НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

# ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

зі спеціальності 122 Комп’ютерні науки та інформаційні технології
за спеціалізацією Геометричне моделювання в інформаційних системах
на тему : Мобільний застосунок для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами

Виконала: студентка ___4___курсу, групи__ТР-61____

_____Гулак Олена Сергіївна_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____проф. Отрох С. І._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант ____доц., к.т.н. Шалденко О. В._____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

зі спеціальності 122 Комп’ютерні науки та інформаційні технології
за спеціалізацією Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

# ЗАВДАННЯ
**на дипломну роботу студентці**
Гулак Олені Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи Мобільний застосунок для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами
керівник роботи Отрох Сергій Іванович, проф.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи Мобільний застосунок для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Проаналізувати задачу створення мобільного застосунку по моніторингу та ідентифікації незаконно розміщеної зовнішньої реклами
2) Аналіз існуючих рішень та шаблонів проектування при розробці застосувань
3) Моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
4) Аналіз якості та тестування програмного забезпечення
5. Перелік ілюстративного матеріалу
1) Схеми архітектури додатку
2) Схема структурна варіантів використання

3) Діаграма компонентів додатку
4) Схема бази даних
5) Зразки розробленого інтерфейсу додатку
6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | Шалденко О. В., доц., к.т.н | | |

7. Дата видачі завдання ”___”_____201___ р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 25.10.19 | |
| 2. | Вивчення та аналіз задачі | 27.11.19-16.12.19 | |
| 3. | Розробка архітектури та загальної структури системи | 17.12.19-27.01.20 | |
| 4. | Розробка структур окремих підсистем | 28.01.20-06.04.20 | |
| 5. | Програмна реалізація системи | 07.04.20-04.05.20 | |
| 6. | Оформлення пояснювальної записки | 05.05.20-04.06.20 | |
| 7. | Захист програмного продукту | 10.06.20 | |
| 8. | Передзахист | 10.06.20 | |
| 9. | Захист | | |

Студентка _____ Гулак О. С.
(підпис) (прізвище та ініціали,)

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

# АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 21 рисунок, 15 таблиць, додатки, 20 джерел.

Дипломний проект присвячений розробці комплексу задач моніторингу та ідентифікації незаконного розміщення зовнішньої реклами. Мета розробки - розробити зручний сервіс для створення та передачі заявок між інспекторами та супервізорами, що може бути використано українськими комунальними підприємствами.

В дипломному проекті були розглянуті: методи боротьби з незаконно розміщеною зовнішньою рекламою, архітектурні шаблони проектування програмного забезпечення.

У першому розділі було проведено змістовну постановку задачі. За результатами аналізу існуючих методів вирішення проблеми усунення незаконно розміщених оголошень в різних розвинених країнах було сформовано власний алгоритм.

У розділі з аналізу існуючих рішень та шаблонів проектування при розробці застосувань було визначено найбільш оптимальний патерн для додатку. Огляд існуючих аналогів програмного забезпечення допоміг сформувати перелік головних вимог до розроблюваного рішення.

У розділі з моделювання програмного забезпечення було сформовано функціональні та нефункціональні вимоги, визначено архітектуру програмного забезпечення, розроблено структуру бази даних. У четвертому розділу було розроблено користувацький інтерфейс, представлено діаграму послідовності при створенні заявки про незаконне розміщення реклами, спроектовано програмне забезпечення для платформи iOS. В останньому розділі представлено взаємодію користувача з системою.

Ключові слова: архітектурний патерн, створення заявки, незаконно розміщена зовнішня реклама, інспектор, супервізор, метод автодозвону.

# ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 21 figures, 15 tables, applications, 20 sources.

Diploma project is dedicated to the development of a set of tasks for monitoring and identification of illegal placement of outdoor advertising. The purpose of the development is to develop a convenient service for creating and transferring applications between inspectors and supervisors, which can be used by Ukrainian utilities.

The diploma project considered: methods of combating illegally placed outdoor advertising, architectural templates for software design.

In the first section, a meaningful statement of the problem was made. Based on the analysis of existing methods of solving the problem of eliminating illegally placed ads in different developed countries, an own algorithm was formed.

In the section on the analysis of existing solutions and design patterns in the development of applications, the most optimal pattern for the application was determined. A review of existing software analogues helped to form a list of the main requirements for the developed solution.

In the section on software modeling, functional and non-functional requirements were formed, the software architecture was defined, and the database structure was developed. In the fourth section, a user interface was developed, a sequence diagram for creating an application for illegal advertising was presented, and software for the iOS platform was designed. The last section presents the user's interaction with the system.

Key words: architectural pattern, application creation, illegally placed outdoor advertising, inspector, supervisor, autodial method.

# ЗМІСТ

# ВСТУП

В наш час зовнішня реклама дуже популярна та високоефективна. Вона поєднує в собі безліч переваг: широкий обхват аудиторії, доступну вартість і гнучкість дії. На відміну від реклами в друкованих виданнях, на радіо і телебаченні, ніхто цілеспрямовано не купує її, не вмикає радіо і телевізор. Вона не супроводжує розважальні програми для залучення аудиторії. Її розміщують найчастіше в тих місцях, де потік людей досить високий, адже чим більше людей побачить рекламу, тим вище буде її ефективність.

З приростом попиту на зовнішню рекламу виросла і кількість незаконно розміщених оголошень, що породжує нову проблему - забруднення міста. Комунальні підприємства активно борються з порушниками та залучають інспекторів різних організацій, таких як КиївРеклама та Благоустрій, до боротьби з незаконно розміщеними оголошеннями.

З початку 2017 року в Києві запрацювала нова рекламна реформа, що описує норми розміщення оголошень, а саме заборона на розміщення реклами на паркінгах, дорожніх розвязках, перехрестях, жилих комплексах, підземних і наземних переходах. Через це кількість реклами зменшилась у 3 рази. Для того, щоб підтримати рівень доходу, було прийнято рішення збільшити вартість реклами, що і спровокувало потік незаконного розміщення реклами [1].

В наш час внаслідок збільшення доступності мобільного інтернету спостерігається тенденція до використання мобільних додатків для зв’язку з контактними центрами. На даний момент для того, щоб повідомити про несанкціоноване розміщення реклами, необхідно зателефонувати до диспетчера Управління з питань реклами Київської міської державної Адміністрації [2], в той час як додаток дозволяє користувачеві уникнути необхідності самостійно здійснювати телефонний дзвінок, який може обернутися тривалим очікуванням, недоступністю абонента, тратою грошей. Крім того, не завжди користувач опиняється в умовах тиші і має можливість говорити по телефону, що робить використання мобільного додатку зручнішим для здійснення дзвінка. Київ - туристичне місто, тому задля покращення його зовніш-

нього вигляду, в першу чергу необхідно позбавитись від незаконно розміщених вивісок, які закривають історичні будівлі та шкодять фасаду споруд. На сьогодні така реклама остогидла не тільки туристам, але і самим мешканцям міста, а отже люди стають більш свідомими, зростає кількість тих, хто хоче покращити стан зовнішнього вигляду міста. Цим обумовлена актуальність теми розробки застосунку для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами.

Основні завдання роботи:

—виконати аналіз предметної області;

—виконати аналіз існуючих шаблонів проектувань;

—спроектувати та реалізувати iOS-додаток;

—провести тестування застосунку.

У роботі наступний зміст розділів пояснювальної записки:

Перший розділ “Постановка задачі створення додатку для усунення незаконно розміщеної реклами на платформі iOS” описує мету роботи, порівняння методів боротьби з проблемою у різних країнах, постановку задачі реалізації застосунку.

У другому розділі проводиться аналіз існуючих шаблонів проектування програмних рішень.

Третій розділ “Моделювання програмного забезпечення” містить опис та аналіз вимог до застосунку, опис архітектури додатку, варіанти використання програмного забезпечення а також структуру бази даних.

Четвертий розділ “Реалізація програмного забезпечення” містить подробиці реалізації програмного зебезпечення для платформи iOS, UML діаграму послідовності при створенні заявки, аналіз безпеки даних.

П’ятий розділ присвячено опису взаємодії користувача з клієнтським додатком, представленню основних можливостей застосунку, а також результатам тестування.

У висновку описано основні результати, отримані при виконанні дипломної роботи.

# 1. ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ ДОДАТКУ НА ПЛАТФОРМІ iOS ДЛЯ УСУНЕННЯ НЕЗАКОННО РОЗМІЩЕНОЇ ЗОВНІШНЬОЇ РЕКЛАМИ

Метою дипломної роботи є створення мобільного застосунку для моніторингу та ідентифікації незаконно розміщених оголошень, впровадження якого дасть можливість інспекторам комунальних підприємств, а також жителям Києва очищувати своє місто від незаконної реклами.

Боротьба з рекламою проблема не тільки України, але й багатьох інших країн. Наприклад, в червні 2013 року мер Торонто Роб Форд особисто зривав нелегальні оголошення з автобусних зупинок в рамках запуску кампанії Clean Toronto Together. Передбачалося, що представники муніципалітету будуть все літо патрулювати головні вулиці міста, чистити від оголошень стовпи і коробки інженерних мереж, зупинки та поштові скриньки. А також одночасно виписувати порушникам штрафи на суми від 300 до 500 доларів. При цьому оголошення, які розвішуються самими жителями округу з інформацією про втрачену власність, гаражний розпродаж або релігійну подію - під штраф не підпадали. Крім того, добровольці організації Illegal Signs повідомляють в мерію про нелегальні об'єкти (в Торонто, зокрема, заборонена об'ємна реклама на зовнішніх носіях), після чого компанії-порушники демонтують об'єкт і просто переміщують його в інше місце міста. Їм вдалося влаштувати своїх співробітників в усі великі компанії, які пропонують рекламу, і ті повідомляють Illegal Signs оперативну і точну інформацію про нові нелегальні конструкції. Але для запуску такої кампанії необхідно чимало коштів та бажаючих допомагати у вільний час.

Навесні 2012 року житель американського міста Окленд-Парк Тім Лонерган став співпрацювати з місцевим муніципалітетом в спробі зрозуміти, чим він може допомогти своєму місту, потопаючому в оголошеннях з рекламою спа-салонів, чищенням килимів і скупкою старих автомобілів. Він звернув увагу на те, як з такими оголошеннями боролися в сусідньому Голлівуді. Там влаштували змагання серед жи-

телів з нагородою в 500 доларів. Нагороду отримував той, хто збере більше всіх нелегальних оголошень. Вулиці міста очистилися, але через якийсь час оголошення почали з'являтися знову. Тоді влада вирішила випробувати метод автодозвону. Його суть проста: за номером телефону, вказаним в рекламному оголошенні, кожні 30 секунд дзвонить робот, тим самим блокуючи лінію для інших, і зачитує записане повідомлення. У ньому йшлося про те, що автодозвон припиниться, як тільки буде оплачений штраф. В результаті такі дзвінки привели до зниження кількості нелегальних оголошень на 70%, при цьому операція з телефонними «терористами» коштувала всього 300 доларів. Низьковартість данного методу дуже велика перевага, але в Україні купівля телефонного номеру низьковартістна процедура, тому будуть з'являтись ті ж самі оголошення, але з іншими номерами і з часом добровольцям набридне повідомляти номера незаконно розміщенних оголошень і постійно телефонувати диспетречам Управління з питань реклами.

У Нью-Йорку орудує група добровольців Illegal Billboards. Їх мета полягає в тому, щоб визначити, які рекламні оголошення і конструкції в місті є нелегальними, і домогтися того, щоб їх прибрали з вулиць. За оцінкою активістів, половина вивісок в Нью-Йорку розміщуються нелегально. Зовнішня реклама стала фактично чорним ринком, на якому заправляють корпорації, і їх не зупинити офіційними штрафами. Тому потрібна кропітка робота активістів, які розслідують всі підозрілі вивіски і домагаються їх видалення через офіційні механізми. Illegal Billboards навчає всіх бажаючих методам боротьби з нелегальними оголошеннями. Для цього вони проводять регулярні семінари і розміщують на своєму сайті покрокові інструкції. Головний секрет полягає в тому, щоб знати, на які закони посилатися, а також яку інформацію в які органи повідомляти. Навіть в тих випадках, коли представники міста підтверджують легальність тієї чи іншої реклами, активісти перевіряють їх дані. Проблема Нью-Йорка схожа з проблемою Києва, адже у людних місцях незаконної реклами у рази більше, ніж законної. Але добровольців, готових витрачати свій вільний час менше. Отже, і цей метод далекий від реальності Києва.

Більш схожа проблема у Греції, де як і у Києві проблема не тільки з рекламними об’явами, але і з нелегальними конструкціями, на кшталт наших рекламних причепів. Існуючий закон в Греції, який забороняє розміщення реклами уздовж доріг,

дотримувалися тільки під час Олімпіади в Афінах у 2004 році. Жителі столиці довго скаржилися на мера, якого підозрювали у змові з нелегальними рекламодавцями. У підсумку в 2010 році вони домоглися того, що Міністерство транспорту та інфраструктури запустило сайт і спеціальний додаток для iPhone, через який можна повідомляти про нелегальну рекламу [3].

Даний метод є оптимальним і для нашого міста, адже людина, яка хоче повідомити про незаконну рекламу має витратити декілька хвилин для того, щоб зробити знімок незаконної реклами, завантажити його в додаток, встановити геолокацію та вказати номери телефонів зазначених в оголошенні. Після чого вповноважені підприємства переглядають, відстежують та ліквідують усі незаконні об’яви. Система працює на підставі наказу, затвердженого рішенням Київської міської ради від 25 грудня 2008 року N 114.1.7.051/1051. “У разі самовільного розміщення рекламних засобів або інших оголошень рекламного змісту в м. Києві на номер абонента, зазначений в рекламі, може здійснюватись автодозвін (дзвінки з інтервалом у декілька хвилин) з інформуванням рекламодавця автовідповідачем про порушення ним встановлених правила та зобов'язання негайного усунення порушення” [4].

Призначення програмного продукту полягає в створенні заявки на ліквідацію реклами, відображенні мапи з відміченими на ній правопорушниками, моніторингу статусу створених заявок. Програмне забезпечення необхідне для вирішення наступного спектру завдань:

- робити фотографію оголошення;
- встановлювати місцезнаходження оголошення;
- редагувати інформацію про заявку;
- вносити в базу даних нові об’єкти порушення або видаляти ті, які вже були додані до бази;
- надавати інформацію про сервіс;
- шукати на карті існуючі за номером телефону;
- шукати на карті існуючі за номером оголошення.

Крім того, програма має володіти інтуїтивно зрозумілим інтерфейсом і дизайном, що забезпечує комфортну роботу з додатком. Основні завдання розробки відповідного програмного забезпечення:

- аналіз існуючих шаблонів проектувань при розробці застосувань;
- вибір найбільш відповідного патерну;
- вибір засобів розробки програмного забезпечення;
- налаштування коректної роботи з сервером;
- реалізація додатку під платформу iOS;
- проведення тестування.

Для забезпечення взаємодії бажаючих прибрати незаконно розміщене оголошення і супервізора, який перевірятиме чи дійсно номер телефону відповідає інформації поданій за оголошенням, система повинна включати в собі додатки для активістів та робітників комунальних служб під різні платформи, сервер системи і додаток для супервізорів. В рамках даної роботи було реалізовано мобільний додаток під платформу iOS.

# 2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ШАБЛОНІВ ПРОЕКТУВАННЯ ПРИ РОЗРОБЦІ ЗАСТОСУВАНЬ

Перш за все при розробці мобільного додатку потрібно обрати шаблон проектування або так званий патерн. На відміну від готових функцій чи бібліотек, патерн не можна імпортувати в програму копіюванням коду. Патерн являє собою загальний принцип вирішення певної проблеми, який треба підлаштовувати для потреб тієї чи іншої програми. Зазвичай шаблон не є закінченим зразком, а це лише приклад розв'язання задачі, який використовують в різних умовах.

Патерн — це високорівневий опис рішення, реалізація якого може бути унікальною для тої, чи іншої програми. Патерни відрізняються між собою за рівнем своєї деталізації, за рівнем складності та охоплення проектованої системи. Найбільш прості та низькорівневі патерни — ідіоми. Їх вважають не дуже універсальними, оскільки вони мають сенс в рамках лише однієї мови програмування. Натомість найбільш універсальні патерни — архітектурні, адже їх можна реалізувати за допомогою практично будь-якої мови програмування. Вони широко використовуються для проектування всієї програми, а не лише окремих її компонентів.

Об'єктно-орієнтовані шаблони визначають взаємодії та відносини між об'єктами або класами, без визначення того, які кінцеві об’єкти чи класи будуть використовуватися в додатках. За допомогою архітектурних патернів користувацький інтерфейс, модель даних додатку, взаємодія з користувачем розділені на три окремі модулі так, щоб будь-які зміни в одному з компонентів мінімально впливали на інші [5].

Для порівняння розглянемо наступні архітектурні патерни: MVC, MVP, MVVM, VIPER.

## 2.1. Model-View-Controller

У фундаментальному патерні MVC (Model-View-Controller) є три основні складові: View (представлення, відображення, користувацький інтерфейс), Model (модель, бізнес логіка) і Controller (контролер, вміщує в собі логіку за зміну моделі при взаємодії користувача з представленням, реалізує Use Case). Стандартну схему архітектури «Модель-Представлення-Контролер» подано на Рисунку 2.1.

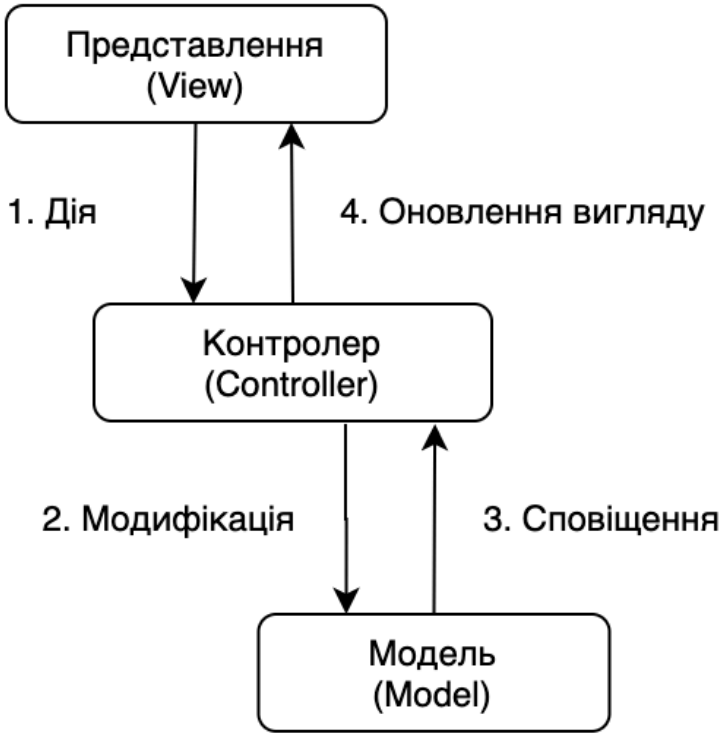



Рисунок 2.1 - Схема архітектури MVC

MVC знайшов застосування в розробці багатьох програм, дав розвиток новим технологіям і кожного дня полегшує життя програмістам. Основна ідея цього шаблону полягає в тому, що і представлення, і контролер залежать від моделі, але модель від цих двох компонентів зовсім не залежить. Ця особливість і дозволяє розробляти, змінювати і тестувати бізнес логіку, нічого не знаючи про представлення та контролери. В свою чергу, контролер не повинен нічого знати про відображення і для одного представлення можна перемикати контролери. Так само один і той же контролер

може використовуватися для різного інтерфейсу (наприклад, контролер, який буде виконуватися, може залежати від статусу користувача, який увійшов в систему). Користувач бачить відображення, взаємодіє з ним, ці дії частина представлення перенаправляє контролеру і підписується на зміни даних моделі. Контролер в свою чергу виконує певні функції з моделлю даних, відображення отримує останній стан моделі і оновлює представлення користувачеві.

Головною метою архітектурного шаблону MVC є створення гнучкого дизайну програмного забезпечення, який в подальшому полегшує розширення та зміни програми, а також надає можливість повторно використовувати окремі компоненти. Окрім того використання даного патерну у великих системах сприятиме впорядкуванню їх структури та робитиме ці системи зрозумілішими за рахунок зменшення складності [6].

Розглянемо більш детально кожен елемент архітектурного патерну MVC.

Модель - центральний компонент MVC. Відповідальний за поведінку додатку, яка не залежить від користувацького інтерфейсу. Модель вміщує в собі функції керування, збереження даних та їх структури, керування логікою додатку, відповідає за алгоритми, розрахунки, внутрішні устрої системи.

Представлення - модуль, відповідальний за виведення інформації, яка надходить із системи або в систему, відображення даних моделі користувачу, внаслідок зміни моделі. Для однієї і тієї ж інформації можуть співіснувати одночасно декілька представлень (виглядів). Представлення не відповідає за обробку введених даних користувача.

Контролер - модуль керування введеням і виведенням даних, забезпечує “зв’язок” між користувачем та системою. Контролер передає дані від користувача до системи і навпаки. Отримуючи вхідні дані, він перекладає їх на команди для інших частин системи. Для реалізації необхідної функції використовує дані з компонентів вигляду та моделі [7]. Контролер відповідальний за передану інформацію в систему, адже на основі введених даних саме він повинен визначити:

виконати запит на повторне введення даних та виводити повідомлення про помилку (модуль представлення має оновити вигляд сторінки, виведену інформацію);

- передати введені дані в модель системи.

### 2.1.1. Недоліки патерну MVC

1. Необхідно використовувати більшу кількість ресурсів. Складність обумовлена тим, що всі три фундаментальні компоненти  абсолютно незалежні між собою та взаємодіють виключно шляхом передачі даних. Контролер завжди повинен завантажувати (і за необхідності створювати) всі можливі комбінації змінних і передавати їх в модель. Компонент моделі, в свою чергу, має завантажити всі дані для візуалізації і передати їх у представлення. Наприклад, в модульному методі, модуль має можливість безпосередньо візуалізувати дані і обробляти змінні без завантаження їх в окремі відділи пам'яті.

2. Ускладнений алгоритм поділу програми на модулі. У концепції MVC наявність трьох блоків (Модель, Представлення, Контролер) задано жорстко. Відповідно кожен функціональний компонент має складатися з трьох секцій, що в свою чергу, дещо ускладнює та навантажує архітектуру функціональних модулів програми.

3. Погіршений механізм розширення функціоналу. Проблема досить схожа на вищезазначену. Недостатньо просто написати функціональний модуль і застосувати його в одному місці програми. Кожна функціональний частина повинна складатися з трьох модулів, і кожен з цих модулів повинен бути підключеним у відповідному блоці.

### 2.1.2. Переваги патерну MVC

1. Єдина концепція системи. Безсумнівною перевагою MVC є єдина глобальна архітектура програми. Навіть в складних системах, розробники (як ті, які від самого початку працювали над розробкою системи, так і ті, які нещодавно приєдналися) можуть легко зорієнтуватися в програмних блоках. Так, наприклад, якщо виникла помилка в логіці обробки даних (компонент моделі), розробник одразу відкидає обидва інші компоненти програми (контролер і представлення) і займається поглибленим вивченням третього компоненту (моделі). Дана властивість значно спрощує локалізація проблеми і витрати часу на пошук необхідного для виправлення компоненту зменшуються.

## 2.2. Model-View-Presenter

Model-View-Presenter (MVP) - архітектурний патерн, який походить від MVC, в основному використовується задля побудови призначеного для користувача інтерфейсу. Він поділяє користувацький інтерфейс (відображення) та методи обробки подій у різні класи. Модуль пред’явлення (Presenter) в даному патерні відповідає за функціональність посередника (аналогічно до контролера в шаблоні MVC) і додатково виступає відповідальним за управління подіями, які були призначені для користувацького інтерфейсу аналогічно тому, як в інших архітектурних патернах зазвичай відповідальний модуль представлення [8]. Стандартну схему архітектури «Модель-Представлення-Пред’явник» подано на Рисунку 2.2.

Модель - клас для визначення даних, над якими проводитимуться певні дії або які відображатимуться у інтерфейсі користувача.

Представлення - клас для керування елементами на сторінці, перенаправляє події до класу пред’явника.

Пред'явник  - клас, який містить логіку реакцій на події, відповідає за функції, що викликатимуться внаслідок цих реакцій, оновлює модель (даних з програми, бізнес-логіки) і, в свою чергу, виконує маніпуляції над станом представлення. Для полегшення його тестування, клас пред'явника повинен замість посилання на конкретну реалізацію мати посилання на інтерфейс класу представлення. Як наслідок, це дає можливість легко оновити діюче представлення на макет для виконання різноманітних тестів.

Якщо порівняти його з MVC, то головною відмінністю є клас пред’явника, який відповідає за логіку обробки подій, управління представленням та форматування даних. Компонент системи, який відповідає за вигляд, в даному шаблоні розширюється, адже він включає в собі дочірні класи UIViewController, Activity або Fragment і їх зв'язки з класами інших контролів. Таким чином ми "розвантажуємо"

класи аналогічні контролеру від тих обов'язків, яким вони не повинні займатися, залишаючи всередині тільки код ініціалізації представлення і анімацій.


Рисунок 2.2 - Схема архітектури MVP

Шаблон проектування MVP по-перше використовується для того, щоб чітко розподілити відповідальність між класами. Це особливо актуально, якщо над проектом працює команда розробків. Варто розуміти, що більшість людей не мислить однаково. Наприклад, код форматування даних в результаті може опинитися як в контролері, так і в елементі таблиці, так і в коді класу, що відповідає за обробку даних, які надійшли з сервера. Всю роботу щодо форматування і підготовку моделі для представлення ми переносимо в клас пред’явника, тому неоднозначності в тому, де форматувати дані, не залишається.

По-друге, неодмінним плюсом використання патерну MVP є його універсальність для Android та iOS. Адже логіка пред’явника буде універсальною і якщо розробка додатків відбувається по черзі, то адаптація на іншій платформі проходитиме передбачуваніше і швидше, оскільки код логіки в класі представлення практично однаковий [9]. Більше того, на етапі початку роботи з розробки додатку для іншої платформи вже будуть готові тест-кейси.

### 2.2.1. Недоліки патерну MVP

1. В порівнянні з використанням архітектурного патерну MVC, на реалізацію шаблону MVP витрачається більше часу, а отже швидкість розробки дещо сповільнюється.

### 2.2.2. Переваги патерну MVP

1. Справжнє розділення логіки представлення від логіки моделі, на відміну, наприклад, від MVC.

2. Чіткий поділ відповідальності між класами:

2.1. View - візуальна частина, анімації, переходи між екранами;

2.2. Presenter - форматування, реакція на події, логіка і управління представленням (View);

2.3. Model - робота із завантаженням даних через API, вилучення даних з глобальної БД і їх кешування.

3. Майже універсальна реалізація інтерфейсів на мобільних платформах для "звичайних" проектів.

4. Інтеграційне тестування, яке виконується на рівні пред’явника.

5. Розбиття завдань, поставлених задач в команді в рамках розробки одного екрану на задачу представлення (разом з класом пред’явника) і моделі.

6. Можливість представити інтерактивну демо-версію замовнику, без наявної працюючої частини бекенду. В подальшому при наданні реального сервісу, весь написаний код використовується, а вже під нього адаптується прикладний програмний інтерфейс.

7. Розширення і додавання нового функціоналу не викликатиме труднощів у нових учасників розробки проекту, через набір простих правил розміщення коду в суворо визначених модулях. Ця характерність MVP дозволяє легко зорієнтуватися в розташуванні певного функціоналу.

## 2.3. Model-View-ViewModel

Model-View-ViewModel (MVVM) - архітектурний патерн, який дозволяє відділити логіку від візуальної частини додатку (представлення) та задає загальну архітектуру застосунку. Даний патерн було представлено головним архітектором Microsoft Azure Джоном Госсманом (John Gossman) в 2005 році в якості модифікації шаблону Presentation Model і в першу чергу був націлений на розробку додатків в Windows Presentation Foundation (WPF). В наш час даний патерн вийшов за рамки WPF і на разі широко застосовується при розробці додатків під iOS, Android платформи. Як випливає з назви, MVVM складається з трьох компонентів: моделі (Model), моделі представлення (ViewModel) і представлення (View). Стандартну схему архітектури «Model-View-ViewModel» подано на Рисунку 2.3.

MVVM використовується для розмежовування моделі та її відображення. Можливість змінювати їх незалежно одна від одної викликає дану необхідність. Наприклад, дизайнер працює над розробкою користувацького інтерфейсу, а програміст — з логікою роботи даних.

MVVM зручно використовувати замість класичного архітектурного шаблону MVC та йому подібних у тих випадках, коли на платформі, де ведеться розробка, виконується «зв'язування даних» (Binding).

Модель - компонент шаблону MVVM, який описує використовувані в додатку дані. Моделі можуть містити в собі і логіку, яка пов'язана безпосередньо з цими даними (наприклад, логіку апробації різних властивостей моделі). У той же час модель ніяк не відповідає за логіку, пов’язану із реакцією візуальних елементів управління та відтворенням даних [10].

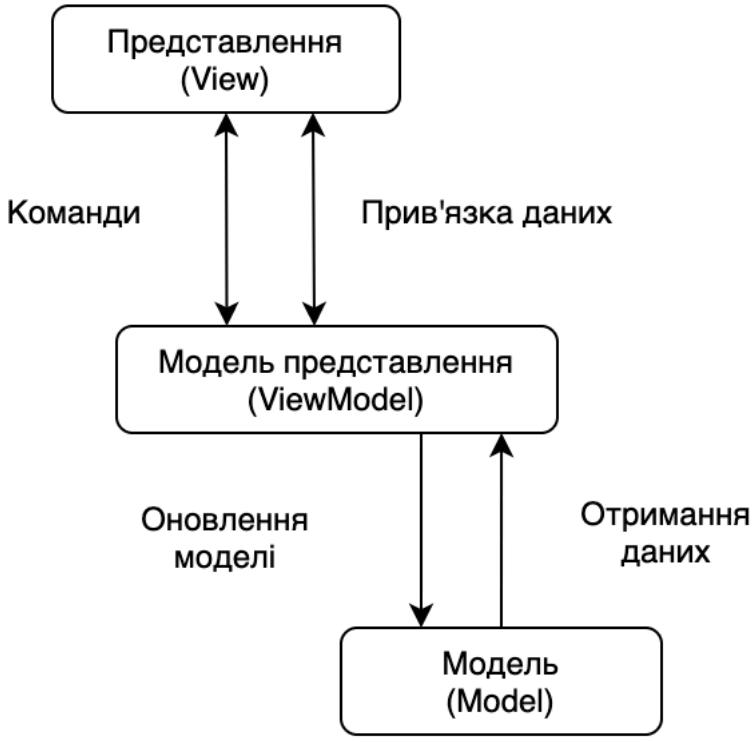



Рисунок 2.3 - Схема архітектури MVVM

Досить часто модель включає реалізацію інтерфейсів, які дозволяють передавати повідомлення в систему щодо змін властивостей моделі. Це дає можливість полегшити зв’язування з представленням, хоча пряма взаємодія між компонентами представлення та моделі відсутня.

View або представлення, як і в попередньо проаналізованих архітектурних патернах MVC, MVP, визначає візуальний користувацький інтерфейс, через який проходить взаємодія з додатком. Представлення не виконує обробку жодних подій, хоча в деяких ситуаціях і трапляються винятки, а виконує дії в основному за допомогою команд. В обов’язки представлення входить відображення даних, які надходять від компоненту моделі. Однак, представлення не може впливати безпосередньо на модель, вважається, що представлення має доступ лише до зчитування даних.

Модель представлення є своєрідною абстракцією представлення та надає обгортку даних з сегменту моделі, які підлягають сполученню. Тобто, вона включає в собі модель, яка перетворена до представлення, а також містить у собі всі команди, якими може користуватися представлення, щоб впливати на модель.

Ознаки моделі представлення:

– двосторонній зв’язок з представленням;

– модель представлення – це абстракція представлення. Зазвичай це означає, що властивості представлення співпадають з властивостями сегменту моделі представлення або моделі;

– відсутнє посилання на інтерфейс представлення (UIView). Завдяки механізму поєднання даних (Bindings), зміна стану представлення автоматично змінює модель представлення і навпаки;

– одна модель представлення пов’язана тільки з одним представленням.

При використанні цього шаблону клас представлення не відповідає за реалізацію відповідного інтерфейсу. Представлення повинно містити посилання на джерело даних, яким у даному випадку виступає View-модель.

Елементи представлення пов'язані з відповідними властивостями і подіями моделі представлення. У свою чергу, модель представлення реалізує спеціальний вигляд, який використовується для автоматичного оновлення кожного елементу інтерфейсу.

### 2.3.1. Недоліки патерну MVVM

1. Для невеликих проектів цей підхід може бути не виправданим.

2. Якщо логіка зв’язування (Binding) даних занадто складна - налагоджувати додаток буде важче.

### 2.3.2. Переваги патерну MVVM

1. Гнучкість розробки. Цей підхід підвищує злагодженість роботи в команді, оскільки в той час, поки один член команди працює над стилізацією і компонуванням екрану - інший, має можливість одночасно працювати над описанням логіки отримання даних та їх обробки;

2. Тестування. Така структура спрощує процес написання тестів. Також, в більшості випадків відпадає потреба в автоматизованому UI-тестуванні, оскільки сам компонент представлення моделі можна покрити тестами;

3. Розмежування логіки. За рахунок більшого розмежування код стає більш гнучким і простим у підтримці, не кажучи вже про властивості його читабельності. Кожен модуль відповідає за свою конкретну функцію і тільки.

## 2.4. VIPER

VIPER - архітектурний патерн, який є розширенням MVC. У ньому більш детально розділяються відповідальності між шарами і привноситься поняття модульності. Шар View збігається в обох архітектурах, Interactor є активною моделлю і переймає обов'язки моделі і контролера, Presenter - шар, який відповідає за підготовку даних до відображенню і передачу для користувача подій до рівня Interactor (в MVC за виконання цих задач був відповідальний контролер), Entity - пасивна модель, Router (часто також званий Wireframe) - шар, який відповідає за навігацію. VIPER вирішує проблеми MVC патерну з приводу дотримання SOLID принципів, але в той же час значно збільшується кількість класів, інтерфейсів і як наслідок - коду. Стандартну схему взаємодії компонентів архітектури «VIPER» подано на Рисунку 2.4.

Взаємодія між компонентами відбувається через інтерфейси (у класі, який належить до будь-якого з VIPER-компонентів окрім Entity існує два протоколи - вводу і виводу), тому один VIPER-модуль містить як мінімум п'ять класів і вісім інтерфейсів. Звідси випливає головний недолік архітектурного шаблону VIPER- велика кількість шаблонного коду, з яким частково можна боротися за допомогою бібліотечних кодогенерацій. Однією з відомих та популярних утиліт для генерації Swift-коду є Generamba, розроблена iOS-командою компанії Rambler, яка створює порожній VIPER-модуль з усіма класами і інтерфейсами та автоматично додає його в файлову систему проекту і в файл проекту .pbxproj.

Представлення (View) - клас, у якому є весь код відповідає за те, щоб показати користувачеві інтерфейс програми та забезпечити взаємодію. Після отримання сигнали про дії користувача, цей клас передає повідомлення класу пред’явника (Presenter).

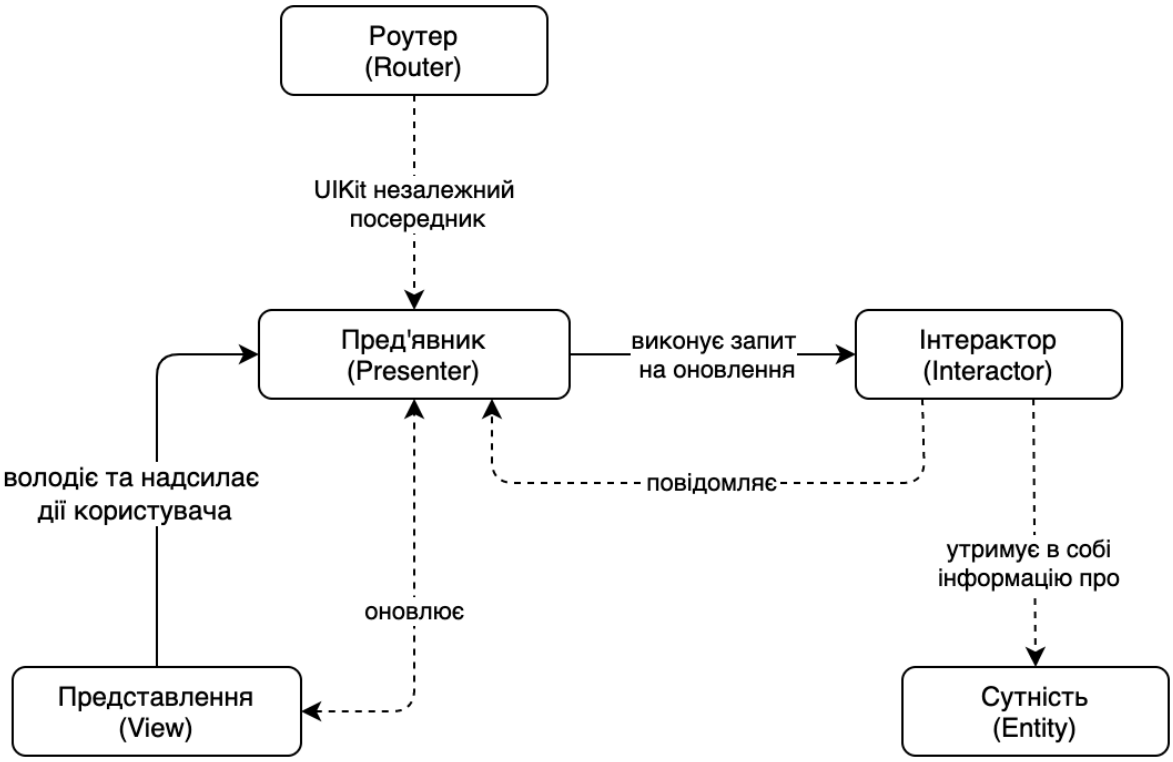



Рисунок 2.4 - Схема взаємодії компонентів архітектури VIPER

Пред’явник (Presenter) - ядро модуля. Він отримує відповідь користувача від представлення (View) та викликає відповідну функцію подальших дій. Це єдиний клас в модулі, який призначений для спілкування з усіма іншими компонентами. Він викликає компоненти маршрутизатора (Router) для встановлення зв’язків, Інтерактор для отримання даних (дані з мережі або локальні дані), та представлення (View) для оновлення інтерфейсу користувача. Предявник зберігає інформацію про стан всього модуля, виступає в якості компоненту введеня, виведення даних модуля і не просто передає команди компоненту представлення в інтерактор, а ще й приймає рішення, в якому випадку що саме треба зробити.

Інтерактор (Interactor) - має бізнес-логіку програми. В основному відповідальний за API-виклики, отримує дані з серверу, а також за здійснення викликів даних, але не обов'язково від себе.

Маршрутизатор (Router) - компонент моделі, який обробляє навігацію від одного екрану до іншого відповідно до тієї каркасної моделі, яка була створена проек-

тувальником взаємодії. Тобто маршрутизатор несе відповідальність за навігацію між модулями.

Сутність (Entity) - містить прості класи моделей, які використовує інтерактор і які не несуть в собі ніякої бізнес-логіки.

VIPER-додаток містить набір класів (які є ядром додатку) і набір модулів, кожен з яких є відокремленою групою класів. Модуль з'єднується з ядром за допомогою класів - ModuleBuilder [11]. Таким чином, логіка додатку розподіляється по модулях, отже, зменшується зв'язаність функціоналу, спрощується додавання або видалення модулів.

### 2.4.1. Недоліки патерну VIPER

1. Різке збільшення кількості класів в проекті, викликає складності при створенні нового модуля.
2. Деякі з принципів не відповідають безпосередньо підходам Apple.
3. Відсутність у відкритому доступі набору конкретних рекомендацій, кращих рішень і прикладів складних додатків.

### 2.4.2. Переваги патерну VIPER

1. Робить зручнішим тестування шару пред’явника додатків.
2. Повністю незалежні один від одного модулі - це дозволяє розробляти їх окремо та перевикористовувати як в одному додатку, так і в декількох.
3. Передача проекту іншим розробникам, або прихід нового, дається набагато простіше, оскільки загальні підходи до архітектури визначені заздалегіть.

## 2.5. Опис існуючих рішень

Додаток “StopSpam” - єдиний аналог представлений в магазині App Store [12]. При запуску додатку, на головному екрані з заголовком “Мапа” відображено карту, камера карти встановлена на поточне місцезнаходження користувача. На головному

екрані присутнє поле для вводу зі знаком пошуку, при введені даних в якому нічого не відбувається. В меню додатку є наступні пункти:

- вихід;
- мапа;
- оголошення;
- історія дзвінків;
- створити.

Можливості авторизуватися або зареєструватися в додатку немає, оскільки при першому ж запуску застосунку є тільки кнопка “Вихід”, яка показує інформаційне вікно з запитом на вихід.

Пункт меню “Мапа” повертає на головну активність. Як зазначалося раніше, даний екран відображає карту та мітку з поточним місцезнаходженням.

Вкладка “Оголошення” та “Історія дзвінків” відкриває активність з полем для введення запиту на пошук. При спробі знайти будь-яке оголошення або дзвінок, екран залишається незмінним. Результатів пошуку або ж повідомлення про їх відсутність не виведено.

Останнім надано пункт “Створити”. З відповідною іконкою є кнопка на головному екрані.  При натисненні на будь-який з цих двох елементів, додаток дає збій та вилітає (Crash).

## 2.6. Висновки до розділу

У даному розділі було розглянуто найбільш росповсюджені архітектурні шаблони для мобільних платформ: MVC, MVP, MVVM та VIPER. Всі зазначені патерни реалізують розділення відповідності між класами, що забезпечує високе перевикористання та гнучкість коду. Проте, при масштабуванні додадку абстракцій MVC недостатньо для того, щоб розділити функціонал між класами і повністю дотримувати-

ся принципів SOLID [13]. Використання таких шаблонів, як MVP, MVVM або VIPER, дозволяє вирішити дану проблему. В той же час, патерни MVVM, VIPER в невеликих проектах, або в проектах де немає перевикористання коду є невиправданими. В MVVM занадто складна логіка зв'язування даних (Binding), а VIPER передбачає велику кількість класів і шаблонного коду. Тому, в результаті проведених досліджень, для реалізації додатку було обрано патерн MVP.

Огляд аналогів показав, що додаток в першу чергу має бути якісним, функціонуючим, надавати можливість авторизуватися.

Користувачу повинно бути зрозуміло, чому при запиті на пошук оголошень немає жодного результату, якщо список заявок виявився пустим. Тобто, необхідно передбачити такий сценарій та виводити відповідне повідомлення.

При розробці додатку важливо запобігти можливих "вильотів" (Crash), диференціювати поведінку додатку при недоступному сервері системи, випдкавому одиничному збої.

Крім того, додаток повинен забезпечувати інтуїтивно зрозумілу роботу з системою.

# 3. МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

При створенні мобільного застосунку використовувались найактуальніші технології та методики, які використовуються у ІТ галузі.

## 3.1. Аналіз вимог до програмного забезпечення

Систему можна розділити на два типи:

а) Сервіс для прийняття та обробки заявок;

б) Клієнтські застосунки, які відповідатиму за надання можливості користувачам створювати заявки, спілкуватимуться з розробленим сервісом та передаватимуть необхідні дані на відправку.

### 3.1.1. Розроблення функціональних вимог

Застосунок для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами повинен відповідати наступним функціональним вимогам:

- надавати користувачу можливість авторизуватися з декількох пристроїв;
- забезпечувати синхронізацію даних на різних пристроях;
- надавати користувачу інформацію про незаконно розміщені оголошення;
- можливість створювати нову заявку (з вказанням адреси, номерів зазначених на оголошенні телефонів, фотозвітом, назвою компанії порушника, типу об’єкту, на якому розміщено рекламу, вказанням коментарів та балансоутримувача) та скасовувати її;
- створення фото або можливість вибору фото з галереї;
- надавати інформацію про сервіс;
- надсилати сповіщення користувачу у разі зміни статусу створюваної ним заявки;

− надавати можливість пошуку заявки за номером вказаних у ній телефонів або безпомередньо за номером створених заявок;

− визначати місцезнаходження користувача або давати можливість самостійно вказати положення на карті.

### 3.1.2. Розроблення нефункціональних вимог

В результаті аналізу функціональних вимог та огляду існуючих архітектурних шаблонів проектування додатків, були сформовані наступні нефункціональні вимоги:

− додаток має бути написано на мові Swift під платформу iOS;

− додаток повинен бути реалізований з використанням шаблону MVP;

− інтерфейси користувача повинні бути інтуїтивно зрозумілими, мають відповідати сучасним вимогам, які описані в Apple Human Interface Guidelines;

− графічний інтерфейс має стабільно працювати з будь-яким розрішенням екрану, починаючи з iPhone 4S.

## 3.2.Варіанти використання системи

Структурну схему варіантів використання мобільного додатку представлено на рисунку 3.1.

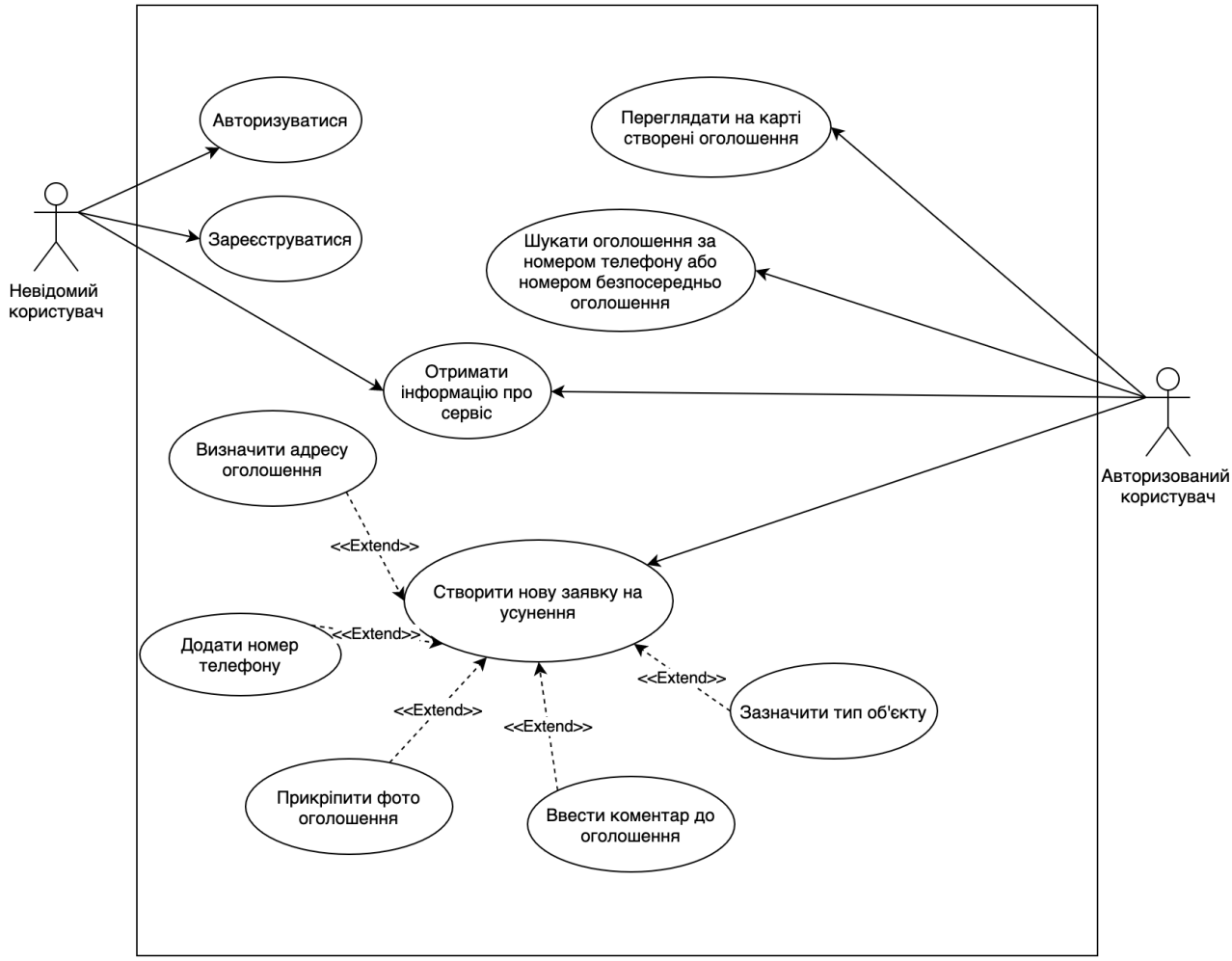



Рисунок 3.1 - Варіанти використання мобільного додатку

Невідомий користувач може зареєструватися в додатку, вказавши своє ім’я, електронну адресу, номер телефону та пароль.

Невідомий користувач може авторизуватися в додатку через номер телефону або електронну адресу.

Будь-який користувач, як невідомий, так і авторизований, може отримати інформацію про сервіс.

Авторизований користувач може переглянути на карті сворені оголошення. Тобто, обравши деяке місце на карті і натиснувши клавішу “шукати в цій області”, його увазі буде представлено місця, в яких вже створено оголошення за допомогою спеціальних відміток. У випадку, якщо оголошень занадто багато, виконується кластеризація, мітки групуються і показується загальне число оголошень в цій зоні.

Авторизований користувач може шукати оголошення, тобто отримати список оголошень, які співпадають з даними пошуку за номером телефону або за номером самого оголошення.

Авторизований користувач може створити нову заявку на усунення оголошення, обов’язково вказавши при цьому інформацію про адресу (зазначити місто, район), номери телефонів, назву компанії порушника, та прикріпивши фото оголошення. Додатково при створенні заявки можна вказати коментар до оголошення, тип об’єкту, на якому розміщено оголошення та балансоутримувач.

## 3.3.Архітектура iOS-додатку

На рисунку 3.2 подано діаграму компонентів застосунку.

Інтерфейс користувача наданий задля взаємодії з усією системою.

Компонент авторизації потрібний для визначення необхідності авторизуватися та надає можливість авторизації.

Компонент інформації про сервіс призначений для надання користувачу можливості дізнатися про закон, на підставі якого працює система та електронну адресу, за якою можна звертатися у разі виникнення запитань, скарг або пропозицій.

Компонент оголошень надає інформацію про заявки і відповідає за пошук оголошення за номером вказаного в ній телефону або ж за номером самої заявки.

Компонент створення заявки дозволяє користувачу створити нову заявку про незаконно розміщене оголошення з вказанням необхідних даних.

Компонент оновлення даних оновлює інформацію про об’єкти в базі даних застосунку, якщо пристрій авторизовано, сервер системи доступний і пройшов деякий час від попереднього оновлення даних.

HTTP клієнт відповідає за взаємодію з сервером системи через звернення до REST-інтерфейсу серверу, реалізує методи запитів на сервер, конфігураційні дані для з’єднання з сервером.

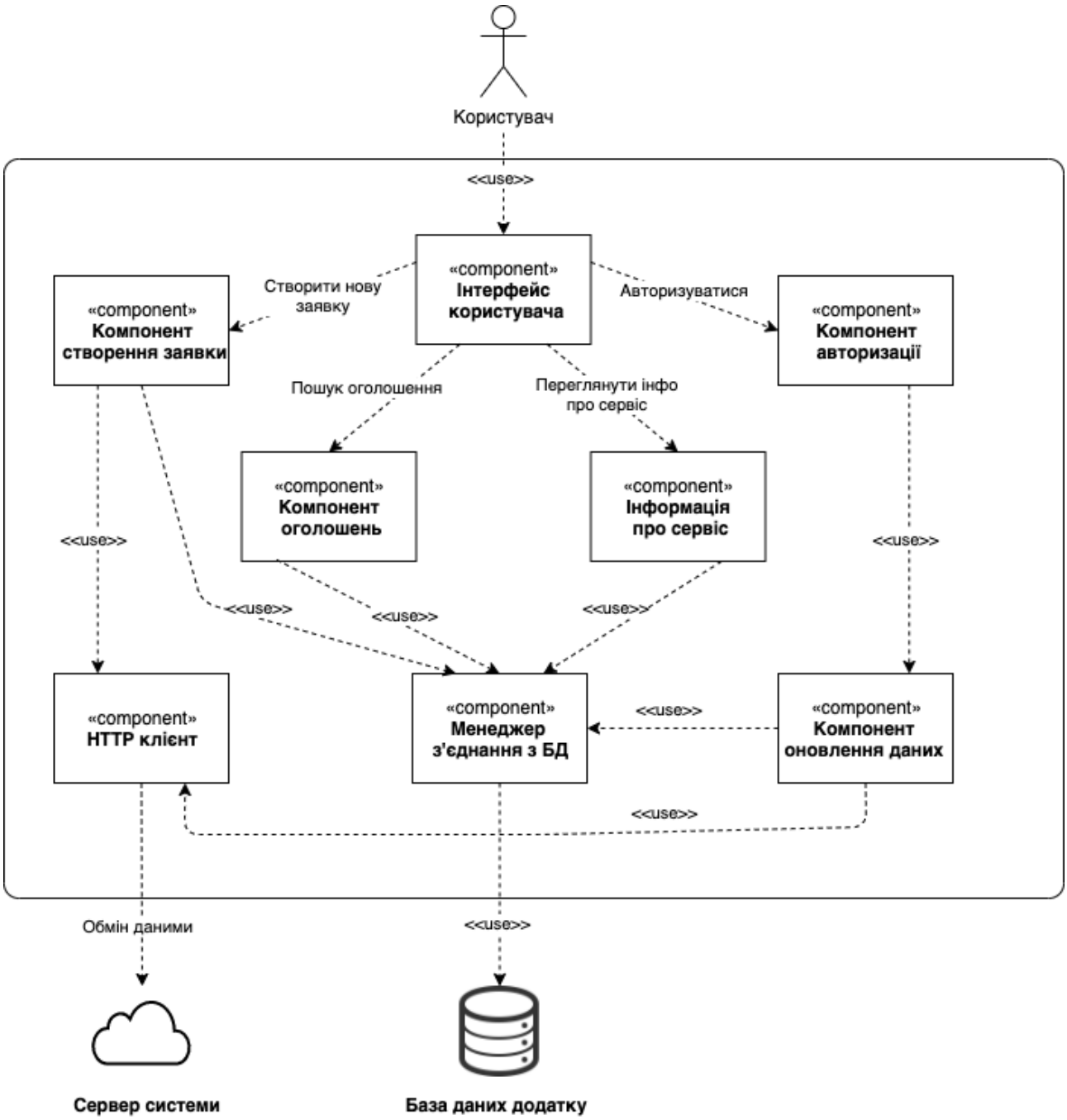



Рисунок 3.2 - Компоненти iOS-додатку

Менеджер з’єднання з базою даних містить в собі методи ініціалізації бази даних, а також конфігураційні дані для з’єднання з базою даних.

## 3.4.Схема бази даних

Для збереження інформації про користувача, токен, оголошення та про сервіс було розроблено базу даних, схема якої представлена на рисунку 3.3

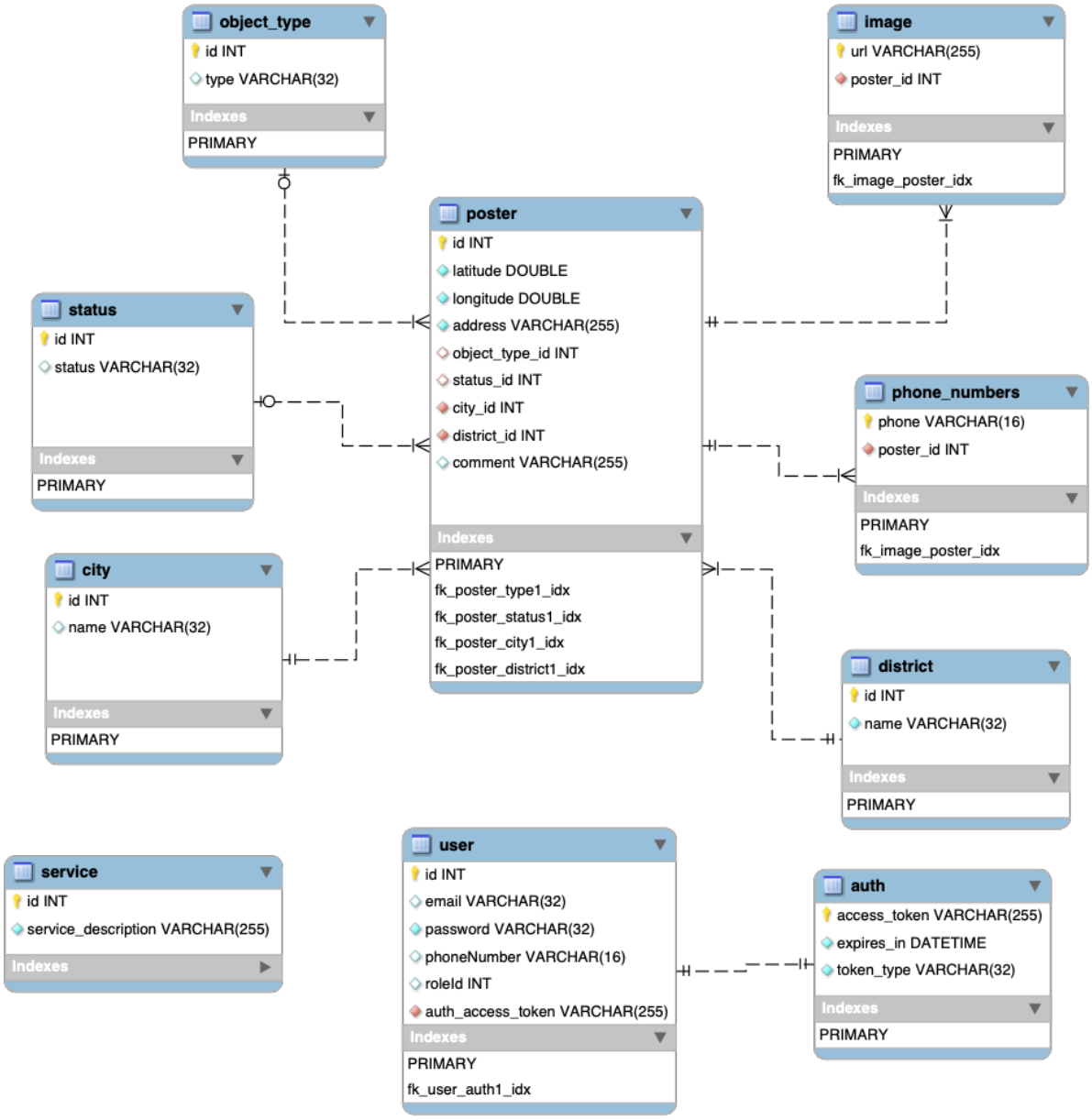



Рисунок 3.3 - Схема бази даних мобільного додатку

Далі у таблицях 3.1 - 3.10 наводиться детальний опис кожної з таблиць бази даних. В кожному описі надано 5 колонок, де “Ключ” має три типи даних. Позна-

чення FK (foreign key) вказує, що поле є зовнішнім ключем, PK (primary key) - первинним ключем, а також дана помітка може бути відсутня.

Таблиця 3.1 — Опис таблиці “Poster”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор оголошення | Int | |
| | latitude | Широта місцезнаходження | Double | |
| | longitude | Довгота місцезнаходження | Double | |
| | address | Адреса оголошення | Varchar | 255 |
| FK | object_type_id | Ідентифікатор типу об’єкту | Int | |
| FK | status_id | Ідентифікатор статусу оголошення | Int | |
| FK | city_id | Ідентифікатор міста | Int | |
| FK | district_id | Ідентифікатор району | Int | |
| | comment | Коментар до оголошення | Varchar | 255 |

Таблиця 3.2 - Опис таблиці “Object_type”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор об’єкту | Int | |
| | type | Назва типу об’єкту | Varchar | 32 |

Таблиця 3.3 - Опис таблиці “Status”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор статусу | Int | |
| | status | Назва статусу | Varchar | 32 |

Таблиця 3.4 - Опис таблиці “City”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор міста | Int | |
| | name | Назва міста | Varchar | 32 |

Таблиця 3.5 - Опис таблиці “Image”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | url | Посилання до прикріпленого зображення | Varchar | 255 |
| FK | poster_id | Ідентифікатор оголошення | Int | |

Таблиця 3.6 - Опис таблиці “Phone_numbers”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | phone | Номер телефону | Varchar | 16 |
| FK | poster_id | Ідентифікатор оголошення | Int | |

Таблиця 3.7 - Опис таблиці “District”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор району | Int | |
| | name | Назва району | Varchar | 32 |

Таблиця 3.8 - Опис таблиці “Service”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор сервісу | Int | |
| | service_description | Опис додаткової інформації, яка надається користувачу | Varchar | 255 |

Таблиця 3.9 - Опис таблиці “User”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | id | Ідентифікатор користувача | Int | |
| | email | Електронна адреса користувача | Varchar | 32 |
| | password | Пароль користувача | Varchar | 32 |
| | phoneNumber | Номер телефону | Varchar | 16 |
| | roleId | Ідентифікатор ролі користувача | Int | |
| FK | auth_access_token | Токен | Varchar | 255 |

Таблиця 3.10 - Опис таблиці “Auth”

| Ключ | Назва поля | Опис | Тип | Розмір |
|---|---|---|---|---|
| PK | access_token | Токен | Varchar | 255 |
| | expires_in | Дата завершення дії токену | Datetime | |
| | token_type | Тип токену | Varchar | 32 |

## 3.5.Висновки до розділу

У ході написання третього розділу було проведено аналіз вимог до застосунку, визначено функціональні та нефункціональні вимоги до програмного забезпечення, розроблено архітектуру програмного продукту, визначено варіанти використання додатку. Було спроектовано структуру бази даних мобільного додатку під платформу iOS, детально її описано.

# 4. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1. Проектування програмного забезпечення

Для створення клієнтської частини системи для платформи iOS використовувалась IDE Xcode та багатопарадигмова мова програмування Swift. Для локального збереження даних використовується база даних, яка створена за допомогою вбудованих сервісів UserDefaults та СУБД SQLite, яка підтримується платформою iOS.

Взаємодія з сервером, обробка даних з серверу виконується за допомогою імплементації широковикористовуваного фреймворку Alamofire. Alamofire - обгортка над URLSession, яка значно спрощує роботу з сервером.

Для роботи з картою використовувалися Maps SDK, яка дозволяє додавати карти на основі даних Google. SDK автоматично обробляє доступ до серверів Google Maps, оновлює відображення карти та відповідь на жести користувачів, такі як кліки, перетягування, масштабування. Google Maps SDK дозволяє додати маркери, наземні накладки, інформаційні вікна, кнопку пошуку місцезнаходження користувача та переміщення камери на це положення [14]. Всі ці об’єкти дають додаткову інформацію та дозволяють користувачу взаємодіяти із картою.

В проекті було розроблено такі класи:

- CreateAnAdViewController;
- LoginViewController;
- RegistrationViewController;
- MapViewController;
- SearchViewController;
- APIService;
- NetworkAPI;
- UserLocationManager;
- UserDefaultsManager.

**CreateAnAdViewController.** Клас для створення заявки про незаконно розміщене оголошення.

У класі використовуються такі поля:

- allPhones - номери зазначених телефонів;
- violationImageView - прикріплене фото оголошення;
- intruderTextField - назва компанії порушника;
- latitude - широта місцезнаходження:
- longitude - довгота місцезнаходження;
- address - текстова адреса місцезнаходження;
- district - район, де знаходиться оголошення;
- objectType - тип об’єкту, на якому розміщено рекламу;
- balance - балансоутримувач;
- comment - коментар до оголошення.

А також у класі є наступні методи:

- addNumber(_ sender: UIButton) - надає можливість вказати більше ніж 1 номер телефону, але менше 5. Для цього у користувача в списку номерів з’являється ще одне поле для вводу, розширюється стек номерів телефону;
- presentAlert(withTitle title: String) - у разі виклику методу виводить вікно сповіщення з текстом помилки/попередження, кнопкою можливості перейти до налаштувань телефону та кнопкою закриття сповіщення.
- violationPhotoTapped(_ sender: UIButton) - перевіряє доступ до галереї та у разі наданих прав дає можливість вибрати фото з галереї;
- createPoster(_ sender: UIButton) - “підтягує” всі введені дані, форматує їх та надсилає POST запит з введеними параметрами на створення заявки.

**LoginViewController.** Клас, який відповідає за авторизацію користувача, перевірку коректності введених даних.

У класі наявні наступні параметри:

- phoneNumber - номер телефону;
- email - електронна адреса;
- password - пароль;
- profile - структура даних типу Profile для збереження інформації про кори-

стувача у разі успішної авторизації.

Також у класі описані такі методи:

- handleSendingPass() - метод обробки даних про введений номер телефону, який у разі правильності введеного номеру показує поле для вводу паролю та робить доступною для натискання кнопку “Увійти”;

- handleEnter() - робить запит на отримання доступу з введеними в додатку параметрами;

- getUser() - виконує запит на отримання даних про користувача з використанням токену, отриманого в методі handleEnter(), виконує обробку результату та запам’ятовує параметри в UserDefaultManager.

**RegistrationViewController.** Клас, який створений для реєстрації нового користувача.

Представлені змінні даного класу:

- fullName - повне ім’я користувача (ПІБ);

- email - електронна пошта користувача;

- phoneNumber - номер телефону користувача;

- password - створений пароль.

В класі реєстрації подано такі методи:

- isCorrect() - метод перевірки правильності введених даних;

- registration() - виконує запит на реєстрацію нового користувача в системі.

**MapViewController.** Клас відображення карти, додавання маркерів-міток створених оголошень на карту.

Представлені змінні даного класу:

- markersArray - масив існуючих маркерів для оголошень на карті;

- posters - масив існуючих оголошень;

- mapView - безпосереднє представлення мапи;

- searchBar - поле для пошуку необхідного повідомлення.

Даний клас відповідальний за виконання наступних методів:

- addPostersToMap() - додає на карту маркери кожного оголошення;

- plotMarker() - метод генерації маркеру (його зображення, позиції, заголовку, ідентифікаційного номеру);

- didTapMyLocationButton() - функція, яка у разі натискання на кнопку знаходження поточних геоданних користувача, переміщує камеру карти на це місце-розташування, масштабує карту до стандартного наближення та повертає кут вигляду до стандартного;

- checkLastSelectedMarker() - при натисканні на один з маркерів, змінює зображення даного маркеру.

**SearchViewController.** Клас відображення карти, додавання маркерів-міток створених оголошень на карту.

Представлені змінні даного класу:

- searchTextField - поле пошуку;

- resultPosters - оголошення, які співпадають за номером оголошення або за вказаними номерами порушників з текстом з поля пошуку.

У класі наявні наступні методи:

- setupSearchBar() - налаштування зовнішнього вигляду пошуку;

- textFieldDidChange() - метод відслідковування змін поля введеня. У разі оновлення даних пошуку, надсилає новий запит, оновлює таблицю згідно з результатами відповіді від серверу.

**APIService.** Клас, який імплементує бібліотеку Alamofire. Містить обгортку над методами даної бібліотеки, налаштовує конфігурацію, створює сесію з конфігураційними даними і за допомогою даної сесії виконує стандартні методи, такі як:

- request() - створює запит для отримання вмісту даних за вказаним посиланням, методом запиту, параметрами та заголовками;

- upload() - відповідає за запит з метаданими (наприклад, надсиланням фото).

**NetworkAPI.** Клас, який є наслідником від класу APIService, використовує породжуючий патерн Одинак (Singleton) та реалізує наступні методи:

- getCode() - виконує запит на перевірку наявності введеного номеру телефону в базі даних користувачів;

- token() - метод запиту на отримання токену користувача з введеними даними (логін, пароль);

- getUser() - отримання даних про користувача;

- getPhoto() - отримати фото;

- getPosters() - отримати масив створених оголошень;
- postPoster() - надіслати на сервер нову заявку;
- searchPoster() - пошук в базі даних серверу необхідної заявки.

**UserLocationManager.** Клас, який імпортує стандартну бібліотеку від Apple CoreLocation, використовує породжуючий патерн Одинак (Singleton).

В класі надані наступні параметри:

- locationManager -об’єкт, який є точкою входу до служби локації;
- currentLocation - поточне місцезнаходження користувача.

Даний клас відповідальний за такі методи:

- requestAccess() - перевіряє доступ до геоданих користувача і в залежності від отриманих даних виконує один із сценаріїв:

а) якщо користувач надав дозвіл на використовування даних, то продовжує роботу програми;

б) якщо ж в доступі було відмовлено, викликає вікно з перенаправленням на налаштування програми та відповідним текстовим повідомленням;

- startUpdatingLocation() - повідомляє locationManager про необхідність оновлювати геоданні;
- stopUpdatingLocation() - повідомляє locationManager про необхідність припинити оновлювати геоданні;
- getCurrentLocation() - повертає дані про змінну currentLocation.

**UserDefaultsManager.** Клас, який реалізує “гетери” і “сетери” для параметрів, які використовуються в додатку.

В класі зібрані наступні поля:

- profile - інформація про авторизованого користувача;
- profileImage - фото користувача;

А також один метод:

- clearProfileInfo() - метод, який очищує дані про авторизованого користувача у разі виходу з профілю.

## 4.2.Підтримка різних версій платформи iOS

Для будь-якого iOS-проекту необхідно зазначати мінімальну версію платформи iOS, яку буде підтримувати розроблюваний додаток. Чим менша версія, тим більша кількість пристроїв буде сумісна з застосунком, але буде обмежена робота з деякими можливостями API більш сучасних версій платформи [15].

Згідно з офіційними даними поданими на сайті Apple [16], на момент 27 січня 2020 року відсоток девайсів, представлених за останні чотири роки та на які встановлена 13 версія iOS складає 77%, 12 версія - 17%, більш пізні версії - 6%. Тому в якості мінімально підтримуваної версії було обрано iOS 12.

## 4.3.Реалізація користувацького інтерфейсу

Дизайн додатку був розроблений з виконанням вимог та принципів представлених в Apple Human Interface Guidelines. HIG (Human Interface Guidelines) - комплексна концепція, яка містить рекомендації для створення логічних, візуально інтуїтивних інтерфейсів, врегульованих розташувань елементів та тимчасових файлів [17].

Оскільки для реалізації додатку було обрано архітектурний патерн MVP, модуль вигляду було розбито на “підекрани” в XML-файли з розширенням .storyboard. Усі повторювані елементи вигляду, такі як комірка таблиці, було винесено в окремі файли з розширенням .xib. Таке розподілення допомагає розділити задачі та роботу над підзадачами з мінімальною кількістю виникнення конфліктів при злитті (merge), а в випадку такого конфлікту - спрощує їх вирішення. Для кожного елементу вигляду створено окремі класи-наслідники від UIView, де зібрано методи налаштування зовнішнього вигляду, ініціалізації, перемалювання та інші методи, які допомагають зробити один вигляд універсальним, з можливістю подальшого перевикористання. В проекті інтерфейс користувача має такі модулі вигляду:

- LaunchScreen - початковий екран або екран запуску, який з’являється при

відкритті додатку;

- Map - екран відображення карти;
- Login - екран для авторизації;
- Menu - бокове меню, яке з’являється при натисканні на кнопку “бургер”, або свайпі з крайнього лівого положення до середини екрану у випадку, якщо користувач не авторизований;
- MenuInspectors - бокове меню, яке з’являється при натисканні на кнопку “бургер”, або свайпі з крайнього лівого положення до середини екрану у випадку, якщо користувач авторизований;
- CreateAnAd - екран з створенням нової заявки.

## 4.4.Створення заявки на усунення оголошення

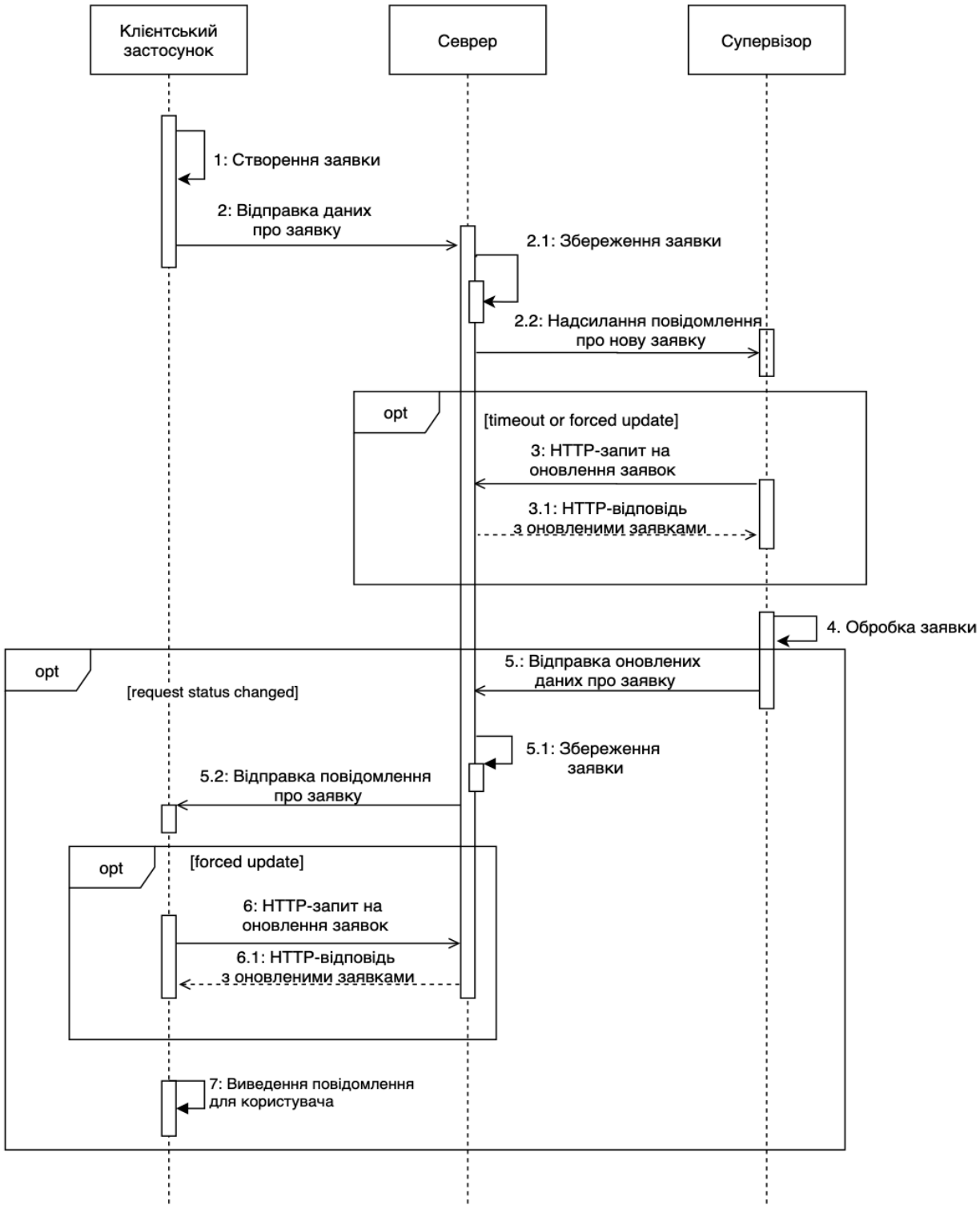
На рисунку 4.1 через взаємодію компонентів системи зображено процес створення заявки на усунення оголошення.

Як тільки користувач створює нову заявку на усунення незаконно розміщеного оголошення, додаток відправляє дані про заявку на сервер. Сервер зберігає отримані дані в базу даних і відправляє супервізору повідомлення про нову заявку та прикріплену інформацію.

Робота супервізора полягає в обробці отриманих заявок. Супервізор телефонує на вказаний номер та переконується в тому, що він дійсно належить порушнику, діяльність якого відповідає вказаній на рекламі та він дійсно розміщував оголошення з заявки. Така процедура унеможливлює випадки шахрайства. Далі на номер спрацьовує система автодозвону згідно з чинним законодавством.

Якщо в результаті обробки супервізором заявки, її статус змінився, дані на сервері системи оновлюються. Сервер зберігає дані і відправляє повідомлення з інформацією про заявку користувача. Додаток користувача виводить push-нотифікацію з інформацією про зміну статусу заявки.

Рисунок 4.1 - Діаграма послідовності при створенні заявки




## 4.5.Безпека даних

Паролі авторизації користувачів не зберігаються в системі у відкритому доступі. Для шифрування даних було обрано стандарт кодування Base64. Метод кодування Base64 відноситься до групи binary-to-text encoding алгоритмів, які перетворюють

набір байтів в ASCII формат. В базі даних зберігаються результати кодування сконкатинованих через двокрапку логіну та паролю.

У разі успішної аутентифікації клієнтського додатку, сервер надсилає сгенерований JWT (JSON Web Token), який в подальшому використовується для авторизації застосунку. JWT - це відкритий стандарт для забезпечення безпеки передачі пакетів між сторонами [18]. Токен передається в закодованому вигляді і складається з трьох частин, кожна з яких відділена крапкою.

## 4.6.Висновки до розділу

У ході написання четвертого розділу було обрано мінімально підтримувану клієнтським застосунком версію iOS. Розроблено користувацький інтерфейс згідно з вимогами Apple Human Interface Guidelines. Представлено діаграму послідовності при створенні заявки про незаконне розміщення реклами. Спроектовано програмне забезпечення для платформи iOS. При цьому детально розглянуто необхідні для коректного функціонування додатку класи, їх методи та поля. Також проаналізовано безпеку даних, описано алгоритм шифрування логіну та паролю.

# 5. ВЗАЄМОДІЯ КОРИСТУВАЧА З СИСТЕМОЮ

## 5.1.Графічний інтерфейс користувача

У даному розділі представлено основні етапи взаємодії користувача з програмним забезпеченням.

При відкритті програми користувач бачить екран запуску (Launch screen or Splash screen), який містить розроблений логотип (рисунок 5.1). Він виконує функцію “першого враження”, поки додаток відкривається і починає свою роботу.


Рисунок 5.1 - Екран запуску програми

Після того, як додаток починає свою роботу, користувач бачить стандартний

запит на використання даних про місцезнаходження (рисунок 5.2).

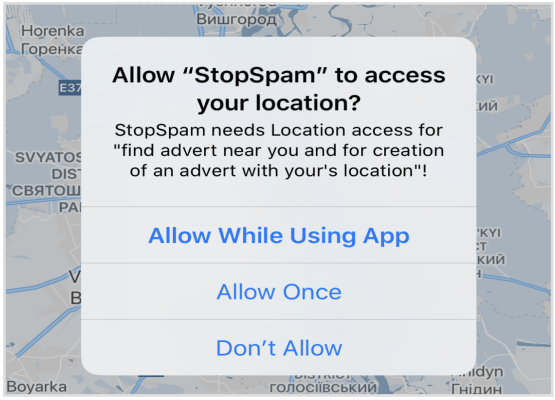



Рисунок 5.2 - Запит на доступ до GPS

Для авторизації користувачу необхідно на головному екрані (рисунок 5.3) натиснути на значок "бургер", який знаходиться в лівому верхньому кутку, або провести по екрану з середини лівого крайнього положення до центру. У результаті відкриється бокове меню (рисунок 5.4), де потрібно натиснути на кнопку "Вхід". Далі користувач повинен авторизуватися за допомогою телефону або електронної пошти (рисунок 5.5).

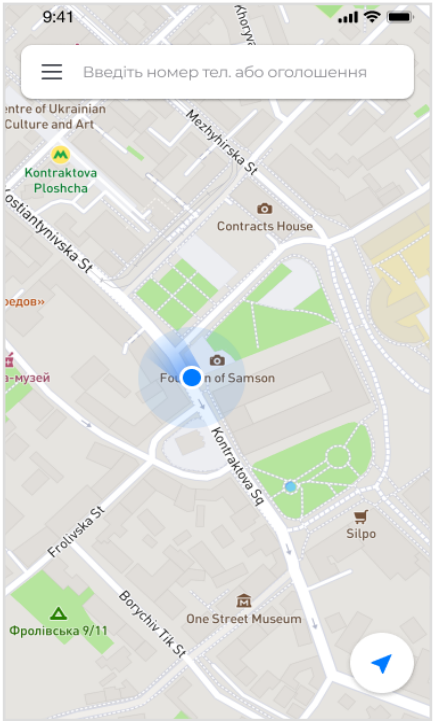



Рисунок 5.3 - Головний екран з картою неавторизованого користувача

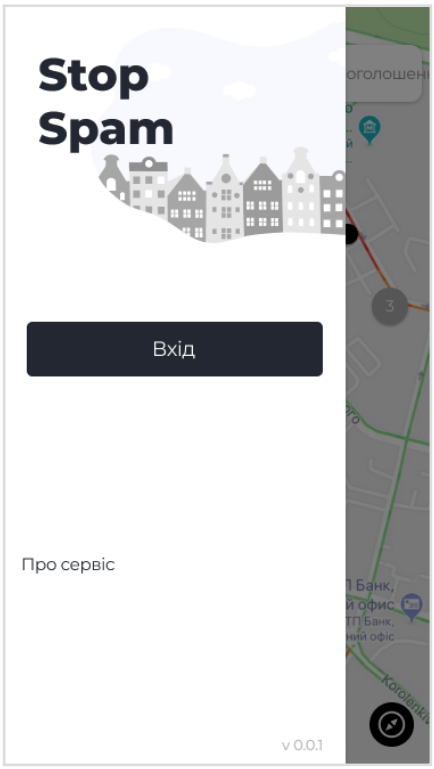



Рисунок 5.4 - Бокове меню неавторизованого користувача

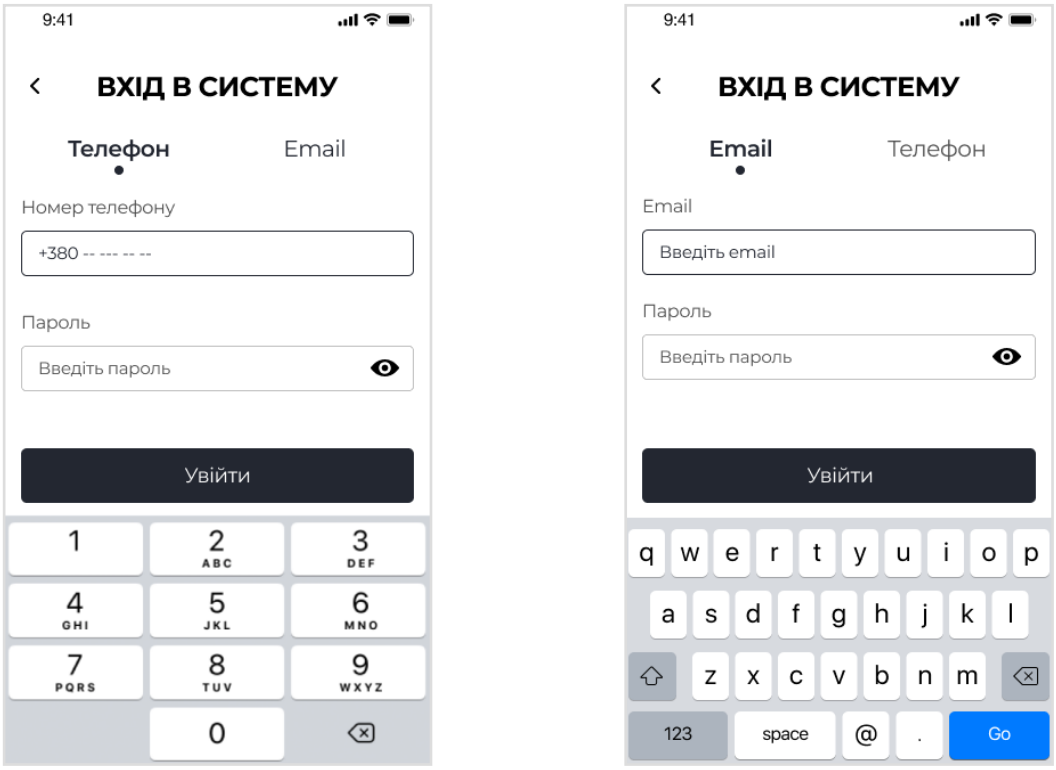



Рисунок 5.5 - Екран авторизації через номер телефону та електронну пошту

Після успішної авторизації оновлюється вигляд бокового меню (Рисунок 5.6), на карті з’являються відмічені існуючі заявки (Рисунок 5.7). Для створення оголошення в меню треба натиснути на кнопку “Створити оголошення”. Користувач побачить нову активність - форму для створення заявки (Рисунок 5.8). Далі необхідно заповнити інформацію про заявку. Щоб обрати фото треба натиснути на зображення фотоапарата або сіру зону навколо нього та вибрати необхідне фото. Після цього прогорнути сторінку вниз (жестом з нижньої точки екрану і до середини) за необхідності вказати коментар, дозаповнити інші поля та натиснути на кнопку “Створити” (Рисунок 5.9).

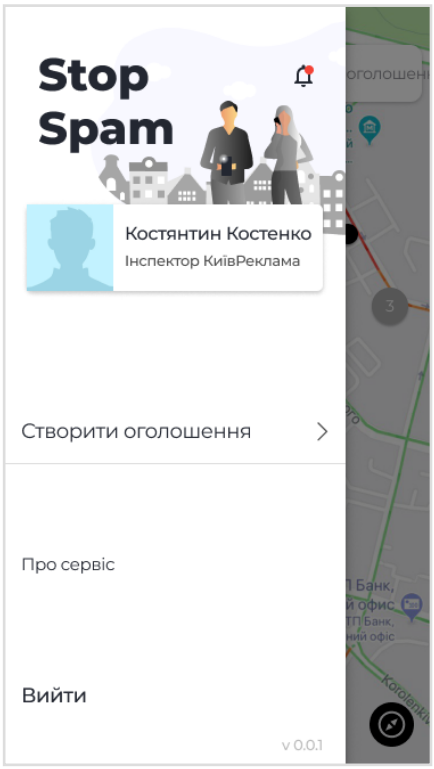



Рисунок 5.6 - Бокове меню авторизованого користувача


Рисунок 5.7 - Головний екран авторизованого користувача

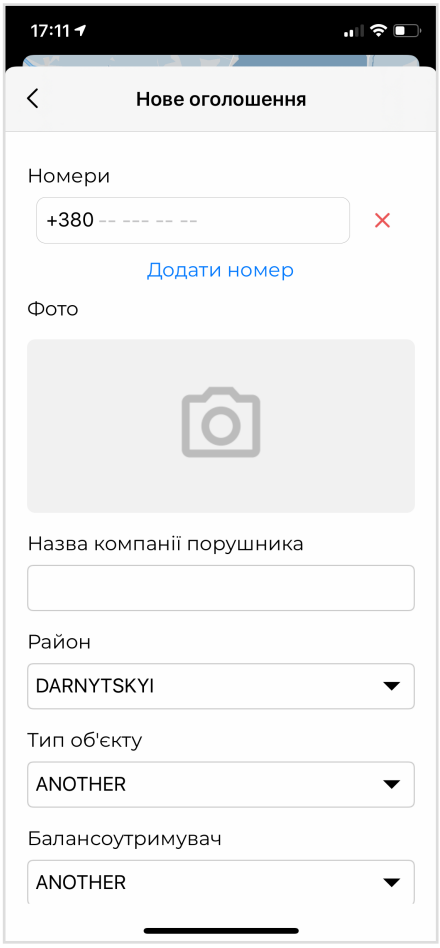



Рисунок 5.8 - Форма створення нової заявки

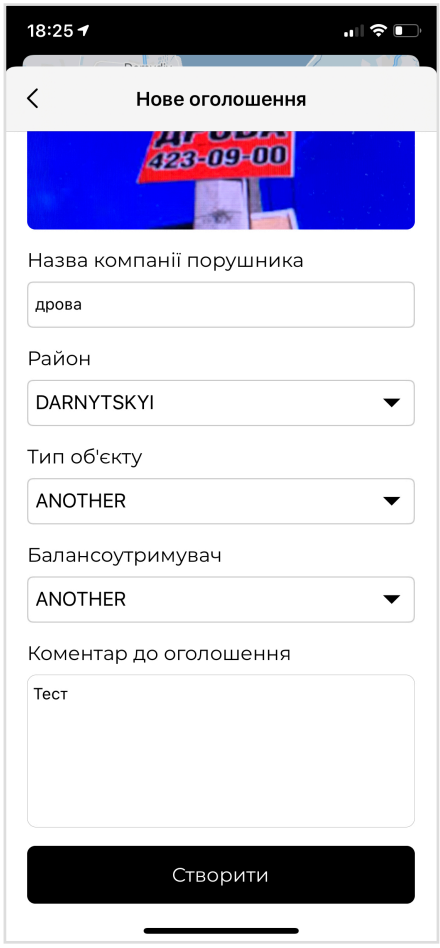



Рисунок 5.9 - Заповнена форма (нижня частина)

Для пошуку заявки необхідно на головному екрані натиснути на поле “Пошук за номером телефону або оголошення”. У користувача з’явиться нова активність з таблицею результатів пошуку або інформацією, яка повідомляє що оголошень не знайдено (Рисунок 5.10). В новій активності потрібно ввести номер шуканого оголошення. В таблиці результатів користувач отримує усі оголошення, в яких номер телефону містить підрядок пошукового запиту, а також в яких ідентифікаційний номер містить комбінацію з запиту (Рисунок 5.11).

За бажанням є можливість переглянути інформацію про сервіс. Для цього потрібно в пункті меню (Рисунок 5.6) натиснути на кнопку “Про сервіс”. Результат подано на Рисунку 5.12.

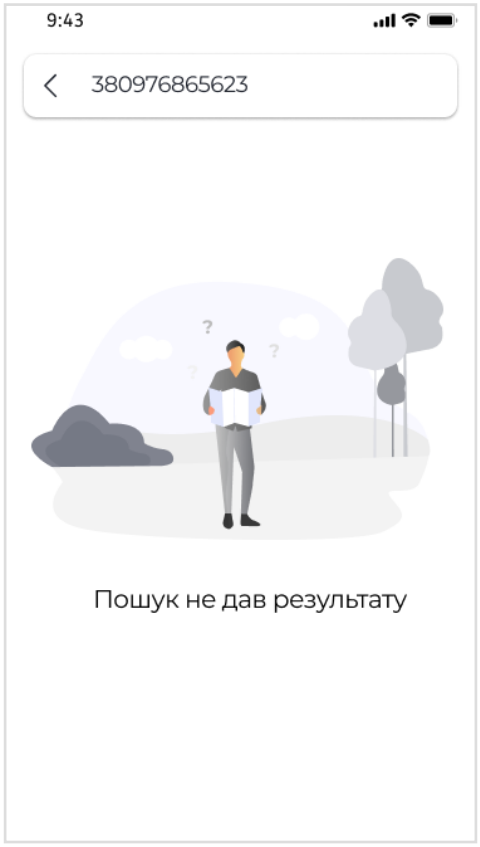



Рисунок 5.10 - Пошук заявки не дав результатів

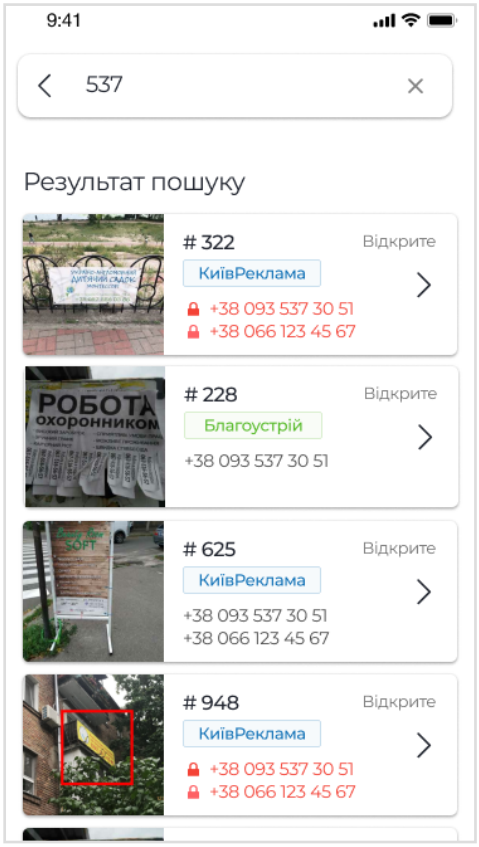



Рисунок 5.11 - Таблиця з результатами пошуку

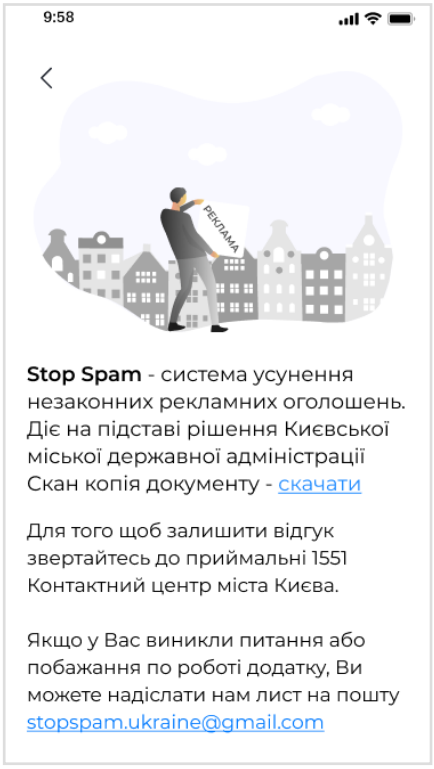



Рисунок 5.12 - Інформаційне вікно про сервіс

Для виходу з профілю авторизованому користувачу необхідно в боковому меню натиснути на кнопку “Вихід”. У разі виходу вигляд меню оновиться до вигляду для неавторизованих користувачів.

## 5.2.Тестування ПЗ

В процесі розробки додатку було проведено поетапне тестування з метою виявлення програмних помилок, перевірки реалізованості функціональних вимог та відповідності ТЗ (технічному завданню).

Тестування iOS-додатку проводилось як вручну, так і за допомогою написання модульних Unit-тестів. Тестований застосунок послідовно запускався, було проведено аналіз його поведінки і за необхідності та за результатами аналізу було введено зміни в коді [19]. Працездатність додатку перевіряється шляхом:

а) введення коректних та некоректних даних (динамічне ручне тестування);

б) динамічного тестування вручну на відповідність усім функціональним вимогам;

в) тестування окремих функцій з використанням Unit-тестів;

г) тестування користувацького інтерфейсу;

ґ) тестування зручності використання.

Під час тестування було перевірено усю функціональність клієнтського програмного забезпечення. Були успішно виконані Unit-тести та вручну перевірені сценарії використання, подані у відповідних таблицях:

- авторизація (таблиця 5.1);
- створення заявки на усунення незаконного оголошення (таблиця 5.2);
- пошук заявки за номером телефону або за номером оголошення (таблиця5.3);
- перегляд інформації про сервіс (таблиця 5.4);
- вихід з системи (таблиця 5.5).

Таблиця 5.1 - Перевірка авторизації

| | |
|---|---|
| Мета тесту | Перевірка можливості авторизуватися |
| Вхідні дані | Логін, пароль користувача |
| Дії | 1. На екрані авторизації ввести номер телефону або електронну пошту.<br>2. Ввести пароль.<br>3. Натиснути на кнопку “Увійти”. |
| Очікуваний результат | Екран авторизації закривається, головним екраном стає активність з картою. При відкритті бокового меню візуальна частина змінюється, показано дані користувача. |
| Стан програмного продукту після проведення тесту | Відображається карта, з’являються мітки з незаконно розміщеними оголошеннями. |
| Тест пройдено? | Так |

Таблиця 5.2 - Перевірка створення заявки на усунення незаконного оголошення

| | |
|---|---|
| Мета тесту | Перевірка можливості створення заявки на усунення незаконного оголошення |
| Вхідні дані | Сценарій №1) Номер телефону вказаного в оголошенні, фото оголошення, адреса<br>Сценарій №2) Спроба створити заявку без обраного фото |
| Дії | Сценарій №1)<br>1.1. В меню обрати пункт “Створити оголошення”.<br>1.2. В полі для вводу номеру оголошення ввести номер.<br>1.3. Для обрання фото натиснути на знак камери.<br>1.4. Надати доступ до галереї.<br>1.5. Обрати фото оголошення з галереї.<br>1.6. Обрати район, тип об’єкту, балансоутримувач, ввести назву компанії порушника та коментар до оголошення.<br>1.7. Натиснути на кнопку “Створити”<br>Сценарій №2) Не вводячи дані та не обравши фото з галереї натиснути на кнопку “Створити” |

Продовження таблиці 5.2

| | |
|---|---|
| Очікуваний результат | Сценарій №1) Заявка буде успішно створена. Користувач побачить інформаційне вікно про успішно створену заявку.<br>Сценарій №2) Заявку створено не буде. Користувач побачить інформаційне вікно про необхідність заповнити дані. |
| Стан програмного продукту після проведення тесту | Сценарій №1) Користувач перенаправляється на головний екран з картою, бачить інформаційне вікно з текстом про результат створення заявки.<br>Сценарій №2) Залишиться відкритою активність з створенням заявки. |
| Тест пройдено? | Так |

Таблиця 5.3 - Пошук заявки за номером телефону або за номером оголошення

| | |
|---|---|
| Мета тесту | Перевірка можливості пошуку заявки за номером телефону або за номером оголошення |
| Вхідні дані | Пошуковий запит |
| Дії | 1. На головному екрані натиснути на поле пошуку<br>2. В полі пошуку ввести шуканий номер заявки або телефону.<br>3. Натиснути на кнопку “Шукати”. |
| Очікуваний результат | 1) Якщо оголошення за запитом знайдені, таблиця даних оновлюється, відображаються шукані оголошення;<br>2) Якщо оголошення за запитом не знайдено, то відображається інформація про те, що оголошень за таким пошуковим запитом в базі даних немає. |
| Стан програмного продукту після проведення тесту | Як тільки програма завершує пошук, інтерфейс оновлюється, відображається або таблиця з результатами, або інформація про відсутність оголошень. |
| Тест пройдено? | Так |

Таблиця 5.4 - Перегляд інформації про сервіс

| | |
|---|---|
| Мета тесту | Перевірка можливості переглянути інформацію про сервіс |
| Вхідні дані | Необхідність відкрити інформацію про сервіс |
| Дії | 1. На головному екрані відкрити меню.<br>2. Вибрати пункт “Про сервіс”. |
| Очікуваний результат | Повинно відкритися вікно з інформацією про сервіс, посиланням на закон на правах якого працює система. |
| Стан програмного продукту після проведення тесту | Стає активним вікно з відображенням даних про сервіс |
| Тест пройдено? | Так |

Таблиця 5.5 - Вихід з системи

| | |
|---|---|
| Мета тесту | Перевірка можливості вийти з профілю, розлогінитись. |
| Вхідні дані | Необхідність вийти з системи |
| Дії | 1. На головному екрані відкрити меню.<br>2. Вибрати пункт “Вихід”.<br>3. В спливаючому вікні натиснути “Вийти” |
| Очікуваний результат | При натисненні користувачем “Вийти”, відображається інформаційне вікно з запитом “Ви впевнені, що хочете вийти?” та двома можливостями вибору:<br>- Вийти<br>- Закрити<br>При закритті меню залишається відкрите.<br>При виході меню закривається і в подальшому його вигляд змінюється на інтерфейс для не авторизованих користувачів. |
| Стан програмного продукту після проведення тесту | При виході з системи, дані про користувача стираються, активним стає головне вікно з відображення карти, змінюється вигляд меню. |
| Тест пройдено? | Так |

Також було протестовано сумісність з різними розмірами екранів. Так, наприклад, iPhone 5 має в ширину 640 пікселів, на довжину 1136 пікселів. В той час, як iPhone XS Max має в ширину 1242 пікселі, в довжину - 2688 [15]. Такий діапазон розмірних характеристик потребує мобільності інтерфейсу. Для уникнення проблем з відносно малими екранами було вирішено на вигляд окремих модульних представлень додати ScrollView. Така дія дає можливість користувачу у випадку, якщо не всі елементи вміщуються в екрані, прогорнути сторінку вниз.

# ВИСНОВКИ

В результаті виконання дипломної роботи було порушено тему незаконно розміщених оголошень, аргументовано актуальність розробки мобільного застосунку на платформі iOS, відповідного тематиці такої проблеми. Проведено змістовну постановку задачі. Для реалізації додатку відповідно до новітніх інформаційних технологій було проаналізовано існуючі шаблони проектування, що допомогло визначитися з найбільш доречним патерном для використання в межах розробки застосунку.

Було проведено змістовний аналіз вимог до застосунку, сформорвано перелік функціональних та нефункціональних вимог. Грунтуючись на них було сформовано сукупність завдань, розроблено сценарії використання додатку.

Базуючись на обраному архітектурному патерні MVP було описано структуру класів програмного забезпечення. Розроблено ефективний алгоритм створення заявки, на основі якого було розроблено клієнтський застосунок.

Застосунок було успішно протестовано на стандартних емуляторах, наданих доповненням до IDE Xcode, Simulator, та на реальних девайсах на платформі iOS (iPhone 7, iPhone XS). Була успішно протестована підтримка різних версій системи, а саме 12.2 - 13.3.1.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Новые правила размещения рекламы в Киеве: все, что нужно знать [Електронний ресурс] – Режим доступу до ресурсу: https://dengi.informator.ua/2018/08/09/novye-pravila-razmeshheniya-reklamy-v-kieve-vse-chto-nuzhno-znat/.
2. Что делать, если мешает реклама на улицах Киева [Електронний ресурс] – Режим доступу до ресурсу: https://kiev.informator.ua/2018/08/09/chto-delat-esli-meshaet-reklama-na-ulitsah-kieva/
3. Иностранный опыт: Как в Греции, США и Канаде борются с нелегальной рекламой [Електронний ресурс] – Режим доступу до ресурсу: https://www.the-village.ru/village/city/abroad/133421-inostrannyy-opyt-kak-goroda-boryutsya-s-nelegalnoy-reklamoy
4. Про внесення змін до правил благоустрою міста Києва: закон України від 15 листопада 2018 року N 49/6100
5. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес: пер. с англ. – С.-Петербург: Питер, 2016. – 368 с.
6. Бек К. Шаблони реалізації корпоративних додатків: пер. з англ. / К. Бек. - М .: Вільямс, 2000. - 224 с.
7. John Charles Olamendy: Model View Presenter Design Pattern [Електронний ресурс] - 2011. - Режим доступу до ресурсу: https://www.c-sharpcorner.com/uploadfile/john_charles/model-view-presenter-design-pattern/
8. Neuburg M. iOS 9 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics / M. NeuburgS.: O'Reilly Media, 2015. – 604 p.
9. Мартін Р. Швидка розробка програм. Принципи, приклади, практика / Р. Мартін, Дж. Ньюкирк, Р. Косс: пер. з англ. - М .: Вільямс, 2004 - 752 с.
10. Фримен Э. Паттерны проектирования / Э. Фримен, К. Сиерра, Б. Бейтс: пер. с англ. – С.-Петербург: Питер, 2016. – 656 с.

11. Zarra M. Core Data: Data Storage and Management for iOS, OS X, and iCloud [Text] / M. Zarra – N.-Y.: Pragmatic Bookshelf, 2013. – 43 с.
12. Додаток в App Store “Stop Spam”. [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/ua/app/stopspam/id1418897282
13. Martin R. Clean Architecture / Martin Robert, 2017. - 432 с.
14. Maps SDK for iOS [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/maps/documentation/ios-sdk/intro?hl=uk.
15. Compare iPhone models [Електронний ресурс] – Режим доступу до ресурсу: https://www.apple.com/uk/iphone/compare/
16. Використання iOS та iPadOS. За підрахунками App Store 27 січня 2020 року. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/support/app-store/
17. Human Interface Guidelines [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/design/human-interface-guidelines/
18. JavaScript and JSON Essentials: Build light weight, scalable, and faster web applications with the power of JSON, 2nd Edition / Bruno Joseph D'mello , Sai Srinivas Sriparasa; Packt Publishing; 2nd Revised edition edition, 2018. — 226 с.
19. Котляров В.П. Основы тестирования программного обеспечения: Учебное пособие / Котляров В. П., Коликова Т.В. — М.: Интернет Университет Информационных Технологий; Бином. Лаборатория знаий, 2006. — 285 с.
20. Гулак О.С., Шалденко О.В. Мобільний застосунок для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами // XVIII-а міжнародна науково-практична конференція молодих вчених та студентів “Сучасні проблеми наукового забезпечення енергетики”.

# ДОДАТОК А

Інструментальні засоби мобільного застосунку
для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами

Специфікація

УКР.НТУУ«КПІ»_ТЕФ_АПЕПС_ТР61105_20Б

Аркушів 2

Київ – 2020

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61105_17Б | Записка Гулак О.С. ТР-61.docx | Текстова частина дипломної роботи |
| Компоненти | | |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61105_20Б 12-1 | CreateAnAdViewController.swift | Компонент створення нової заявки |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61105_20Б 12-2 | LoginViewController.swift | Компонент авторизації в програмі |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61105_20Б 12-3 | SearchViewController.swift | Компонент пошуку оголошень |
| УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61105_20Б 12-4 | AdsForSupervisor.swift | Компонент управління оголошеннями для супервізорі |

# ДОДАТОК Б

Інструментальні засоби мобільного застосунку
для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами

Текст програми

УКР.НТУУ "КПІ". ТР61105_20Б 12-1

Аркушів 16

Київ – 2020

**CreateAnAdViewController.swift**

```
import UIKit
import Photos
import iOSDropDown
import RealmSwift

class SPCreateAnAdViewController: UIViewController, UINavigationControllerDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()
        self.addBackButtonBlack()

        realm = try! Realm()
        let tap = UITapGestureRecognizer(target: self, action:
#selector(self.violationPhotoTapped(_:)))
        self.violationImageView.isUserInteractionEnabled = true
        self.violationImageView.addGestureRecognizer(tap)

        comment.setBorder(radius: 8, color: .lightGray)
        self.getDistricts()
        self.getTypeOfObjects()
        self.getBalancerOfObjects()
        self.registerForKeyboardNotifications()
        self.hideKeyboardWhenTappedAround()
        self.comment.delegate = self
        self.districtDropDown.text = "Солом'янський"
        self.objectType.text = "ANOTHER"
        self.balance.text = "ANOTHER"
    }

    @IBAction func addNumber(_ sender: UIButton) {
        let array = self.arrangedSubviews?.filter({ $0.isHidden })
        if array?.count == 0 {
            Return  }

        array?.first?.isHidden = false
        UIView.animate(withDuration: 0.5) {
            let index = self.arrangedSubviews?.count
            self.phoneNumbersStackView.insertArrangedSubview((array?.first!)!, at: index!)
            self.phoneNumbersStackView.layoutIfNeeded()
        }
    }

    @objc func violationPhotoTapped(_ sender: UITapGestureRecognizer) {
        self.checkGalleryAccess() {
```

```
            DispatchQueue.main.async {
                self.openImagePicker()
            }
        }
    }

    @IBAction func deleteNumberTapped(_ sender: UIButton) {
        sender.superview?.isHidden = true
    }

    @IBAction func createPoster(_ sender: UIButton) {
        let posters = realm.objects(Poster.self)
        if self.violationImageView.accessibilityIdentifier == nil {
            SPAlert.showAlertWithOkButton(self, strTitle: "Виберіть фото", aStrMessage: nil, completion: nil)
            return
        }
        if self.violationImageView.image == UIImage(named: "emptyImage") {
            SPAlert.showAlertWithOkButton(self, strTitle: "Choose photo", aStrMessage: nil, completion: nil)
            return
        }
        let phoneNumbers = allPhones()
        if phoneNumbers == [] {
            SPAlert.showAlertWithOkButton(self, strTitle: "Введіть номер телефону", aStrMessage: nil, completion: nil)
            return
        }
        if posters.filter({$0.number == phoneNumbers?.first}).count > 0 {
            SPAlert.showAlertWithOkButton(self, strTitle: "Даний номер телефону вже блокується", aStrMessage: "Введіть інший номер телефону", completion: nil)
            return
        }
        let advertValue = "{\"external_link\":\"test\",\"description\":\"test\"}"

        guard let currentPosition = UserLocationManager.shared.getCurrentLocation()?.coordinate else {
            return
        }
        let intruder = "{\"name\":\"\(self.intruderTextField.text ?? "")\"}"
        let identifier = self.violationImageView.accessibilityIdentifier ?? ""
        self.violationImageView.image?.accessibilityIdentifier = self.violationImageView?.accessibilityIdentifier ?? ""
        let addresses = "[{\"photo_names\":[\"\(identifier)\"], \"address\":\"test\", \"lat\" :\(currentPosition.latitude), \"lon\":\(currentPosition.longitude)}]"
```

```
        var objectTypeText = objectType.text ?? ""
        var balanceText = balance.text ?? ""
        if objectTypeText == "Не вибраний" || objectTypeText == "" {
            objectTypeText = ""
            balanceText = ""
        }

        var arrayOfParams: [String : String]
        arrayOfParams = ["advert": advertValue,
                         "intruder": intruder,
                         "phoneNumbers" : "\(phoneNumbers!)",
                         "addresses" : addresses,
                         "district" : districtDropDown.text ?? "",
                         "balancer" : balanceText,
                         "objectType" : objectTypeText
            ]
        print(arrayOfParams)
        let newPoster = Poster()
        newPoster.id = posters.count + 1
        newPoster.number = phoneNumbers?.first ?? ""
        store(image: self.violationImageView.image ?? UIImage(), forKey: newPoster.number)
        if let url = filePath(forKey: newPoster.number) {
            newPoster.image = "\(url)"
        }
        try! self.realm.write {
            realm.add(newPoster, update: .all)
        }
        self.dismiss(animated: true, completion: {
        NotificationCenter.default.post(name: NotificationKey.avdertCreatedSuccess,
        object: nil,
        userInfo: nil)})
        NetworkAPI.shared.postPoster(image: self.violationImageView!.image!, arrayOfParamet-
ers: arrayOfParams, completionHandler: {
            self.dismiss(animated: true, completion: {
                NotificationCenter.default.post(name: NotificationKey.avdertCreatedSuccess,
                object: nil,
                userInfo: nil)})
        }) { (code, error) in
            print(error)
        }
    }
}

extension SPCreateAnAdViewController: UITextFieldDelegate {
```

```
    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString string: String) -> Bool {
        guard let textFieldText = textField.text,
            let rangeOfTextToReplace = Range(range, in: textFieldText) else {
                return false
        }
        let substringToReplace = textFieldText[rangeOfTextToReplace]
        let count = textFieldText.count - substringToReplace.count + string.count
        var formattedText = textField.text
        if formattedText!.count == 2 || formattedText!.count == 6 || formattedText!.count == 9{
            formattedText!.append(" ")
        }
        let separator = " "

        if var number = textField.text, string != "" {
            number = number.replacingOccurrences(of: separator, with: "")
            textField.text = formattedText
        }
        return count <= 12
    }
}

extension SPCreateAnAdViewController {

    func getDdistricts() {
        var objectsName: [String] = []
            NetworkAPI.shared.getConstants(searchValue: "districts", completion: { (data) in
            print(data)
            for object in data {
                if let name = object.name {
                    objectsName.append(name)
                }
            }
            self.districtDropDown.optionArray = objectsName
        })
    }

    func getTypeOfObjects() {
        var objectsName: [String] = ["Не вибраний"]
        var objectsType: [Int] = [0]
        NetworkAPI.shared.getConstants(searchValue: "object_types", completion: { (data) in
            print(data)
            for object in data {
                if let name = object.type {
                    objectsName.append(name)
```

```
                objectsType.append(object.id)
            }
        }
        self.objectType.optionArray = objectsName
        self.objectType.optionIds   = objectsType
    })
    self.objectType.didSelect(completion: {
        (selectedText , index ,id) in
        self.balance.optionArray = []
        self.balance.text = ""
        self.balance.layoutIfNeeded()
        self.balance.selectedIndex = nil
        for object in self.allBalance {
            if object.id == id, let name = object.name {
                self.balance.optionArray.append(name)
                self.balance.optionIds?.append(self.balance.optionArray.count)
            }
        }
        self.balance.selectedIndex = 0
        self.balance.text = self.balance.optionArray.first
    })
}

func getBalancerOfObjects() {
    var objects: [String] = ["Не вибраний"]
    var objectsType: [Int] = [0]
    let balanceObject : SPConstants = SPConstants(id: 0, name: "Не вибраний", type: nil, ob-
jectTypeId: nil, objectType: nil)
    NetworkAPI.shared.getConstants(searchValue: "balancers", completion: { (data) in
        print(data)
        for object in data {
            if let name = object.objectType?.name, let id =  object.objectType?.id {
                objects.append(name)
                objectsType.append(id)
            }
            let balanceObject : SPConstants = SPConstants(id: object.objectType!.id, name: ob-
ject.name, type: nil, objectTypeId: nil, objectType: nil)
            self.allBalance.append(balanceObject)
        }
        self.balance.optionArray = objects
        self.balance.optionIds   = objectsType
    })
}

func allPhones() -> [String]? {
```

```
        var result: [String]? = []
        if let phone = phoneNumberTextField.text, phone != "" {
            let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")

            result?.append(finalPhoneNumber)
        }
        if let phone = secondNumberTF.text, phone != "" {
            let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")
            result?.append(finalPhoneNumber)
        }
        if let phone = thirdNumberTF.text, phone != "" {
            let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")
            result?.append(finalPhoneNumber)
        }
        if let phone = fourthNumberTF.text, phone != "" {
            let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")
            result?.append(finalPhoneNumber)
        }
        if let phone = fifthNumberTF.text, phone != "" {
            let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")
            result?.append(finalPhoneNumber)
        }
        return result
    }
    func registerForKeyboardNotifications() {
        NotificationCenter.default.addObserver(self, selector: #selector(onKeyboardAppear(_:)),
name: UIResponder.keyboardWillShowNotification, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(onKeyboardDisappear(_:)),
name: UIResponder.keyboardWillHideNotification, object: nil)
    }

    @objc func onKeyboardAppear(_ notification: NSNotification) {
        let info = notification.userInfo!
        let rect: CGRect = info[UIResponder.keyboardFrameEndUserInfoKey] as! CGRect
        let time = info[UIResponder.keyboardAnimationDurationUserInfoKey] as! NSNumber
        let kbSize = self.view.convert(rect, from: nil)//rect.size
        let safeAreaBottom = view.safeAreaInsets.bottom
        self.saveButtonBottomConstraint?.constant = kbSize.height + safeAreaBottom
        self.scrollViewBottomConstraint?.isActive = false
        var contentInset:UIEdgeInsets = self.scrollView.contentInset
        contentInset.bottom = kbSize.height
        let keyBoardSize = info[UIResponder.keyboardFrameEndUserInfoKey] as! CGRect
        scrollView.contentInset = UIEdgeInsets(top: 0.0, left: 0.0, bottom: keyBoardSize.height,
right: 0.0)
```

```
        scrollView.scrollIndicatorInsets = UIEdgeInsets(top: 0.0, left: 0.0, bottom: keyBoard-
Size.height, right: 0.0)
        UIView.animate(withDuration: time.doubleValue) {
            self.view.layoutIfNeeded()
        }
    }

    @objc func onKeyboardDisappear(_ notification: NSNotification) {
        let info = notification.userInfo!
        let time = info[UIResponder.keyboardAnimationDurationUserInfoKey] as! NSNumber

        self.saveButtonBottomConstraint?.constant = 0
        self.scrollViewBottomConstraint?.isActive = true
        let contentInset:UIEdgeInsets = UIEdgeInsets.zero
        scrollView.contentInset = contentInset
        UIView.animate(withDuration: time.doubleValue) {
            self.view.layoutIfNeeded()
        }
    }

    private func filePath(forKey key: String) -> URL? {
        let fileManager = FileManager.default
        guard let documentURL = fileManager.urls(for: .documentDirectory,
                                    in: FileManager.SearchPathDomainMask.userDomainMask).first
else { return nil }

        return documentURL.appendingPathComponent(key + ".png")
    }

    private func store(image: UIImage,
                       forKey key: String) {
        if let pngRepresentation = image.pngData() {

            if let filePath = filePath(forKey: key) {
                do  {
                    try pngRepresentation.write(to: filePath,
                                                options: .atomic)
                } catch let err {
                    print("Saving file resulted in error: ", err)
                }
            }
        }
    }
}
```

```
extension SPCreateAnAdViewController: UITextViewDelegate {
    func textViewDidBeginEditing(_ textView: UITextView) {
        let textVFrame = textView.frame.origin.y - textView.frame.height - 20
        let scrollPoint : CGPoint = CGPoint.init(x:0, y: textVFrame)
        self.scrollView.setContentOffset(scrollPoint, animated: true)
    }

    func textViewDidEndEditing(_ textView: UITextView) {
        self.scrollView.setContentOffset(CGPoint.zero, animated: true)
    }
}
```

**LoginViewController.swift**

```
import UIKit

enum loginType: String {
    case sendPass       = "sendPass"
    case enterWithPhone = "enterWithPhone"
    case enterWithEmail = "enterWithEmail"

    var typeText: String {
        switch self {
        case .sendPass:
            return "Номер телефону"
        case .enterWithPhone:
            return "Номер телефону"
        case .enterWithEmail:
            return "Електронна пошта"
        }
    }
    var buttonText: String {
        switch self {
        case .sendPass:
            return "Увійти"
        case .enterWithPhone:
            return "Увійти"
        case .enterWithEmail:
            return "Увійти"
        }
    }
}

class SPLoginViewController: UIViewController {

    @IBOutlet weak var phoneMailSegmentedControl: UISegmentedControl!
    @IBOutlet weak var phoneNumberTextField: UITextField!
```

```
    @IBOutlet weak var mailTextField: UITextField!
    @IBOutlet weak var typeFirstLabel: UILabel!
    @IBOutlet weak var phoneNumberView: UIView!
    @IBOutlet weak var enterButton: UIButton!
    @IBOutlet weak var passwordLabel: UILabel!
    @IBOutlet weak var passwordTextField: UITextField!

    var profile: SPProfile?

    var pageType: loginType = .sendPass

    override func viewDidLoad() {
        super.viewDidLoad()
        self.phoneNumberView.setBorder(radius: 5, color: .lightGray)
        self.phoneNumberTextField.delegate = self
        NotificationCenter.default.addObserver(self, selector: #selector(userRegistered(_:)), name: Notifica-
tionKey.didUserRegistered, object: nil)
        // Do any additional setup after loading the view.
    }

    override func viewWillLayoutSubviews() {
        phoneMailSegmentedControl.setTitleTextAttributes([NSAttributedString.Key.foregroundColor:
UIColor.black], for: .selected)
        phoneMailSegmentedControl.setTitleTextAttributes([NSAttributedString.Key.foregroundColor:
UIColor.lightGray], for: .normal)
    }

    // MARK: - Action

    @IBAction func indexChanged(_ sender: Any) {
        switch phoneMailSegmentedControl.selectedSegmentIndex {
        case 0:
            switch self.pageType {
            case .sendPass:
                self.pageType = .enterWithPhone
            case .enterWithPhone:
                self.pageType = .enterWithPhone
            case .enterWithEmail:
                self.pageType = .sendPass
            }
            self.mailTextField.isHidden = true
        case 1:
            self.pageType = .enterWithEmail
            self.mailTextField.isHidden = false
        default:
            break
        }
        self.typeFirstLabel.text = self.pageType.typeText
```

```
        self.enterButton.titleLabel?.text = self.pageType.buttonText
    }

    @IBAction func enterButtonTapped(_ sender: UIButton) {
        switch self.pageType {
        case .sendPass:
            self.handleSendingPass()
            break
        case .enterWithEmail:
            self.handleEnterWithEmail()
            break
        case .enterWithPhone:
            self.handleEnterWithPhone()
        }
    }

    //MARK:  Handle requests

    private func handleSendingPass() {
        guard let phone = phoneNumberTextField.text else {
            return
        }
        let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")
        NetworkAPI.shared.getCode(phoneNumber: finalPhoneNumber, completion: { (access) in
            if (access) {
                self.passwordLabel.isHidden = false
                self.passwordTextField.isHidden = false
                self.enterButton.setTitle("Увійти", for: .normal)
                self.pageType = .enterWithPhone
            }
        })
        if finalPhoneNumber == "380965400848" || finalPhoneNumber == "380935373051" {
            NetworkAPI.shared.token(phoneNumber: finalPhoneNumber, loginType: .phone, completion-
Handler: { [weak self] (obj) in
                self?.dismiss(animated: true, completion: self?.getUser)
                print(obj.token)
            }) { (code, message) in
                print(message)
            }
        } else {
            SPAlert.showAlertWithOkButton(self, strTitle: "Помилка", aStrMessage: "Номер телефону або
пароль вказані невірно", completion: nil)
        }
        self.dismiss(animated: true, completion: self.getUser)
        self.getUser()
    }

    private func handleEnterWithPhone() {
```

```
        guard let phone = phoneNumberTextField.text else {
            return
        }
        self.dismiss(animated: true, completion: self.getUser)
      self.getUser()
        let finalPhoneNumber = "380\(phone)".replacingOccurrences(of: " ", with: "")
        if finalPhoneNumber == "380965400848" || finalPhoneNumber == "380935373051" {
            NetworkAPI.shared.token(phoneNumber: finalPhoneNumber, loginType: .phone, completion-
Handler: { [weak self] (obj) in
                self?.dismiss(animated: true, completion: self?.getUser)
                print(obj.token)
            }) { (code, message) in
                print(message)
            }
        } else {
            SPAlert.showAlertWithOkButton(self, strTitle: "Помилка", aStrMessage: "Номер телефону або
пароль вказані невірно", completion: nil)
        }
    }

    private func handleEnterWithEmail() {
      guard let phone = mailTextField.text else {
        return
       }
            if phone == "alena.hulak@gmail.com" {
                NetworkAPI.shared.token(phoneNumber: phone, loginType: .phone, completionHandler:
{ [weak self] (obj) in
                    self?.dismiss(animated: true, completion: self?.getUser)
                    print(obj.token)
                }) { (code, message) in
                    print(message)
                }
            } else {
                SPAlert.showAlertWithOkButton(self, strTitle: "Помилка", aStrMessage: "Email або пароль
вказані невірно", completion: nil)
            }
        }
}

extension SPLoginViewController: UITextFieldDelegate {
    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacement-
String string: String) -> Bool {
        guard let textFieldText = textField.text,
            let rangeOfTextToReplace = Range(range, in: textFieldText) else {
                return false
        }
        let substringToReplace = textFieldText[rangeOfTextToReplace]
        let count = textFieldText.count - substringToReplace.count + string.count
```

```
        var formattedText = textField.text
        if formattedText!.count == 2 || formattedText!.count == 6 || formattedText!.count == 9{
            formattedText!.append(" ")
        }
        let separator = " "

        if var number = textField.text, string != "" {
            number = number.replacingOccurrences(of: separator, with: "")
            textField.text = formattedText
        }
        return count <= 12
    }
}

extension SPLoginViewController {
    func getUser() {
        NetworkAPI.shared.getUser(completionHandler: { profile in
            UserDefaultsManager.profile = profile
            DispatchQueue.main.async {
                self.profile = profile
                NotificationCenter.default.post(name: NotificationKey.didUserLogin,
                                                object: profile,
                                                userInfo: nil)
            }
        }) { (code, error) in
            DispatchQueue.main.async {

            }
        }
    }

    @objc func userRegistered(_ notification: Notification) {
        self.dismiss(animated: false) {
            NotificationCenter.default.post(name: NotificationKey.didUserRegisteredS,
            object: nil,
            userInfo: nil)
        }
    }
```

**SearchViewController.swift**

```
import Foundation

public protocol SkeletonTableViewDataSource: UITableViewDataSource {
    func numSections(in collectionSkeletonView: UITableView) -> Int
    func collectionSkeletonView(_ skeletonView: UITableView, numberOfRowsInSection section: Int) ->
Int
```

```
    func collectionSkeletonView(_ skeletonView: UITableView, cellIdentifierForRowAt indexPath: Index-
Path) -> String
}

extension SPSearchViewController: SkeletonTableViewDataSource, UITableViewDelegate, UITable-
ViewDataSource {
    func numSections(in collectionSkeletonView: UITableView) -> Int {
        if self.resultPosters.count > 0 {
            self.updateView(true, false)
        } else {
            self.updateView(false, true)
        }
        return self.resultPosters.count
    }

    func collectionSkeletonView(_ skeletonView: UITableView, numberOfRowsInSection section: Int) ->
Int {
        if self.resultPosters.count > 0 {
            self.updateView(true, false)
        } else {
            self.updateView(false, true)
        }
        return self.resultPosters.count
    }

    func collectionSkeletonView(_ skeletonView: UITableView, cellIdentifierForRowAt indexPath: Index-
Path) -> String
    {
        return "cell"
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if self.resultPosters.count > 0 {
            self.updateView(true, false)
        } else {
            self.updateView(false, true)
        }
        return self.resultPosters.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) as! SPPosterIn-
SearchTableViewCell
        let currentPoster = self.resultPosters[indexPath.row]
        var posterImage: UIImage?
        posterImage = UIImage(named: "poster\(currentPoster.content?.id ?? 1)")
        let group = "КиївРеклама"
```

```
        let posterId = "#\(currentPoster.id ?? 1)"
        let secondNumber = currentPoster.content?.phoneNumbers.first?.number
        let firstNumber = currentPoster.content?.phoneNumbers.last?.number

        cell.configure(image: posterImage, posterId: posterId, group: group, firstPhone: secondNumber ??
"", secondPhone: firstNumber)
        cell.setNeedsLayout()
        DispatchQueue.global().async {
        NetworkAPI.shared.getPoster(id: currentPoster.content?.id ?? 1, completionHandler: { (poster) in
            if currentPoster.phoneNumbers != nil {
             let numbers = poster.phoneNumbers.first?.number ?? "PhoneNumbers(id: 1, number: )"
            NetworkAPI.shared.getPosterPhoto(photoName: currentPoster.content?.default_image_path) {
                        (image) in
                        guard let image = image else { return }
                    DispatchQueue.main.async {
                       posterImage = image

                        cell.configure(image: posterImage, posterId: posterId, group: group, firstPhone:
numbers ?? "", secondPhone: secondNumber)
                        cell.setNeedsLayout()

              }}})
             { (code, description) in
               debugPrint(code)
             }
        return cell
    }
}
```

**AdsForSupervisor.swift**

```
import UIKit
import RealmSwift

class AdsForSuperviserViewController: UITableViewController {

    var realm: Realm!

    var postersList: Results<Poster> {     //(2)
     get {
         return realm.objects(Poster.self)
        }
    }

    @IBOutlet weak var tableViewCell: UITableViewCell!

    override func viewDidLoad() {
        super.viewDidLoad()
```

# ДОДАТОК В

Інструментальні засоби мобільного застосунку
для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ АПЕПС_ТР61105_20Б 13-1

Аркушів 9

Київ – 2020

# АНОТАЦІЯ

Додаток містить опис основних програмних класів інструментальних засобів мобільного застосунку для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами, що виконують деякі головні функціональні можливості програми, а саме:

— Можливість авторизації в додатку.

— Можливість створення нової заявки.

— Можливість пошуку оголошення.

— Можливість управління оголошеннями, видаляти їх, блокувати, набору вказаного номеру телефону.

Мобільний застосунок розроблений за допомогою мови програмування Swift з використанням інструментальних засобів Xcode.

ЗМІСТ

# ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис основних програмних класів інструментальних засобів мобільного застосунку для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами, що виконують деякі головні функціональні можливості програми.

У додатку Б міститься програмний код основних програмних модулів.

Мобільний застосунок для роботи потребує мобільний пристрій з операційною з операційною системою iOS версії 11.0 і вище.

Мобільний застосунок розроблений за допомогою допомогою мови програмування Swift з використанням інструментальних засобів Xcode.

# ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Інструментальні засоби мобільного застосунку для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами, надають такі функціональні можливості програми:

1. Наявна можливість атворизації та рестрації.
2. Наявна можливість входу до системи без авторизації, якщо користувач вже входив до свого акаунту і не виходив з нього.
3. Наявна можливість створювати нову заявку (з вказанням адреси, номерів зазначених на оголошенні телефонів, фотозвітом, назвою компанії порушника, типу об’єкту, на якому розміщено рекламу, вказанням коментарів та балансоутримувача) та скасовувати її.
4. Наявна можливість створення фото або можливість вибору фото з галереї.
5. Наявна можливість перегляду інформації про сервіс.
6. Наявна можливість користувачу переглянути інформацію про незаконно розміщені оголошення.
7. Наявна можливість пошуку заявки за номером вказаних у ній телефонів або безпомередньо за номером створених заявок.
8. Наявна можливість визначати місцезнаходження користувача або давати можливість самостійно вказати положення на карті.
9. Наявна можливість телефонувати на номер вказаний в оголошенні.
10. Наявна можливість копіювати в буфер обміну номер телефону вказаного в оголошенні.
11. Наявна можливість блокувати номер телефону, який вказаний в оголошенні.
12. Наявна можливість видаляти створені заявки на блокування номеру.

# ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Згідно з архітектурою, додаток складається з трьох головних модулів:

— модуль представлення.

— модуль моделей.

— модуль логіки реакцій на події.

Модуль представлення містить велику кількість класів, що слугують для демонстрації користувацького інтерфейсу та реактивних змін на екрані. Перенаправляє події до класу пред’явника.

Шар моделей містить в собі класи, що використовуються для моделювання сутностей, якими оперує мобільний застосунок.

Шар пред'явник містить логіку реакцій на події, відповідає за функції, що викликатимуться внаслідок цих реакцій, оновлює модель.

# ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання мобільного застосунку користувач повинен мати мобільний пристрій з операційною системою iOS версії 11.0 і вище.

Для першого використання застосунку користувач має мати підключення до мережі інтернет для авторизації у систему. Після цього, авторизація є автоматичною і всі зміни в системі зберігаються локально, в разі відсутності підключення до мережі інтернет. Після відновлення підключення до мережі всі дані синхронізуються. Обов’язковим є підключення до інтернету при створенні заявки, а також при внесенні номеру телефону в базу порушників.

# ВХІДНІ ДАНІ

Вхідна інформація для інструментальних засобів мобільного застосунку:

1. Вхідними даними системи аутентифікації мають бути електронна пошта та пароль або номер телефону та пароль.
2. Вхідними даними системи створення заявки на усунення оголошення мають бути фото порушення, номер вказаного телефону, назва компанії, тип об’єкту, на якому розміщене оголошення, балансоутримувач, район та коментар.
3. Вхідними даними відображення мапи має бути поточне місцерозташування користувача.

# ВИХІДНІ ДАНІ

Вихідна інформація для інструментальних засобів мобільного застосунку:

1. Вихідними даними системи аутентифікації мають бути інформація привязана до даного акаунту та дані для подальшої автоматичної аутентифікації.
2. Вихідними даними системи перегляду оголошень на карті мають бути сутності оголошень, їх локацій та номерів.
3. Вихідними даними системи управління заявками мають бути надіслані вхідні дані, інформація про порушника.

# ДОДАТОК Г

Мобільний застосунок
для моніторингу та ідентифікації незаконного розміщення зовнішньої реклами

Статті наукових конференцій

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ АПЕПС_ТР61105_20Б

Аркушів 4

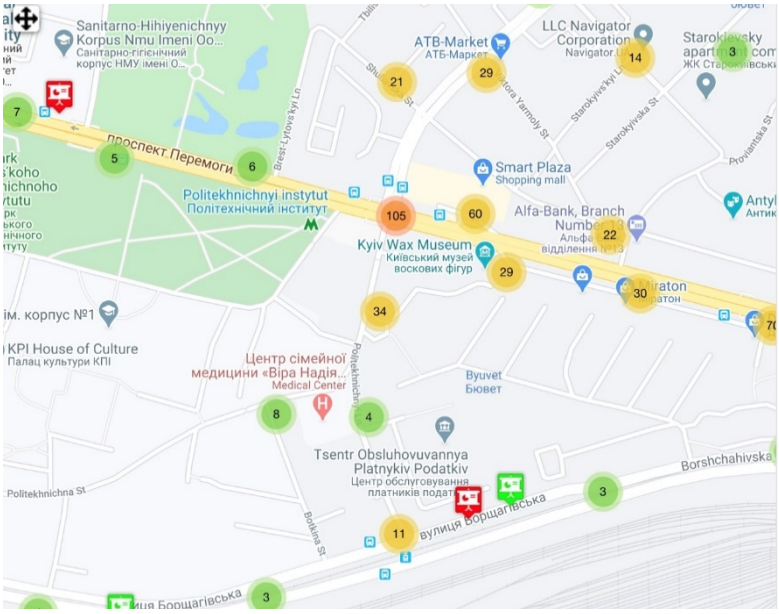
Київ – 2020

*Hulak O.S., student, alena.hulak@gmail.com*
*Shaldenko O.V., docent, candidate of technical sciences, o.shaldenko@gmail.com*
*National Technical University of Ukraine*
*"Kyiv Polytechnic Institute" named Igor Sikorsky, Ukraine*

# ANALYSIS OF MONITORING METHODS AND IDENTIFICATION OF ILLEGAL OUTDOOR ADVERTISING

Nowadays, outdoor advertising is very popular and highly effective. It combines many advantages: a wide audience, affordable cost and flexibility. Unlike advertising in print media, on radio and television, no one purposefully buys it, does not turn on the radio and television. It does not accompany entertainment programs to engage the audience. It is more often placed in places where the flow of people is quite high, because more people see the ad, the higher its effectiveness [1].

With the growing demand for outdoor advertising, the number of illegally placed ads has grown, which creates a new problem - pollution of the city (Picture 1) [4]. Utilities are actively fighting violators and involving inspectors from various organizations, such as KyivReklama and Blagoustriy in combating illegally placed advertisements.




Picture 1 - Map with marked illegally placed advertising

Since the beginning of 2017, a new advertising reform has been introduced in Kyiv, which describes the rules for placing ads, namely the ban on advertising in parking lots,

road junctions, intersections, residential complexes, underground and overground crossings [2]. As a result, the amount of advertising has decreased 3 times. In order to maintain the level of income, it was decided to increase the cost of advertising, which provoked the flow of illegal advertising.

The goal is to create a mobile application for monitoring and identification of illegally placed ads, the implementation of which will allow inspectors of utilities, as well as residents of Kiev to clean their city from illegal advertising.

The fight against advertising is a problem not only in Ukraine but also in many other countries. For example, in June 2013, Toronto Mayor Rob Ford personally disrupted illegal ads from bus stops as part of the Clean Toronto Together campaign. It was assumed that representatives of the municipality will patrol the main streets of the city all summer, clean the posts and boxes of utilities, bus stops and mailboxes. And at the same time to issue fines to violators in the amount of 300 to 500 dollars. At the same time, advertisements posted by the residents of the district with information about lost property, a garage sale or a religious event were not subject to a fine. In addition, Illegal Signs volunteers report illegal objects to the City Hall (in Toronto, in particular, large-scale advertising on outdoor media is prohibited), after which violating companies dismantle the object and simply move it to another part of the city. They have managed to place their employees in all major companies that offer advertising, and they report to Illegal Signs prompt and accurate information about new illegal designs. But to launch such a campaign requires a lot of money and people willing to help in their spare time.

The problem is more similar in Greece, where, as in Kyiv, there is a problem not only with advertisements, but also with illegal constructions, such as our advertising trailers. The existing law in Greece, which prohibits advertising along roads, was complied with only during the 2004 Athens Olympics. Residents of the capital have long complained about the mayor, who was suspected of colluding with illegal advertisers [3]. As a result, in 2010 they managed to get the Ministry of Transport and Infrastructure to launch a website and a special application for the iPhone, through which you can report illegal advertising.

Based on this method, which is the most optimal for the people of our country, we developed our own solution. A person who wants to report illegal advertising must take a

few minutes to take a picture of the illegal advertising, download it to the application, set the geolocation and provide the phone numbers listed in the ad. Authorized companies then review, track, and eliminate all illegal ads. The user will receive push notifications about the change of the ad status.

The purpose of the software product is to create an application for the elimination of advertising, display a map with the offenders marked on it, monitoring the status of the created applications.

To ensure the interaction of those wishing to remove the illegally placed ad and the supervisor, who will check whether the phone number corresponds to the information provided in the ad, the system includes applications for activists and utilities for different platforms, system server and application for supervisors.

In this article, a solution to the problem of monitoring and identification of illegally placed outdoor advertising was proposed and described. As a result of the analysis of existing methods of solving the problem in different developed countries, our own algorithm was formed: to form an application for illegally placed outdoor advertising using the application, send it to the supervisor, the supervisor checks the application and blocks the phone number.

The solution has been successfully applied to the system for eliminating illegal advertisements and will soon be used by Kyiv utilities.

Bibliography

1. New rules for advertising in Kiev: everything you need to know [Electronic resource] – Resource access mode: https://dengi.informator.ua/2018/08/09/novye-pravila-razmeshheniya-reklamy-v-kieve-vse-chto-nuzhno-znat/.
2. What to do if advertising on the streets of Kiev interferes [Electronic resource] – Resource access mode: https://kiev.informator.ua/2018/08/09/chto-delat-esli-meshaet-reklama-na-ulitsah-kieva/
3. City to clamp down on illegal outdoor advertisers [Electronic resource] – Resource access mode: https://www.joburg.org.za/media_/Newsroom/Pages/2016%20&%202015%20Articles/City-to-clamp-down-on-illegal-outdoor-advertisers.aspx.
4. Map with marked illegally placed advertising [Electronic resource] – Resource access mode: http://db.kievreklama.kiev.ua/db/rz.dll