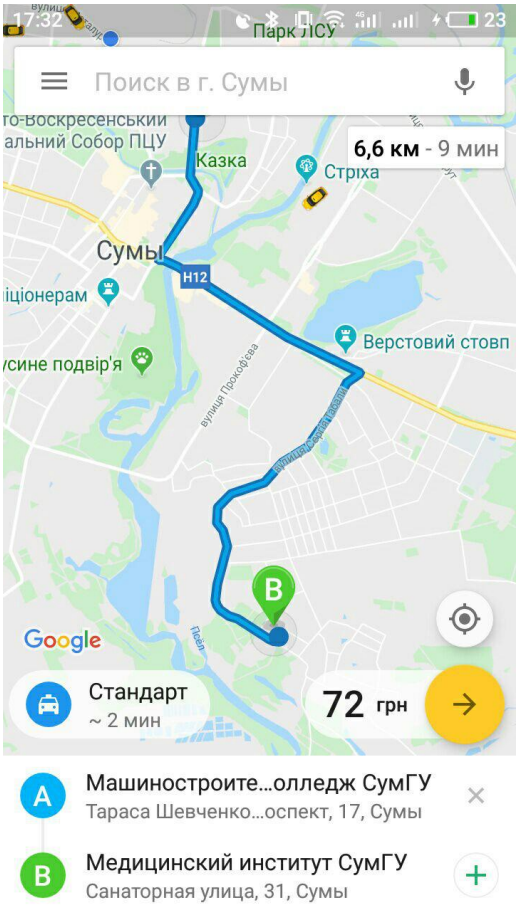


Рисунок 1.6 – Інтерфейс П3

Тепер перейдимо до детальнішого аналізу кожної із програм. Усі додатки мають схожий інтерфейс та функціонал. Що стосується П1, то тут можливо автоматично визначити місце розташування при виклику автомобіля, але доведеться вручну вводити кінцеву точку. На карті маємо іконки автомобілів, які знаходяться поруч з адресою прибуття. Після прийняття замовлення, залишиться лише іконка того, хто прийняв замовлення. Таким чином можна бачити наскільки далеко знаходиться водій. При цьому не показано маршрут руху автомобіля [2].

П2 дуже схожа на П1, але все ж має свої відмінності. Тут уже є побудований маршрут руху. Це додає можливість детальнішого відстежування водія. Решта, з того, що більш-менш близько до розроблюваного симулятора, не буде враховано при аналізі [3].

П3 має такі ж відмінності з П1 та максимально схожа з П2. Тут також

бачимо наявність побудови маршруту, що немало важливо для реалізації майбутнього додатку симулятора. У ПЗ наявні й інші функції, але їх не будемо враховувати при аналізі в даному випадку, так як вони не мають ніякого відношення до розробки майбуньої системи [4].

Також варто зазначити, що ці додатки не враховують можливі затори на дорогах. Тому, виникає ризик збільшення часу поїздки. Для даної роботи це неприпустимо. А, отже, це можна віднести до однієї з найбільших проблем у аналогів.

На основі аналізу даних програмних продуктів створимо порівняльну таблицю (таблиця 1.1).

Таблиця 1.1 – Порівняння аналогів

№	Характеристика	«Тачка»	«Uber»	«OnTaxi»
1	Іконки автомобілів на карті	+	+	+
2	Автоматичний виклик	+-	+-	+-
3	Побудова маршруту	-	+	+
4	Історія викликів	+	+	+
5	Врахування заторів	-	-	-
6	Відстежування руху автомобілів	+-	+-	+-
7	Автоматичне направлення автомобіля	-	-	-

Як бачимо, проаналізовані програмні додатки у більшій мірі не задовольняють тих основних вимог, які мають бути виконані при розробці симулятора системи диспетчерської служби патрульної поліції. Такий результат було отримано через відсутність точних аналогів з даним симулятором. Але завдяки цьому вдалося виділити проблеми, на які варто звертати увагу при подальшій роботі.

Таким чином, дуже важливо, аби в такій програмі була можливість побудови маршрутів, врахування заторів на дорозі, відстежування руху

автомобілів та їх автоматичне направлення на місце виклику з урахуванням найшвидшого реагування на виклик.

У результаті можна виділити такі основні проблеми систем швидкого реагування:

- відсутність побудови маршруту;
- неврахування заторів при побудові маршруту;
- відсутність автоматичного направлення автомобіля на місце виклику;
- вузька спеціалізація.

Тепер перейдимо до того, що стосується технологій та методів, необхідних для створення даних аналогів. Загалом, визначені вище додатки створені за одним і тим же принципом та використовують практично ідентичні програмні засоби для свого функціонування.

У першу чергу для їх повного функціонування необхідно 4 компоненти:

- мобільний додаток для пасажирів;
- мобільний додаток для водіїв;
- панель адміністратора для контролю та управління з боку власника;
- сервер, на якому буде зберігатися інформація про поїхдки, оцінки та інше.

У кожного сервісу існує свій веб-сайт, та він несе більш інформаційну функцію.

Геолокація досягається завдяки Google’s Location APIs на платформі Android та Core Location на iOS. Також в обох випадках в додатки інтегровано Google Maps. Управління платіжними системами не є корисним для розгляду у випадку розробки веб-сервісу за темою. Ще одним важливим пунктом є відправка push-сповіщень чи sms-повідомлень. Що стується другого, то також не є актуальним для даної роботи. А перше має право на існування. Дана функція підтримається завдяки Apple Push Notifications Service (APNs) та Google Cloud Messaging на iOS і Android відповідно. Усі вищевказані технології мають сьогодні найбільшу популярність, тому не дивно, що вони і використовуються. Ще одним

цікавим пунктом є відправка листів на e-mail. Для цього зазвичай використовують SendGrid API. Як можна помітити, усі методи відносяться до роботи з мобільними додатками. Та це також є корисним. Адже на їх основі можна використовувати найбільш схожі засоби для роботи з інтернет-сторінками [2-4].

Отже, при розробці симулятора швидкого реагування диспетчерської служби патрульної поліцій необхідно врахувати ці недоліки. Тобто, треба зробити більшу функціональність інтернет-сервісу за наявні аналоги, враховувати швидкість прибуття того чи іншого автомобіля на місце виклику та на основі цього обирати машину для реагування на виїзд і, зрештою, врахувати всі нюанси, які можуть збільшити час прибуття автомобіля до точки.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі роботи

Уже було зазначено, що науково-дослідницька робота в результаті буде представлена у вигляді web-сервісу для швидкого реагування диспетчерської служби патрульної поліції. Уся суть полягає у якнайшвидшій обробці вхідного виклику та направленні машини в точку за найменший проміжок часу. Таким чином буде доступна інформація про автомобілі в патрулі, їх маршрут, істрою їх викликів, детальна інформація про поточні виклики. Це і є основні функції розроблюваної системи.

Щодо задач, то вони виглядають так:

- дослідження систем швидкого реагування;
- побудова алгоритму вибору найкоротшого маршруту;
- моделювання роботи системи;
- розробка симулятора.

Основною метою проекту є розробка симулятора швидкого реагування на виклики, який у автоматичному режимі зможе визначити автомобіль, що звоже найшвише прибути до точки виклику.

Важливо також зазначити функціональні вимоги до майбутнього симулятора. Основою системи є прийом виклику та відправлення автомобіля за місцем виклику. Та функціонал можна розширити достатньо широко, щоб можна було зберігати інформацію, проводити контроль та інше. Таким чином виділяємо наступні функції:

- прийом виклику службою;
- пошук найближчого автомобіля та відправлення його на місце виклику;
- моніторинг патрулів на карті;

- побудова маршруту патруля;
- збереження інформації про виклики та реагування на них;
- пошук патрульних автомобілей на карті.

У результаті маємо поставлену мету та основний функціонал майбутньої системи. Під час розробки функції можуть мінімально змінитися через можливість ризиків.

Планування робіт розміщено в додатку А.

2.2 Вибір методів дослідження

Перед початком розробки було проведено аналіз методів та технологій для використання. Завдяки цьому можна найбільш ефективно використати можливості програмування та досягти легкої взаємодії усіх частин програмного продукту.

У першу чергу необхідно розробити невеликий блок програмного коду для симуляції виклику у довільний час. Це необхідно більше для тестування, ніж практичного застосування. Також можливий варіант аналізу критичних випадків шляхом більшого навантаження диспетчерської служби викликами. Забігаючи наперед варто зазначити, що симулятор буде розроблений за допомогою мови JavaScript. Це передумовлено тим, що сам інтернет-сервіс буде розроблено на цій мові програмування [5, 6].

Наступним етапом є розробка бази даних, яка буде зберігати інформацію про автомобілі, їх маршрути, виклики та їх стани. Її буде реалізовано у СУБД Mysql Workbench. Це сучасний потужний програмний продукт, який дозволяє створити БД будь-якого типу та складності. Її буде реалізовано за всіма правилами розробки та приведено до 3-ї нормальної форми. Варто сказати, що дана СУБД дозволяє розташувати скрипт БД на своєму локальному хостингу, але перед цим

виконується перевірка тексту скрипта на помилки різного роду такі як дублювання ключів коректності параметрів у типах даних і т. д.

Чим зумовлений вибір саме даної СУБД досить легко пояснити:

- швидкодія – MySQL є однією із найбільш швидких баз даних з тих, що існують на ринку;
- простота використання – MySQL є простою у використанні СУБД, яку значно легко інсталювати та адмініструвати;
- можливості обробки – кількість рядків у таблицях може досягати 50 мільйонів. У даному проекті в тестових цілях це не є важливим, але в подальшому даний плюс може принести свої плоди;
- взаємодія і безпека – MySQL розроблена для роботи в мережі. Таким чином, СУБД наділена сучасною системою захисту від небажаного доступу;
- малий розмір – СУБД MySQL має досить незначний розмір, навіть при великому об’ємі даних [7, 8].

Одним з найбільш важливих завдань є розробка математичного алгоритму побудови маршруту. За основу було вирішено взяти пошук найкоротшого маршруту в графі. Основний алгоритм буде закладено в роботі онлайн-карт. Це уже давно протестований варіант прокладання маршруту, тому не варто вигадувати щось нове. Але при цьому буде враховано додаткові фактори, які можуть впливати на час руху, такі як затори та інші [9, 10].

І нарешті сама основа симулятора, як зазначено вище, буде реалізована мовою JavaScript. Це мова програмування сьогодні є однією з найбільш використовуваних. Варто зазначити, що для розробки використовуватиметься фреймворк Angular 6 від компанії Google [5]. Він призначений для створення клієнтських додатків і в першу чергу націлений на односторінкові web-ресурси. Даний фреймворк надає таку функціональність, як двохстороннє зв‘язування. Це дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому. Однією із ключових властивостей Angular є використання в

якості мови програмування TypeScript [11].

Щодо переваг даного фреймворку, то їх можна розділити на пункти:

- велике ком’юніті – це є значним плюсом. Адже не буде необхідності зупинятися на простих помилках та рішеннях. Усі помилки дуже легко виправити за рахунок досвіду програмістів;
- використання директив – в основі фреймворку використовується HTML. Проте він розширюється за допомогою директив, які додають в код динамічності;
- висока швидкість розробки – використовуючи автоматичну генерацію коду для компонентів, сервісів та іншого можна значно скоритити час на розробку;
- MVC – в Angular використовується модель MVC, яка розділяє логіку, представлення та дані додатку в окремі файли;
- модульність – у фреймворкі можна відокремити модулі від основної програми. Вони можуть як взаємодіяти між собою, так і працювати автономно;
- наявність готових рішень – це дозволяє значно зекономити час при розробці. Готові рішення дозволять не вигадувати заново те, що вже давно реалізовано.

Разом з цим доведеться встановити сервер Node.js і пакетний менеджер npm. Сервер Node.js необхідний для розробки самого продукту. За його допомогою буде налаштовано локально роботу з базою даних та серверною частиною програми. І таким чином проведено взаємодію між front-end та back-end [12, 13].

Також важливим є WAMP сервер. Цей програмний продукт дозволить локально запустити front-end. До того ж без цього локального сервера не вдасться реалізувати базу даних та її наповнення. Таким чином в сукупності два програмних продукти з локальними серверами допоможуть протестувати систему на стадії розробки. [14].

Отже, було визначено технології, за допомогою яких буде створено симулятор для реагування на виклики патрульною поліцією. У подальшому буде проводитися розробка за допомогою визначених тут методів та технологій. Усі програмні продукти та технології зазначені вище будуть використані для досягнення мети проекту та виконання всіх поставлених задач. Вибір інструментів є дуже важливим етапом для подальшої роботи. Тому йому було приділено достатньо уваги.

3 МОДЕЛЮВАННЯ СИМУЛЯТОРА СИСТЕМИ ШВИДКОГО РЕАГУВАННЯ ДИСПЕТЧЕРСЬКОЇ СЛУЖБИ ПАТРУЛЬНОЇ ПОЛІЦІЇ

3.1 Вимоги до системи

Загалом, у результаті система повинна відстежувати рух патрульних автомобілей на карті, приймати виклики, які надходять до неї, надсилади на місце виклику найближчий автомобіль та зберігати інформацію про попередні виклики.

Реалізація буде виконана за допомогою Google maps. Це найбільш поширений сервіс для роботи з мапою в електронному вигляді у нашому регіоні. Вся інформація буде зберігатися в базі даних, до якої матиме доступ конкретний тип користувачів. За запитом можна буде дізнатися інформацію про виклики конкретного патруля чи за датою та інше.

Система повинна:

- відстежувати рух патрулів на карті;
- приймати виклик від користувача;
- за допомогою математичного алгоритму шукати автомобіль, який найшвидше прибуде на місце виклику;
- працювати без втручання оператора;
- надавати за запитом інформацію про попередні виклики.

3.2 Структура системи

Для розуміння роботи системи та її складових побудуємо її архітектуру за допомогою абстрактної моделі (рис. 3.1). Вона включає в себе сервер, на якому запущено сам веб-сайт та базу даних. А також клієнта, на якому запускається веб-браузер, а відводіно розроблюваний веб-ресурс.

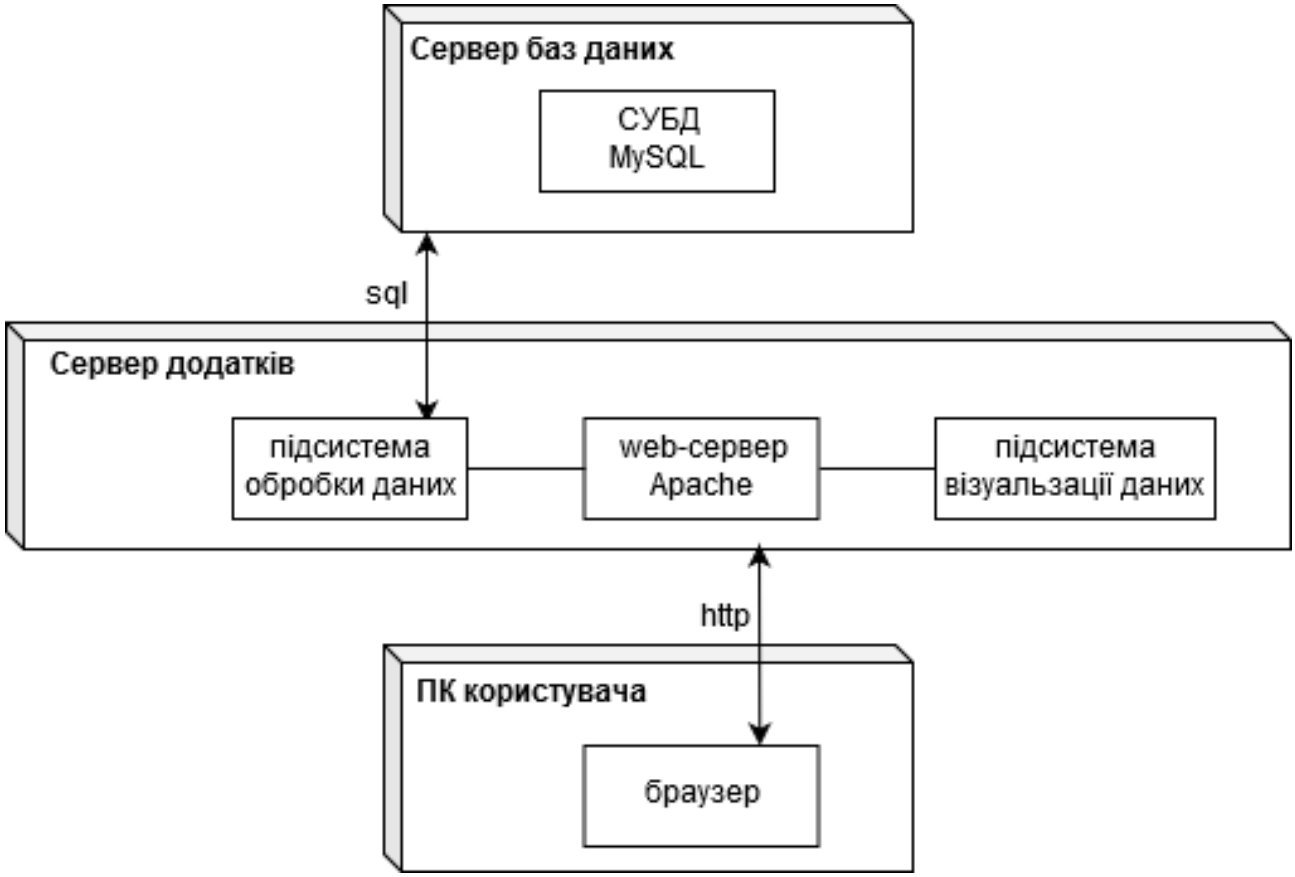


Рисунок 3.1 – Архітектура системи

3.3 Моделювання системи

ред розробкою інформаційної системи необхідно змоделювати її роботу, показати усі процеси, порядок їх виконання та можливі варіанти розвитку подій всередині самої системи. Для цього треба побудувати контекстну діаграму IDEF0 та варіантів використання(use case).

Першою будуємо контекстну діаграму IDEF0 (рис. 3.2). Головним бізнес-процесом є обробка виклику диспетчерської служби патрульної поліції. На вході маємо інформацію про виклик патрульної поліції, а в результаті роботи системи отримуємо на вихід патруль для реагування на виклик та саме уже дані про оброблений виклик. Механізмами виступають диспетчер, адміністратор, апаратне та програмне забезпечення. Усі процеси системи регулюються нормативною документацією, яка буду використана при розробці, математичною моделлю для розрахунку результату [13, 15].

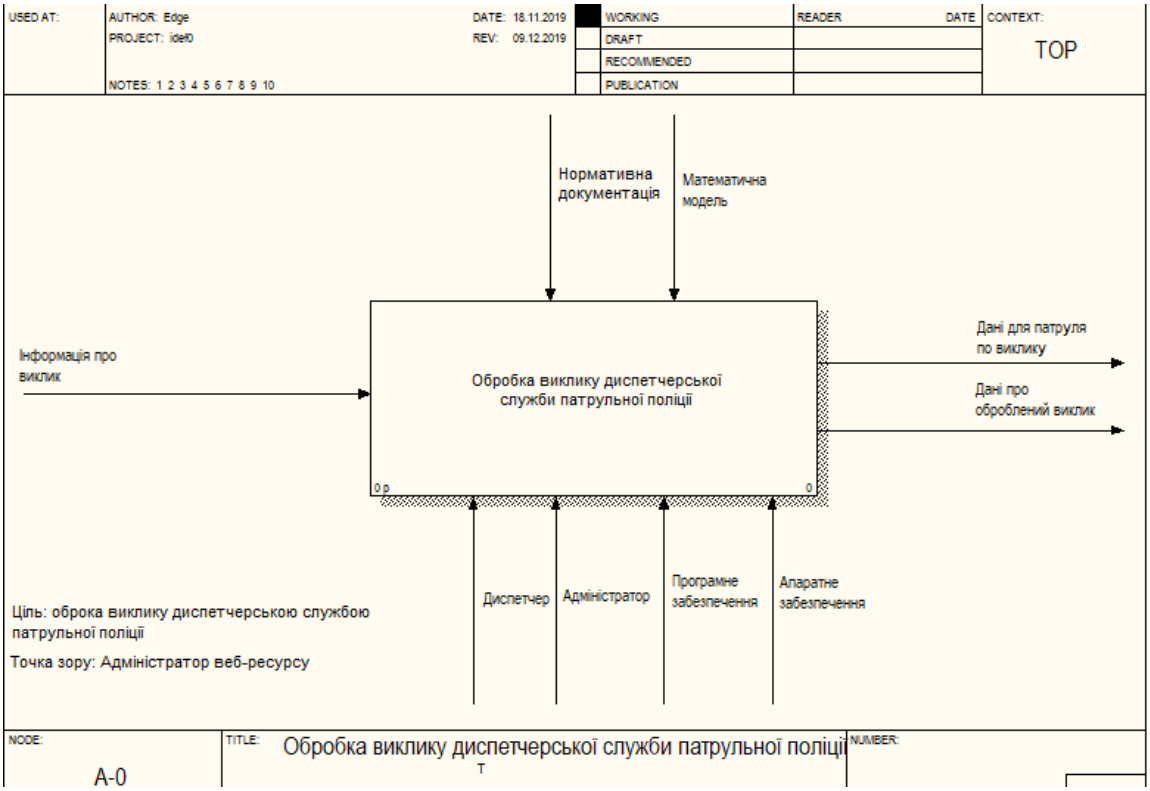


Рисунок 3.2 – Діаграма IDEF0

Далі було проведено декомпозицію даної контекстної діаграми (рис. 3.3). На даній діаграмі чітко показано послідовність процесів системи. На даному рівні декомпозиції було виділено 3 процеси. А саме: прийом виклику диспетчерською службою, пошук найближчого патруля та відправка автомобіля на місце виклику. Диспетчре приймає участь тільки в першому процесі, а адміністратор тільки в останньому. Апаратне та програмне забезпечення приймають участь у всіх процесах, так як вони передбачають роботу з розрахунками та базою даних. Математична модель використовується тільки для пошуку найближчого патруля. А уже нормативна документація регулює процеси, які відбуваються в системі [13, 15].

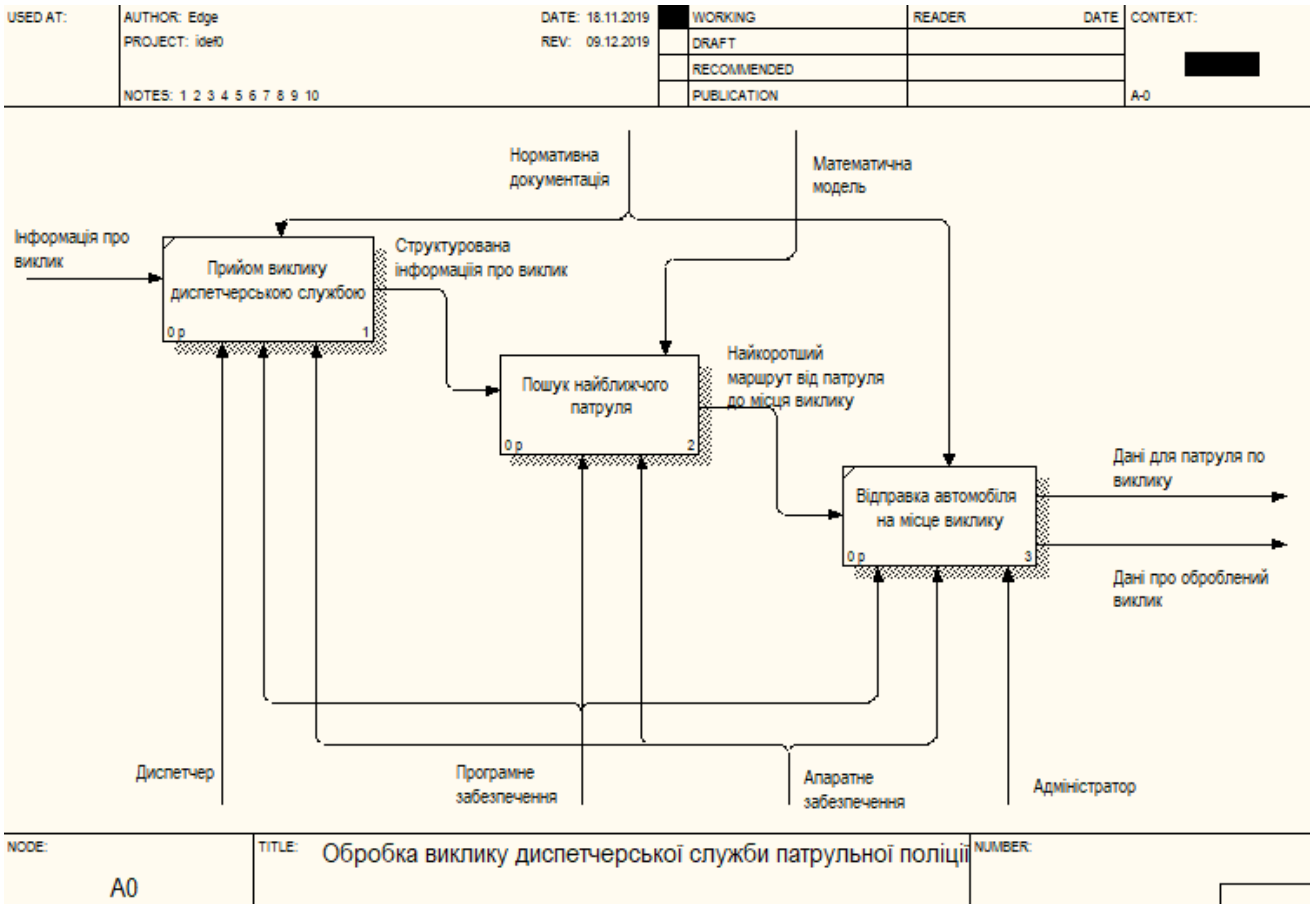


Рисунок 3.3 – Декомпозиція контекстної діаграми IDEF0

Наступним етапом проведемо декомпозицію процесу пошуку найближчого патруля (рис. 3.4). Даний процес декомпозовано на три підпроцеси – збір даних

для алгоритму, розрахунок найкоротшого маршруту для кожного патруля та вибір найкоротшого маршруту серед усіх патрулів. У результаті система має найкоротший маршрут до місця виклику. При цьому використовуємо математичну модель для роботи алгоритму пошуку найкоротшого маршруту. Це відбувається завдяки роботі програмного та апаратного забезпечення. На проміжному етапі, між підпроцесами маємо накоротші маршрути від кожного патруля до місця виклику. Таким чином уже з них буде обрано патруль, якому буде направлено виклик.

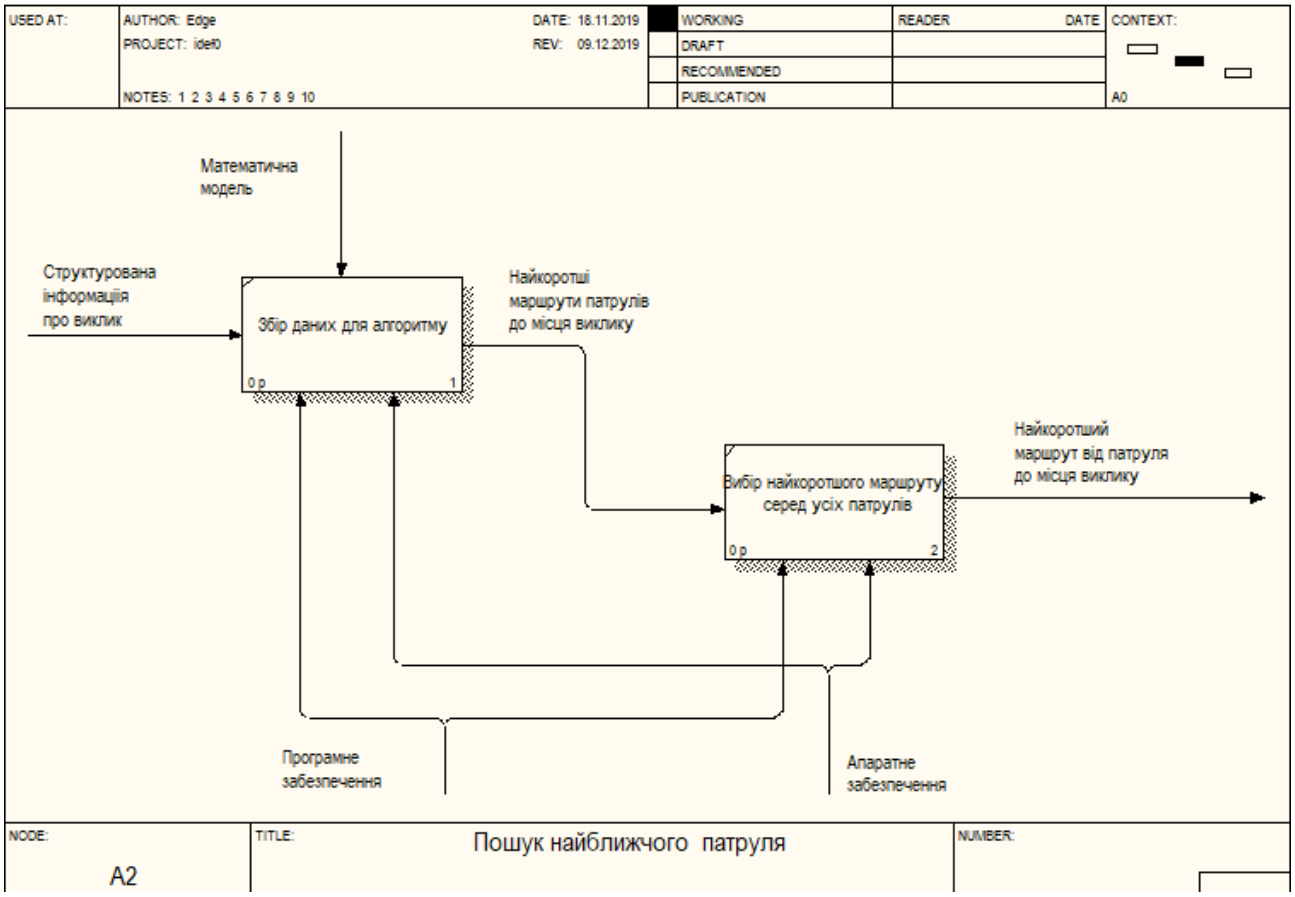


Рисунок 3.4 – Декомпозиція процесу пошуку найближчого патруля

Далі переходимо до створення діаграми варіантів використання. Виділимо три актори у системі – база даних, адміністратор та звичайний користувач. Адміністратор може переглядати інформацію про всі активні патрулі та керувати маршрутами патрулів. Також його завданням є направлення виклику патрулю,

який дістанеться до місця виклику найшвидше. А це буде обраховано системою. Користувач має можливість переглядати інформацію про свій патруль та про виклик, який надано його патрулю. Додатково маємо обробку виклику системою, розрахунок та пошук найближчого патруля до місця виклику та направлення цього патруля. Таким чином маємо зміну даних у БД. Диспетчер ініціює обробку виклику, надаючи інформацію про нього. Після цього система починає працювати з математичною моделлю та передає дані для адміністратора, який уже виконує свої функції керуванням патрулями. У результаті отримуємо діаграму варіантів використання для даної інформаційної системи (рис. 3.5) [16].

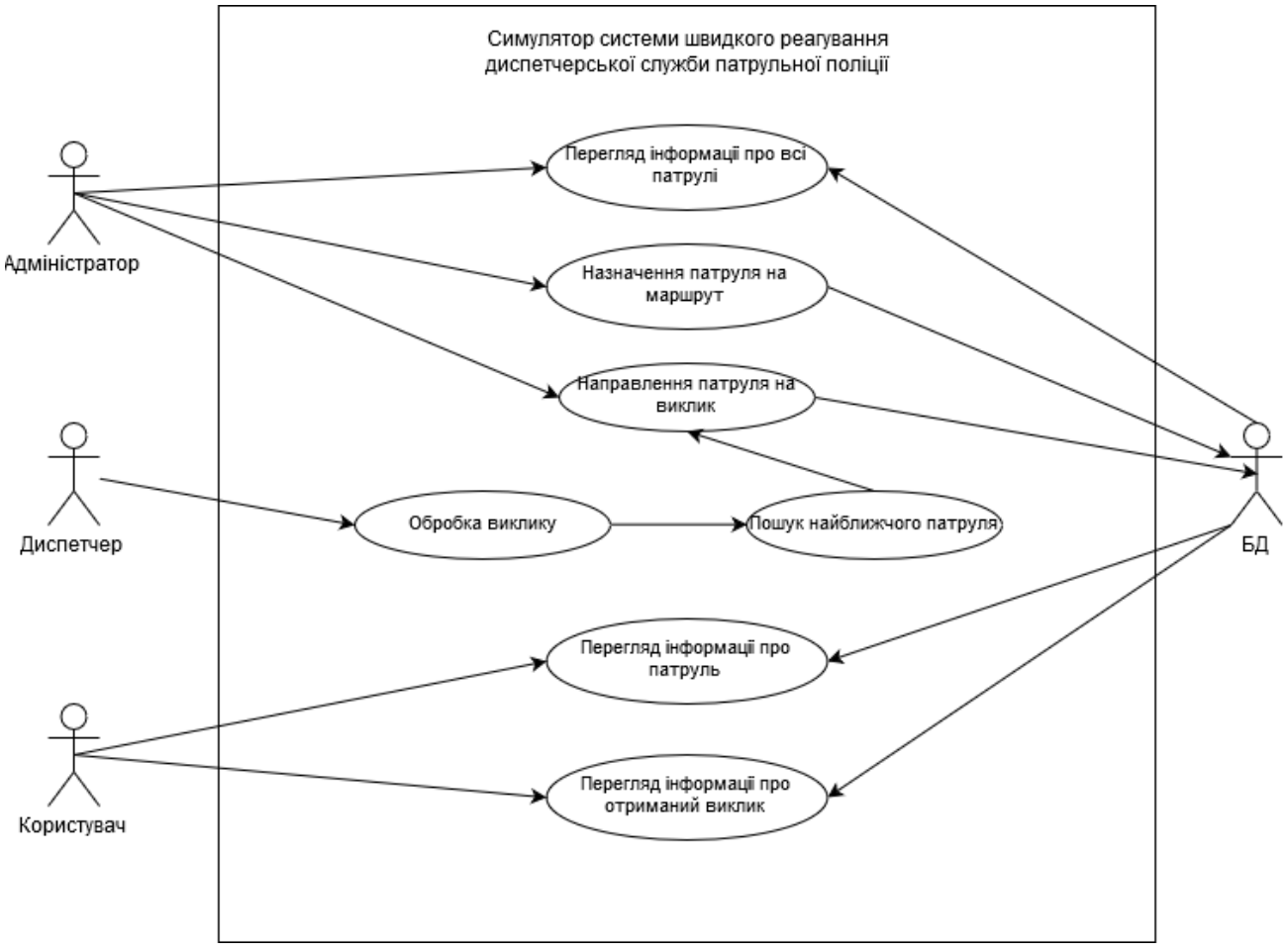


Рисунок 3.5 – Діаграма варіантів використання

3.4 Проектування бази даних

Для роботи системи потрібно спроектувати базу даних для збереження інформації про патрулі, виклики та маршрути. Дані в БД будуть додаватися адміном та автоматично при надходженні виклику та направленні патруля на нього. Також інформація буде використана для візуального відображення стану патрулів та викликів на карті.

Після аналізу та моделювання роботи майбутньої системи було зроблено висновки про необхідність наступних сутностей:

- користувач;
- координати;
- автомобіль;
- патруль;
- маршрути;
- виклик.

Після нормалізації бази даних було додано ще дві сутності для запобігання відношення множина до множини. Вони виглядають так:

- список координатів;
- список поліцейських.

У результаті проектуємо роботу бази даних та отримуємо схему БД (рис. 3.6). Проаналізуємо таблиці БД. Абсолютно усі сутності мають унікальний ідентифікатор, який використовує автоінкремент. Сутність «Користувач» додатково має поле роль. Тут міститимуться статичні дані про тип користувача, а саме адмін чи простий користувач. Детальніше про неї описано в таблиці 3.1 [17, 18].

Таблиця 3.1 – Сутність «Користувач»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	user_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	role	varchar(20)	-		Приймає значення “admin” або “user”. Означає роль користувача, що ввійшов у систему
3	username	varchar(30)	-		Логін користувача
4	user_password	varchar(20)	-		Пароль користувача
5	name	varchar(30)	-		Ім’я користувача
6	surname	varchar(30)	-		Прізвище користувача
7	birthdate	date	-		Дата народження користувача
8	sex	int(1)	-		Стать користувача
9	policeman_rank	varchar(30)	+		Звання користувача

Сутність «Координати» містить широту, довготу та значення для визначення порядкового номеру цієї точки у маршруті (табл. 3.2).

У таблиці «Автомобіль» міститься інформація про автопарк поліцейської дільниці. Це модель, колір та номерний знак (табл. 3.3).

«Патруль» є основоною сутністю та зберігає дані з інших сутностей, зв’язаних зовнішніми ключами(табл. 3.4).

Сутність «Маршрут» зберігає дані про діючі маршрути для патрулювання вулиць міста. Детальніше описано в таблиці 3.5.

Таблиця 3.2 – Сутність «Координати»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	coordinate_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	latitude	double	-		Широта в точці координат
3	longitude	double	-		Довгота в точці координат

Таблиця 3.3 – Сутність «Автомобілі»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	car_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	model	varchar(100)	-		Модель автомобіля
3	color	varchar(45)	+		Колір автомобіля
4	car_number	varchar(15)	-		Номер автомобіля

Таблиця 3.4 – Сутність «Патруль»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	patrol_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	patrol_status	int(1)	-		Статус патруля(приймає значення 1 – в роботі, 0 – знятий з роботи)
3	car_id	int(11)	-		Зовнішній ключ на сутність «Автомобіль»
4	route_id	int(11)	-		Зовнішній ключ на сутність «Маршрут»

Таблиця 3.5 – Сутність «Маршрут»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	route_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	route_name	varchar(100)	-		Назва маршруту

У таблиці «Виклик» маємо дані про сам виклик – дата та координати (табл. 3.6).

Додаткові таблиці «Список координатів» та «Список поліцейських» містять дані для зв’язування двох основних таблиць. У першому випадку це – координати і маршрут, до якого вони відносяться (табл. 3.7). А в іншому прикладі це – поліцейські, які відносяться до конкретного патруля (табл. 3.8).

Таблиця 3.6 – Сутність «Виклик»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	call_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	call_date	datetime	-		Дата та час прийняття виклику
3	call_latitude	double	-		Широта координати виклику
4	call_longotude	double	-		Довгота координати виклику
5	call_status	int(1)	-		Статус виклику(1 – направлений патрулю, 0 – не направлений патрулю)
6	patrol_id	int(11)	+		Зовнішній ключ на сутність «Патруль»

Таблиця 3.7 – Сутність «Список координатів»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	list_routes_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	route_id	int(11)	-		Зовнішній ключ на сутність «Маршрут»
3	coordinate_id	int(11)	-		Зовнішній ключ на сутність «Координата»

Таблиця 3.8 – Сутність «Список поліцейських»

№	Позначення в БД	Тип даних	Null	Додатково	Пояснення
1	list_patrol_id	int(11)	-	Автоінкре-мент	Первинний ключ
2	patrol_id	int(11)	-		Зовнішній ключ на сутність «Патруль»
3	user_id	int(11)	-		Зовнішній ключ на сутність «Користувач»

Далі було додано усі індекси та зовнішні ключі як посилання на сутності. Завдяки ним дані будуть зв’язуватися між сутностями, що показує їх цілісність. Використовуючи MySQL Workbench було візуалізовано схему бази даних для того, щоб показати її окремим скриншотом (рис. 3.6). Після цього скрипт можна експортувати в файл формату .sql та використати його для розташування на локальному сервері. Такий підхід допомагає зекономити час, який міг би бути використаний при ручному прописуванні скрипта бази даних. А вже після заповнення таблиць БД, можна знову експотрувати файл бази даних та викласти його на хостингу.

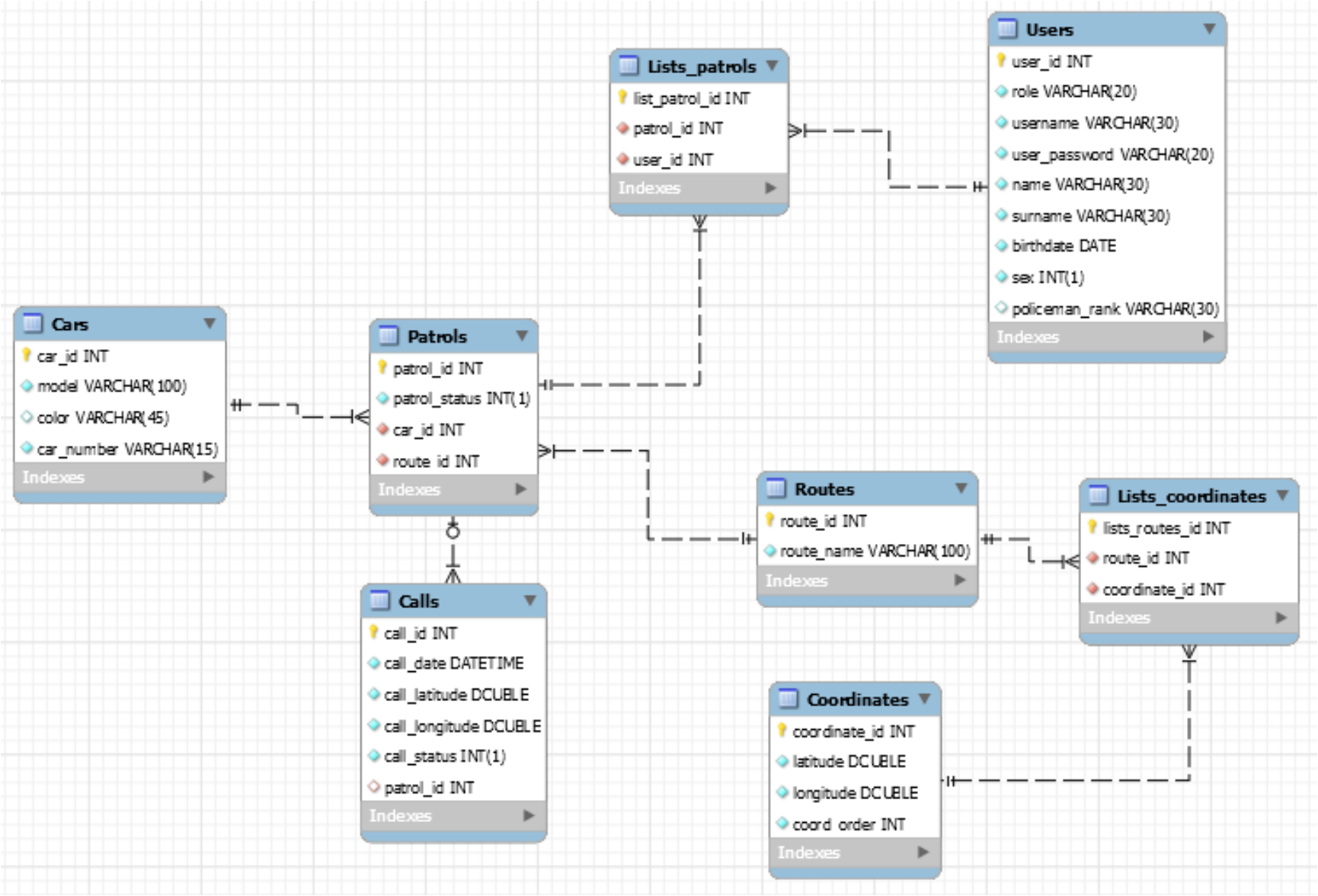


Рисунок 3.6 – Схема бази даних

3.5 Математична модель

Для реалізації пошуку найкоротшого шляху, а у даному випадку патруля, який найшвидше прибуде на місце виклику, буде використано алгоритм Дейкстри. Даний алгоритм дозволяє знайти шлях від однієї вершини графа до решти. До того ж він працює лише з додатніми вагами ребер, що в точності потрібно для симулятора системи.

Розберемо детальніше те, як він працює. Маємо зважений орієнтований граф *G(V, E)* без ребер негативної ваги. Необхідно знайти шляхи від деякої вершини *a* графа *G* до всіх інших вершин графа.

Кожній вершині із *V* поставимо мітку – мінімальну відому відстань від цієї вершини до *a*. Алгоритм працює покроково – на кожному кроці він проходить одну вершину і зменшує значення її мітки. Робота алгоритму завершається, коли буде пройдено усі вершини.

Мітки початкової вершини *a* привласнюємо значення *0*, мітки решти вершин – значення бескінечності. Це означає, що значення відстані від початкової вершини *a* поки що невідомі. Усі вершини графа позначаються як невідомі (рис. 3.7).

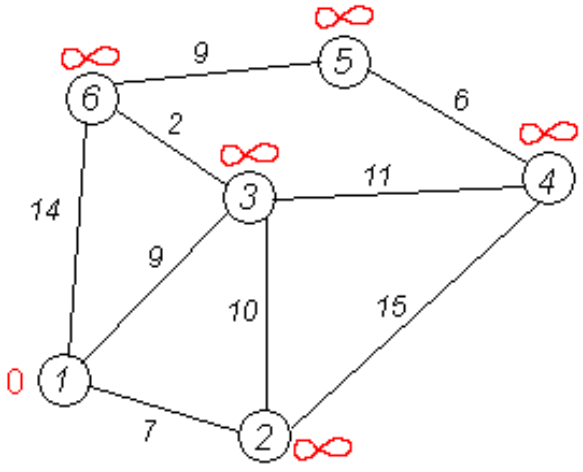


Рисунок 3.7 – Граф підготовлений для обходження

Якщо всі вершини були пройдені, алгоритм завершається. У іншому випадку, из тих вершин, що не були відвідані, обираємо вершину *u,* яка має мінімальну мітку. Розглядаємо всі можливі маршрути, в яких *u* є передостаннім пунктом. Вершини, в які ведуть ребра з *u,*окрім відмічених як відвідані, розглядаємо нову довжину шляху, яка рівна сумі значень поточної мітки *u* і довжини ребра,що з'єднує *u* з цим сусідом. Якщо отримане значення довжини менше значення мітки сусіда, то заміняємо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів помітимо вершуну *u* як відвідану і повторимо крок алгоритму. Кінцевий вигляд графа із всімнайкоротшими відстанями до усіх вершин показано на рисунку 3.8 [19, 20].

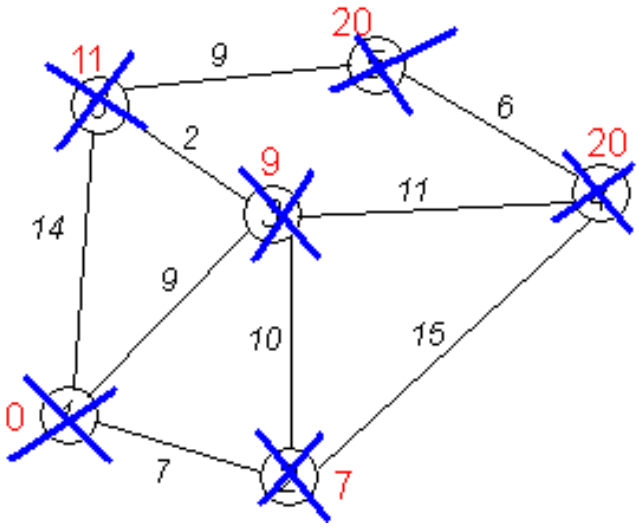


Рисунок 3.8 – Кінцевий вигляд графу

Таким чином отримаємо найкоротший шлях кожного патруля до місця виклику. Наступним етапом необхідно обрати серед усіх патрулів той, що прибуде за найменший час. Для цього обираємо найменше значення маршруту до точки виклику, знайдене раніше. При цьому варто враховувати додатково затори, напрям руху автомобіля та статус готовності патруля до отримання нового виклику.

Даний математичний алгоритм уже влаштований у Google maps. Тож його будемо просто використовувати. Та після його роботи програма буде обирати патруль, що прибуде на місце виклику швидше за всіх. Додатково враховуватимуться фактори, про які вказано вище. Таким чином у результаті матимемо рішення у вигляді одного патруля, якому буде надано виклик. А програмний продукт в комбінації з алгоритмом закладеним онлайн картами надаватиме це рішення.

4 РОЗРОБКА СИМУЛЯТОРА СИСТЕМИ ШВИДКОГО РЕАГУВАННЯ ДИСПЕТЧЕРСЬКОЇ СЛУЖБИ ПАТРУЛЬНОЇ ПОЛІЦІЇ

4.1 Налаштування середоваща для розробки

Перед початком роботи необхідно обрати середовща для написання програмного коду. Як було зазначено раніше, для розробки бази даних було використано MySQL Workbench (рис. 4.1), а також завдяки простішому доступу деколи використовувався phpMyAdmin (рис. 4.2).

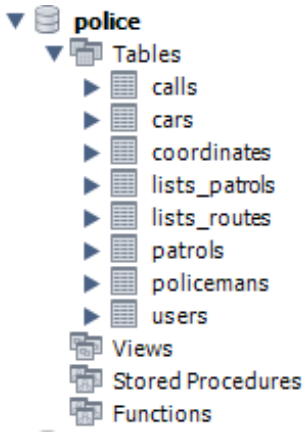


Рисунок 4.1 – База даних на локальному сервері

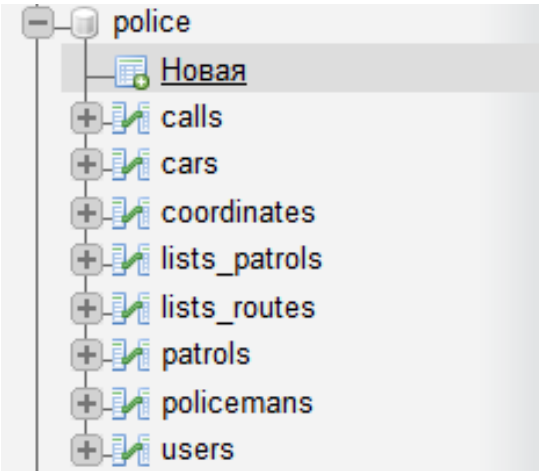


Рисунок 4.2 – Використання phpMyAdmin

Для написання коду було обрано Visual Studio Code (рис. 4.3). Даний програмний продукт досить невимогливий. Як і всі сучасні IDE дозволяє використовувати доповнення для простішої роботи з кодом (рис. 4.4).

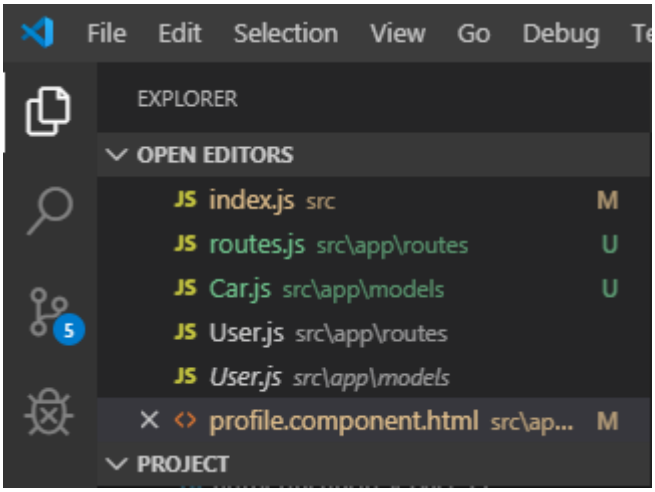


Рисунок 4.3 – Visual Studio Code

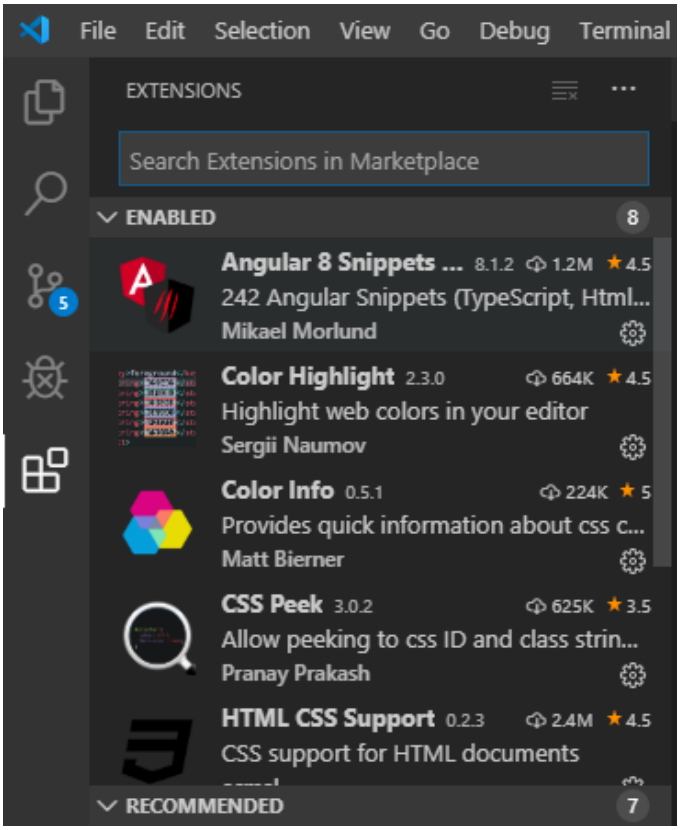


Рисунок 4.4 – Visual Studio Code

Таким чином було організовано роботу для розробки програмного продукту. За допомогою командного рядка та пакетного менеджера npm було додано усі необхідні бібліотеки в проект (рис. 4.5). Усі вони знаходяться у файлі package.json для того, щоб було простіше розвернути проект на іншому комп’ютері (рис. 4.6).

```
PS D:\University\diplom\project> npm install express
[..................] / rollbackFailedOptional: verb npm-session f468b1f57c0ee65f
```

Рисунок 4.5 – Приклад завантаження пакету в проект

```
"alert-node": "^2.0.3",
"body-parser": "^1.17.2",
"cors": "^2.8.4",
"ejs": "^3.0.1",
"express": "^4.17.1",
"express-jwt": "^5.3.1",
"jsonwebtoken": "^7.4.2",
"mysql": "^2.14.1",
"mysql2": "^1.6.1",
"rxjs": "~6.4.0",
"sequelize": "^5.21.2",
"tslib": "^1.10.0",
"zone.js": "~0.9.1"
},
```

Рисунок 4.6 – Вміст залежностей файлу package.json

Для більш комфортної роботи було розміщено проект на веб-ресурсі GitHub (рис. 4.7). Даний сервіс дозволяє зберігати проект програмного продукту та керувати його версіями. Таким чином отримуємо можлтвість безпечної розробки. Та при необхідності та виникненні помилок можна просто повернутися до попередньої версії програми, тим самим уникнувши ризиків при розробці [22].

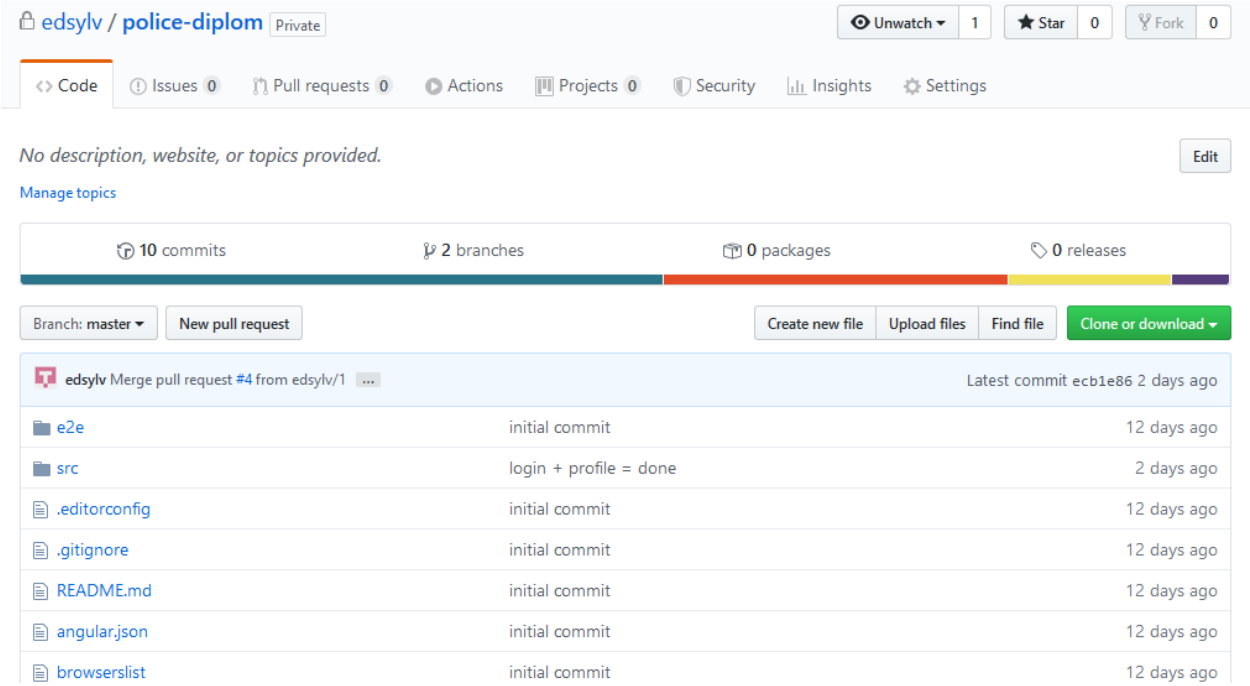


Рисунок 4.7 – Проект на GitHub

4.2 Налаштування роботи з базою даних

У першу чергу було підключено базу даних та налаштовано роботу з нею. Для цього використано бібліотеку Sequelize (рис. 4.8). Це сучасний підхід для роботи з базами даних у даному стеку технологій. Бібліотеку додано до проекту за допомогою вищевказаного пакетного менеджеру npm. У якості даних для вхожу вказано схему бази даних, хост, діалект, користувача та пароль. Даний модуль було експортовано для використання в інших файлах проекту. Винисення підключення в окремий файл дозволяє спростити зміну налаштувань підключання у випадку, якщо це буде необхідо зробити у майбутньому або під час самої розробки [23].

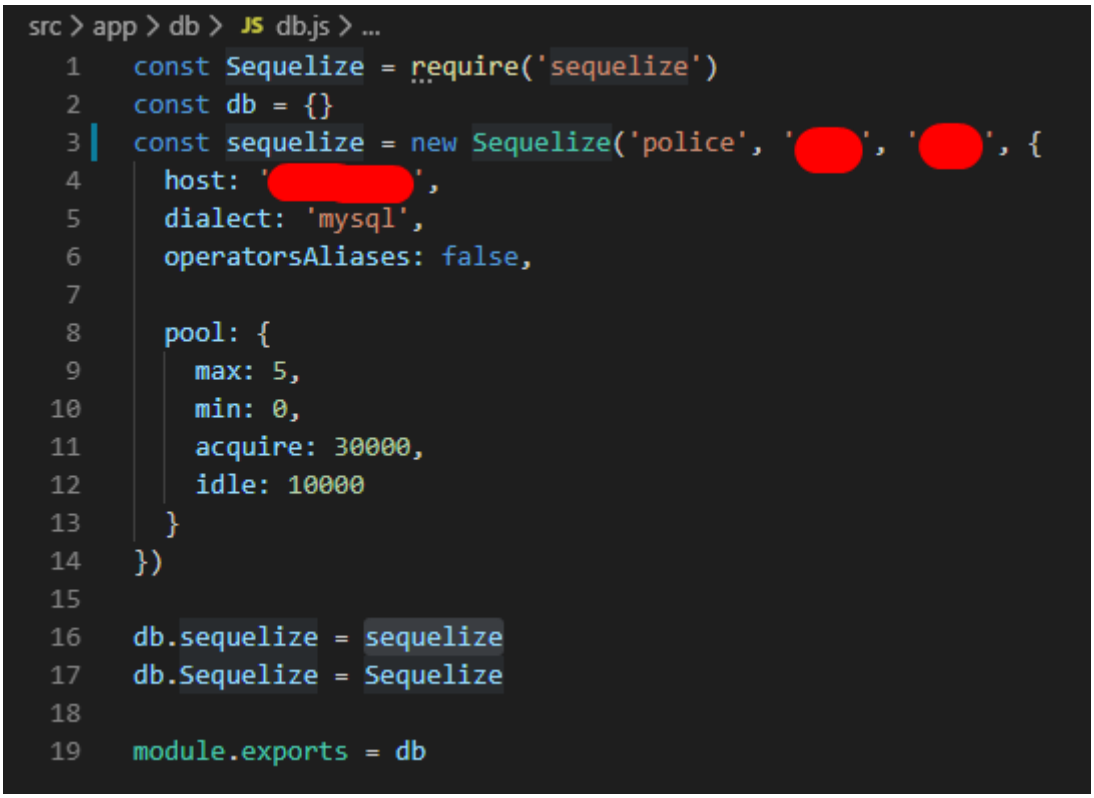
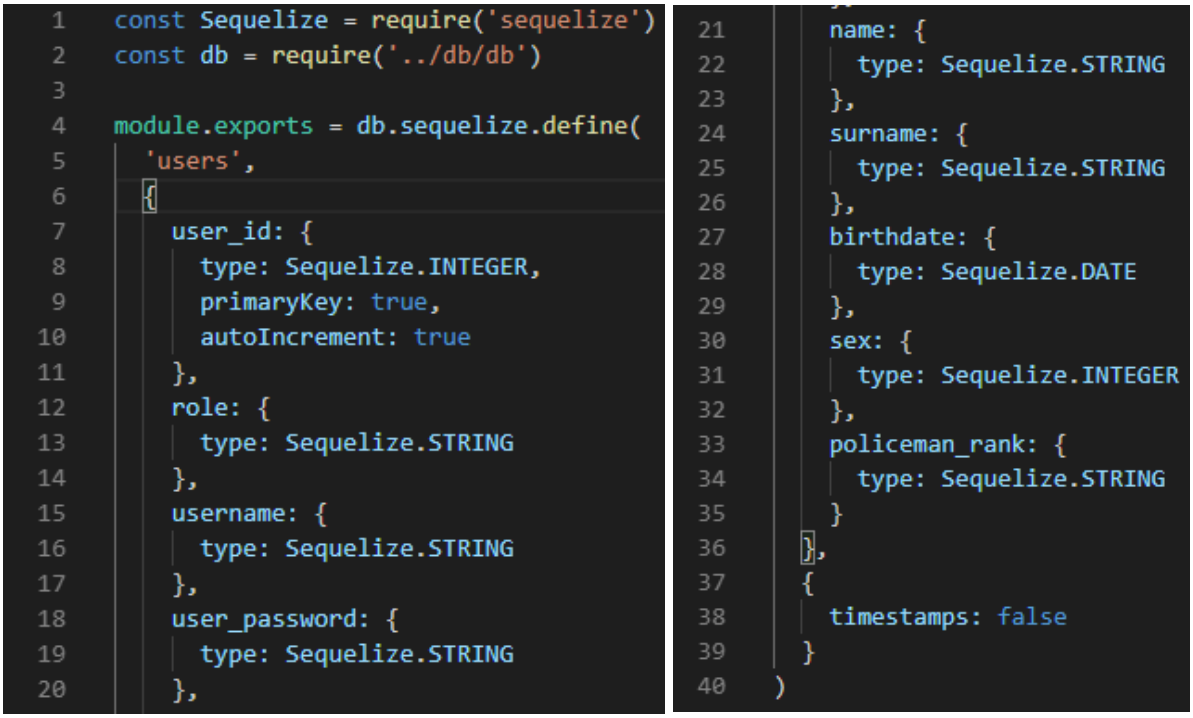


Рисунок 4.8 – Підключення до бази даних

Сама бібліотека Sequelize передумовує створення моделей для сутностей бази даних. Таким чином робимо таку собі копію сутності, але уже у вигляді об’єкту в Javascript. Для прикладу наведено модель бази даних користувача на рисунку 4.9. Як бачимо, тут прописуємо типи даних та властивості об’єкта для подальшого звернення до нього [23].



```
const Sequelize = require('sequelize')
const db = require('../db/db')

module.exports = db.sequelize.define(
  'users',
  {
    user_id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    role: {
      type: Sequelize.STRING
    },
    username: {
      type: Sequelize.STRING
    },
    user_password: {
      type: Sequelize.STRING
    },
    name: {
      type: Sequelize.STRING
    },
    surname: {
      type: Sequelize.STRING
    },
    birthdate: {
      type: Sequelize.DATE
    },
    sex: {
      type: Sequelize.INTEGER
    },
    policeman_rank: {
      type: Sequelize.STRING
    }
  },
  {
    timestamps: false
  }
)
```

Рисунок 4.9 – Модель сутності «Користувач»

4.3 Робота з локальним сервером

Для подальшої розобки та підключення до бази даних необхідно створити локальний сервер за допомогою Node.js [12]. Зробити це досить просто, але даний етап є важливим. Помічником є бібліотека express, яка була встановлена як і решта бібліотек, за допомогою пакетного менеджеру npm та додана автоматично в файл package.json. При налаштуванні підключення вказуємо порт та власне відслідковуємо саме підключення (рис. 4.10).

```
var express = require('express')
var cors = require('cors')
var bodyParser = require('body-parser')
var app = express()
var port = process.env.PORT || 3000
app.use(bodyParser.json())
app.use(cors())
app.use(
  bodyParser.urlencoded({
    extended: false
  })
)

var Users = require('./app/routes/User')
var Routes = require('./app/routes/routes')

app.use('/users', Users)
app.use('/', Routes)

app.listen(port, function() {
  console.log('Server is running on port: ' + port)
})
```

Рисунок 4.10 – Налаштування локального серверу

4.4 Розробка веб-системи

Як уже зазначалося вище, для розробки програмного продукту було обрано фреймворк Angular 6. Він є сучасним і в той же час не найбільш новим. Такий вибір можна пояснити тим, що дана версія фреймворку уже показала свою роботоздатність та повністю задовольняє усім вимогам, які задані даною системою. Суть фреймворку полягає у створенні компонентів та їх використанні на веб-сторінці. Таким чином компонент – це частина веб-сторінки у вигляді блоку або веб-сторінка вцілому. Даний підхід дозволяє легко маніпулювати даними та спрощувати доступ до окремих блоків коду. Це

відбувається за рахунок розподілення компонентів за функціями та категоріями [24].

Для прикладу розглянемо розробку головної сторінки, яка у даному випадку не надає ніякого функціоналу окрім переходу до авторизації. Саме так і має бути, адже увесь функціонал має бути закритим. Вигляд гололовної сторінки представлено на рисунку 4.11.

Рисунок 4.11 – Головна сторінка сайту

На даному скриншоті маємо дві кнопки, які виконують однакову функцію – перехід до входу в систему. Зупинемося детальніше на самій структурі файлів для роботи системи (рис. 4.12).

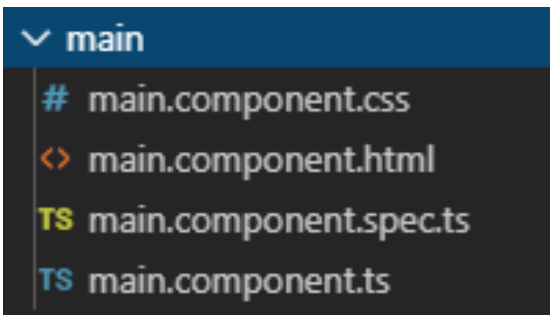


Рисунок 4.12 – Компонент головної сторінки

Як бачимо, до самого компоненту входять 4 файли. Файл стилів, у який прописуємо всі стилі для компонента, файл розмітки – те, що буде представлено на екрані користувачу, файл класу компонента, в якому будуть проведені всі маніпуляції з даними та необов'язковий файл з описом компонента.

Таким чином, у коді було встановлено власний тег для даного компонента, який буде використовуватися у розмітців подальшому для його виклику (рис. 4.13). У даному блоці коду також задано шляхи до файлів розмітки та стилів.

```
@Component({
  selector: 'app-main',
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css']
})
```

Рисунок 4.13 – Компонент

Ієрархія компонентів може бути якою завгодно. Власноруч створений компонент може викликатися іншим таким же компонентом. Так, на даному прикладі, окремим компонентом було винесено «шапку» сайту (4.14-4.15). Це зроблено для того, щоб не прописувати однаковий код на кожній сторінці, а просто використовувати компонент. А вже завдяки перевірками на авторизацію та роль відображати необхідні розділи системи (рис.4.16).

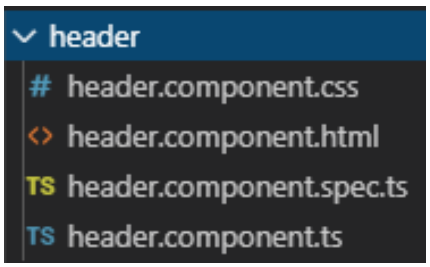


Рисунок 4.14 – Файли компонента «шапки» сторінки

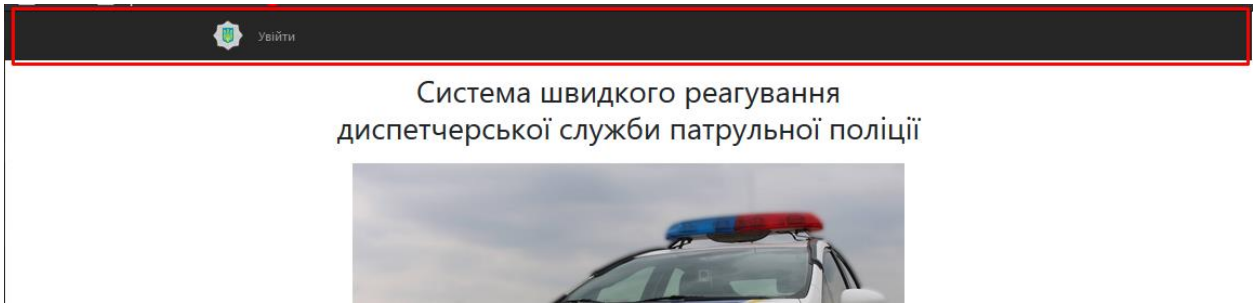


Рисунок 4.15 – «Шапка» сторінки

```
<div class="container d-flex flex-column flex-md-row justify-content-left">
    <a class="py-2 d-none d-md-inline-block" href="/">
        <img class="favicon" src="../../../favicon.ico" width="40" height="40" alt="favicon">
    </a>
    <a *ngIf="!auth.isLoggedIn()" class="py-2 pt-3 d-none d-md-inline-block" routerLink="/login">
        Увiти
    </a>
    <a *ngIf="auth.isLoggedIn()" class="py-2 pt-3 d-none d-md-inline-block" routerLink="/">
        Головна
    </a>
    <a *ngIf="auth.isLoggedIn()" class="py-2 pt-3 d-none d-md-inline-block" routerLink="" (click)="aut
        Вийти
    </a>
</div>
```

Рисунок 4.16 – Відображення розділів меню за умовою

Такі можливості Angular дозволяють зробити веб-сайт більш динамічним та простішим з точки зору розробки.

Важливим етапом є налаштування маршрутів для веб-сайту. Для цього використовуємо пакет Router. У головному компоненті (рис. 4.17), який власне відповідає за відображення всього на сторінках, налаштовуємо усі шляхи наявні в системі та їх відповідність компонентам (рис. 4.18).

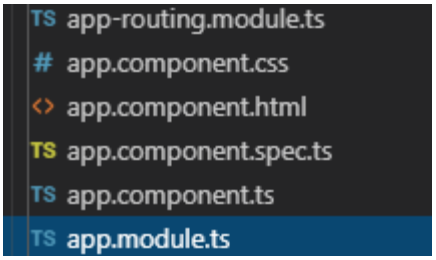


Рисунок 4.17 – Головний компонент

```
const appRoutes: Routes = [
  {path: '', component: MainComponent},
  {path: 'login', component: LoginComponent},
```

Рисунок 4.18 – Маршрути веб-сайту

Ще одним важливим етапом є налаштування авторизації та перевірка виконання входу в систему. Для цього було використано сервісе AuthenticationService. При виконанні входу, користувачу привласнюється автоматично згенерований токен, за яким і визначається те, чи в даний момент виконано вхід (рис. 4.19).

```
constructor(private http: HttpClient, private router: Router) {}

private saveToken(token: string): void {
  localStorage.setItem('usertoken', token)
  this.token = token
}

private getToken(): string {
  if (!this.token) {
    this.token = localStorage.getItem('usertoken')
  }
  return this.token
}
```

Рисунок 4.19 – Привласнення токена користувачу

Вище вже було вказано про роботу з маршрутами на веб-сайті. Та варто ще пояснити спосіб взаємодії адреси сайту або маршруту з сервером. На боці серверу виконується відслідковування переходу за конкретними маршрутами. Таким чином, коли виконується перехід на вказану для сервера адресу, ініціюється виконання того чи іншого запиту, записаного в коді (рис. 4.20). На даному рисунку показано приклад POST запиту. Тобто, після введення логіна

та пароля, користувач натискає кнопку входу. При цьому поточна адреса на сайті «../login». Та виконання запиту відбудеться лише після натиснення відповідної кнопки. Після цього відбувається перевірка на вхід і маємо результат.

```
users.post('/login', (req, res) => {
    User.findOne({
      where: {
        username: req.body.username,
        user_password: req.body.user_password
      }
    })
      .then(user => {
        if (user) {
          let token = jwt.sign(user.dataValues, process.env.SECRET_KEY, {
            expiresIn: 1440
          })
          res.json({ token: token })
        } else {
          res.send('User does not exist')
        }
      })
      .catch(err => {
        res.send('error: ' + err)
      })
})
```

Рисунок 4.20 – Виконання запиту сервером

На боці користувача це виглядає наступним чином. На рисунку 4.21 зображено саму форму входу. Необхідно увести всі дані для продовження. У протилежному випадку відбудеться повідомлення про помилку (рис. 4.22). Якщо дані було введено неправильно, то помилка буде виглядати відповідно (рис. 4.23).

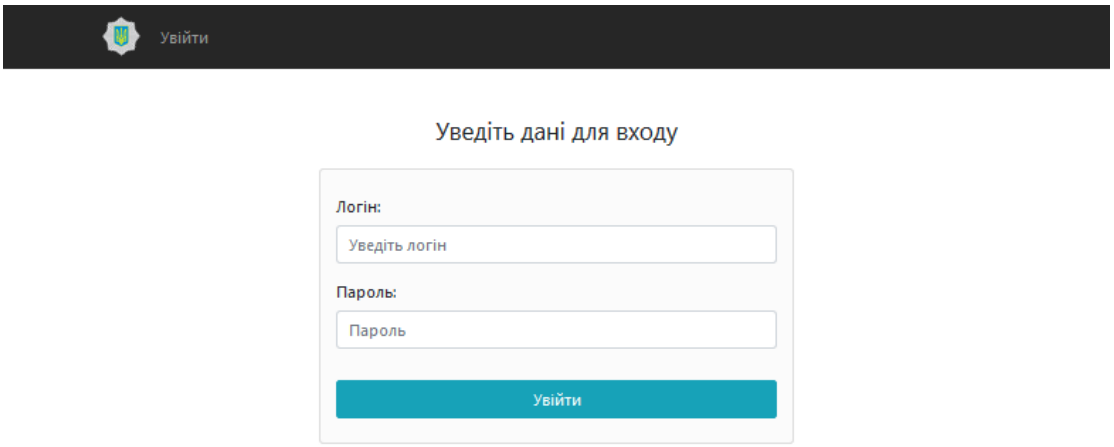


Рисунок 4.21 – Форма входу

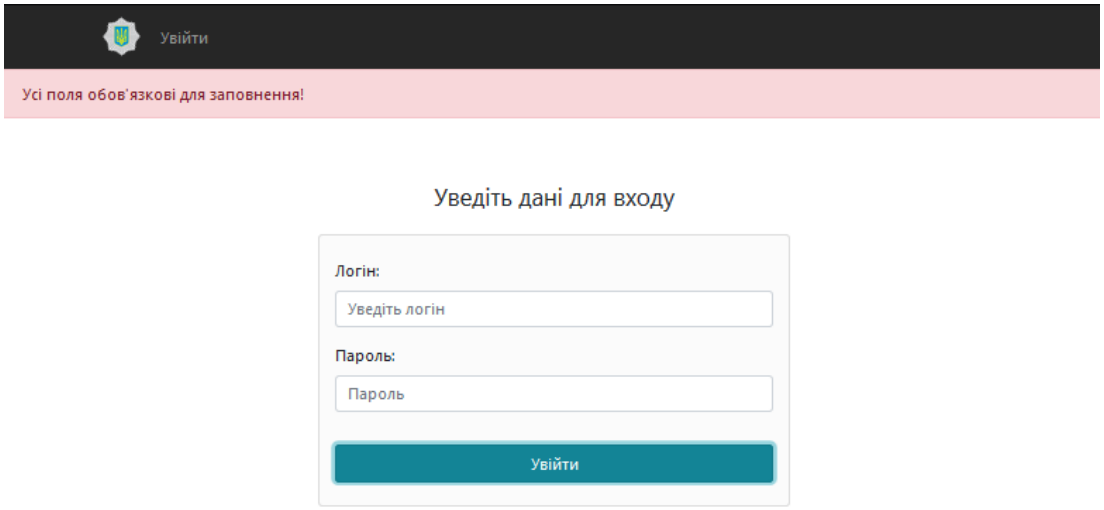


Рисунок 4.22 – Помилка про необхідність увести всі дані

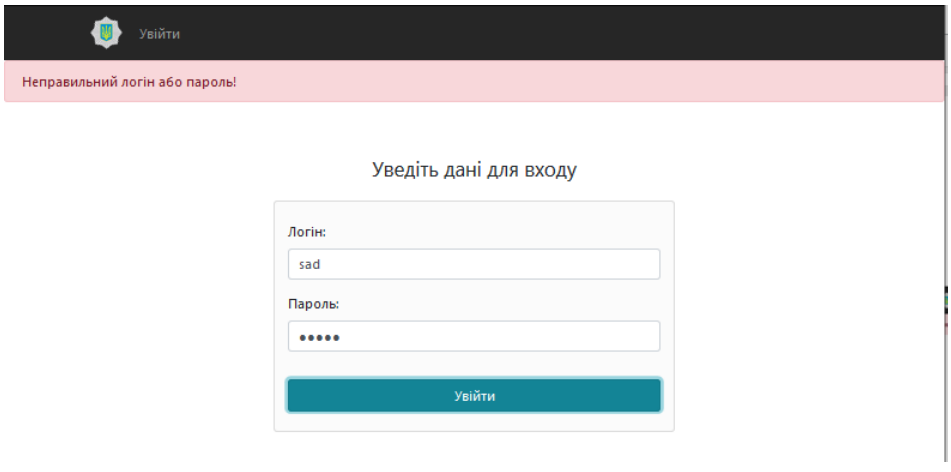


Рисунок 4.23 – Помилка про неправильність уведених даних

Варто зазначити ще використання вбудованих стилів Bootstrap. Було додано посилання на дану бібліотеку в основному файлі розмітки html (рис. 4.24).

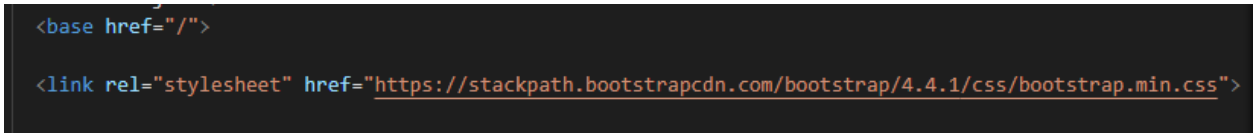


Рисунок 4.24 – Посилання на бібліотеку стилів

Використання стилів таким чином дуже просте та не потребує значних знань. Можна за допомогою лише одного класу надати блоку необхідного вигляду. Це може бути як попередження чи успішне введення даних. Також є багато варіантів візуалізувати кнопки та форми (рис. 4.25).

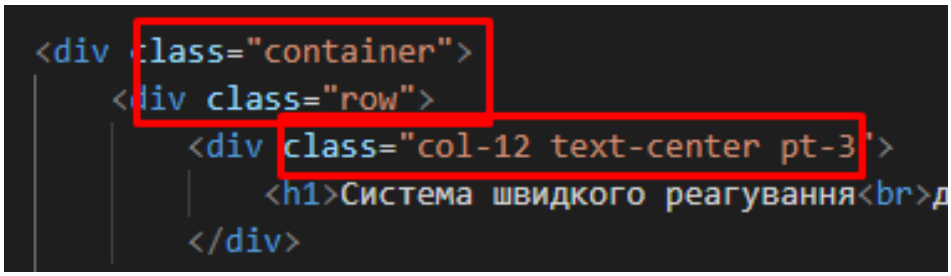


Рисунок 4.25 – Використання bootstrap класів

Подальша розробка проходила в такому ж форматі. Тобто, використання компонентів є основою. Увесь сайт реалізовано таким же чином. Кожна веб-сторінка має окремий компонент, який включає розмітку та сам клас. Варто також зазначити, що файли цього класу мають розширення .ts (рис. 4.26).

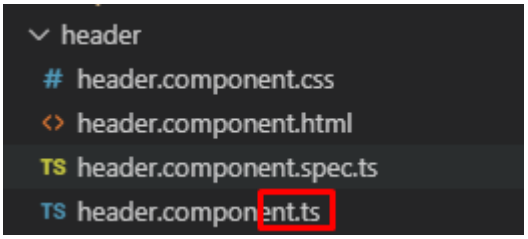


Рисунок 4.26 – Розширення файлу класу

Такі файли приймають ту ж мову Javascript, але вже нову її версію. Проте при самій розробці це не особливо впливає на процес.

Подальшого представлення компонентів не буде винесено сюди. Програмний код викладено у додатку Б.

ВИСНОВКИ

У результаті виконання роботи було проведено повноцінну розробку програмного продукту, а саме, симулятора системи швидкого реагування диспетчерської служби патрульної поліції. Починаючи з аналізу предметної області і завершуючи самою розробкою. Таким чином пройдено усі стадії реалізації проекту.

Після аналізу даної сфери діяльності та саме аналогів, які доступні на ринку, було визначено важливість розробки та роботи в цьому напрямі. Наступним етапом було сплановано роботу та саме виділено пункти, з яких складається процес реалізації системи. Важливим етапом є встановлення дедлайнів та часу виконання тих чи інших процесів. Також було продумано ризики, які могли б виникнути у процесі роботи над проектом та виконано необхідні дії для їх уникнення.

Наступний етап – моделювання роботи системи дозволив чітко окреслити межі майбутньої розробки та технології для використання. Тут же було розроблено схему бази даних та продумано всі варіанти її використання.

І, врешті-решт, розробка системи проводилася протягом всього строку виконання проекту, починаючи після завершення аналізу аналогів.

У результаті, система має чітко окреслену сферу діяльності та повінстю задовольняє поставлені на етапі моделювання задачі. Сюди входить адекватна робота математичного алгоритму, тобто, пошуку патруля, який прибуде на місце виклику найшвидше. Також варто зазначити, що база даних повністю задовольняє вимогам системи та працює достатньо швидко для зручної роботи користувачів. Інтерфейс веб-сайту є зрозумілим та легким для сприйняття навіть найбільш недосвідченим користувачем. Забезпечення захисту даних дозволяє перейти на відповідні сторінки тільки авторизованим користувачам та відповідним ролям.

Отже, можна з упевненістю говорити про успішне виконання проекту. Головна мета виконана, система працює, функціонал розроблено саме той, який задумувалося на початку та винесено у вимоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільний додаток “My Pol”[Електронний ресурс] – Режим доступу: https://appmypolice.com/ua/

2. Тачку [Електронний ресурс] – Режим доступу: https://umbrellait.com/ru/how-to-build-app-like-uber- development-guide-to-a-million-dollar-project/

3. Создаем приложение с механикой Uber [Електронний ресурс] – Режим доступу: http://www.tachku.com

4. OnTaxi [Електронний ресурс] – Режим доступу: https://ontaxi.com.ua

5. Флэнаган, Дэвид JavaScript. Карманный справочник / Дэвид Флэнаган. - М.: Вильямс, 2015. - 320 с.

6. Козловский, Павел Разработка веб-приложений с использованием AngularJS / Павел Козловский , Питер Бэкон Дарвин, Павел Козловский. - Москва: Мир, 2014. - 394 с.

7. Аткинсон Л. MySQL. Библиотека профессионала / Аткинсон Л. – М.: Вильямс, 2010. – 624 с.

8. Уилтон П. SQL для начинающих / Уилтон П., Колби Дж. – М.: Вильямс, 2011. – 496 с.

9. Ананий В. Алгоритмы: введение в разработку и анализ./ Ананий В., ЛевитинА.// В сб.:Introduction to The Designand Analysis of Aigorithms. – iМ.: «Вильямс», 2010. – с. 212-215.

10. Свами М. Графы, сети и алгоритмы / Свами М., Тхуласираман К. – М: Мир, 1984. – 455 с.

11. Steve, Fenton TypeScript For JavaScript Programmers / Steve Fenton. - Москва: Наука, 2012. - 505 с.

12. Кантелон, М. Node.js в действии / М. Кантелон. - М.: Питер, 2015.

- 810 с.

13. Основы npm [Електронний ресурс] – Режим доступу: https://nodejs-junior-developer-traini.gitbooks.io/super-node-js-book-2/osnovi-npm.html

14. WAMPSERVER [Електронний ресурс] – Режим доступу: http://www.wampserver.com/ru/

15. Вичугова, А. А. Методы и средства концептуального проектирования информационных систем: сравнительный анализ структурного и объектно-ориентированного подходов / А.А. Вичугова. - М.: Синергия, 2014. - 631 с.

16. Ипатова, Э. Р. Методологии и технологии системного проектирования информационных систем. Учебник: моногр. / Э.Р. Ипатова. - М.: Флинта, 2016. - 300 с.

17. Буч, Гради Введение в UML от создателей языка / Гради Буч , Джеймс Рамбо , Ивар Якобсон. - М.: ДМК Пресс, 2015. - 496 с.

18. Завадський І.О. Основи баз даних:[Навч. посіб.] / І.О. Завадський. – К. : Видавець І.О. Завадський, 2014. – 192 с.

19. Мартишин, С.А. Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench: Методы и средства проектирования информационных систем и техноло / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. - М.: Форум, 2018. - 61 с.

20. Л. Є. Базилевич. Дискретна математика у прикладах і задачах : теорія множин, математична логіка, комбінаторика, теорія графів. — Математичний практикум. — Львів, 2013. — 486 с.

21. Дискретна математика: підручник / Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина ; за наук. ред. В. В. Пасічника ; М-во освіти і науки. молоді та спорту України. — 3-тє вид., виправл. та доповн. — Львів: Магнолія-2006, 2013. — 432 с. : іл. — (Серія «Комп'ютинг»). — Бібліогр.: с. 430—431

22. GitHub [Електронний ресурс] – Режим доступу: https://github.com/

23. Sequelize [Електронний ресурс] – Режим доступу: https://sequelize.org/

24. Angular и TypeScript. Сайтостроение для профессионалов / Я. Файн, А. Моисеев; Питер. 2018. — 464 с.

ДОДАТОК А

А.1 Ідентифікація мети IT-проекту методом SMART

Мета проекту: проектування симулятора швидкого реагування диспетчерської служби патрульної поліції. Сюди входить вибір математичного алгоритму пошуку найкоротшого маршуруту, розробка системи у вигляді веб-додатка. Важливим пунктом є максимальне збільшення швидкості реагування та точності розрахунку при направленні автомобіля на місце виклику.

А.2 Планування змісту структури робіт IT-проекту

WBS – це графічне подання згрупованих елементів проекту у вигляді пакета робіт, які ієрархічно пов’язані з продуктом проекту. На верхньому першому рівні WBS фіксується продукт проекту – система швидкого реаугвання диспетчерської служби патрульної поліції. Наступний рівень відповідає основним діям проекту. Далі проводиться розбиття цих дій до елементарних робіт, тобто, тих, які мають один чіткий результат з одним відповідальним та на яку можна обчислити витрати праці та тривалість виконання.

Продуктом у даному проекті є система швидкого реагування диспетчерської служби патрульної поліції. Основними діями в проекті є: аналіз предметної області систем швидкого реагування, розробка симулятора системи та формування документації. Після цього кожну з вищевказаних дій розбиваємо до елементарних робіт. Найважливішою і складною гілкою у

проекті є розробка симулятора системи. Даний пункт розбиваємо на проектування БД, створення додатку та тестування. При проектуванні бази даних маємо стандартні кроки і відповідно зазначаємо їх – побудова логічної моделі, створення фізичної моделі та реалізація взаємодії БД з програмою. Для створення програмного додатку визначено такі кроки: розробка симулятора виклику, формування дизайну, реалізація функціоналу та алгоритму розрахунку. При тестуванні будемо використовувати такі його види: функціональне, тестування продуктивності, зручності використання та інтерфейсу користувача.

Кінцевий вигляд WBS структури представлено на рисунку А.1.

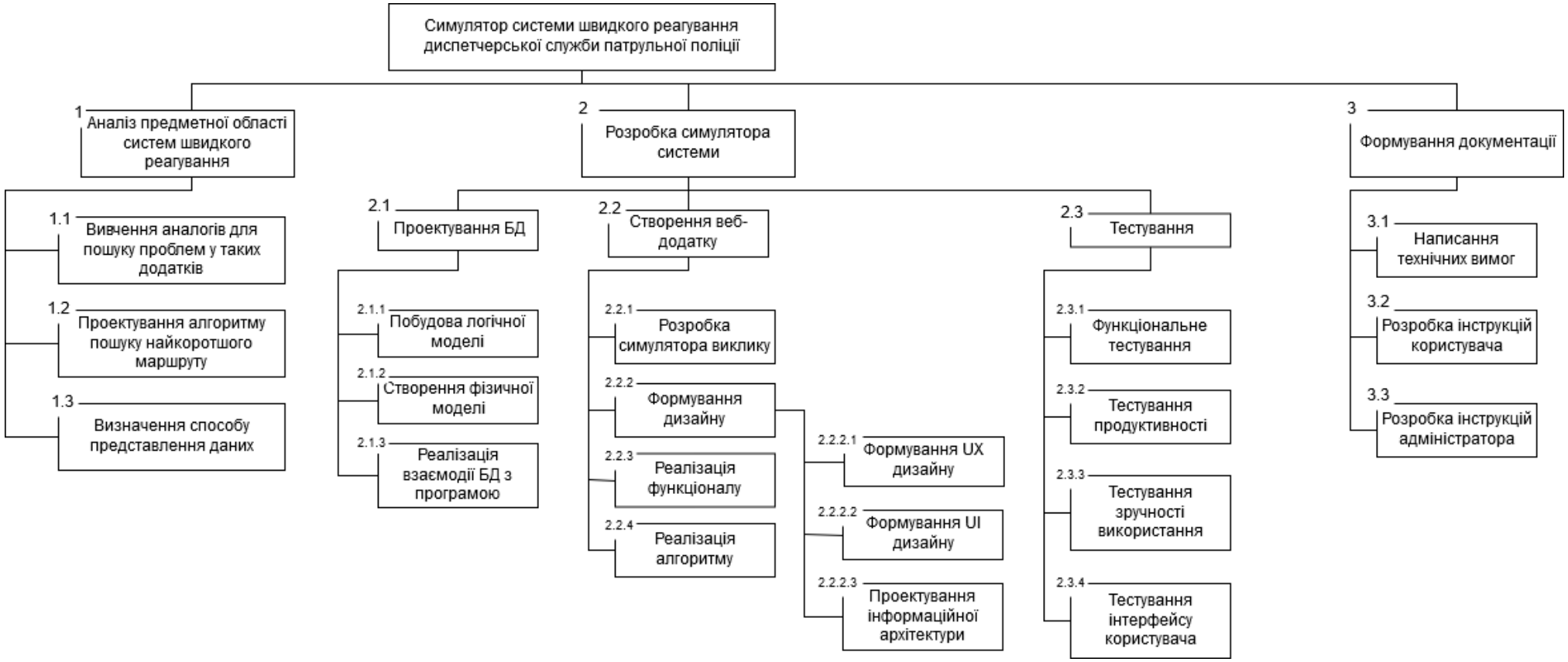


Рисунок А.1 – WBS структура

Також на основі виществореної структури робіт необхідно розробити OBS структуру. OBS-структура проекту – організаційна структура виконавців проекту. Визначається за переліком пакетів робіт нижнього рівня кожної гілки WBS-структури. Представляється відповідальними за виконання елементарних робіт.

Організаційна структура представляє собою графічне відображення учасників проекту та їх відповідальних осіб, які задіяні в реалізації проекту. На верхньому рівні OBS розташована команда проекту.

На наступному рівні фіксуються виконавці: організації, відділи тощо. Потім, рівнем нижче, для кожного виконавця вказують прізвища конкретних осіб, які будуть відповідати за виконання елементарних робіт.

Таким чином маємо два учасники проекту – розробник (Силенко) та дипломний керівник (Шендрик). Зрозуміло, що більшу частину робіт виділено на розробника. Проте деякі пункти неможливо зробити без втручання керівника. Будь то допомога при аналізі чи підказки у ході розробки. Остаточний вигляд OBS структури маємо на рисунку А.2.

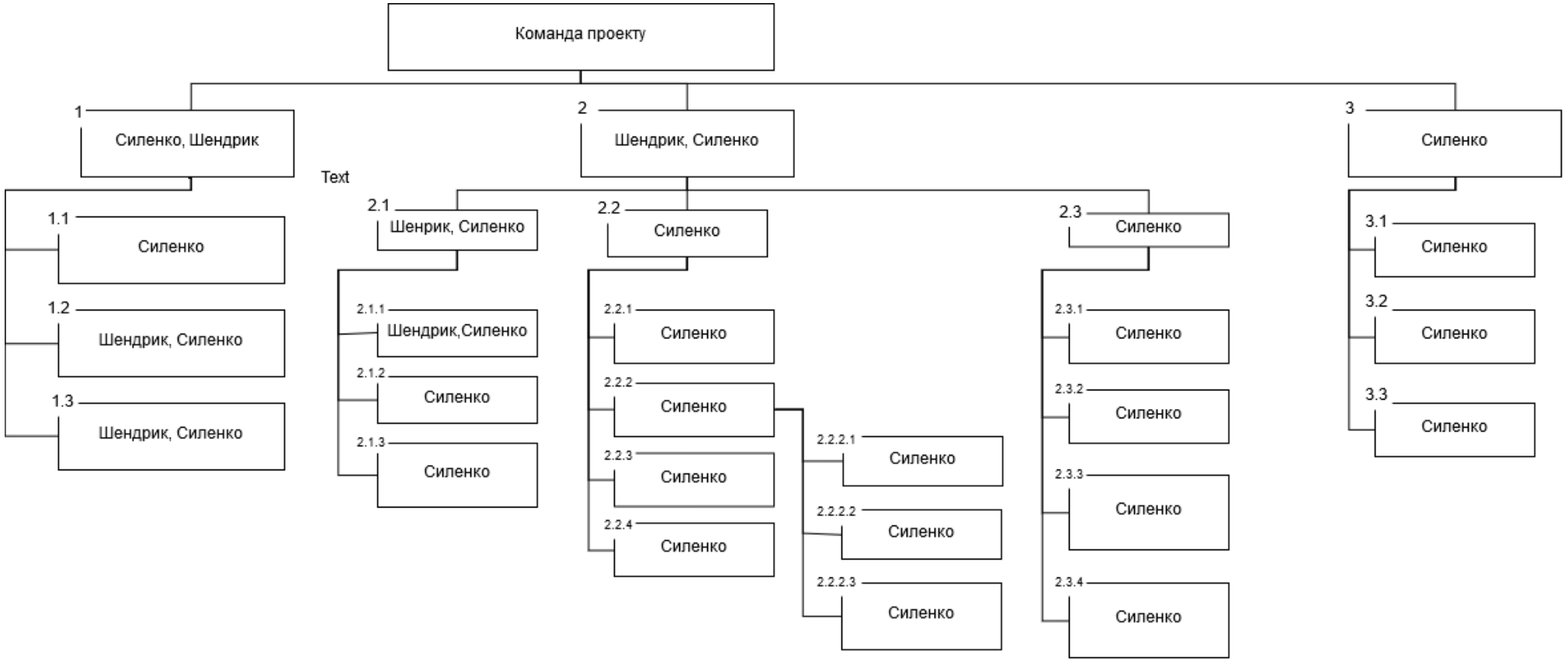


Рисунок А.2 – OBS структура

А.3 Побудова календарного графіку виконання IT - проекту

Для того щоб мати реальне уявлення про тривалість виконання робіт з урахуванням обмеженості у використанні ресурсів було побудовано календарний графік робіт у вигляді діаграми Ганта, яку наведено на рисунку А.3. Вона є реальним розподілом робіт за календарними датами, тобто своєрідним розкладом виконання робіт.

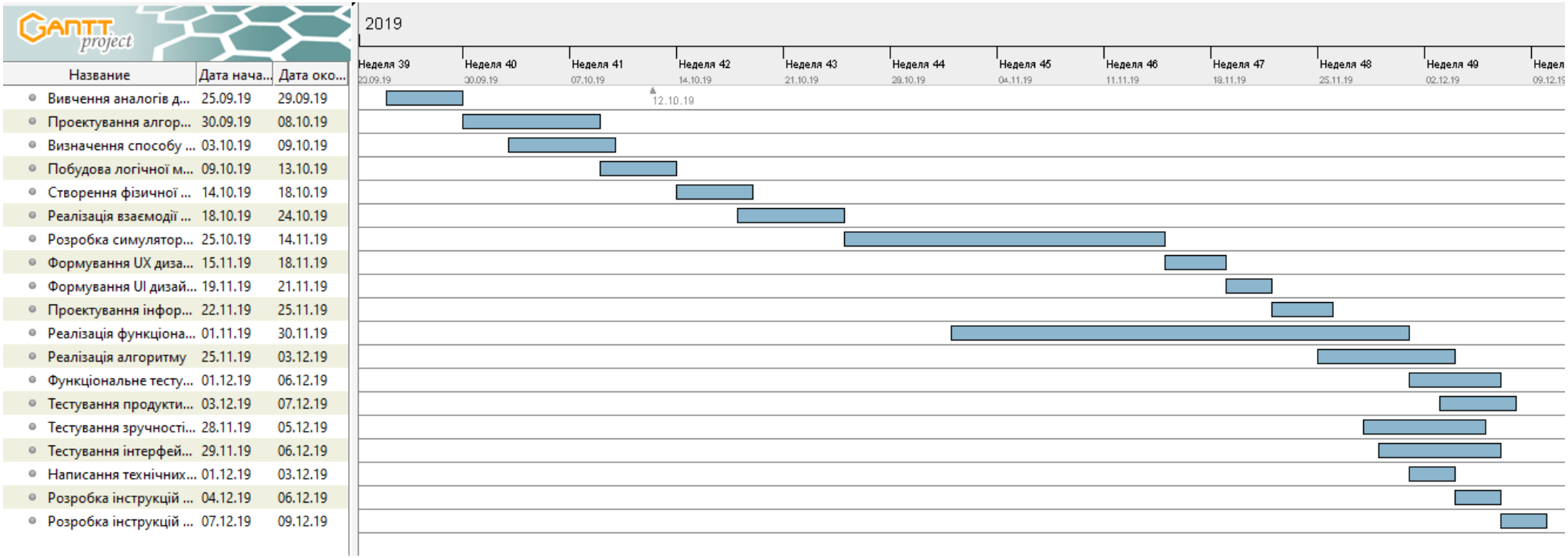


Рисунок А.3 – Діаграма Ганта

За початок роботи було обрану дату отримання теми дипломного проекту. Починаючи з цієї точки було визначено тривалість виконання усіх робіт відповідно діаграмі WBS, дати їх початку та завершення. Також за діаграмою Ганта можна побачити дату завершення роботи над проектом.

А.4 Планування ризиків проекту

Після визначення всіх робіт проекту та календарних планів проекту необхідно продумати можливі ризики, які можуть виникнути у ході реалізації проекту.

Було виділено наступні ризики у даному проекті:

- R1 – пропущені помилки у ході розробки;
- R2 – недотримання календарного плану;
- R3 – зміна ТЗ на етапі розробки;
- R4 – хвороба розробника;
- R5 – затримка правок замовника;
- R6 – неоптимізована робота БД;
- R7 – проблеми з технічним забезпеченням.

Далі шляхом експертної оцінки було визначено ймовірність виникнення усіх ризиків і побудовано відповідну таблицю (таблиця А.1).

Таблиця А.1 – Ймовірність виникнення ризиків

Ймовірність виникнення	R1	R2	R3	R4	R5	R6	R7
слабкоймовірний							
малоймовірний							
ймовірний							
дуже ймовірний							
майже можливий							

Далі було побудовано таблицю можливих втрат при виникненні ризиків (таблиця А.2).

Таблиця А.2 – Втрати при виникненні ризиків

Значимість втрат	R1	R2	R3	R4	R5	R6	R7
мінімальна							
низька							
середня							
висока							
максимальна							

На основі цих двох таблиць було побудовано матрицю ймовірність-втрати для оцінки ризиків (таблиця А.3). У даній матриці світлим кольором позначено неважливі ризики, темнішим – помірні, темним – критичні.

Таблиця А.3 – Матриця ймовірність-втрати

Ймовірність			R1		
		R4			
	R6				
		R7	R5	R2	
				R3	
	Втрати				

Отже, було визначено оддин критичний ризик – пропущення помилок у ході розробки. Що стосується цього ризику, то він може проявлятися не так уже й рідко. Більшість помилок у подальшому будуть знайдені на етапі тестування і продукт буде повернений для їх усунення розробнику.

ДОДАТОК Б

Налаштування локального серверу.

```
var express = require('express')
var cors = require('cors')
var bodyParser = require('body-parser')

var app = express()

var port = process.env.PORT || 3000

app.use(bodyParser.json())
app.use(cors())
app.use(
  bodyParser.urlencoded({
    extended: false
  })
)

var Users = require('./app/routes/User')

app.use('/users', Users)

app.listen(port, function() {
  console.log('Server is running on port: ' + port)
})
```

Підключення до бази даних.

```
const Sequelize = require('sequelize')
const db = {}
const sequelize = new Sequelize('police', 'root', '', {
  host: 'localhost',
  dialect: 'mysql',
  operatorsAliases: false,

  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
})
```

```
db.sequelize = sequelize
db.Sequelize = Sequelize

module.exports = db
```

Налаштування моделі сутності «Користувач»

```
const Sequelize = require('sequelize')
const db = require('../db/db')

module.exports = db.sequelize.define(
  'users',
  {
    user_id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    role: {
      type: Sequelize.STRING
    },
    username: {
      type: Sequelize.STRING
    },
    user_password: {
      type: Sequelize.STRING
    },
    name: {
      type: Sequelize.STRING
    },
    surname: {
      type: Sequelize.STRING
    },
    birthdate: {
      type: Sequelize.DATE
    },
    sex: {
      type: Sequelize.INTEGER
    },
    policeman_rank: {
      type: Sequelize.STRING
    }
  },
  {
    timestamps: false
  }
)
```

Сервіс аутентифікації

```
import { Injectable } from '@angular/core'
import { HttpClient } from '@angular/common/http'
import { Observable, of } from 'rxjs'
import { map } from 'rxjs/operators'
import { Router } from '@angular/router'

export interface UserDetails {
  user_id: number
  role: string
  username: string
  user_password: string
  name: string
  surname: string
  birthdate: string
  sex: number
  policeman_rank: string
  exp: number
  iat: number
}

interface TokenResponse {
  token: string
}

export interface TokenPayload {
  user_id: number
  role: string
  username: string
  user_password: string
}

@Injectable()
export class AuthenticationService {
  private token: string

  constructor(private http: HttpClient, private router: Router) {}

  private saveToken(token: string): void {
    localStorage.setItem('usertoken', token)
    this.token = token
  }

  private getToken(): string {
    if (!this.token) {
      this.token = localStorage.getItem('usertoken')
    }
    return this.token
  }
```

```
public getUserDetails(): UserDetails {
  const token = this.getToken()
  let payload
  if (token) {
    payload = token.split('.')[1]
    payload = window.atob(payload)
    return JSON.parse(payload)
  } else {
    return null
  }
}

public isLoggedIn(): boolean {
  const user = this.getUserDetails()
  if (user) {
    return user.exp > Date.now() / 1000
  } else {
    return false
  }
}

public register(user: TokenPayload): Observable<any> {
  return this.http.post(`/users/register`, user)
}

public login(user: TokenPayload): Observable<any> {
  const base = this.http.post(`/users/login`, user)

  const request = base.pipe(
    map((data: TokenResponse) => {
      if (data.token) {
        this.saveToken(data.token)
      }
      return data
    })
  )

  return request
}

public profile(): Observable<any> {
  return this.http.get(`/users/profile`, {
    headers: { Authorization: ` ${this.getToken()}` }
  })
}

public logout(): void {
  this.token = ''
  window.localStorage.removeItem('usertoken')
```

```
    this.router.navigateByUrl('/')
  }
}
```

Файл розмітки «шапки» сайту.

```
<nav class="site-header py-1">
    <div class="container d-flex flex-column flex-md-row justify-content-
left">
        <a class="py-2 d-none d-md-inline-block" href="/">
            <img class="favicon" src="../../../favicon.ico" width="40" height="40
" alt="favicon">
        </a>
        <a *ngIf="!auth.isLoggedIn()" class="py-2 pt-3 d-none d-md-inline-
block" routerLink="/login">
            Увійти
        </a>
        <a *ngIf="auth.isLoggedIn()" class="py-2 pt-3 d-none d-md-inline-
block" routerLink="/">
            Головна
        </a>
        <a *ngIf="auth.isLoggedIn()" class="py-2 pt-3 d-none d-md-inline-
block" routerLink="" (click)="auth.logout()">
            Вийти
        </a>
    </div>
</nav>
```

Файл класу компонента «шапки» сайту.

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService } from '../../services/authentication.service'

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {

  constructor(public auth: AuthenticationService) { }

  ngOnInit() {
  }

}
```

Файл розмітки головної сторінки сайту.

```
<app-header></app-header>

<div class="container">
    <div class="row">
        <div class="col-12 text-center pt-3">
            <h1>Система швидкого реагування<br>диспетчерської служби патрульної п
оліції</h1>
        </div>
    </div>
    <div class="row pt-3 pb-3">
        <div class="col-2"></div>
        <div class="col-8 text-center">
            <img src="assets/img/patrul_car.png" width="100%" height="100%" alt="
patrul_car">
        </div>
        <div class="col-2"></div>
    </div>
    <div class="row">
        <div class="col-12">
            <p class="text-under-img font-weight-light my-2 p-2 text-
center card card-outline-secondary">
                Система дозволяє відстежувати патрульні автомобілі на мапі та кер
увати їх маршрутами.
                Виклики поліції фіксуються та оброблюються в автоматичному режимі
.<br>
                Пройдіть авторизацію для роботи з системою. Якщо у вас немає дани
х для входу,
                зверніться до адміністратора.<br>
                Дякуємо за розуміння!
            </p>
        </div>
    </div>
    <div *ngIf="!auth.isLoggedIn()" class="row">
        <div class="col-12 py-4">
            <a class="btn btn-outline-secondary btn-
block" href="/login" role="button">Перейти до авторіації</a>
        </div>
    </div>
</div>
```

Файл класу головної сторінки сайту.

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService } from '../../services/authentication.service'

@Component({
  selector: 'app-main',
```

```
  templateUrl: './main.component.html',
  styleUrls: ['./main.component.css']
})
export class MainComponent implements OnInit {

  constructor(public auth: AuthenticationService) { }

  ngOnInit() {
  }

}
```

Файл розмітки сторінки входу.

```
<app-header></app-header>

<div *ngIf="!isDisplayDbErr" #alert class="alert alert-danger alert-
dismissible fade show" role="alert">
    {{ errMsg }}
</div>

<div class="container">
    <div class="row pt-5 pb-3">
        <div class="col-12 text-center">
            <h4 class="">Уведіть дані для входу</h4>
        </div>
    </div>
    <div class="row">
        <div class="col-3"></div>
        <div class="input-container col-6 card py-4 px-6 card">
            <form (submit)="login()">
                <div class="row">
                    <div class="form-group col-12">
                        <label for="login">Логін:</label>
                        <input type="text"
                               [(ngModel)]="credentials.username"
                               name="username"
                               class="form-control"
                               id="inputUserName"
                               placeholder="Уведіть логін"
                               required>
                    </div>
                </div>
                <div class="row pb-3">
                    <div class="form-group col-12">
                        <label for="login">Пароль:</label>
                        <input type="password"
                               [(ngModel)]="credentials.user_password"
```

```
                                name="user_password"
                                class="form-control"
                                id="inputPassword"
                                placeholder="Пароль"
                                required>
                        </div>
                    </div>

                    <button class="btn btn-info btn-
block" type="submit">Увійти</button>
                </form>
            </div>
            <div class="col-3"></div>
        </div>
</div>
```

Файл класу сторінки входу.

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService, TokenPayload } from '../../services/authenticatio
n.service'
import { Router } from '@angular/router'

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  isDisplayDbErr = true;
  errMsg = "";

  credentials: TokenPayload = {
    user_id: 0,
    role: '',
    username: '',
    user_password: ''
  }

  constructor(private auth: AuthenticationService, private router: Router) {}

  login() {
    this.auth.login(this.credentials).subscribe(
      () => {
        this.router.navigateByUrl('/profile')
      },
      err => {
        this.isDisplayDbErr = false;
```

```
        if(this.credentials.username && this.credentials.user_password) {
          this.errMsg = "Неправильний логін або пароль!";
        } else {
          this.errMsg = "Усі поля обов'язкові для заповнення!";
        }
        console.error(err)
      }
    )
  }

  ngOnInit() {
  }

}
```

Файл розмітки сторінки профілю.

```
<app-header></app-header>

<div class="container emp-profile">
    <form method="post">
        <div class="row pb-5">
            <div class="col-md-4 pb-3">
                <div class="profile-img text-center">
                    <img src="../../../assets/img/user_man.png" width="20" height
="20" alt="user img">
                </div>
            </div>
            <div class="col-md-4 text-head">
                <div class="profile-head pb-3">
                    <h4>
                        {{ details?.name }} {{ details?.surname }}
                    </h4>
                    <h5>
                        {{ details?.policeman_rank }}
                    </h5>
                </div>
            </div>
            <div class="col-md-2 text-head">
                <input *ngIf="details?.role=='admin'" type="submit" class="edit-
btn-my profile-edit-btn" name="btnAddMore" value="Редагувати">
            </div>
            <div class="col-md-2 text-head">
                <input *ngIf="details?.role=='admin'" type="submit" class="edit-
btn-my profile-edit-btn" name="btnAddMore" value="Редагувати">
            </div>
        </div>
        <div class="row">
```

```
            <div class="col-md-4">
            </div>
            <div class="col-md-8">
                <div class="row">
                    <div class="col-md-6 text-title">
                        <label>Дата народження</label>
                    </div>
                    <div class="col-md-6">
                        <p>{{ details?.birthdate }}</p>
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-6 text-title">
                        <label>Стать</label>
                    </div>
                    <div class="col-md-6">
                        <p *ngIf="details?.birthdate">
                            <img src="../../../assets/img/male.png" width="15" he
ight="15" alt="male">
                            Чоловіча
                        </p>
                        <p *ngIf="!details?.birthdate">
                            <img src="../../../assets/img/female.png" width="15"
height="15" alt="female">
                            Жіноча
                        </p>
                    </div>
                </div>
            </div>
        </div>
    </form>
```

Файл класу сторінки профіля.

```
import { Component, OnInit } from '@angular/core';
import { AuthenticationService, UserDetails } from '../../services/authentication
.service';

@Component({
  selector: 'app-profile',
  templateUrl: './profile.component.html',
  styleUrls: ['./profile.component.css']
})
export class ProfileComponent implements OnInit {
  details: UserDetails

  constructor(private auth: AuthenticationService) { }

  ngOnInit() {
```

```
    this.auth.profile().subscribe(
      user => {
        this.details = user
      },
      err => {
        console.error(err)
      }
    )
  }

}
```