МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

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

Виконавець роботи: студент групи ІТз-51с Пітерцев Даніїл Вадимович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____ « ___»_____ 2020 р.

Науковий керівник _____ к. т. н, Нагорний В.В.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Голова комісії _____ Шифрін Д. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

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

Студент _____
(підпис)

Суми-2020

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

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

____ В. В. Шендрик
«__» _____2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Пітерцев Даніїл Вадимович

1 Тема роботи *Інформаційна система фіксації порушень благоустрою міста Конотоп*

керівник роботи *Нагорний Володимир В'ячеславович, к.т.н.*

затверджені наказом по університету від « 15» травня 2020 р. № 0582-III

2 Строк подання студентом роботи «6» червня 2020 р**.**

3 Вхідні дані до роботи Зображення проблеми, інформація про користувача та інформація про місце.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області, Моделювання та проектування інформаційної системи фіксування порушень благоустрою міста Конотоп, Розробка інформаційної системи, Висновки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *презентація 20 слайдів*: тема дипломного проекту, актуальність, постановка задачі, аналіз програмних продуктів – аналогів, порівняння сайтів-аналогів, функціональні вимоги до інформаційної системи, зовнішня структура інформаційної системи, моделювання роботи інформаційної системи, діаграма варіантів використання, архітектура порталу, засоби реалізації, демонстрація інформаційної системи, висновки.

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі	До 15.04.2020	
2	Моделювання системи	До 16.04.2020	
3	Створення сайту	До 04.05.2020	
4	Тестування сайту	До 05.05.2020	
5	Створення мобільного додатку	До 25.05.2020	
6	Здача пояснювальної записки	До 01.06.2020	

Студент _____ Пітерцев Д.В.
(підпис)

Керівник роботи _____ к. т. н, Нагорний В.В.
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра: «Інформаційна система фіксації порушень благоустрою міста Конотоп».

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел, додатків. Кваліфікаційна робота містить 67 сторінки, 4 таблиці, 35 рисунків, список літератури 20 найменувань, 3 додатки.

В роботі досліджено та визначено проблеми предметної області майбутнього проекту та методи для розробки інформаційної системи. Проведено аналіз існуючих програмних продуктів, які схожі за темою і метою.

Було сформульовано перелік вимог, постановку задачі та мету, яка полягає у створенні інформаційної системи фіксації порушень благоустрою міста Конотоп.

Було проаналізовано та обрано існуючі моделі, методи, технології, засоби для реалізації поставленої задачі, обґрунтована актуальність розробки проекту.

Результатом проведеної роботи є інформаційна система у вигляді веб-сайту та мобільного додатку.

Практичне значення роботи – фіксування порушення благоустрою міста Конотоп та повідомлення відповідних комунальних служб.

Ключові слова: інформаційна система, м. Конотоп, благоустрій міста, Laravel, mvc, php , android studio

ЗМІСТ

ВСТУП

Все більше адміністративних процесів переходять від паперової тяганини до електронної обробки. За останні два роки Україна зробила в сфері відкритих даних більше, ніж за останнє десятиліття [1].

Актуальність теми зумовлена розвитком інформаційних систем у адміністративному просторі. Перевага електронних документів замість паперових.

Об’єктом дослідження є процес фіксування порушень благоустрою міста Конотоп.

Предметом дослідження є інформаційна система фіксування порушень благоустрою міста Конотоп.

Метою кваліфікаційної роботи бакалавра є розроблення інформаційної системи фіксації порушень благоустрою міста Конотоп.

Для досягнення мети кваліфікаційної роботи потрібно вирішити такі задачі:

- провести аналіз аналогів інформаційних систем фіксування порушень благоустрою;
- сформулювати функціональні вимоги до інформаційної системи;
- провести моделювання інформаційної системи;
- реалізувати інформаційну систему у вигляді веб-сайту та мобільного додатку;
- провести тестування інформаційної системи;

Причиною створення системи є потреба в інформуванні людей та відповідних комунальних служби про проблеми рідного міста, що і визначає актуальність даної роботи.

Результатом проекту буде інформаційна система, що надаватиме інформацію про проблемні місця в місті, та буде відправляти запити з проханням їх вирішити у компетентні органи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження сучасного стану інформаційних систем – аналогів

Розробку інформаційної системи потрібно починати з аналізу аналогів, та дослідження «переваг» та «недоліків» систем.

Для дослідження аналогів були використані такі інформаційні системи: «Сервіси всеросійської мультисервісної платформи громадянської активності igrajdanin.ru», «Карта проблем Челябінська», «CitiPower & Powercor», «We Energies Outage Map».

1.1.1 Сервіси всеросійської мультисервісної платформи громадянської активності igrajdanin.ru

«Сервисы всероссийской мультисервисной платформы гражданской активности igrajdanin.ru» (рис. 1.1) – це незалежний і безкоштовний інструмент для поліпшення міського середовища та сільських територій. За допомогою платформи жителі всіх населених пунктів Росії можуть в зручній для себе формі контактувати з міськими службами, департаментами муніципалітетів і наглядовими органами, повідомляючи владі хвилюючі їх проблеми та вимагаючи їх вирішення.

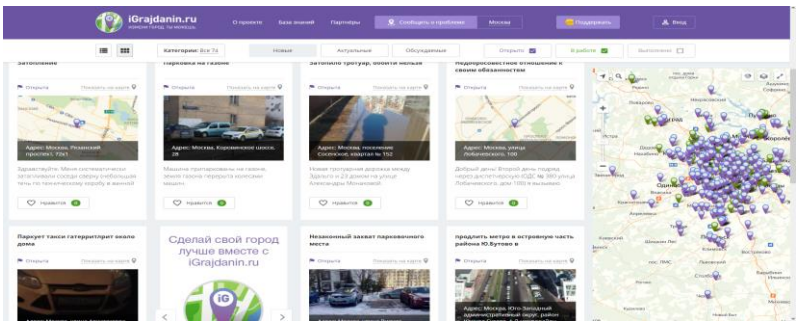

Рисунок 1.1 – Головна сторінка

Можливості: перегляд карток на головній сторінці з поточними проблемами міста, реєстрація для доступу до основного функціоналу (рис. 1.2), наявність фільтру для сортування інформації за категорією (рис. 1.3), система лайків, сортування за актуальністю.

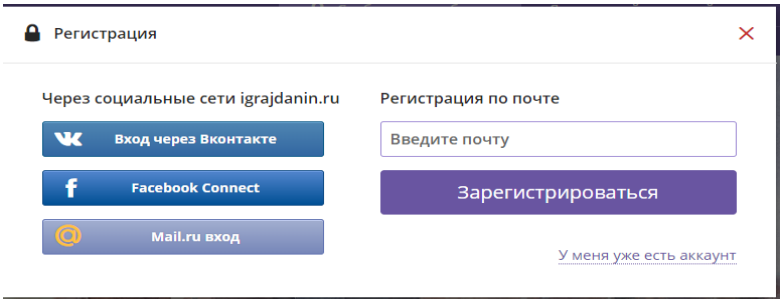

Рисунок 1.2 – Вікно реєстрації

Сайт використовує технологію Ajax для підвантаження даних, це технологія для взаємодії з сервером без перезавантаження сторінок [2].

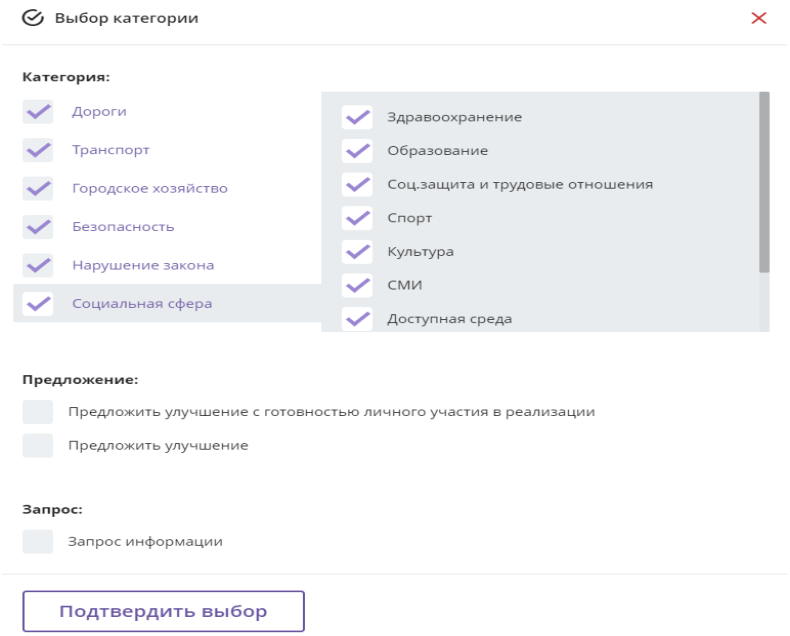

Рисунок 1.3 – Вікно вибору категорії

Інформаційна система має сторінку зі списком контактів усіх служб (рис 1.4).

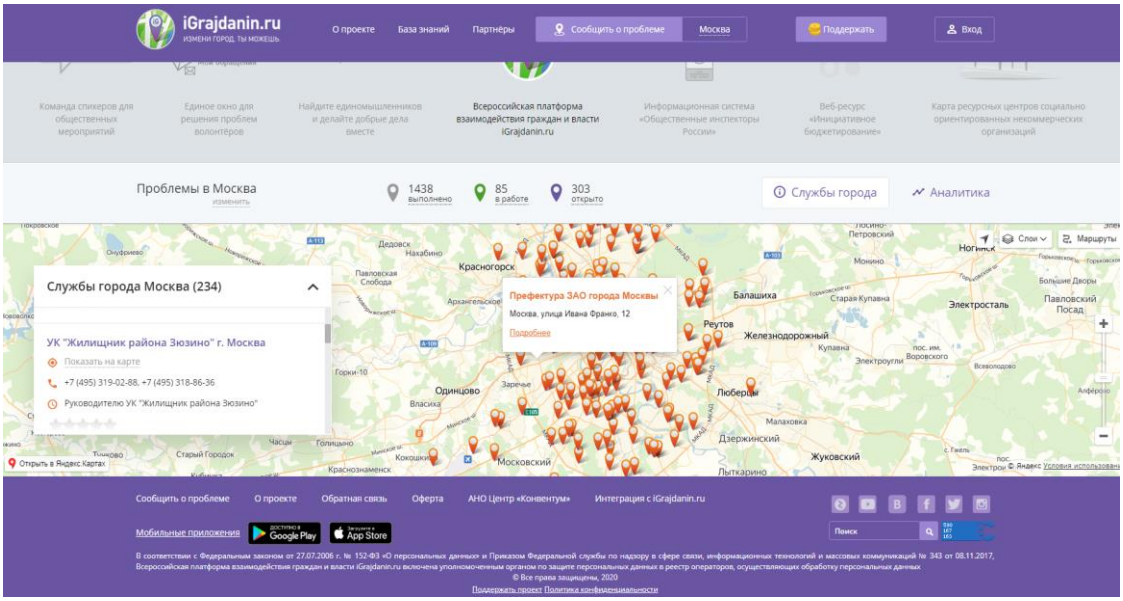

Рисунок 1.4 – Сторінка перегляду служб міста

1.1.2 «CitiPower & Powercor»

CitiPower & Powercor – цей сервіс відображає поточні проблеми з освітленням вулиць (рис. 1.5). Має простий діловий дизайн. Із недоліків потрібно виділити те, що не має можливості додати інформацію про проблеми надання послуг електроспоживання.

Можливості: різні типи міток на карті, заплановане відключення та незаплановане (рис. 1.6), перевірка статусу освітлення за поштовим індексом (рис. 1.7).

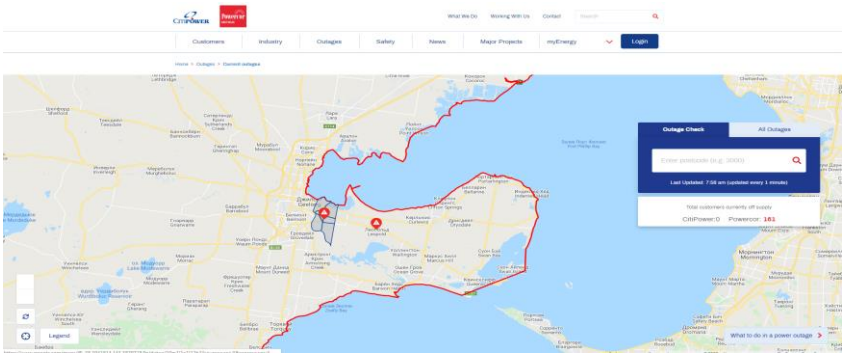

Рисунок 1.5 – Карта CitiPower & Powercor

До карти йде модальне вікно з інформацією про мітки на карті (рис. 1.6).

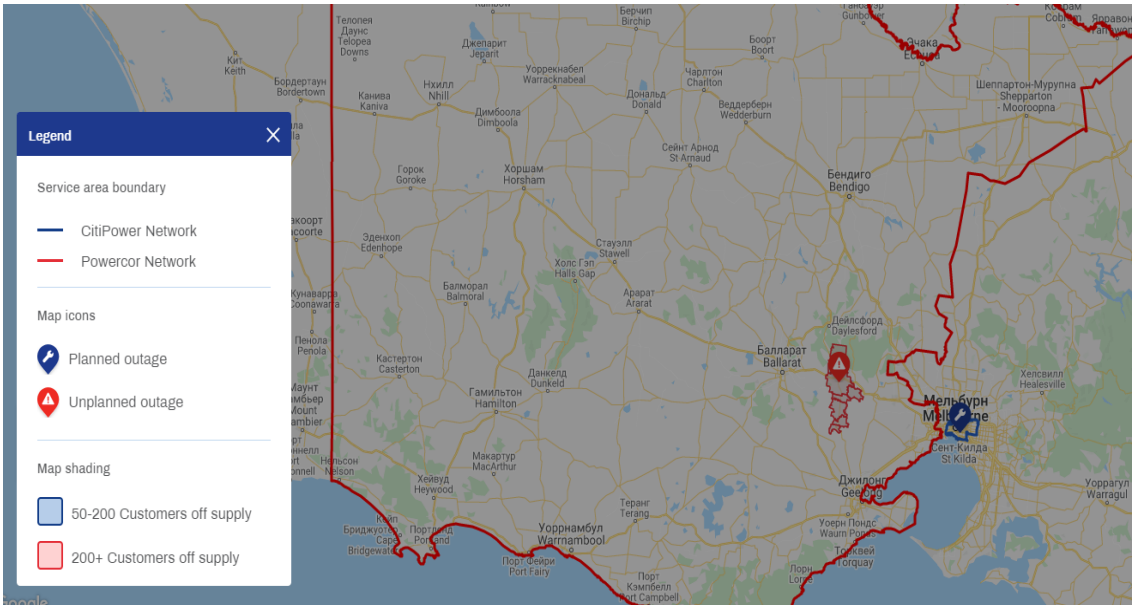

Рисунок 1.6 – Легенда до карти

Для відображення мапи проект використовує бібліотеку OpenStreetMap.

OpenStreetMap – проект, метою якого є створення безкоштовної географічної бази світу. Її метою є врешті-решт мати записи про кожен окремий географічний об’єкт на планети [3].

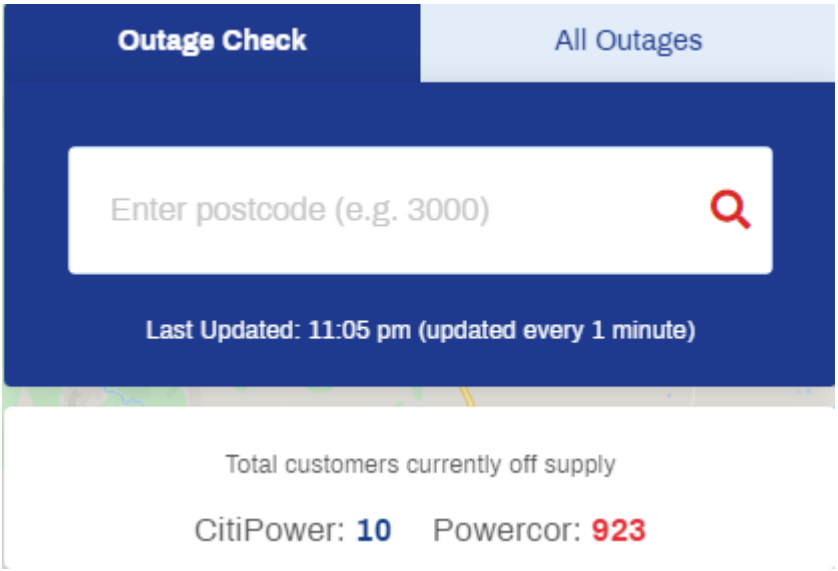

Рисунок 1.7 – Вікно перевірки за поштовим індексом

1.1.3 «We Energies Outage Map»

We Energies Outage Map – цей сервіс відображає останні проблеми надання послуг електроспоживання та займається їх ремонтом (рис. 1.8).

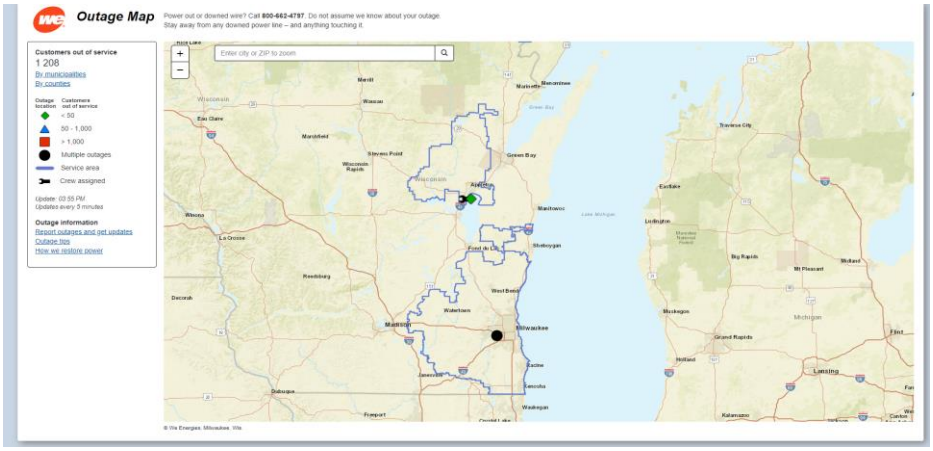

Рисунок 1.8 – Карта We Energies Outage Map

Можливості: має декілька міток які залежать від кількості людей які мають проблеми з електроспоживанням, якщо призначено екіпаж то з'являється друга мітка у виді гайкового ключа, є пошук інформації про несправності за назвою міста та почтовим індексом, є фільтр інформації за муніципалітетами та кількістю несправностей.

Приклади роботи інформаційної системи представлений на рисунках 1.9–1.10.

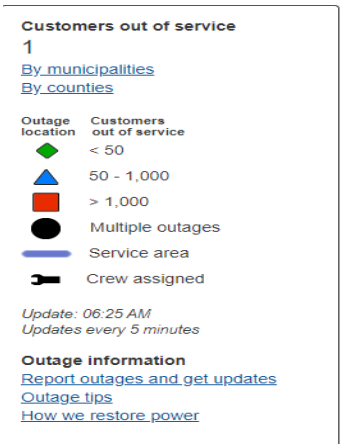

Рисунок 1.9 – Легенда до карти

Інформаційна система не потребує реєстрації, але щоб додати інформацію потрібно вказати поштовий індекс та телефон або номер особистого рахунку (рис. 1.10).

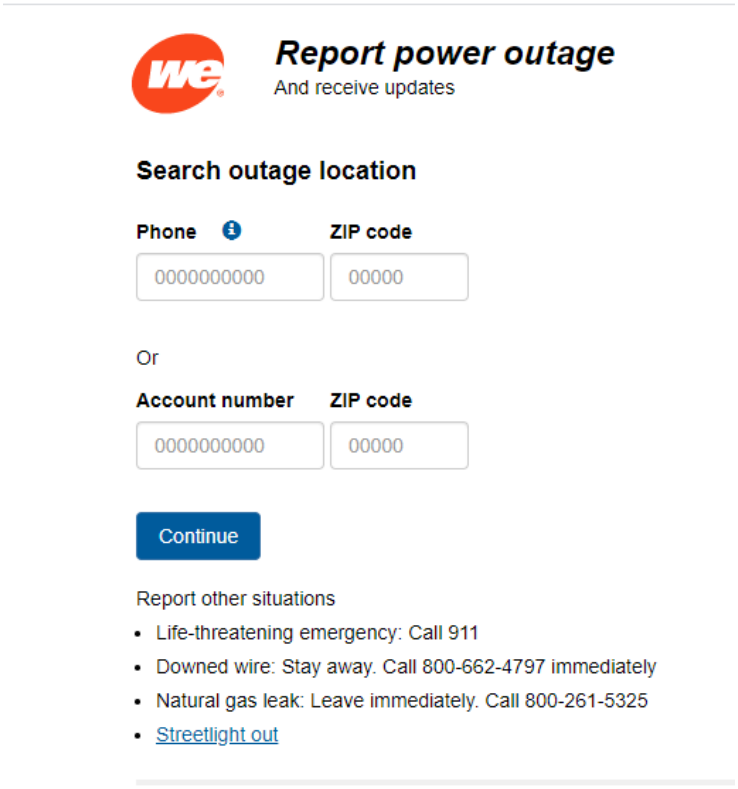

Рисунок 1.10 – Сторінка для відправки інформації про проблему

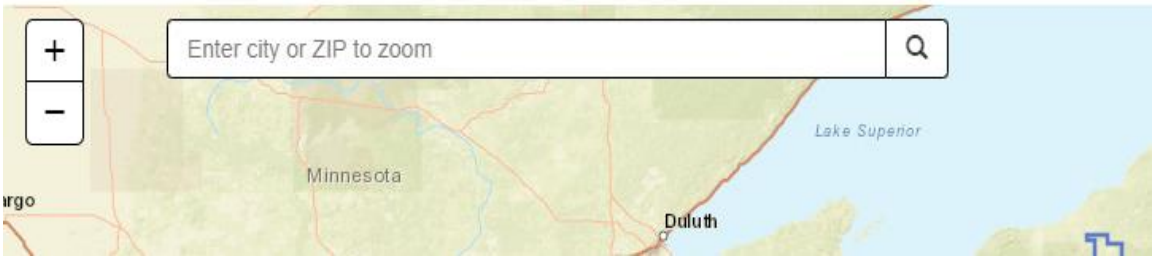

Рисунок 1.11 – Поле для пошуку

1.1.4 Карта проблем Челябінська

«Карта проблем Челябінська» надає інформацію на проблемні місця та віддає цю інформацію адміністрації міста (рис. 1.12).

Можливості: має функціонал фільтрування міток за категорією порушення, на кожній категорії є блок з інформаційними фактами на карті, є сторінка з рейтингом доглянутості районів міста.

Приклади роботи інформаційної системи представлено на рисунках 1.12–1.17.

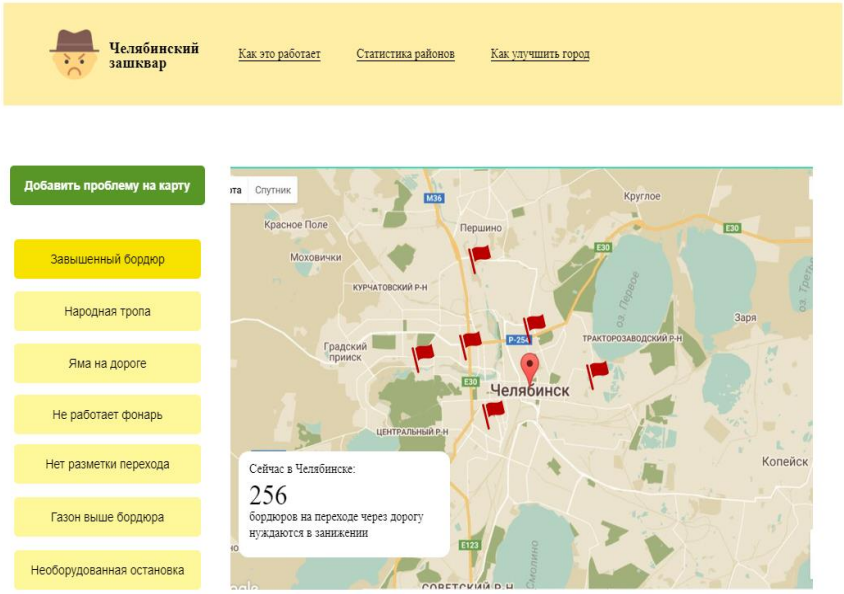

Рисунок 1.12 – Карта проблем Челябінська

Рисунок 1.13 – Фільтр до карти

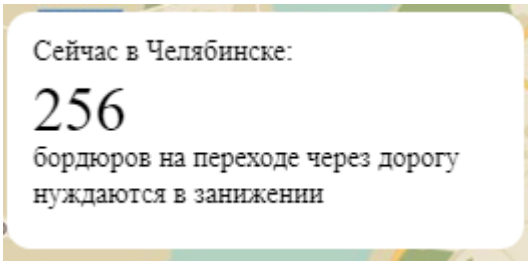

Рисунок 1.14 – Вікно з цікавими фактами

Інформаційна система не вимагає реєстрації, щоб зафіксувати порушення (рис 1.15), що є небезпечно, така інформація може бути не достовірною. Після відправки порушення система інформує користувача, що порушення буде передане в відповідні комунальні служби (рис 1.16).

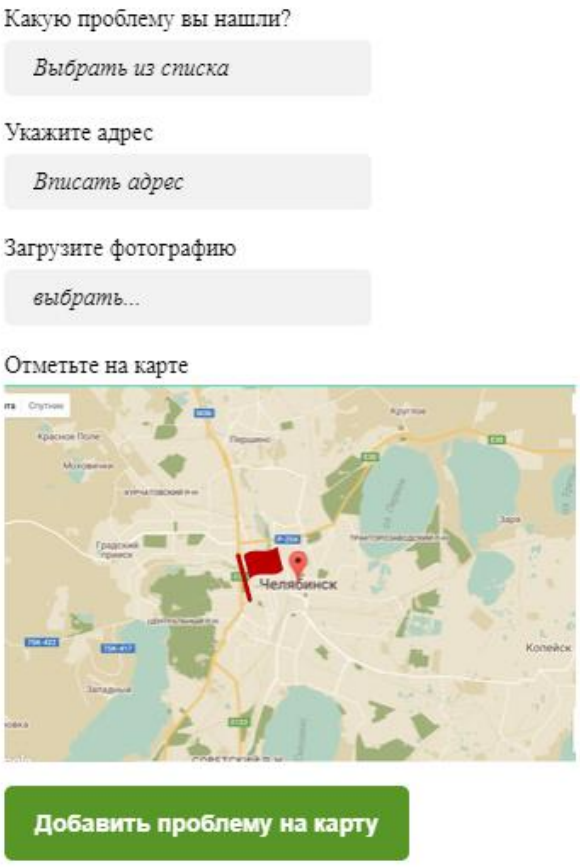

Рисунок 1.15 – Сторінка відправки порушення

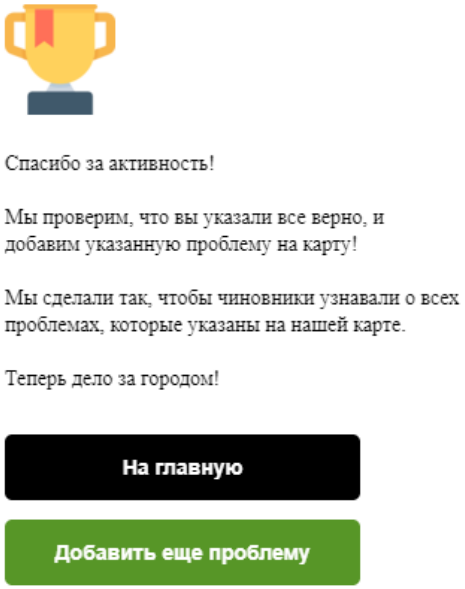

Рисунок 1.16 – Сторінка подяки

Також система має рейтинг найбільш благоустроєних районів міста (рис 1.17).

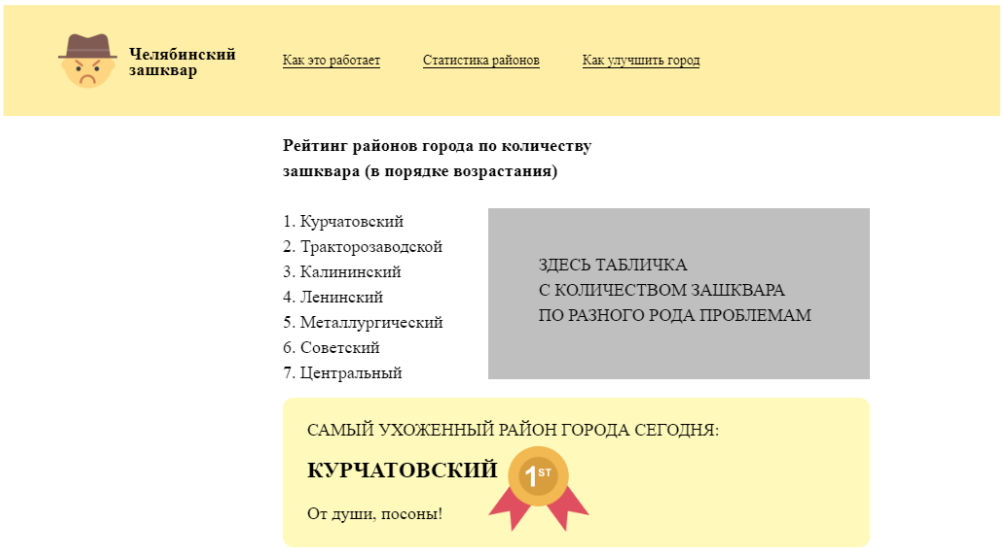

Рисунок 1.17 – Сторінка статистики районів міста

Проаналізувавши схожі інформаційні системи в інтернеті, було проведено порівняльний аналіз (табл. 1.1), де було виявлено переваги та недоліки, які при створені дипломного проекту треба враховувати.

Основні види показників, при яких можлива оцінка якості інформаційного середовища, критерії ІС – це безпека, достовірність і надійність [4].

Таблиця 1.1 – Аналіз розглянутих інформаційних систем

Критерії	igrajdanin.ru	CitiPower & Powerco	We Energies Outage Map	Карта проблем Челябинска	4house-office
Безпека	+	+	+	-	+
Достовірність	+	+	+	-	+
Надійність	+	+	+	-	+
Фільтр	+	-	+	+	+
Інтерфейс	+	+	-	+	+
Пошук	+	-	+	-	+
Легенда	-	+	+	-	+
Пошук за районом	+	+	+	-	-
Авторизація	+	-	-	-	+

1.2 Аналіз програмних продуктів для вирішення поставленої задачі

Для вирішення поставлених задач використовувалися методи проектування ПЗ, методи оптимізації та покращення css стилів та javascript коду.

1. При дослідженні предметної області використовувався метод аналізу.

Аналіз – метод дослідження, що полягає в уявному або практичному розчленуванні цілого на складові частини, кожна з яких аналізується окремо у межах єдиного цілого [5].

2. При дослідженні предметної області використовувався метод аналогії.

Аналогія - метод наукового дослідження; завдяки якому досягається пізнання одних предметів і явищ на основі їх подібності з іншими [5].

3. При моделюванні використовувався метод моделювання.

Моделювання – метод наукового пізнання, що ґрунтується на заміні предмета або явища, що досліджуються, на їх аналог – модель, що містить істотні риси оригіналу [5].

Для реалізації веб-сайту було вирішено використовувати за основу фреймворк Laravel.

Це найпопулярніший на даний момент PHP-фреймворк, що надає величезну кількість інструментальних засобів на всі випадки життя і по-справжньому спрощує розробку сайтів [6].

Однією з головних переваг Laravel фреймворк вважається наявність досить-таки розгорнутої і зрозумілою документації. Навколо цього фреймворка реалізована потужна екосистема. В інтернеті можна без проблем знайти навчальні матеріали. Також регулярно проводяться різні конференції та курси, що збирає величезну кількість веб-розробників, зацікавлених у подальшому розвитку цього фреймворка. З огляду на ці факти, можна не сумніватися в тому, що Laravel очікує непогане комерційне майбутнє.

Для бази даних було обрано MySQL. Архітектура MySQL дуже відмінна від архітектур інших серверів баз даних, що робить цю СУБД корисною для одних цілей, але одночасно невдалим вибором для інших. MySQL не ідеальна, але досить гнучка для того, щоб добре працювати в дуже вимогливих середовищах, наприклад, у веб-додатках [7].

Для пошуку інформації вирішено використовувати пошукову систему Sphinxsearch. Під час обробки запиту користувача Sphinx використовує повнотекстовий індекс для швидкого перегляду кожне повнотекстове співвідношення, то є документ, відповідний вказаним ключовим словам [8].

Для створення мобільного додатку було вирішено використовувати середовище Android Studio. Середовище Android Studio, анонсована в 2013 році на конференції розробників Google, в даний час вважається основним середовищем розробки Android додатків [9].

1.3 Постановка задачі проекту

Метою кваліфікаційної роботи бакалавра є розроблення інформаційної системи фіксації порушень благоустрою міста Конотоп.

Для досягнення поставленої мети необхідно вирішити такі задачі:

Проаналізувати аналогічні інформаційні системи фіксування порушень благоустрою.

Сформулювати функціональні вимоги до інформаційної системи.

Для кращого розуміння бізнес-процесів, що протікають в інформаційній системі змоделювати бізнес процеси інформаційної системи.

Використовуючи фреймовок Laravel реалізувати веб-сайт, а для розробки мобільного додатку використовувати Android studio.

Зробити тестування інформаційної системи, це дозволить знайти помилки та критичні місця.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ФІКСУВАННЯ ПОРУШЕНЬ БЛАГОУСТРОЮ МІСТА КОНОТОП

2.1 Структурно-функціональне моделювання бізнес-процесів інформаційної системи

Розглянемо процес фіксування прошення. Було використано модель робіт AS-IS (Як є), для відображення структури бізнес-процесу. Дана модель використовується у розробці програми як стартова точка.

Контекстна діаграма – це початкова діаграма моделі, що характеризує функції системи взагалі (без деталізації) і зв’язки системи з навколишнім середовищем.

Методологія функціонального моделювання IDEF0 – це технологія опису системи в цілому як безлічі взаємозалежних дій, або функцій. Важливо відзначити функціональну направленість IDEF0 – функції системи досліджуються незалежно від об'єктів, які забезпечують їх виконання [10].

Кожна з чотирьох сторін функціонального блоку має своє певне значення (роль). Стрілка, що входить у відповідну сторону має ту ж назву, верхня сторона має значення «Управління» (Control), ліва сторона має значення «Вхід» (Input), права сторона має значення «Вихід» (Output), нижня сторона має значення «Механізм» (Mechanism) [11].

Контекстну діаграму процесу фіксування порушення, в нотації IDEF0, представлено на рисунку. 2.1.

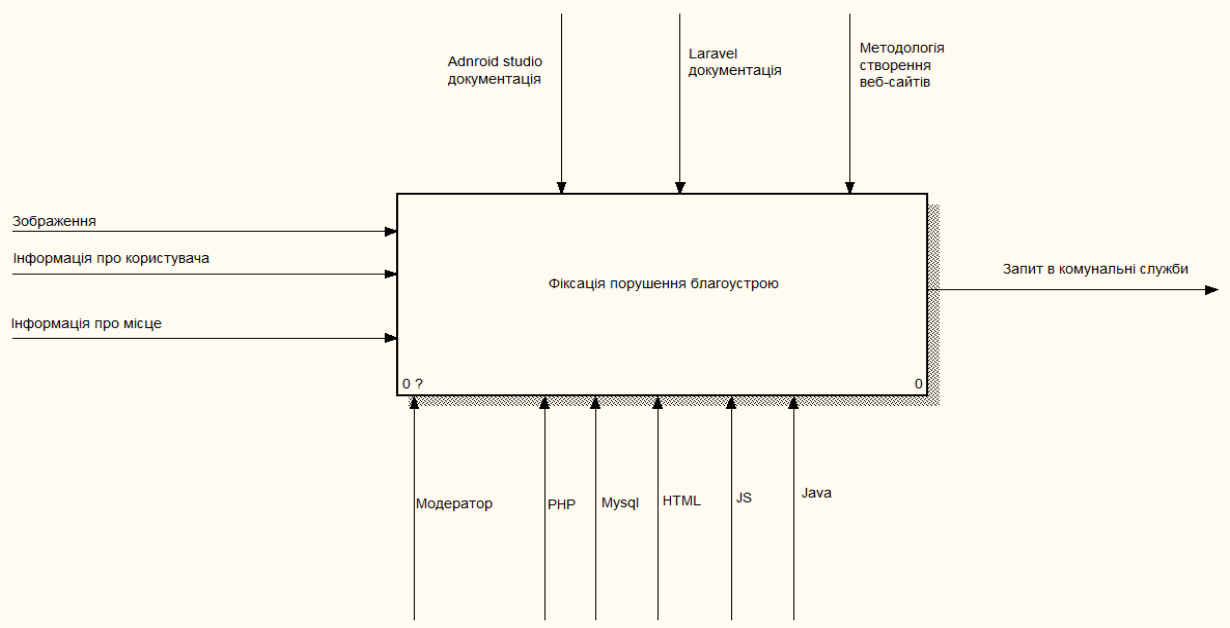

Рисунок 2.1 – Контекстна діаграма

Вхідними даними моделі є зображення проблеми, інформація про користувача та інформація про місце. Вимоги формують android studio документація, Laravel документація, та методологія створення веб-сайтів Механізми, що реалізують проект є Модератор, PHP, Mysql, HTML, JS, та Java. Результатом роботи є запит в комунальні служби.

Декомпозиція діаграми «Фіксація порушення благоустрою» зображена на рисунку 2.2, вхідними даними якої є зображення проблеми, інформація про користувача та інформація про місце. Управляючою дією є android studio документація, Laravel документація, та методологія створення веб-сайтів. За допомогою Модератора, PHP, Mysql, HTML, JS, та Java буде реалізований проект.

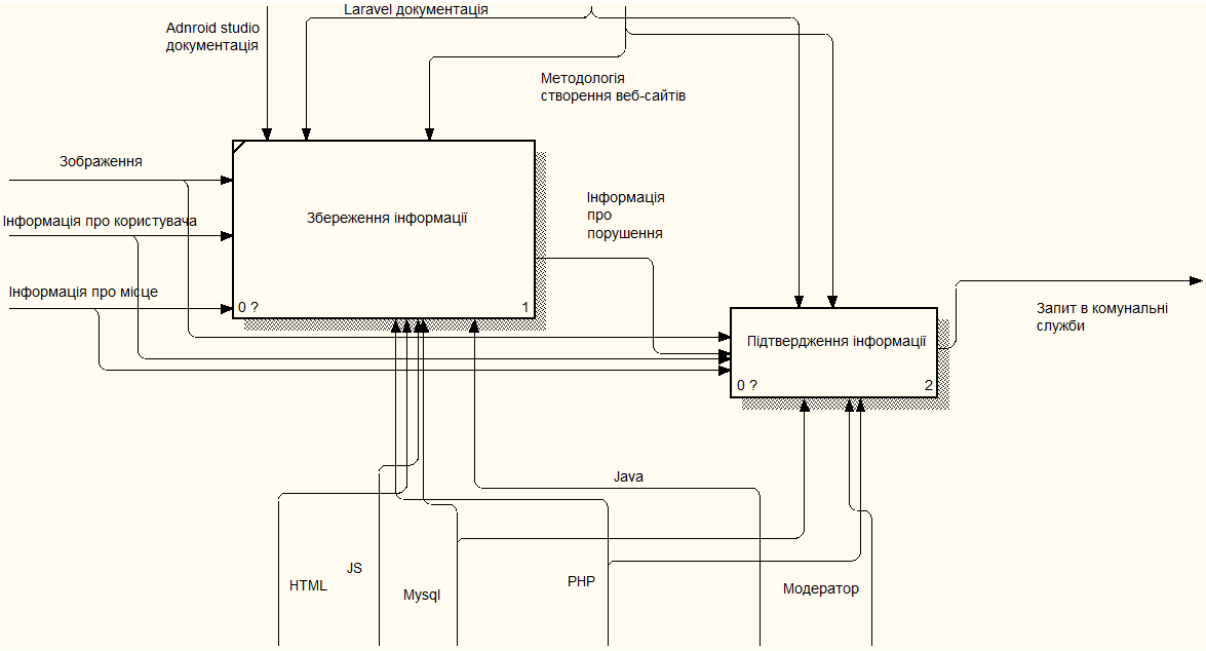

Рисунок 2.2 – Діаграма декомпозиції блоку «Фіксація порушення благоустрою» (в нотації IDEF0)

Декомпозиція діаграми «Підтвердження інформації» зображена на рисунку 2.3, вхідними даними якої є зображення проблеми, інформація про користувача та інформація про місце. Далі заява встає в чергу на відправку, після цього по заявам генерується почтове повідомлення.

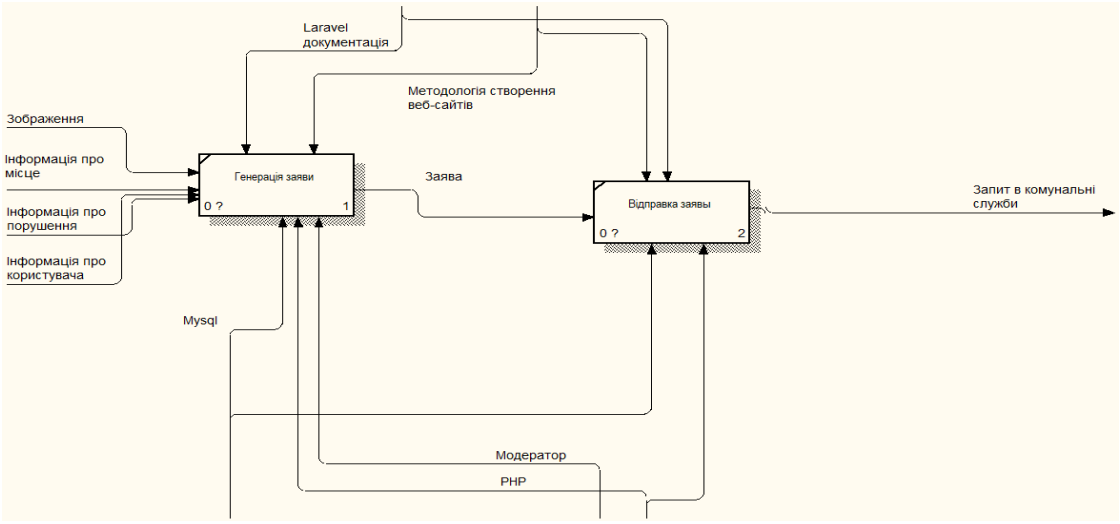

Рисунок 2.3 – Діаграма декомпозиції блоку «Підтвердження інформації» (в нотації IDEF0)

2.2 Моделювання варіантів використання інформаційної системи

UML – це, перш за все, опис об'єкта або явища, а також і дещо інше, а саме все, що авторам UML вдалося включити в мову, не порушуючи принципу уніфікації [12].

Для опису об'єкта використовують так званих акторів.

Таким чином було виділено три актори, які взаємодіють з інформаційною системою:

Користувач – користувач інформаційної системи, має можливість переглядати сторінки.

Модератори – адміністратори інформаційної системи, мають можливість редагувати всі дані.

Компанія – організація яка отримує інформацію про порушення благоустрою.

Також були виділені такі варіанти використання:

− ВВ 1 Авторизація – ВВ надає доступ до додавання координат з порушенням

− ВВ 2 Авторизація (Модератор) – ВВ надає доступ до редагування даних

− ВВ 3 Авторизація (Модератор) – ВВ надає доступ до додавання координат з порушенням

− ВВ 4 Перегляд сторінок – ВВ надає доступ до перегляду сторінок

− ВВ 5 Перегляд сторінок (Компанія) – ВВ надає доступ до перегляду сторінок

Приклад діаграми варіантів використання надано на рисунку 2.4.

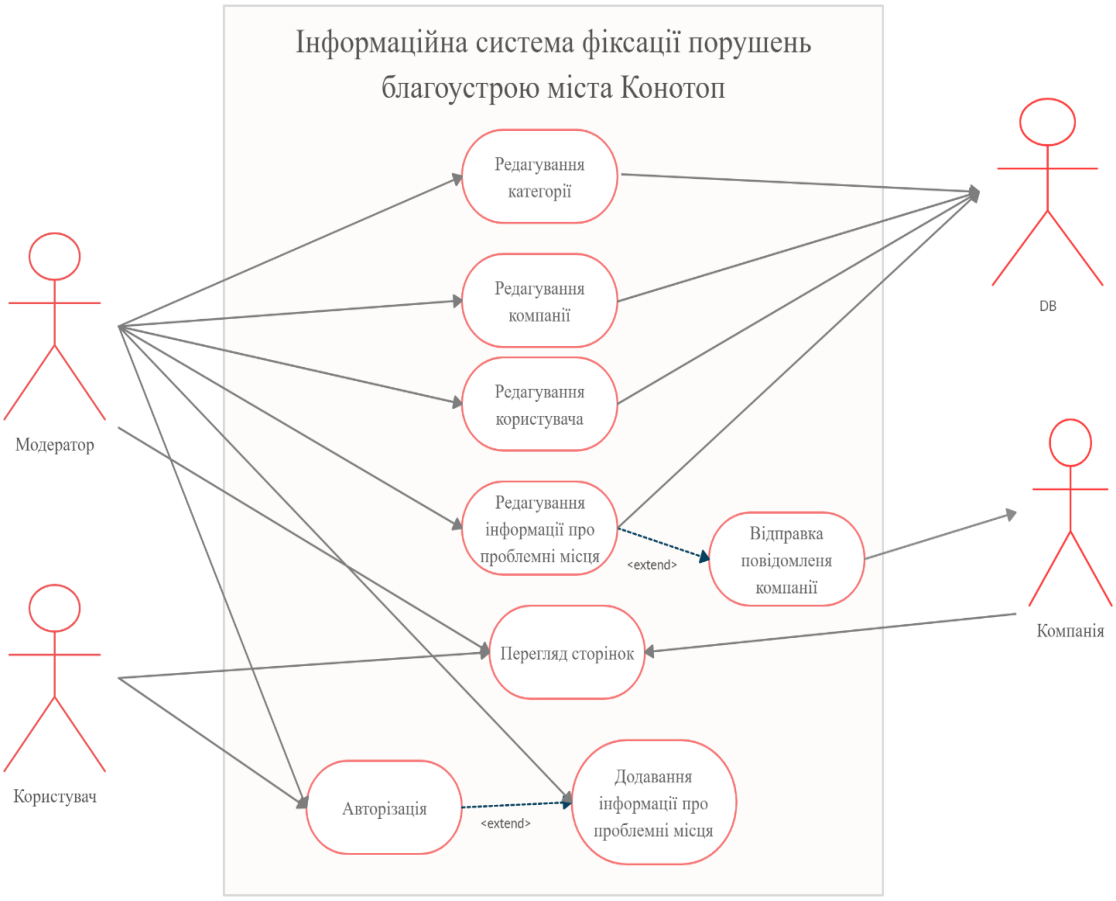

Рисунок 2.4 – Діаграма варіантів використання

2.3 Проектування моделі бази даних

Для проектування моделі бази даних, з яким має працювати інформаційна система, потрібно сформувати інформаційні сутності, атрибути та відношення між ними. Для відображення структури даних та відношення між ними буде використана ER-модель.

Основною концепцією ER-моделі є тип сутності (entity type), який представляє групу об'єктів реального світу, що володіють однаковими властивостями [13].

Основними сутностями бази даних можна виділити:

– інформація про користувача;

– інформація про токени користувача;

– інформація про міграції;

− інформація про запит на зміну паролю;

− інформація про черги;

− інформація про провалені черги;

− інформація про категорії;

− інформація про компанії;

− інформація про координати.

Кожна із зазначених сутностей має свій набір атрибутів, які наведені у таблиці 2.1.

Таблиця 2.1 – Таблиця опису сутностей та їх атрибутів

Назва таблиці	Назва сутності	Назва поля	Опис атрибуту
1	2	3	4
users	Інформація про користувача	id	id користувача
		name	ім'я користувача
		role	роль користувача
		email	email користувача
		email_verified_at	дата перевірки email
		password	пароль користувача
		remember_token	токен користувача
		created_at	дата створення
		updated_at	дата оновлення
users_tokens	Інформація про токени користувача	id	id токена

Продовження таблиці 2.1

1	2	3	4
		user_id	id користувача
		token	токен користувача
		created_at	дата створення
		updated_at	дата оновлення
migrations	Інформація про міграції	id	id міграції
		migration	інформація про міграцію
		batch	статус міграції
password_resets	Інформація про запит на зміну паролю	email	email користувача
		token	токен на зміну email користувача
		created	дата створення
jobs	Інформація про черги	id	id черги
		queue	інформація про чергу
		payload	Статус
		attempts	спроби виконати чергу
		reserved_at	дата запуску черги

Продовження таблиці 2.1

1	2	3	4
		available_at	дата коли буде виконана команда в черзі
		created_at	дата створення черги
failed_jobs	Інформація про провалені черги	id	id проваленої черги
		connection	з'єднання
		queue	інформація про чергу
		payload	статус
		exception	інформація про помилку
		failed_at	дата помилки
categories	Інформація про категорії	id	id категорії
		name	ім'я категорії
		status	статус категорії
		created_at	дата створення
		updated_at	дата оновлення
companies	Інформація про компанії	id	id компанії
		name	ім'я компанії
		description	опис компанії
		contacts	контакти компанії
		image	зображення компанії

Продовження таблиці 2.1

1	2	3	4
		status	статус компанії
		created_at	дата створення
		updated_at	дата оновлення
points	Інформація про координати	id	id координати
		category_id	Категорія Координати
		user_id	власник координати
		company_id	відповідальний за координату
		title	назва координати
		lat	lat-координата
		lng	lng-координата
		comment	коментарій к координаті
		address	адреса координати
		status	статус координати
		image	зображення координати
		created_at	дата створення
		updated_at	дата оновлення

На основі виділених сутностей та їх атрибутів можна створити ER-діаграму.

ER-діаграму інформаційної системи можна побачити на рисунку 2.5.

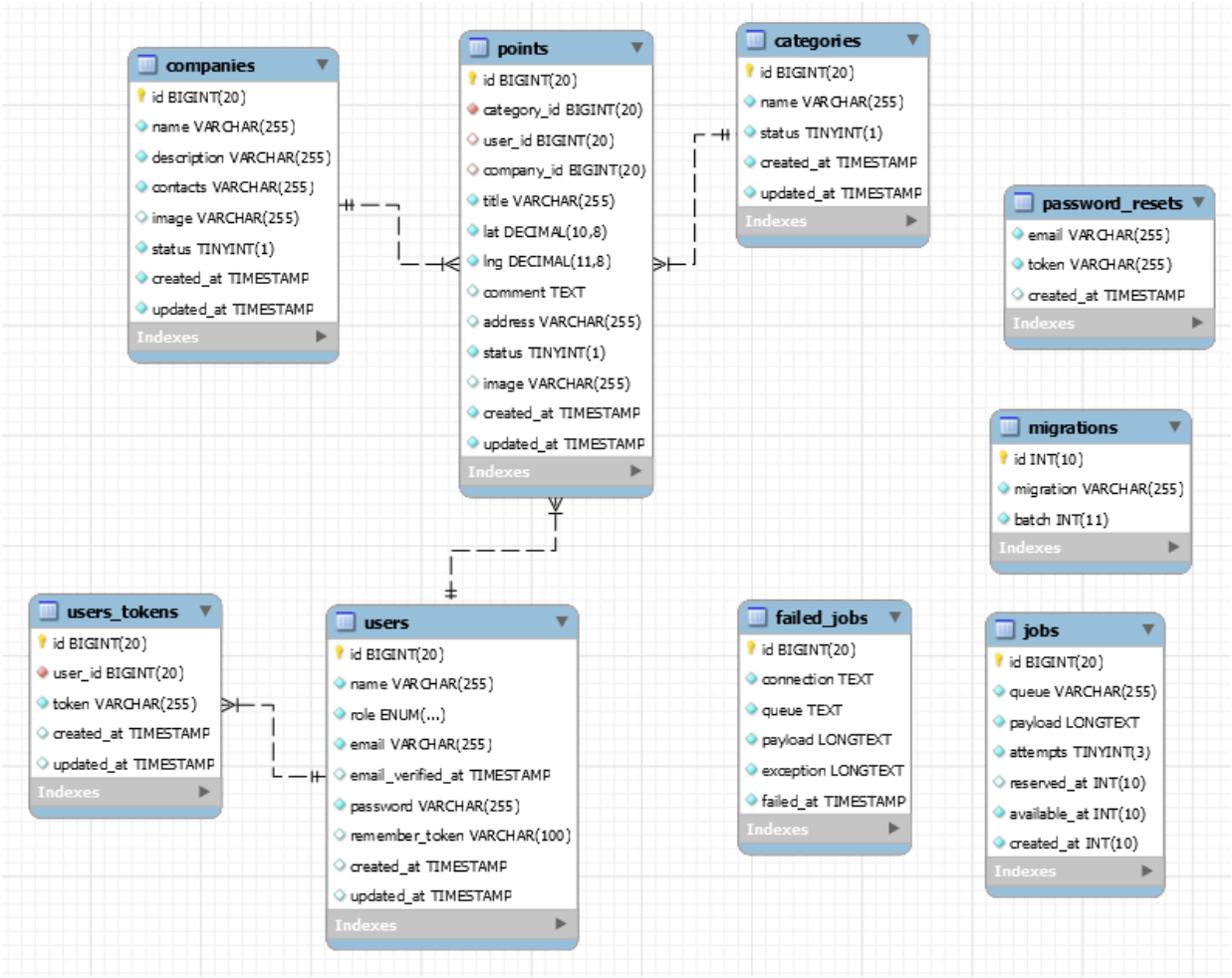

Рисунок 2.5 – Діаграма бази даних типу «сутність-зв’язок»

На даній діаграмі зображено зв’язки між таблицями.

Зв'язок – це асоціація. встановлена між кількома сутностями, і показує як взаємодіють сутності між собою [14].

Так, наприклад, між таблицею category є зв’язок з таблицею points типу оди до багатьох. Між таблицею companies є зв’язок з таблицею points типу оди до багатьох. Між таблицею users є зв’язок з таблицею points та users_tokens типу один до багатьох.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Архітектура програмного додатку

Модель, вид і контролер (MVC) – це добре відома триярусна архітектура розробки, використовувана для розробки веб-додатків.

MVC є архітектурним паттерном, що зазвичай використовується в веб-додатках. Він передбачає три основні шари; модель, вид та контролер. Багато розробників використовують MVC як стандартний шаблон дизайну [15].

Model – це клас, що працює з базою даних, отримує та оновлює дані в базі даних

View – це шаблон, що готує интерфейс програми, яким буде користуватися користувач.

Controller – це класс, що приймає запити від користувачів, обробляє їх, та передає дані до шаблона.

Діаграма MVC в фреймворку Laravel надана на рисунку 3.1

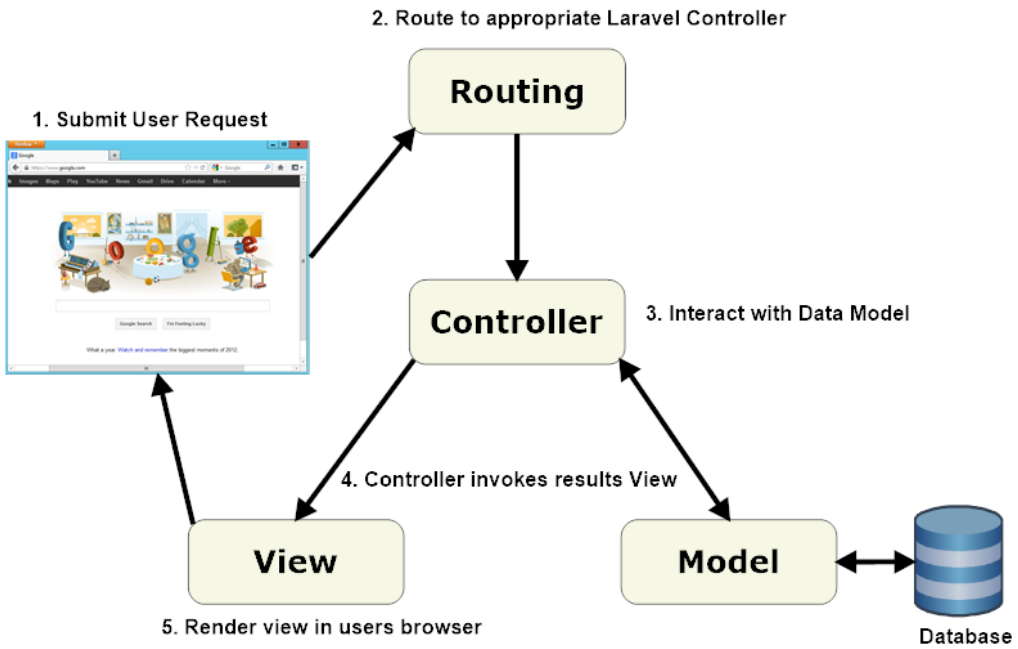

Рисунок 3.1 – Діаграма MVC у Laravel

Далі розробимо список класів та їх атрибутів. Перелік класів та їх атрибутів наведено у таблиці 3.1

Таблиця 3.1 – Опис класів та атрибутів

Назва класу	Опис класу	Назва атрибуту	Опис атрибуту
1	2	3	4
AccountController	Клас акаунту	points	Масив міток
		credentials	Перевірка на авторизацію
AdminCategoryController	Адмінскій клас категорії	categories	Масив категорій
		category	Інформація про категорію
		action	Урл
		method	Тип запиту
		to_update	Масив даних для оновлення
		to_create	Масив даних для створення
		request	Інформація про запит
AdminCompaniesController	Адмінскій клас компанії	companies	Масив компаній
		company	Інформація про компанію
		action	Урл
		method	Тип запиту
		image	Зображення
		to_update	Масив даних для оновлення

Продовження таблиці 3.1

1	2	3	4
		to_create	Масив даних для створення
AdminPointsController	Адмінскій клас міток	points	Масив міток
		point	Інформація про мітку
		request	Інформація про запит
ApiCategoryController	Api клас категорії	id	Id категорії
		request	Інформація про запит
		to_update	Масив даних для оновлення
ApiCompanyController	Api клас компанії	id	Id компанії
		request	Інформація про запит
		to_update	Масив даних для оновлення
ApiPointController	Api клас мітки	id	Id мітки
		request	Інформація про запит
		to_update	Масив даних для оновлення
		image	Зображення

Продовження таблиці 3.1

1	2	3	4
ApiUserController	Api клас користувача	request	Інформація про запит
		credentials	Перевірка на авторизацію
		token	Токен користувача
		userInfoJson	Інформація про користувача
CompaniesController	Клас компаній	companies	Масив компаній
HomeController	Клас головної сторінки	points	Масив міток
		points_card	Масив міток
		categories	Масив категорій
		companies	Масив компаній
PointsController	Клас міток	id	Id мітки
		point	Інформація про мітку
		search_value	Пошукове слово
		category_id	Id категорії
		company_id	Id компанії
		ids	Масив індикаторів міток

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відображати, зокрема, різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їх внутрішню структуру і типи відносин. На даній діаграмі не вказується інформація про тимчасові аспекти функціонування системи. З цієї точки зору діаграма класів може служити подальшим розвитком концептуальної моделі проектованої системи [16].

Діаграми класів інформаційної системи можна побачити на рисунках 3.2–3.4

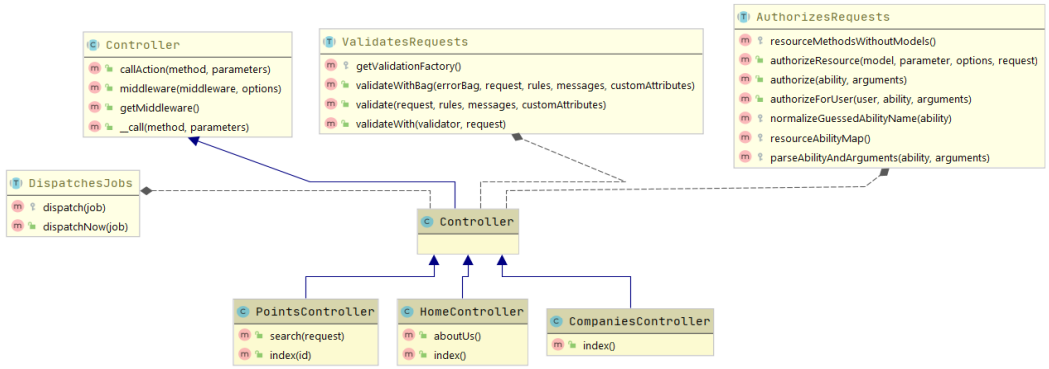

Рисунок 3.2 – Діаграма класів клієнтської частини

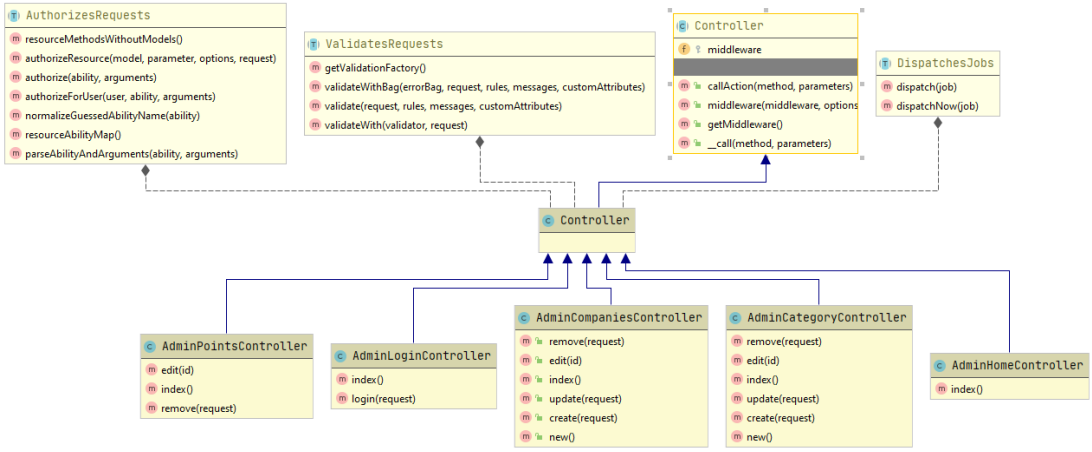

Рисунок 3.3 – Діаграма класів адмін-панелі

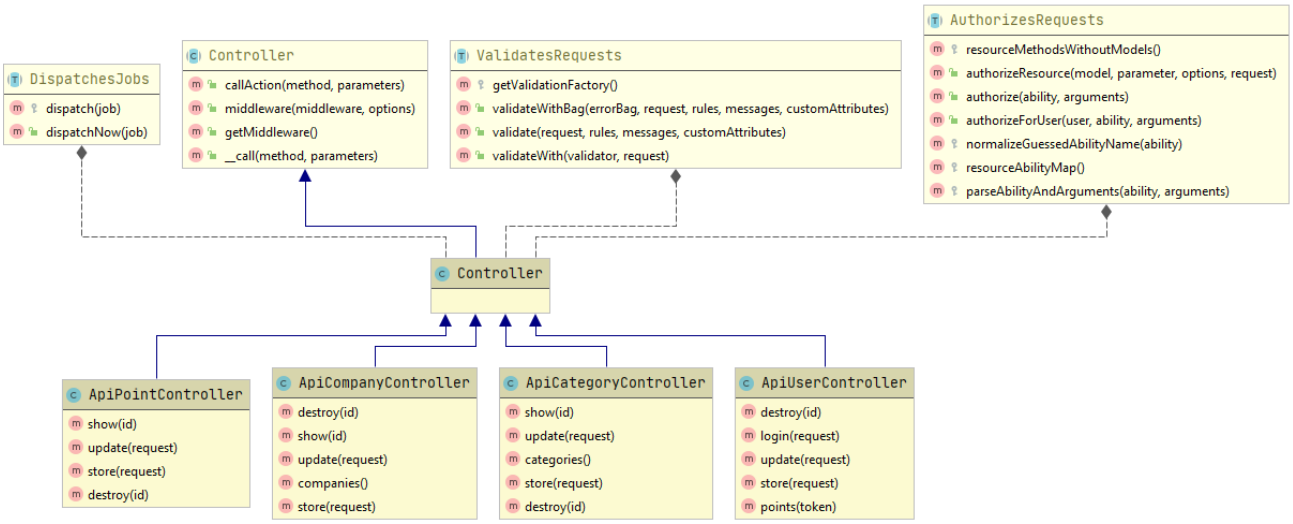

Рисунок 3.4 – Діаграма класів Api

3.2 Програмна реалізація

3.2.1 Створення веб-додатку

Було створено новий проект Laravel та всі його компоненти.

Завершальним етапом налаштування проекту є створення конфігураційного файлу Nginx в директорії /etc/nginx/sites-available.

За угодою в контексті, що починається ключовим словом server, знаходиться опис «віртуального сервера». так називається логічний набір ресурсів, зіставлений із значенням директиви server_name [17].

Приклад конфігураційного файлу Nginx:

```
server {

    listen 80 ;
    listen [::]:80 ;

    root /var/www/4house-office/public;

    index index.php;
```

```
    server_name 4house-office.dan;

    location / {
        try_files \$uri \$uri/ /index.php?\$query_string;
    }

    location /public/css/ {
        alias /var/www/4house-office/public/css/;
    }

    location /public/js/ {
        alias /var/www/4house-office/public/js/;
    }

    location ~ \.php$ {
       try_files $uri $uri/ =404;
        include /etc/nginx/fastcgi.conf;
        fastcgi_pass unix:/run/php/php7.2-fpm.sock;
    }
}
```

Сучасні фреймворки такий як Laravel дозволяють легко визначати структуру вашої бази даних за допомогою міграцій на основі коду. Кожна нова таблиця, стовпець, індекс і ключ можуть визначатися в коді, а кожне нове середовище - переноситися з пустої бази даних в ідеальну схему вашої програми за лічені секунди [18].

Міграція - це окремий файл, який визначає модифікації для бази даних, які необхідно виконати при запуску цієї міграції на верх і вниз.

Приклад міграції :

```
public function up()
    {
        Schema::create('points', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->bigInteger('category_id')->unsigned();
            $table->bigInteger('user_id')->nullable()->unsigned();
            $table->bigInteger('company_id')->nullable()->unsigned();;
            $table->string('title');
```

```
                $table->decimal('lat', 10, 8);
                $table->decimal('lng', 11, 8);
                $table->text('comment')->nullable();
                $table->string('address')->nullable();
                $table->boolean('status')->default(false);
                $table->string('image',255)->nullable();
                $table->timestamp('created_at')->useCurrent();
                $table->timestamp('updated_at')->default(DB::raw('CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP'));
            });

        }
```

Далі для створення Api потрібно визначити майбутні url, які знаходяться в routes/api.php. Приклад урлів:

```
Route::post('/user/login', 'Api\ApiUserController@login');
Route::put('/user', 'Api\ApiUserController@update');
Route::post('/user', 'Api\ApiUserController@store');

Route::get('/user/points/{token}', 'Api\ApiUserController@points');
Route::get('/categories', 'Api\ApiCategoryController@categories');
Route::get('/companies', 'Api\ApiCompanyController@companies');
Route::get('/points', 'Api\ApiPointController@points');

Route::resource('/point', 'Api\ApiPointController',
    ['only' => ['show', 'store', 'update', 'destroy']])->parameters([
    'point' => 'id',
]);
Route::resource('/company', 'Api\ApiCompanyController',
    ['only' => ['show', 'store', 'update', 'destroy']])->parameters([
    'company' => 'id',
]);
```

Було створено новий контролер для Api:

```
<?php
namespace App\Http\Controllers\Api;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
```

```
use App\Company;
class ApiCompanyController extends Controller{
    public function __construct()    {
        $this->middleware('is_admin', ['only' => ['store', 'update']]);
    }
    public function show($id) {
        return Company::find($id);
    }
    public function companies(){
        return Company::all();
    }
    public function store(Request $request){
        Company::insert([
            'name'          => $request->name,
            'description'   => $request->description,
            'contacts'      => $request->contacts,
        ]);
    }
    public function update(Request $request){
        $to_update = Company::clearData($request->all());
        Company::where('id', $request->id)->update($to_update);
    }
    public function destroy($id){
        Company::destroy($id);
    }
}
```

3.2.2 Створення мобільного додатку

Android Studio використовує систему збирання Gradle для компіляції коду програми в файл APK. Gradle також управляє залежностями проекту – зокрема, включенням в процес складання бібліотек, використовуваних додаток [19].

Для мобільного додатку створимо новий проект в Android Studio, та додамо необхідні бібліотеки, для роботи з мапою та http запитами у файл залежностей.

Приклад файлу залежностей:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
```

```
        useLibrary 'org.apache.http.legacy'
        defaultConfig {
            applicationId "com.example.a4house_office"
            minSdkVersion 21
            targetSdkVersion 29
            versionCode 1
            versionName "1.0"
            testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        }
        buildTypes {
            release {
                minifyEnabled false
                proguardFiles          getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
            }
        }
        compileOptions {
            sourceCompatibility JavaVersion.VERSION_1_8
            targetCompatibility JavaVersion.VERSION_1_8
        }
    }

    dependencies {
        implementation fileTree(dir: 'libs', include: ['*.jar'])
        implementation 'androidx.appcompat:appcompat:1.0.2'
        implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
        implementation 'androidx.lifecycle:lifecycle-extensions:2.0.0'
        implementation 'com.google.android.material:material:1.1.0'
        implementation 'androidx.annotation:annotation:1.1.0'
        implementation 'androidx.legacy:legacy-support-v4:1.0.0'
        testImplementation 'junit:junit:4.12'
        androidTestImplementation 'androidx.test:runner:1.1.1'
        androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
        implementation 'com.squareup.okhttp3:okhttp:3.10.0'
        implementation 'com.loopj.android:android-async-http:1.4.9'
        implementation 'com.mapbox.mapboxsdk:mapbox-android-sdk:9.1.0'
        implementation 'com.mapbox.mapboxsdk:mapbox-android-plugin-annotation-v9:0.8.0'
        implementation 'com.google.android.gms:play-services-location:17.0.0'
        implementation 'org.apmem.tools:layouts:1.10@aar'
    }
```

Далі створимо інтерфейс для сторінок. Це робиться в файлі формату XML.

Приклад інтерфейсу:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/login_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.main.LoginActivity">

    <RelativeLayout
        android:layout_width="411dp"
        android:layout_height="276dp"
        android:layout_centerVertical="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.499"
        tools:ignore="MissingConstraints">

        <EditText
            android:id="@+id/et_email"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:hint="@string/email"
            android:inputType="textEmailAddress"
            android:textColor="@color/nav_item_news" />

        <EditText
            android:id="@+id/et_password"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_below="@id/et_email"
            android:layout_centerHorizontal="true"
            android:hint="@string/password"
            android:inputType="textPassword"
            android:textColor="@color/nav_item_news" />
```

```
        <Button
            android:id="@+id/btn_authorization"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_below="@+id/et_password"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="30dp"
            android:layout_marginBottom="30dp"
            android:background="@drawable/action_btn"
            android:text="@string/action_enter"
            android:textColor="@color/nav_item_news" />

        <Button
            android:id="@+id/btn_registration"
            android:layout_width="300dp"
            android:layout_height="wrap_content"
            android:layout_below="@+id/btn_authorization"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="5dp"
            android:layout_marginBottom="30dp"
            android:background="@drawable/action_btn"
            android:text="@string/action_register"
            android:textColor="@color/nav_item_news" />

    </RelativeLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Для кожної сторінки зробимо класс контролера. Приклад файлу контролера:

```
package com.example.a4house_office.ui.main;

import androidx.lifecycle.ViewModelProviders;

import android.content.Intent;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;

import com.example.a4house_office.R;

public class AccountPointsFragment extends Fragment {

    private AccountPointsViewModel mViewModel;

    public static AccountPointsFragment newInstance() {
        return new AccountPointsFragment();
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container, @Nullable Bundle savedInstanceState) {

        return inflater.inflate(R.layout.account_points_fragment, container, false);
    }

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        mViewModel = ViewModelProviders.of(this).get(AccountPointsViewModel.class);
    }

}
```

3.3 Приклад роботи інформаційної системи

Для початку роботи з веб-сайтом користувачу потрібно авторизуватися у системі. Сторінка авторизації наведена на рисунку 3.5.

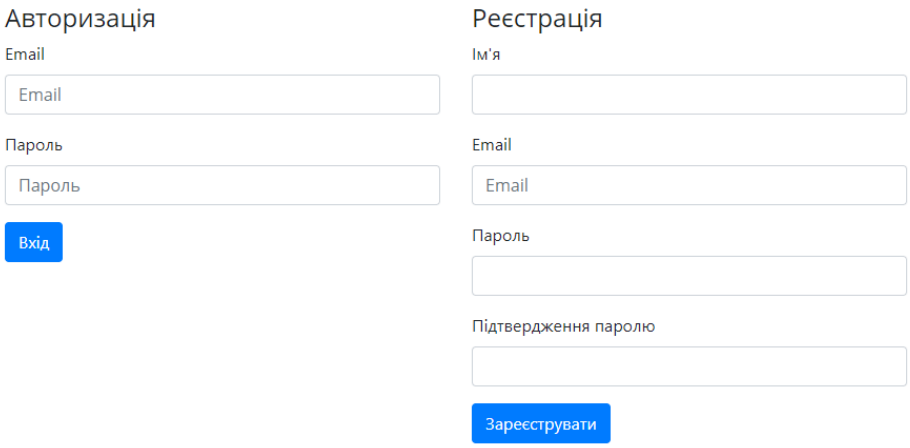

Рисунок 3.5 – Сторінка авторизації

Щоб додати координату на мапі де виявлено порушення потрібно знайти це місце на мапі та натиснути на неї. У спливаючому вікні натиснути на кнопку “Додати” (рис. 3.6).

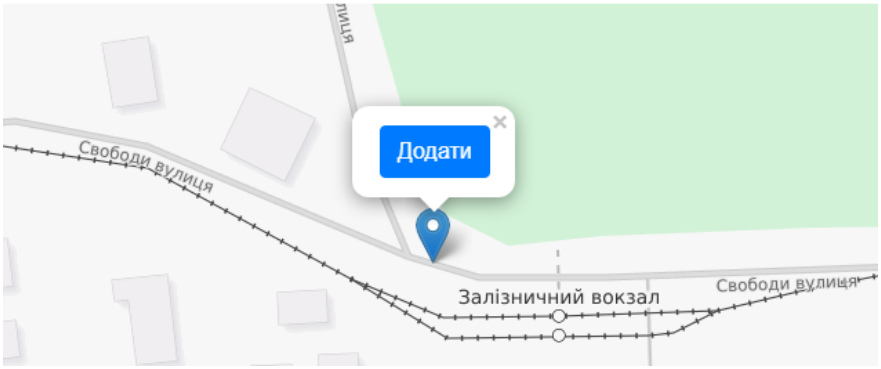

Рисунок 3.6 – Вибір координати

У формі, що з’явилася необхідно заповнити дані. Приклад форми надано на рисунку 3.7.

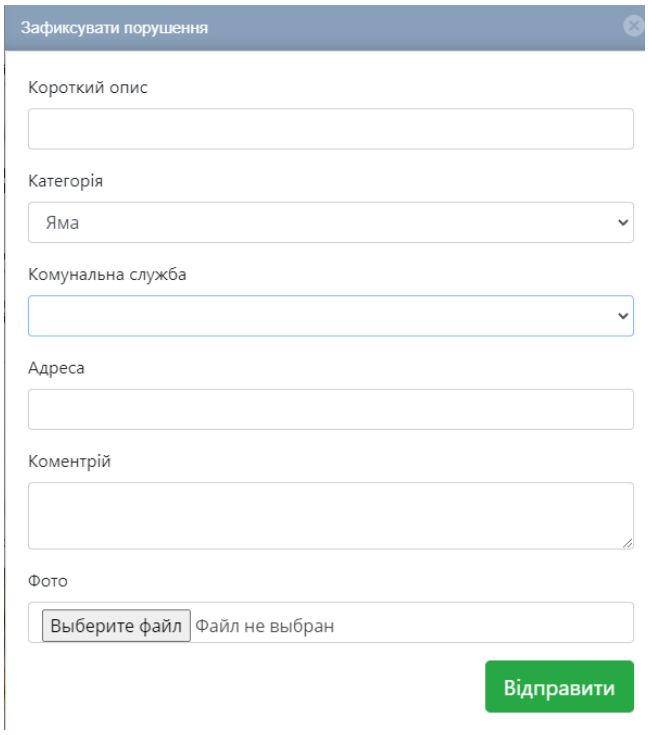

Рисунок 3.7 – Форма фіксації порушення

Після того як користувач заповнить форму та натисне кнопку “Відправити”, дані занесутся до бази.

Для початку роботи з мобільним-додатком користувачу потрібно авторизуватися у системі (рис. 3.8).

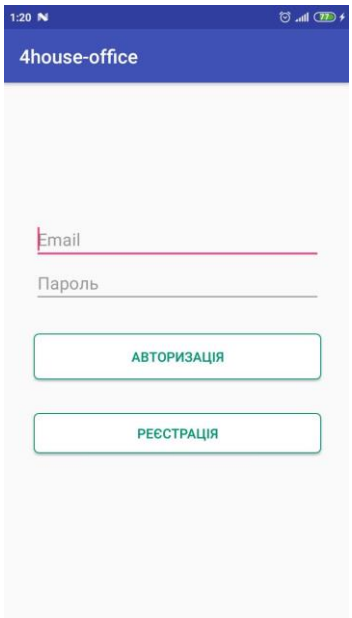

Рисунок 3.8 – Форма авторізації

В тому випадку якщо користувач не зареєстрований, передбачено форму реєстрації, форма реєстрації надано на рисунку 3.9

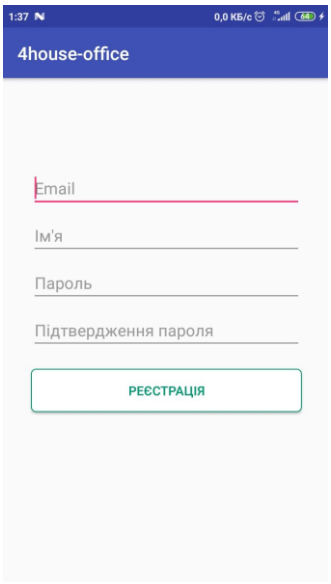

Рисунок 3.9 – Форма реєстрації

Після цього, щоб додати порушення потрібно натиснути на кнопку “Повідомити про проблему” (рис. 3.8).

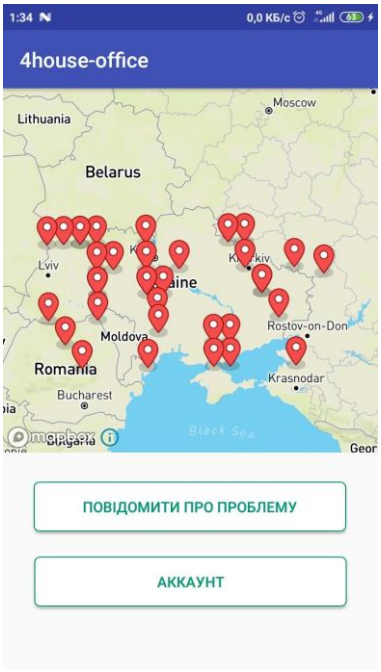

Рисунок 3.10 – Головна сторінка

Далі зробити зображення та заповнити форуму (рис. 3.11).

Тривалі операції, а також операції, які блокують виконання програми до їх завершення (мережеві операції, звернення до файлів) повинні виконуватися поза потоком графічного інтерфейсу [20]. Тому усі запити було зроблено асинхронно.

Рисунок 3.11 – Форма фіксації порушення в мобільному додатку

ВИСНОВКИ

В ході кваліфікаційної роботи було проаналізовано стан сучасних інформаційних систем – аналогів;

У ході аналізу було визначено актуальність та мета роботи, яка полягає у створенні інформаційної системі фіксації порушення благоустрою міста Конотоп, у вигляді веб-сайту та мобільного додатку, використання якого поліпшить благоустрій міста.

Після закінчення аналізу було сформоване технічне завдання, де було визначено мета, призначення та вимоги.

Для кращого розуміння бізнес процесів інформаційної системи було створена контекстна діаграма та декомпозиції у нотації IDEF0.

Була створена діаграма варіантів використання для визначення послідовності дій.

Було створено веб-сайт на фреймворку Laravel, створено Api для мобільного додатку.

За допомогою Andorid studio було створено мобільний додаток.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Про розвиток сфери відкритих даних в Україні – Режим доступу: https://habr.com/ru/post/306414/.

2 Rebecca M. Riordan: Head First Ajax: A Brain-Friendly Guide Видавництво: "O'Reilly Media" –2008 – 5 с.

3 Jonathan Bennett: OpenStreetMap Видавництво: "Pact" –2010 – 25 с.

4 Оцінка якості інформаційної системи [Електронний ресурс]. – Режим доступу: https://helpit.me/articles/ocenka-kacestva-informacionnoi-sistemy/.

5 В.М. Кислий «Методологія та організація наукових досліджень». Видавництво: СумДУ, –2009. – 23 с.

6 Дронов В. А. «Laravel. Швидка розробка сучасних динамічних Web-сайтів на PHP, MySQL, HTML і CSS» Видавництво: БХВ-Петербург, –2017. –329 с.

7 Шварц Б., Зайцев П., Ткаченко В «MySQL по максимуму. 3-е изд» Видавництво:Пітер, –2018 –28 с.

8 Andrew Aksyonoff «Introduction to Search with Sphinx» Видавництво: O'Reilly Media, –2011. –21 с.

9 Дейтел Пол «Android для розробників» Видавництво: Пітер, –2016. – 56 с.

10 Черемных С.В., Семенов И.О., Ручкин В.С. Моделирование и анализ систем. IDEF-технологии практикум Видавництво "Финансы и статистика" –2006 –26с.

11 Д.Ю. Киселев, Ю.В. Киселев, А.В. Вавилин: Функциональное моделирование на базе стандарта IDEF0 Видавництво: "СГАУ" –2014 – 5 с.

12 Моделирование на UML Учебно-методическое пособие [Електронний ресурс]. – Режим доступу: https://books.ifmo.ru/file/pdf/722.pdf.

13 Thomas Connolly, Carolyn Begg «Database Systems: A Practical Approach to Design, Implementation, and Management Third Edition» Видавництво: Addison Wesley, –2016. –399 с.

14 П.В. Бураков, В.Ю. Петров: Введение в системы баз данных –2010 – 28 с.

15 Abdul Majeed, Ibtisam Rauf. MVC Architecture: A Detailed Insight to the Modern Web Applications Development. Peer Rev J Sol Photoen Sys .1(1). PRSP.000505. –2018 –1с.

16 Александр Леоненков, Самоучитель UML Видавництво: Бхв-петербург –2004 –133 с.

17 Димитрий Айвалиотис: Администрирование сервера NGINX Видавництво: "ДМК" –2013 – 34 с.

18 Matt Stauffer. Laravel: Up & Running: A Framework for Building Modern PHP Apps Видавництво: "O'Reilly Media" –2020 –111 с.

19 Харди Б: Программирование под Android. Для профессионалов Видавництво: "Питер" –2016 – 250 с

20 Ян Дарвин: Android. Сборник рецептов: задачи и решения для разработчиков приложений Видавництво: " Диалектика" –2018 – 629 с

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку інформаційної системи
фіксації порушень благоустрою міста Конотоп

Суми 2020

1 Призначення й мета створення інформаційної системи

1.1 Призначення інформаційної системи

Інформаційна система призначена для не байдужих людей які хочуть допомогти рідному місту, та для допомоги місту в усуненні проблем з благоустроєм.

1.2 Мета створення інформаційної системи

Метою роботи є розробка інформаційної системи для фіксації порушень благоустрою міста Конотоп.

2 Вимоги до інформаційної системи в цілому

2.1 Вимоги до структури й функціонування інформаційної системи

Інформаційна система повинна бути реалізована у вигляді веб-сайту та мобільного додатку, доступного в мережі Інтернет. Сайт повинен складатися із взаємозалежних розділів із чітко розділеними функціями. Мобільний складатися додаток чітко розділеними функціями.

2.2 Вимоги до персоналу

Для роботи з веб-сайтом треба мати загальні навички з стандартним веб-браузером (наприклад, Google Chrome).

2.3 Вимоги до стилістичного оформлення сайту

Веб-сайт повинен бути розроблений з використанням мови html та php як серверна мова, містити на головній сторінці мапу. Стилистично веб-сайт повинен бути в діволому тоні. Мобільний додаток повинен бути написаний на мові Java або Kotlin містити на головній сторінці мапу.

3 Основні вимоги

3.1 Структура інформаційної системи

Інформаційна система повинна складатися з наступних розділів:

- сторінка Головна – містить мапу з позначками проблемних місць;

- сторінка Авторизація / Реєстрація – містить форму для аутентифікації;
- сторінка Особистий кабінет – містить список позначок та їх статус;
- сторінка Контактна інформація – містить інформацію про веб-сайт та інформацію про компетентні органи;
- сторінка Адмін панель – містить інформацію про заяви та має можливість їх редагувати;

Мобільний додаток повинен складатися з наступних розділів:

- сторінка Головна – містить мапу з позначками проблемних місць;
- сторінка Особистий кабінет – містить список позначок та їх статус;
- сторінка Авторизація / Реєстрація – містить форму для аутентифікації;

3.2 Вимоги до програмного забезпечення

Програмне забезпечення повинне задовольняти наступним рекомендованим вимогам:

- Веб-браузер: Internet Explorer 8.0 і вище, або Firefox 8.5 і вище, або Opera 9.5 і вище, або Safari 7.1 і вище, або Chrome 70.0 і вище;
- Включена підтримка JavaScript.

3.3 Функціональні вимоги

- перегляд мапи;
- перегляд поміток;
- додавання нових поміток;
- генерування заяв;
- відправка заяв на поштову скриньку.

ДОДАТОК Б

ПЛАНУВАННЯ РОБІТ

1 Ідентифікація ідеї проекту

Дипломний проект призначений для того, щоб мешканці міста допомогли виявити проблемні місця, а керівництво міста отримувало дані, та швидко й ефективно вирішувало проблеми.

Інформаційна система повинна бути реалізована у вигляді веб-сайту та мобільного додатку. Веб-сайт та мобільний додаток повинен бути оформлений у діловому стилі.

2 Деталізація мети методом SMART

Конкретна (Specific). Створити програмний продукт для поліпшення благоустрою міста.

Вимірювана (Measurable). Розробити якісний програмний продукт.

Досяжна (Achievable). Поставлена мета буде впливати у на благоустрій міста та її привабливість.

Реалістична (Relevant). Є всі необхідні дані та технічні засоби.

Обмежена у часі (Time—framed). Ціль має часове обмеження. Терміни досягнення мети проекту визначаються за датою здачі кваліфікаційної работы.

1 Постановка задачі

1.1 Аналіз предметної області

1.1.1 Аналіз аналогічних інформаційних систем

1.1.2 Аналіз інструментів розробки

2 Моделювання системи

3 Створення сайту

3.1 Розробка арі

3.2 Створення БД

3.3 Створення шаблону

4. Тестування сайту

4.1 Написання unit тестів

4.2 Ручне тестування

5 Створення мобільного додатку

5.1 Підключення до Api

5.2 Тестування мобільного додатку

WBS-структура для даного проекту представлена на рисунку Б.1

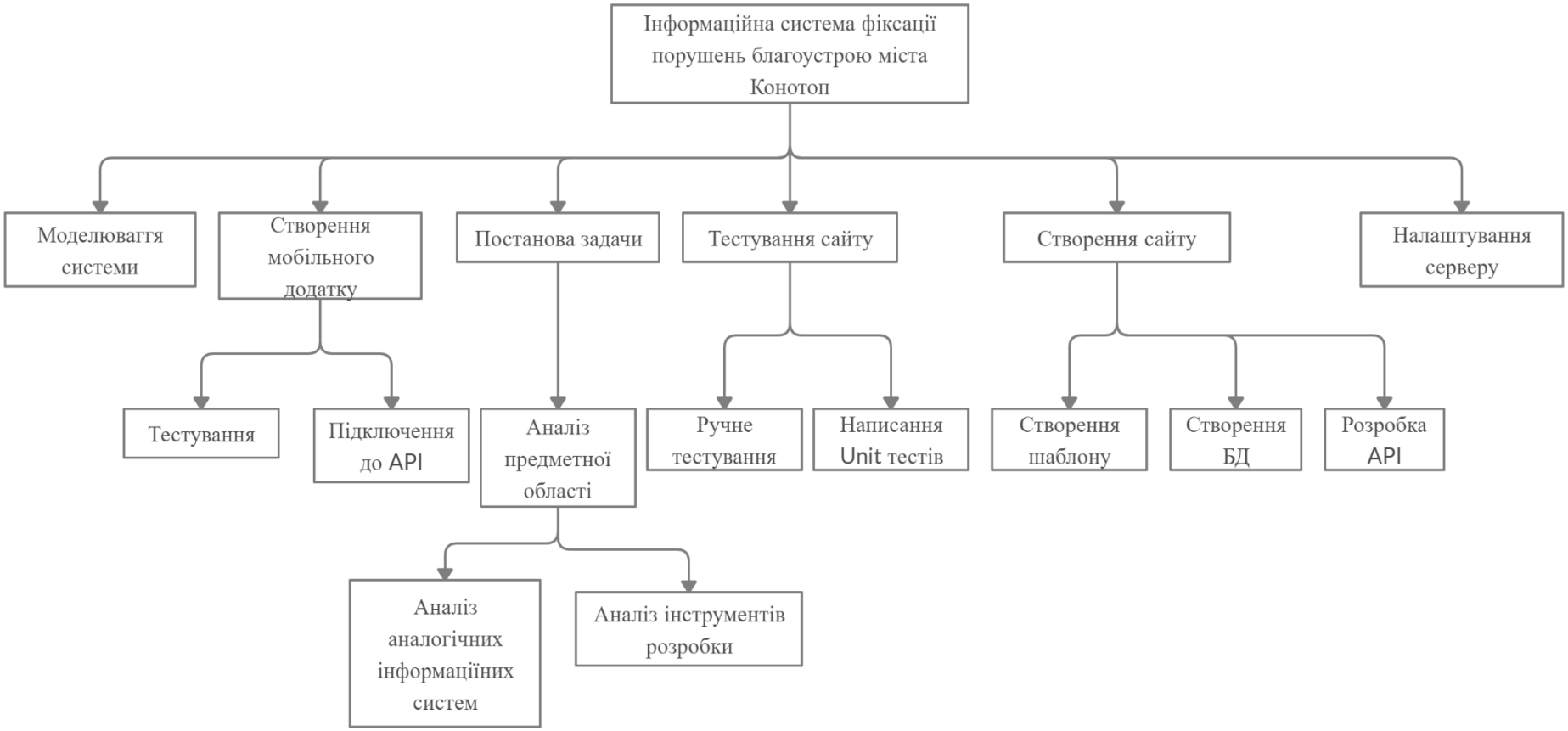

Рисунок Б.1 – WBS-структура інформаційної системи

3 Побудова матриці відповідальності (виконавців пакетів робіт)

Матриця відповідальності представлена в табл. Б.1

Таблиця Б.1 – Матриця відповідальності на основі календарного плану

WBS\OBS	Пітерцев Д. В	Нагорний В. В
1 Постановка задачі	+	+
1.1 Аналіз предметної області	+	+
1.1.1 Аналіз аналогічних інформаційних систем	+	+
1.1.2 Аналіз інструментів розробки	+	+
2 Моделювання системи	+	
3. Створення сайту	+	
3.1 Розробка api	+	
3.2 Створення БД	+	
3.3 Створення шаблону	+	
4. Тестування сайту	+	
4.1 Написання unit тестів	+	
4.2 Ручне тестування	+	
5 Створення мобільного додатку	+	
5.1 Підключення до Api	+	
5.2 Тестування мобільного додатку	+	

4 Побудова календарного графіку виконання ІТ—проекту

Діаграма Ганта – діаграма, яка використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з засобів планування та управління проектами.

Графік виконання дипломного проекту на основі календарного плану, представлено у вигляді Діаграми Ганта (рис. Б.2).

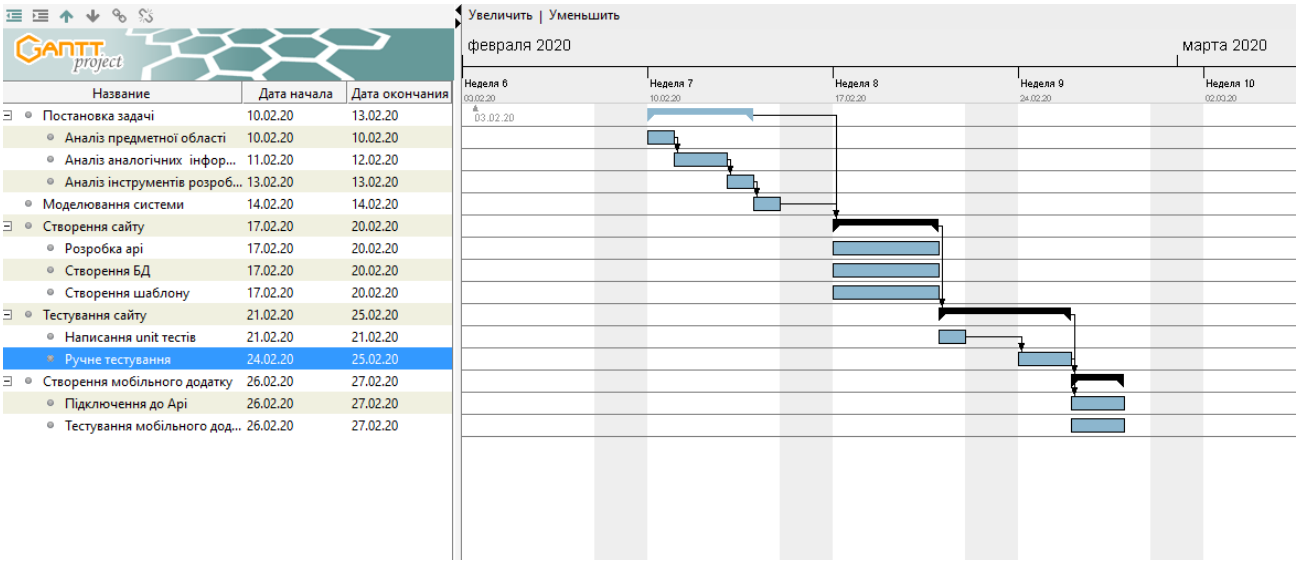

Рисунок Б.2 – Діаграма Ганта

ДОДАТОК В

```
<?php
namespace App\Http\Controllers\Account;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\User;
use App\Point;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
class AccountController extends Controller{
    public function index(){
        $points = Point::where(['user_id'=> auth()->id()])->paginate(25);
        return view('account.account_page',['points' => $points]);
    }
    public function login(){
        return view('account.login');
    }
    public function auth(Request $request){
        $credentials = $request->only('email', 'password');
        if (Auth::attempt($credentials)) {
            return redirect()->to('account');
        }
    }

    public function signup()   {
        return view('account.registration');
    }

    public function logout(){
        Auth::logout();
        return redirect()->to('/');
    }

    public function edit(){
        return view('account.edit');
    }
}

<?php
namespace App\Http\Controllers\Admin;
```

```
use App\Category;
use App\Company;
use App\Http\Controllers\Controller;
use App\Point;
use Illuminate\Http\Request;
use App\Jobs\SendPointJob;

class AdminCategoryController extends Controller{
    public function index(){
        $categories = Category::paginate(25);
        return view('admin.categories', ['categories' => $categories]);
    }

    public function edit($id) {
        $category = Category::find($id);
        $action = route('admin.category.update',['id' => $category->id]);
        $method = "PUT";
        return view('admin.category', ['category' => $category,'action' => $action, 'method'
=> $method]);
    }

    public function update(Request $request){
        $to_update = Category::clearData($request->all());
        Category::where('id', $request->id)->update($to_update);
        return redirect()->to('admin/categories');
    }

    public function new(){
        $action = route('admin.category.create');
        return view('admin.category',['action' => $action]);
    }
    public function create(Request $request){
        $to_create = Category::clearData($request->all());
        Category::insert($to_create);
        return redirect()->to('admin/categories');
    }
    public function remove(Request $request) {
        Category::destroy($request->id);
    }

}
<?php
```

```
namespace App\Http\Controllers\Admin;
use App\Company;
use App\Http\Controllers\Controller;
use App\Point;
use Illuminate\Http\Request;
use App\Jobs\SendPointJob;
class AdminCompaniesController extends Controller{
    public function index()    {
        $companies = Company::paginate(25);
        return view('admin.companies', ['companies' => $companies]);
    }

    public function edit($id)    {
        $company = Company::find($id);
        $action = route('admin.company.update',['id' => $company->id]);
        $method = "PUT";
        return  view('admin.company_edit',  ['company'  =>  $company,'action'  =>  $action,
'method' => $method]);
    }

    public function update(Request $request){
        $to_update = Company::clearData($request->all());
        if ($request->has('remove_image') && $request->remove_image) {
            $to_update['image'] = '';
        }
        if ($request->hasFile('photo')) {
            $image= $request->photo->store('company','public');
            $to_update['image'] = $image;
        }
        Company::where('id', $request->id)->update( $to_update);
        return redirect()->to('admin/companies');
    }

    public function new(){
        $action = route('admin.company.create');
        return view('admin.company_edit',['action' => $action]);
    }

    public function create(Request $request){
        $to_create = Company::clearData($request->all());
        Company::insert($to_create);
        return redirect()->to('admin/companies');
```

```
    }

    public function remove(Request $request) {
        Company::destroy($request->id);
    }
}

<?php
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Route;
class AdminHomeController extends Controller{
    public function index()    {
        return view('admin.home',[]);
    }
}

<?php
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use App\Point;
use Illuminate\Http\Request;
use App\Jobs\SendPointJob;
use Illuminate\Support\Facades\Auth;


class AdminLoginController extends Controller{
    public function index() {
        return view('admin.login');
    }

    public function login(Request  $request){
        $credentials = $request->only('email', 'password');

        if (Auth::attempt($credentials)) {
            if(Auth::user()->role=='admin'){
                return redirect()->to('admin');
            }else{
                Auth::logout();
                return redirect()->to('admin/login');
            }
        }
```

```
            return redirect()->to('admin/login');
        }
    }

    <?php
    namespace App\Http\Controllers\Admin;
    use App\Category;
    use App\Company;
    use App\Http\Controllers\Controller;
    use App\Point;
    use Illuminate\Http\Request;
    use App\Jobs\SendPointJob;
    class AdminPointsController extends Controller{
        public function index()    {
            $points = Point::with('category')->paginate(25);
            return view('admin.points', ['points' => $points]);
        }

        public function edit($id)  {
            $point = Point::find($id);
            $categories = Category::all();
            $companies = Company::all();

            return view('admin.point_edit', ['point' => $point, 'categories' => $categories,
'companies' => $companies]);
        }

        public function remove(Request $request)    {
            Point::destroy($request->id);
        }
    }

    <?php
    namespace App\Http\Controllers\Api;
    use Illuminate\Http\Request;
    use App\Http\Controllers\Controller;
    use App\Category;

    class ApiCategoryController extends Controller{
        public function show($id){
            return Category::find($id);
        }
```

```
    public function categories(){
        return Category::all();
    }

    public function update(Request $request)  {
        $to_update = Category::clearData($request->all());
        Category::where('id', $request->id)->update($to_update);
    }

    public function destroy($id) {
        Category::destroy($id);
    }
}
<?php

namespace App\Http\Controllers\Api;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Point;
use Illuminate\Support\Facades\Storage;

class ApiPointController extends Controller{
    public function __construct() {
        $this->middleware('point_validator', ['only' => [ 'store', 'update' ]]);
        $this->middleware('is_admin', ['only' => [ 'update' ,'destroy']]);
    }

    public function show($id){
        return Point::find($id);
    }

    public function points(){
        return Point::take(50)->get();
    }

    public function store(Request $request) {
        $image = '';
        if ($request->hasFile('photo')) {
            $image = $request->photo->store('points','public');
        }
```

```
        Point::insert([
            'category_id'   => $request->category_id,
            'lat'           => $request->lat,
            'lng'           => $request->lng,
            'title'         => $request->title,
            'comment'       => $request->comment,
            'image'         => $image,
        ]);
    }

    public function update(Request $request) {
        $to_update = Point::clearData($request->all());
        if ($request->hasFile('photo')) {
            $to_update['image'] = $request->photo->store('points','public');
        }
        if($request->hasFile('remove_photo') AND $request->remove_photo){
            $to_update['image']='';
        }
        Point::where('id', $request->id)->update($to_update);
    }
    public function destroy($id) {
        Point::destroy($id);
    }
}

<?php
namespace App\Http\Controllers\Api;
use App\Point;
use App\UsersToken;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\User;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
class ApiUserController extends Controller{
    public function __construct() {
        $this->middleware('user_validator', ['only' => ['destroy']]);
    }

    public function login(Request $request) {
        $credentials = $request->only('email', 'password');
        Auth::logout();
```

```
    if (Auth::attempt($credentials)) {
        if($request->has('need_token')){
            $token = bin2hex(random_bytes(15));
            UsersToken::insert(['user_id' => Auth::user()->id, 'token' => $token]);
        }
        $userInfoJson = array(
            'id'    => Auth::user()->id,
            'name'  => Auth::user()->name,
            'email' => Auth::user()->email
        );
        if($request->has('need_token')){
            $userInfoJson['token'] = $token;
        }
        return $userInfoJson;
    }
    return ['error'=>''];
}

public function points($token) {
    $user_token = UsersToken::where('token', $token)->first();

    if (!empty($user_token)) {
        return Point::where('user_id', $user_token->user_id)->get()->toArray();
    } else {
        return ['error' => 'user_not_found'];
    }
}

public function store(Request $request) {
    $errors = [];
    $to_create = [];
    if ($request->has('email')) {
        $check_user = User::where('email', $request->email)->first();

        if (empty($check_user)) {
            $to_create['email'] = $request->email;
        } else {
            $errors[] = 'email_exists';
        }
    } else {
        $errors[] = 'email';
    }
```

```
            if ($request->has('name')) {
                $to_create['name'] = $request->name;
            } else {
                $errors[] = 'name';
            }
            if ($request->has('password') && $request->has('password_confirm') && $request->password == $request->password_confirm) {
                $to_create['password'] = $request->password;
            } else {
                $errors[] = 'password_confirm';
            }
            if (!$errors) {
                $user = User::create([
                    'name'      => $request->name,
                    'email'     => $request->email,
                    'password'  => Hash::make($request->password)        ]);

                auth()->login($user);
                return ['success' => 1];
            } else {
                return ['errors' => $errors];
            }
        }

        public function update(Request $request)  {
            $errors = [];
            if (Auth::check()) {
                $to_update = [];
                if ($request->has('name')) {
                    $to_update['name'] = $request->name;
                } else {
                    $errors[] = 'name';
                }
                if ($request->has('email')) {
                    if (Auth::user()->email != $request->email) {
                        $to_update['email'] = $request->email;
                    }
                } else {
                    $errors[] = 'email';
                }
                if ($request->has('password')) {
```

```
                    if ($request->has('password_confirm') && $request->password == $request-
>password_confirm) { $to_update['password'] = Hash::make($request->password);
                    } else {
                        $errors[] = 'password_confirm';
                    }
                }

                if ($to_update && empty($errors)) {
                    User::where('id', Auth::user()->id)->update($to_update);

                    return ['success' => 1];
                } else {
                    return ['errors' => $errors];
                }
            } else {
                return ['errors' => ['not_logged']];
            }
        }
        public function destroy($id) {
            User::destroy($id);
        }
    }

    <?php
    namespace App\Http\Controllers;
    use App\Company;
    class CompaniesController extends Controller{
        public function index(){
            $companies = Company::paginate(25);
            return view('companies', ['companies' => $companies]);
        }
    }

    <?php
    namespace App\Http\Controllers;
    use App\Company;
    use App\Point;
    use App\Category;
    use App\Search;
    class HomeController extends Controller{
        public function index() {
            $data['points'] = Point::all()->toJson();
```

```
        $data['points_card'] = Point::orderBy('id', 'DESC')->take(20)->get();
        $data['categories'] = Category::all();
        $data['companies'] = Company::all();
        Search::searchInSphinx('asd');
        return view('home',$data);
    }
}

<?php
namespace App\Http\Controllers;
use App\Category;
use App\Company;
use App\Point;
use App\Search;
use Illuminate\Http\Request;
class PointsController extends Controller{
    public function index($id)    {
        $point = Point::find($id);
        return view('point_info', ['point' => $point]);
    }

    public function search(Request $request)   {
        $data['search_value'] = '';
        $data['appends'] = [];
        if ($request->has('search')) {
            $data['search_value'] = $request->search;
            $data['appends']['search'] = $request->search;
            if ($request->has('category_id')) {
                $data['category_id'] = $request->category_id;
                $data['appends']['category_id'] = $request->category_id;
            }
            if ($request->has('company_id')) {
                $data['company_id'] = $request->company_id;
                $data['appends']['company_id'] = $request->company_id;
            }
            $ids = Search::searchInSphinx($request->search);
            $points = Point::whereIn('id', $ids)->paginate(25);
            $data['points'] = $points;
        }
        $data['categories'] = Category::all();
        $data['companies'] = Company::all();
        return view('search_page', $data);    }}
```