**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СИСТЕМИ ДИСТАНЦІЙНОГО КЕРУВАННЯ ДОСТУПОМ ДО БУДИНКУ»**

Виконала: студентка 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
М.І. Барнаш
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Барнаш Марії Іванівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка системи дистанційного керування доступом до будинку

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Створення системи віддаленого контролю за доступом до житла.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 07.03.2024 р.

## К А Л Е Н Д А Р Н И Й  П Л А Н

| № | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Розробка плану роботи. | 11.03.2024 | |
| | | | |
| 2. | Збір вихідних даних. | 25.03.2024 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 22.04.2024 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 24.06.2024 | |
| | | | |
| 5. | Розробка третього розділу. | 02.09.2024 | |
| | | | |
| 6. | Розробка четвертого розділу. | 28.10.2024 | |
| | | | |
| 7. | Оформлення та нормоконтроль | 20.11.2024 | |
| | кваліфікаційної роботи магістра. | | |
| | | | |
| 8. | Захист кваліфікаційної роботи. | 11.12.2024 | |

Студент _____ М.І. Барнаш
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

## Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

# РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка системи дистанційного керування доступом до будинку»: 65 с., 42 рис., 21 джерело, 2 додатки.

ДИСТАНЦІЙНЕ КЕРУВАННЯ ДОСТУПОМ, РОЗУМНИЙ ДІМ, РОЗУМНИЙ ДОМОФОН.

Об’єкт дослідження – системи розумного входу.

Мета роботи: розробка системи дистанційного керування доступу до багатоквартирних будинків.

Методи дослідження: методи ООП, методи програмної інженерії (розробка вебзастосунків засобами PHP Laravel та VueJS з використанням MySQL, розгортання за допомогою Docker та Kubernetes).

В умовах швидкого технологічного розвитку та поширення концепції «розумного будинку», системи управління доступом набувають все більшої актуальності та значення. Дане дослідження зосереджене на розробці системи дистанційного керування доступом до багатоквартирних будинків. У ході дослідження було проведено ретельне вивчення існуючих підходів до розробки систем розумного входу, а також аналіз потреб і вимог користувачів у цьому напрямку. Застосовуючи методи об’єктно-орієнтованого програмування та програмної інженерії, було розроблено архітектуру системи та вебдодатки за допомогою PHP Laravel та VueJS. Особлива увага була приділена тестуванню розробленої системи, що дозволило перевірити її функціональність, надійність та безпеку в різних умовах експлуатації. Завершальним етапом було розгортання системи за допомогою Docker та Kubernetes, що забезпечило зручність установки та управління. У результаті цієї роботи було створено ефективну та надійну систему дистанційного керування доступом до будинку «SmartEntry», яка відповідає сучасним вимогам безпеки та зручності користувачів у вході до будинку.

# SUMMARY

Master’s qualifying paper “Development of a System of Remote Control of Access to the House”: 65 pages, 42 figures, 21 references, 2 supplements.

REMOTE ACCESS CONTROL, SMART HOME, SMART INTERCOM.

The object of the study is smart entry systems.

The aim of the study is to develop a remote access control system for apartment buildings.

The methods of research are OOP methods, software engineering methods (development of web applications using PHP Laravel and VueJS with MySQL, deployment using Docker and Kubernetes).

In the context of rapid technological development and the proliferation of the «smart home» concept, access control systems are becoming increasingly relevant and significant. This study focuses on the development of a remote access control system for apartment buildings. A thorough examination of existing approaches to the development of smart entry systems was conducted, as well as an analysis of user needs and requirements in this area. By applying object-oriented programming methods and software engineering techniques, the system architecture and web applications were developed using PHP Laravel and VueJS. Special attention was given to testing the developed system, which allowed verification of its functionality, reliability, and security under various operating conditions. The final stage involved deploying the system using Docker and Kubernetes, which ensured ease of installation and management. As a result of this work, an effective and reliable remote access control system «SmartEntry» was created, meeting modern security requirements and providing user convenience for entering the building.

**ЗМІСТ**

Завдання на кваліфікаційну роботу……..…..2
Реферат ..... 4
Summary ..... 5
Вступ..... 8
1 Теоретична частина..... 10
1.1 Аналіз літератури..... 10
1.2 Постановка задачі ..... 12
1.3 Аналіз існуючих аналогів..... 14
1.4 Огляд засобів розробки ..... 16
1.4.1 Laravel ..... 17
1.4.2 Spatie/laravel-permissions..... 18
1.4.3 Owen-it/laravel-auditing..... 19
1.4.4 Moonshine..... 19
1.4.5 VueJS ..... 21
1.4.6 Inertia ..... 22
1.4.7 Agora Video SDK..... 24
1.5 Огляд засобів розгортання ..... 24
1.5.1 Docker..... 25
1.5.2 Kubernetes ..... 26
1.5.3 Хмарні платформи ..... 27
1.6 Висновки до розділу 1 ..... 28
2 Проєктування..... 29
2.1 Визначення прецедентів..... 29
2.2 Визначення схеми даних ..... 30
2.3 Проєктування додатків..... 34
2.3.1 Проєктування адміністративної панелі ..... 35
2.3.2 Проєктування клієнтського застосунку..... 39

2.4 Висновки до розділу 2 ..... 41
3 Реалізація ..... 42
3.1 Реалізація адміністративної панелі ..... 42
3.2 Реалізація клієнтського застосунку ..... 44
3.3 Висновки до розділу 3 ..... 46
4 Розгортання ..... 47
4.1 Створення Docker образів ..... 47
4.2 Створення ресурсів Kubernetes ..... 48
4.3 Висновки до розділу 4 ..... 54
Висновки ..... 55
Перелік посилань ..... 57
Додаток А Приклади роботи адміністративної панелі ..... 59
Додаток Б Приклади роботи клієнтського додатку ..... 62

# ВСТУП

**Актуальність теми.** У сучасних умовах швидкого технологічного розвитку та поширення концепції «розумного будинку» системи управління доступом набувають все більшої актуальності та значення. Сучасні наукові дослідження та розробки спрямовані на вдосконалення систем управління «розумними домами», де використовуються різноманітні датчики та алгоритми для забезпечення безпеки, енергоефективності та комфорту мешканців. Важливо зазначити, що більшість сучасних досліджень і розробок зосереджені на вдосконаленні технологій для приватних будинків, тоді як питання інтеграції таких систем у багатоквартирних та багатоповерхових будівлях потребують більш глибокого вивчення. Такий підхід може сприяти підвищенню безпеки та зручності у житті мешканців багатоквартирних будинків, що є важливим аспектом соціального та технологічного розвитку. Тому розробка інноваційної системи керування доступом, яка відповідає потребам сучасного міста, стає нагальною необхідністю.

**Мета і завдання дослідження.** Метою даного дослідження є розробка та вдосконалення системи розумного входу, яка відповідала б потребам сучасного міста, де багатоквартирні та багатоповерхові будівлі є стандартом. Завдання дослідження включають:

– вивчення існуючих підходів до розробки систем розумного входу;

– аналіз потреб і вимог користувачів;

– розробка архітектури системи за допомогою методів об’єктно-орієнтованого програмування та програмної інженерії;

– створення вебдодатків за допомогою PHP Laravel та VueJS;

– тестування системи для перевірки її функціональності, надійності та безпеки;

– розгортання системи за допомогою Docker;

– написання документації, яка описує архітектуру, функціональні можливості та інструкції з використання системи.

**Об’єкт дослідження.** Об’єктом дослідження є системи дистанційного керування доступом у багатоквартирних та багатоповерхових будівлях.

**Предмет дослідження.** Предметом дослідження є методи та засоби розробки системи «розумного» входу, яка забезпечує ефективне управління доступом, високий рівень безпеки та зручність використання.

**Методи дослідження.** У дослідженні використовувалися методи об’єктно-орієнтованого програмування та програмної інженерії.

**Наукова новизна одержаних результатів.** Наукова новизна полягає в розробці інноваційної системи контролю доступу, яка відповідає потребам сучасного міста, забезпечуючи ефективне управління доступом, безпеку та зручність використання в багатоквартирних будинках.

**Практичне значення одержаних результатів.** У результаті цієї роботи було створено ефективну та надійну систему дистанційного керування доступом до будинку «SmartEntry», яка відповідає сучасним вимогам безпеки та зручності користувачів у вході до будинку.

**Апробація одержаних результатів.** Результати дослідження були представлені на XV всеукраїнській, XXI регіональній науковій конференції молодих дослідників Запорізького національного університету «Актуальні проблеми математики та інформатики».

# 1 ТЕОРЕТИЧНА ЧАСТИНА

## 1.1 Аналіз літератури

Тема дистанційного керування доступом до будинку є актуальною в контексті швидкого розвитку «розумних» технологій, які впроваджуються в побутове середовище. Зростання інтересу до цієї теми обумовлене прагненням користувачів до підвищення зручності, безпеки та енергоефективності у своїх будинках. Із збільшенням популярності «розумних будинків» виникає потреба в розробці ефективних систем керування доступом, які забезпечували користувачів не лише безпекою, але й зручністю та надійністю.

В рамках вивчення літератури по темі було розглянуто 10 статей з кількох підтем.

**Опитування користувачів щодо систем «розумного будинку».**

Стаття [1] аналізує сприйняття користувачами переваг та ризиків «розумних» технологій для дому. Автори визначають позитивне визнання функціональності таких технологій, але також вказують перешкодою проблему автономності та приватності. Стаття розглядає рекомендації для підтримки розвитку ринку розумних технологій для дому.

У статті [2] визначається важливість контролю доступу в «розумних» будинках для забезпечення конфіденційності та безпеки. Автори розглядають існуючі схеми контролю доступу та вказують на недоліки, які потребують уваги. У статті вказано на необхідність більш гнучких та динамічних систем контролю доступу для розумних будинків.

**Технології та принципи для розробки систем «розумного будинку».**

Стаття [3] пропонує систематичний огляд застосувань IoT в «розумних» будинках. Автори провели дослідження, під час якого визначили чотири класи статей, що відображають різноманітні аспекти застосування IoT в розумних будинках. Вони звернули увагу на мотивацію використання IoT в «розумних»

будинках, відкриті виклики та рекомендації щодо поліпшення прийняття та використання додатків для розумного будинку.

Стаття [4] зосереджена на архітектурі та технологіях для розумних будинків. Автори аналізують його характеристики, складові та обладнання, застосування якого є ключовим. У статті також описано ключові технології для розумного будинку та надано висновки щодо досягнень у цій галузі.

Стаття [5] пропонує принципи керування для розумних будинків, акцентуючи увагу на потребах родин та принципах гнучкості. Автори виокремлюють сім принципів, які вони вважають ключовими для побудови систем керування розумним будинком, зокрема, принципи еластичності та підтримки винятків та сліпі точки.

**Розробка систем доступу до будинків.**

Стаття [6] зосереджується на розробці системи керування доступом до будинку за допомогою IoT. Вона відзначається використанням сучасних технологій, таких як Bluetooth та Arduino UNO, для створення механізму безпеки, що можна керувати за допомогою смартфона. Авторами впроваджено систему контролю доступу на основі мікроконтролера Arduino, що дозволяє забезпечити безпеку за доступною ціною та без значних витрат на конструкцію та монтаж.

У статті [7] розглядається розробка системи смарт-замка для дверей, заснованої на технології IoT. Вона відзначається використанням бездротового зв’язку Wi-Fi та Android-програмного забезпечення для віддаленого керування замком дверей за допомогою смартфона. У статті наголошено на використанні бездротових технологій для підвищення зручності та безпеки в будинку.

У статті [8] досліджується важливість розумних систем доступу до будинку для розширення їхнього застосування за межами дому. Автори статті наголошують на потребі створення систем доступу, які дозволяють спільноті забезпечувати доступ до смарт-пристроїв ззовні будинку. Ця стаття вирізняється своїм підходом до розширення області застосування смарт-систем доступу та акцентує на потребі спільнотного керування доступом.

Стаття [9] описує систему безпеки для розумного будинку, що базується на технології IoT та керується за допомогою смартфона. Вона відрізняється використанням різноманітних сенсорів та модулів для виявлення руху, зйомки зображень та контролю доступу до дверей. Ця стаття відзначається своїм комплексним підходом до забезпечення безпеки в будинку та використанням сучасних засобів зв'язку для віддаленого керування системою.

У статті [10] представлена інтегрована система розумного доступу до квартири, що базується на технології IoT. Особливістю цієї статті є наголос на використанні смартфона для керування доступом та контролю стану дверей, що спрощує процес та забезпечує зручність користувачам. Автори статті відзначають, що ця система може підвищити безпеку та комфорт в апартаментах за допомогою IoT технологій.

Аналіз статей свідчить про значний інтерес до теми «розумних» систем для побутового використання, зокрема систем дистанційного керування доступом до будинків. Основні підтеми, вивчені у дослідженнях, включають сприйняття користувачами ризиків та переваг «розумних» технологій, аналіз технологій та принципів розробки систем «розумного» будинку, а також безпосередню розробку систем доступу до будинків на базі IoT. Більшість досліджень зосереджена на приватних будинках, проте, враховуючи те, що в Україні значна частина населення проживає в багатоквартирних будинках, розвиток теми систем «розумного» доступу для такого типу будинків є перспективним та актуальним напрямком для подальших досліджень. Такий підхід може сприяти підвищенню безпеки та зручності у житті мешканців багатоквартирних будинків, що є важливим аспектом соціального та технологічного розвитку.

## 1.2 Постановка задачі

У рамках цього проєкту було прийнято рішення поділити систему на два окремих додатки: адміністративну панель та клієнтський додаток. Це дозволяє

чітко розділити функції управління та безпосереднього користування системою.

Адміністративна панель забезпечує керування всіма аспектами системи, включаючи управління користувачами, об’єктами та правами доступу. Розроблювана адміністративна панель повинна забезпечувати кілька ключових функцій.

Серед них – підтримка аутентифікації та авторизації, що дозволяє адміністраторам входити в систему за допомогою ідентифікатора та пароля. Повинно бути передбачено також розмежування рівнів доступу залежно від ролей користувачів.

Застосунок має надавати можливості для ефективного управління користувачами. Це включає додавання, редагування та видалення інформації про них, а також призначення адміністраторів для обслуговування певних будинків або районів.

Крім цього, система повинна забезпечувати інструменти для роботи з об’єктами, такими як будинки, входи чи квартири. Адміністратори зможуть додавати, змінювати чи видаляти відповідну інформацію, а також визначать відповідальних осіб для конкретних об’єктів.

Функціонал управління доступом передбачає надання або обмеження прав користувачів на вхід до певних будинків чи територій. Крім того, система фіксуватиме й аналізуватиме події, пов’язані з доступом.

Окремо має бути реалізовано журнал дій, який фіксуватиме всі зміни в базі даних і дії, пов’язані з керуванням доступом. Цей інструмент забезпечить прозорість та контроль за операціями в системі.

Клієнтський додаток, в свою чергу, призначений для мешканців та відвідувачів.

Клієнтський додаток для системи дистанційного керування доступом до будинку повинен включати декілька функціональних можливостей.

Для аутентифікації та авторизації повинно бути передбачено реєстрацію користувачів за допомогою електронної пошти або номера телефону. Вхід до

системи здійснюється за логіном і паролем, а підтвердження реєстрації відбувається через одноразовий код, надісланий на пошту чи телефон. У разі втрати пароля користувачі можуть скористатися функцією його відновлення.

Додаток також має дозволяти керувати профілем, надаючи можливість змінювати особисту інформацію, наприклад, ім’я та контактні дані.

Щодо доступу до об’єктів, користувачі мають бути в змозі подавати заявки на отримання статусу жителя конкретної квартири, відкривати двері до власного помешкання, переглядати список заблокованих відвідувачів із можливістю їх розблокування, а також переглядати журнал подій, пов’язаних з їх квартирою (-ами).

Взаємодія мешканців із відвідувачами має включати отримання викликів через додаток, перегляд відео та аудіо від відвідувача в режимі реального часу, блокування небажаних відвідувачів і відкриття дверей для них за необхідності.

Ці функції мають забезпечувати користувачам зручність, безпеку та контроль над доступом до їхньої квартири.

Чіткий поділ системи на адміністративну панель та клієнтський додаток дозволяє забезпечити високий рівень безпеки та зручності для всіх користувачів, що є ключовим для успішного впровадження сучасної системи дистанційного керування доступом до будинку.

### 1.3 Аналіз існуючих аналогів

Щоб повністю оцінити актуальність нової системи дистанційного контролю доступу до будинку та її новизну, необхідно провести ретельний огляд існуючих аналогів. Дослідження сучасних систем допоможе визначити їхні сильні і слабкі сторони та виявити ті області, які ще не розвинуті. Нижче представлено огляд деяких популярних систем дистанційного доступу до будинку.

August Smart Lock дозволяє користувачеві віддалено відкривати і закривати двері через додаток на смартфонах. Підтримує інтеграцію з голосовими асистентами, такими як Alexa і Google, а також з іншими рішеннями смарт-дому. Головним недоліком є невелика можливість інтеграції з іншими системами безпеки. Система не може здійснювати комплексний моніторинг, як, наприклад, відеоспостереження або контроль над станом всього будинку.

Schlage Encode – смарт-замок, оснащений вбудованим Wi-Fi, що дозволяє дистанційно керувати дверима без необхідності додаткового хабу. Schlage Encode підтримує інтеграцію з Amazon Key для доставки товарів безпосередньо до будинку. Але, незважаючи на високу зручність використання, система не має підтримки різних сценаріїв доступу та обмежену гнучкість налаштувань для користувачів.

Real Living Assure Lock пропонує широку інтеграцію з розумними системами через Z-Wave та Zigbee. Система дозволяє дистанційне керування замком та підтримує мобільні додатки. Але, у цій системі відсутнє комплексне рішення для управління доступом та інтеграції з іншими системами безпеки, що залишає можливості для покращення. Також присутні складнощі з налаштуванням системи для непідготовлених користувачів.

Ring Alarm – це комплексна система безпеки, що включає датчики руху, відеокамери та смарт-замки. Користувачі можуть контролювати систему через мобільний додаток та отримувати сповіщення про будь-які події. Хоча ця система пропонує широкий спектр функцій, інтеграція з різними системами доступу іноді може викликати проблеми. Крім того, Ring Alarm має високу вартість.

Система SimpliSafe забезпечує повний спектр функцій для домашньої безпеки, включаючи сенсори для дверей та вікон, камери спостереження та можливість віддаленого контролю. Незважаючи на комплексний підхід, ця система не надає спеціалізованих рішень для керування доступом, що може бути важливо для користувачів, які шукають гнучкі рішення для різних сценаріїв доступу.

Таким чином, аналіз існуючих рішень показує, що більшість сучасних систем пропонують хороші функціональні можливості, проте мають певні обмеження, які залишають простір для інновацій. Основні недоліки включають обмежену інтеграцію з іншими системами безпеки, відсутність гнучкості у налаштуваннях та високий вартісний бар'єр для входу. Відтак, розроблювана система може запропонувати унікальні переваги та задовольнити важливі потреби користувачів завдяки кільком важливим аспектам.

Система буде орієнтована на простоту у використанні, що дозволить залучити ширше коло користувачів, включаючи мешканців багатоквартирних будинків в Україні.

Система забезпечить централізоване керування доступом до будинку, що дозволить більш ефективно контролювати та перевіряти тих, хто має доступ до під'їзних дверей. Це підвищить рівень безпеки та забезпечить зручність для користувачів.

Система буде розроблена з урахуванням доступності для широкого кола користувачів, що дозволить знизити вартісний бар'єр для входу та спростить процес встановлення.

Вбудована функція відеодзвінків дозволить користувачам переконатися в особі відвідувача перед тим, як надати доступ, що значно підвищить рівень безпеки.

Таким чином, розроблювана система дистанційного керування доступом до будинку є досить унікальною та необхідною, оскільки вона відповідає наявним викликам та задовольняє потреби користувачів, які залишаються не вирішеними існуючими рішеннями.

### 1.4 Огляд засобів розробки

При розробці програмної системи будуть використовуватися потужні технології та бібліотеки для забезпечення ефективності, безпеки та зручності

управління та розробки. Вибір вказаних засобів зумовлений їхньою популярністю, готовністю рішень та здатністю ефективно вирішувати конкретні завдання адміністративної частини системи «Smart Entry». Кожен інструмент вносить свій внесок у зручність та швидкість розробки, безпеку системи та швидкість впровадження нового функціоналу.

### 1.4.1 Laravel

Laravel є одним з найпопулярніших PHP-фреймворків, завдяки своїй гнучкості, потужності та зручності використання. Ця платформа швидкої розробки додатків та її інструменти дозволяють швидко створювати нові сайти та додатки з чистим, зрозумілим кодом [11, 12].

Laravel має декілька ключових компонентів.

Одним із найважливіших інструментів є Eloquent ORM, який дозволяє зручно працювати з базами даних завдяки інтуїтивному синтаксису для виконання CRUD-операцій та складних SQL-запитів. Це робить код чистим і зрозумілим.

Крім того, Laravel підтримує автоматичну генерацію коду через утиліту Artisan, яка спрощує створення міграцій, моделей, контролерів і форм, прискорюючи початок роботи над проєктом.

У фреймворк вбудовано систему аутентифікації, яка забезпечує швидку реалізацію реєстрації, входу, скидання паролів і авторизації користувачів.

Для розширення функціоналу можна інтегрувати сторонні бібліотеки та пакети через Composer, додаючи нові можливості до проєкту.

Велика увага приділяється документації та підтримці спільноти, що робить освоєння фреймворку зручним, а розв’язання проблем – швидким завдяки численним ресурсам і активним форумам.

Фреймворк забезпечує високу швидкість розробки, оскільки автоматизує багато процесів і надає готові компоненти.

Код, створений у Laravel, вирізняється чистотою та зрозумілістю, оскільки фреймворк заохочує використання кращих практик програмування.

Його гнучкість і розширюваність дозволяють адаптувати платформу до будь-яких потреб за допомогою модулів і пакетів.

Laravel також приділяє особливу увагу безпеці, захищаючи від SQL-ін’єкцій, CSRF-атак та інших поширених загроз.

Нарешті, активна спільнота забезпечує обмін досвідом, швидке розв’язання проблем і постійний розвиток фреймворку, роблячи його одним із найпопулярніших інструментів для веброзробки.

Таким чином, Laravel є чудовим вибором для розробки сучасних вебдодатків, забезпечуючи баланс між простотою використання, продуктивністю та функціональністю.

### 1.4.2 Spatie/laravel-permissions

Spatie Laravel Permissions – це одна з найпопулярніших бібліотек для Laravel, яка спрощує управління ролями та дозволами в вебдодатках. Вона дає змогу створювати, змінювати й видаляти ролі та дозволи, а також призначати їх користувачам, забезпечуючи організовану структуру доступів. Завдяки можливості групування ролей і дозволів, управління стає ще більш зрозумілим і ефективним.

Ця бібліотека підтримує перевірку дозволу на доступ до функцій або розділів додатка на основі ролей і дозволів користувачів, використовуючи middleware та blade-директиви для легкої інтеграції. Кешування дозволів значно підвищує продуктивність додатка, а простий API забезпечує зручність у використанні навіть у складних або інтеграцію у вже існуючі проєкти.

Spatie Laravel Permissions активно оновлюється, залишаючись сумісною з новими версіями Laravel, і має розгорнуту документацію та велику спільноту розробників. Це робить її ідеальним вибором для будь-якого проєкту, де важливе гнучке та ефективне управління доступом користувачів.

### 1.4.3 Owen-it/laravel-auditing

Owen-it/laravel-auditing – це функціональна бібліотека для Laravel, що допомагає реалізувати аудит змін у моделях. Вона автоматично відстежує всі модифікації в базі даних і створює записи, які містять інформацію про старі та нові значення атрибутів, ідентифікатор користувача, що здійснив зміни, а також часові мітки [13].

Цей інструмент спрощує контроль за історією змін, пропонуючи гнучкі налаштування, які дозволяють вибірково обирати атрибути для аудиту. Простота інтеграції та зрозуміла документація дають змогу швидко впровадити бібліотеку у проєкт. Завдяки докладним аудиторським записам розробники отримують прозорий огляд дій користувачів, що підвищує безпеку даних і відповідає вимогам, таким як GDPR. Крім того, збережені дані можна використовувати для відновлення попередніх версій у разі потреби.

Ця бібліотека є важливим інструментом для проєктів на Laravel, де необхідно забезпечити ефективне відстеження змін і безпеку інформації.

### 1.4.4 Moonshine

Moonshine – це потужна і функціональна адміністративна панель, спеціально створена для використання у проєктах, побудованих на основі Laravel. Вона забезпечує простий і швидкий доступ до управління даними, дозволяючи розробникам з легкістю створювати інтуїтивні та ефективні інструменти адміністрування для своїх вебдодатків. Завдяки цьому інструменту можна не лише оптимізувати процеси розробки, а й значно полегшити подальше адміністрування, зберігаючи при цьому високі стандарти продуктивності та захисту.

Однією з ключових особливостей Moonshine є її інтеграція з Laravel. Це означає, що панель органічно вписується в екосистему фреймворку,

використовуючи всі його можливості. Для розробників, які вже знайомі з Laravel, налаштування та робота з Moonshine буде максимально інтуїтивною та зрозумілою.

Особливу увагу розробники Moonshine приділили дизайну інтерфейсу. Сучасний, зручний і логічно організований користувацький інтерфейс дозволяє швидко знаходити потрібні інструменти та виконувати завдання без зайвих зусиль. Це робить процес управління даними комфортним навіть для новачків.

Moonshine вирізняється високою масштабованістю, що робить її універсальним рішенням для проєктів будь-якого масштабу. Від невеликих вебсайтів до складних корпоративних систем – панель здатна забезпечувати стабільну продуктивність навіть при роботі з великими обсягами даних. Завдяки модульній архітектурі її функціональність легко адаптується під конкретні потреби проєкту. Наприклад, можна додавати нові модулі, змінювати існуючі функції або інтегрувати сторонні рішення для досягнення бажаного результату.

Безпека також є пріоритетом у Moonshine. Панель використовує надійні механізми аутентифікації та авторизації, які надає Laravel, що забезпечує високий рівень захисту даних. Такий підхід гарантує, що конфіденційна інформація залишається під надійним захистом навіть у випадку складних проєктів із підвищеними вимогами до безпеки.

Крім того, Moonshine підтримується активною спільнотою розробників, що постійно вдосконалює цей інструмент, додаючи нові можливості та оновлення.

Таким чином, Moonshine є універсальним і потужним рішенням для створення адміністративних панелей у проєктах на Laravel. Завдяки своїй зручності, гнучкості та безпеці, вона стає незамінним інструментом для ефективного управління даними та забезпечення високої якості вебдодатків.

### 1.4.5 VueJS

Vue.js – це прогресивна JavaScript-бібліотека для створення користувацьких інтерфейсів. Вона лежить в основі екосистеми, яка дозволяє створювати потужні клієнтські додатки [14]. Вона створена з метою бути легкою для інтеграції в існуючі проєкти і одночасно достатньо потужною для розробки складних односторінкових додатків.

Основна ідея Vue.js полягає в інкрементальній прийнятності, що робить її універсальним інструментом як для створення нових проєктів, так і для поступового покращення вже існуючих. Цей підхід дозволяє розробникам впроваджувати бібліотеку поступово, починаючи з невеликих компонентів і поступово розширюючи її використання. Така стратегія дає змогу інтегрувати Vue.js у проєкти різної складності, забезпечуючи гнучкість та зручність.

Однією з ключових переваг Vue.js є її простота в освоєнні. Завдяки інтуїтивно зрозумілому синтаксису та докладній документації, навіть новачки можуть швидко розпочати роботу з цією бібліотекою. Численні приклади й активна підтримка спільноти сприяють легкому входженню у розробку. Особливу увагу розробники Vue.js приділяють реактивній системі, яка автоматично синхронізує стан даних із відображенням на сторінці. Це дозволяє створювати інтерактивні інтерфейси, мінімізуючи ручне оновлення DOM та дозволяючи зосередитися на бізнес-логіці програми.

Компонентний підхід, що лежить в основі Vue.js, значно спрощує розробку та підтримку додатків. Кожен компонент є незалежним модулем, що містить свою логіку, шаблони та стилі, що полегшує тестування та повторне використання коду. Така модульна архітектура робить Vue.js ідеальним для створення масштабованих додатків, адже вона дозволяє розбивати складні системи на керовані частини.

Екосистема Vue.js включає численні офіційні інструменти, такі як Vuex для керування станом, Vue Router для маршрутизації та Vue Native для розробки мобільних додатків. Ці інструменти забезпечують розширену функціональність і легко інтегруються в будь-який проєкт. Окрім цього, Vue.js

демонструє високу продуктивність завдяки використанню віртуального DOM, який мінімізує кількість змін у реальному DOM, забезпечуючи швидкодію навіть у складних додатках.

Однією з найсильніших сторін Vue.js є її велика й активна спільнота. Розробники постійно створюють нові плагіни, бібліотеки та інструменти, а також діляться своїм досвідом, що допомагає знаходити вирішення будь-яких технічних викликів. Завдяки цьому Vue.js отримує постійне оновлення та залишається актуальною для сучасних вебтехнологій.

Гнучкість цієї бібліотеки робить її однаково зручною для малих проєктів і великих систем. Почати можна з розробки одного невеликого компонента, поступово розширюючи функціональність додатка, не змінюючи архітектуру значною мірою. Vue.js дозволяє розробникам створювати сучасні, динамічні вебдодатки, зберігаючи при цьому простоту, ефективність і масштабованість.

Завдяки своїм перевагам та постійному вдосконаленню Vue.js залишається одним із найбільш популярних інструментів для фронтенд-розробки, що відкриває широкі можливості для створення інноваційних вебрішень.

### 1.4.6 Inertia

Inertia.js – це сучасна бібліотека для розробки вебдодатків, яка дозволяє створювати єдину сторінку додатка (SPA) з використанням серверного рендерингу. Inertia.js не є фреймворком, а швидше інструментом, що допомагає об’єднати серверну і клієнтську частину додатка в єдине ціле. Вона працює як міст між традиційними серверними фреймворками (наприклад, Laravel, Rails) і сучасними клієнтськими фреймворками (React, Vue, Svelte) [16].

Основна ідея Inertia.js полягає в тому, щоб зменшити кількість дублікації коду і спростити процес розробки, залишаючи можливість використовувати

всі переваги серверного рендерингу разом з динамічністю і інтерактивністю клієнтських SPA.

Основною перевагою Inertia.js є спрощений процес розробки, що дозволяє уникнути дублювання коду на серверній та клієнтській стороні. Завдяки цьому розробники можуть писати менше коду і зосередитися на створенні функціоналу додатка.

Inertia.js легко інтегрується з існуючими серверними фреймворками, такими як Laravel, Rails, Django, і клієнтськими фреймворками, як-от React, Vue, Svelte. Це дозволяє розробникам використовувати знайомі інструменти і технології.

Також Inertia.js забезпечує повноцінний серверний рендеринг, що покращує SEO і зменшує час завантаження сторінки. Це особливо корисно для великих додатків, які потребують швидкого рендерингу.

Завдяки використанню Inertia.js, навігація по додатку стає більш плавною і швидкою, оскільки він завантажує лише необхідні дані і не перезавантажує всю сторінку.

Важливим є те, що Inertia.js зменшує кількість залежностей у проєкті, оскільки не потребує використання додаткових бібліотек для управління станом чи роутингом на клієнтській стороні.

Також, завдяки Inertia.js, серверні валідації стають більш інтегрованими у процес розробки, що дозволяє ефективніше обробляти помилки і покращує загальну стабільність додатка.

Використання Inertia.js забезпечує високу реактивність додатка, оскільки він швидко оновлює тільки ті частини сторінки, які змінилися, без необхідності повного перезавантаження.

Inertia.js є потужним інструментом для розробників, які прагнуть об’єднати переваги серверного і клієнтського рендерингу, створюючи швидкі, інтерактивні і зручні вебдодатки.

### 1.4.7 Agora Video SDK

Agora’s Video SDK – це програмний набір для розробників, створений для інтеграції високоякісних відеокомунікацій у мобільні, веб та десктопні додатки. SDK надає інструменти для створення відеоконференцій, стрімінгу та відеодзвінків з мінімальною затримкою і високою якістю зв’язку [17].

Agora’s Video SDK пропонує низку переваг, що роблять його потужним інструментом для розробників. Серед них – висока якість відео та аудіо: підтримка HD-відео до 1080p, автоматичне налаштування якості залежно від пропускної здатності мережі та низька затримка для забезпечення реального часу спілкування.

Рішення легко масштабується, дозволяючи працювати як з невеликими відеодзвінками, так і з великими конференціями, підтримуючи при цьому велику кількість учасників. Завдяки кросплатформенності SDK працює на iOS, Android, Windows, macOS та Web, надаючи розробникам єдиний API для всіх платформ, що значно спрощує інтеграцію.

Інструмент також забезпечує можливості інтеграції з іншими сервісами через API, а також підтримує сторонні плагіни та додатки для розширення функціоналу. Безпека є пріоритетом: шифрування відео та аудіо захищає дані, а функції аутентифікації обмежують доступ до сесій.

Крім цього, SDK пропонує інтерактивні можливості, такі як обмін екраном, інтерактивні дошки, обмін файлами, реакції в реальному часі та чати. Завдяки цим функціям Agora’s Video SDK стає ідеальним вибором для створення сучасних, інтерактивних рішень у різних галузях.

## 1.5 Огляд засобів розгортання

Для розгортання багатокомпонентних систем, які складаються з кількох незалежних частин, важливо забезпечити стабільну інфраструктуру, що

дозволяє мінімізувати ручні налаштування, спрощує оновлення, підтримує легке масштабування та автоматично відновлюється у разі збою. Docker і Kubernetes є перевіреними інструментами для таких завдань, надаючи все необхідне для управління розподіленими компонентами системи.

### 1.5.1 Docker

Docker – це платформа для контейнеризації, яка дозволяє ізолювати кожен компонент системи в окремому контейнері. Цей підхід дає можливість зосередитися на конкретному середовищі кожної частини системи, гарантуючи стабільність та простоту в управлінні [18, 19].

Використання Docker для розгортання багатокомпонентних систем має низку вагомих переваг.

Передусім, Docker забезпечує уніфіковане середовище виконання, де кожен компонент, наприклад, адміністративна панель чи клієнтський застосунок, упакований у власний контейнер із усім необхідним для роботи. Це мінімізує проблеми несумісності, оскільки контейнери функціонують однаково у будь-якому середовищі – від локальних серверів до production-середовищ.

Також Docker пропонує гнучкість в оновленні та розгортанні: кожен компонент можна оновлювати, тестувати чи деплоїти незалежно від інших. Наприклад, зміни в адміністративній панелі можна внести без впливу на клієнтський застосунок, що знижує ризик збоїв під час оновлень.

Мобільність контейнерів є ще однією важливою перевагою. Оскільки Docker не залежить від інфраструктури, контейнери легко переміщуються між платформами, забезпечуючи передбачувану продуктивність у будь-якому середовищі. Це значно спрощує життя розробникам, дозволяючи швидко запускати проєкти.

Крім того, Docker оптимізує використання ресурсів завдяки легкій структурі контейнерів. Це дозволяє одночасно працювати з кількома контейнерами на одному сервері з мінімальним навантаженням, забезпечуючи високу продуктивність усіх компонентів системи.

### 1.5.2 Kubernetes

Kubernetes – це платформа для оркестрації контейнерів, що дозволяє автоматизувати розгортання, масштабування та контроль стану додатків, запущених у контейнерах. Застосування Kubernetes у багатокомпонентних системах надає значні переваги в управлінні складними архітектурами [20, 21].

Kubernetes пропонує ефективні рішення для управління контейнеризованими додатками завдяки своїм ключовим перевагам. Одна з них – масштабованість на основі навантаження: платформа автоматично збільшує або зменшує кількість контейнерів залежно від попиту. Наприклад, у разі великого обсягу запитів до клієнтського застосунку Kubernetes оперативно додає екземпляри, забезпечуючи стабільну роботу навіть за пікових навантажень.

Ще однією важливою перевагою є автоматичне відновлення. Kubernetes моніторить стан контейнерів і автоматично перезапускає їх у разі збоїв, гарантуючи високу доступність системи. Балансування навантаження також є невід’ємною складовою: платформа рівномірно розподіляє навантаження між контейнерами, уникаючи перевантаження окремих компонентів і забезпечуючи стабільну продуктивність.

Крім того, Kubernetes спрощує управління конфігураціями та секретами. Вбудовані механізми дозволяють безпечно зберігати та використовувати ключі доступу, конфігураційні файли та інші чутливі дані, зменшуючи необхідність ручного управління.

Поєднання можливостей Docker і Kubernetes створює надійне, масштабоване й автоматизоване середовище, яке ідеально підходить для розгортання складних систем.

### 1.5.3 Хмарні платформи

Застосування хмарних платформ для розгортання багатокомпонентних систем на основі Docker і Kubernetes є ефективним рішенням, яке надає додаткові можливості для управління інфраструктурою та забезпечує високу доступність і масштабованість системи. Використання хмарної інфраструктури дозволяє автоматизувати ключові аспекти розгортання та обслуговування, знижуючи витрати на підтримку та управління фізичними серверами.

Використання хмарних платформ для розгортання має низку переваг. По-перше, вони забезпечують масштабованість на вимогу, що дозволяє адаптувати ресурси залежно від навантаження. Це особливо корисно для систем із періодами змінного попиту, оскільки забезпечує ефективне використання ресурсів і знижує витрати. По-друге, платформи пропонують автоматичне резервування та відновлення, завдяки чому система залишається доступною навіть за умов збоїв. Крім того, хмарні сервіси дозволяють автоматизувати оновлення, налаштування та контроль інфраструктури, надаючи адміністратору змогу зосередитися на розвитку додатків. Нарешті, моделі оплати за фактичне споживання ресурсів допомагають оптимізувати витрати, що особливо цінно для змінних за інтенсивністю проєктів.

Серед основних хмарних платформ виділяються Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure.

AWS пропонує послуги, такі як Amazon Elastic Kubernetes Service (EKS) і Amazon Elastic Container Service (ECS), із акцентом на безпеку, резервування та глобальну доступність.

GCP забезпечує глибоку інтеграцію Google Kubernetes Engine (GKE) з іншими сервісами Google, зокрема в галузі штучного інтелекту та машинного навчання.

Microsoft Azure зі своїм Azure Kubernetes Service (AKS) надає високий рівень інтеграції з екосистемою Microsoft, підтримує безперервне розгортання та забезпечує гнучке управління доступом до ресурсів.

Таким чином, використання хмарних платформ сприяє ефективному управлінню інфраструктурою, забезпечує безперебійну роботу додатків, спрощує масштабування і допомагає підтримувати конкурентоспроможність системи в умовах динамічного ринку.

## 1.6 Висновки до розділу 1

Таким чином, у 1 розділі було проведено аналіз статей з теми і встановлено, що існує значний інтерес до теми «розумних» систем взагалі та зокрема до систем дистанційного керування доступом до будинків. Але, так як не було жодної роботи, що б описувала використання таких систем у багатоквартирних квартирах, було вирішено зосередитися на розробці саме такої системи. Після цього, було поставлено задачу, вирішено поділити систему на два окремих додатки: адміністративну панель та клієнтський додаток, а також встановлено їхні функціональні вимоги. У розділі 3 було розглянуто засоби розробки, а саме Laravel, VueJS, Inertia, Agora Video SDK, Moonshine, Spatie/laravel-permissions та Owen-it/laravel-auditing. Було розказано про кожен засіб, а також вказано їх основні переваги. У останньому підрозділі було розглянуто сучасні засоби для розгортання багатокомпонентних систем, а саме Docker, Kubernetes та хмарні платформи.

# 2 ПРОЄКТУВАННЯ

## 2.1 Визначення прецедентів

На рисунку 2.1. представлено діаграму прецедентів для адміністративної частини розроблюваного проєкту. Як можна побачити, є 3 основних ролі адміністраторів – головна, яка може виконувати будь які операції з сутностями, менеджер району, який може керувати сутностями в межах свого району (що являє собою об’єднання вулиць) та менеджер будинку, що відповідає за сутності в межах підвладного будинку. Що до сутностей, то маються на увазі: країни, міста, райони, вулиці, будинки, входи, квартири та їх мешканці. Також мається на увазі те, що кожен адмін буде мати змогу відкривати двері до керованих ним будинків.

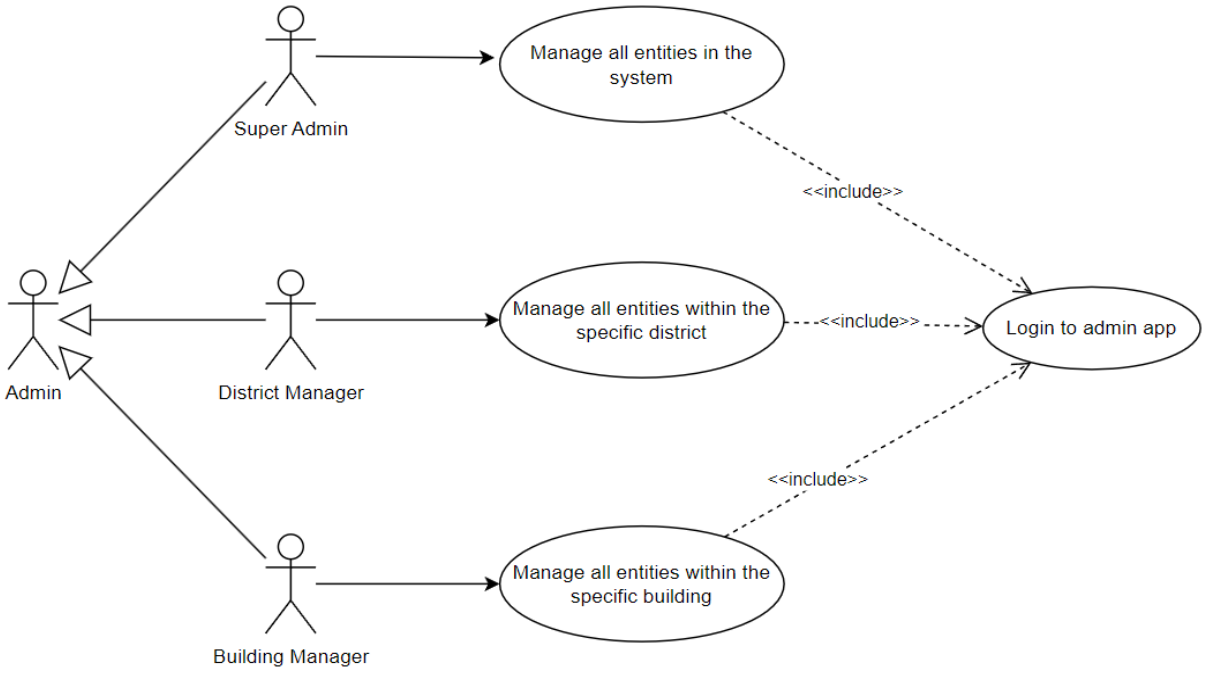



Рисунок 2.1 – Діаграма прецедентів адміністративної панелі

На рисунку 2.2 зображено діаграму прецедентів для клієнтського застоснунку. Як можна побачити, є 2 ролі: відвідувач та мешканець. Відвідувачем є будь який зареєстрований користувач. Він може переглядати

списки наявних під'їздів та квартир, а також зв'язуватися з мешканцем за допомогою відеодзвінка. Також, відвідувач може залишити заяву для отримання статусу мешканця. Після її обробки відповідною структурою (наприклад, керуючою компанією), відвідувач отримає роль мешканця. Мешканець – це користувач, що пов'язаний з однією або кількома квартирами. Він може приймати дзвінки від відвідувачів, блокувати чи впускати їх, а також відкривати двері.

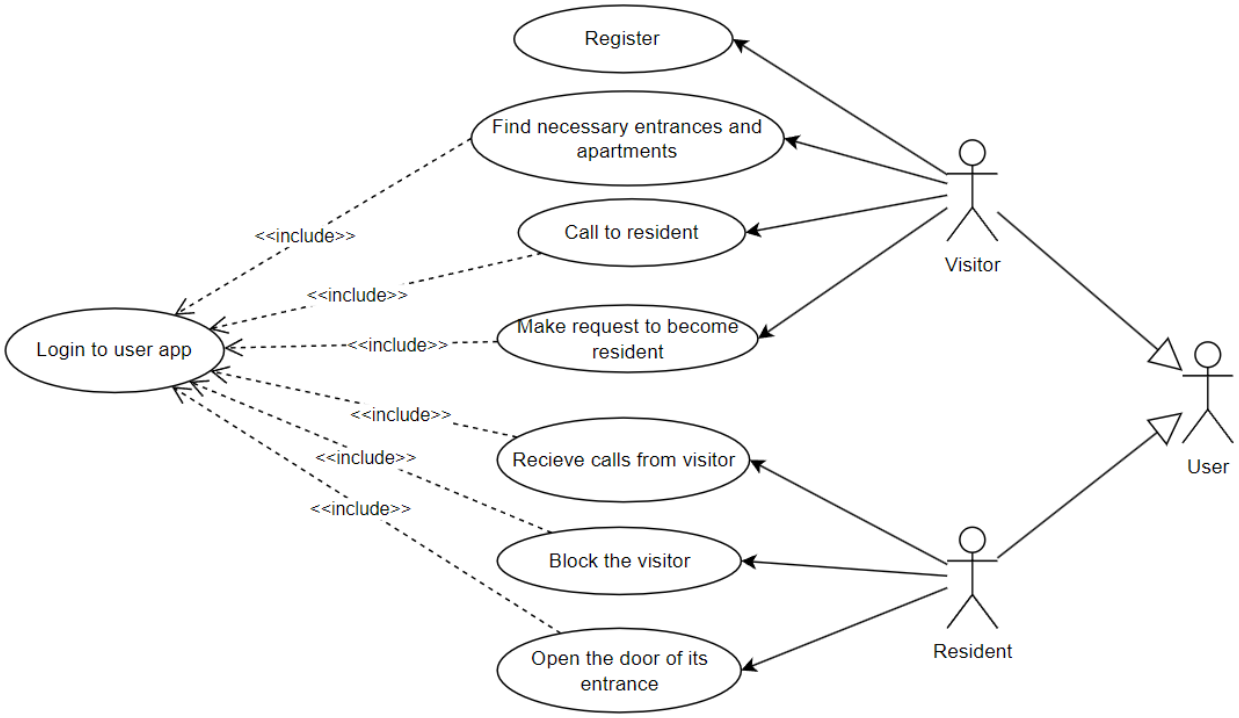



Рисунок 2.2 – Діаграма прецедентів клієнтського застосунку

## 2.2 Визначення схеми даних

Так як ER діаграма досить велика, то її буде представлено частинами.

На рисунку 2.3 зображено схему географічний даних. Цей блок описує географічну структуру розташування. Країни, міста, райони, вулиці та будівлі пов'язані між собою в ієрархії, що забезпечує зручне зберігання та організацію даних про об'єкти.




Рисунок 2.3 – ER-діаграма географічний даних

На рисунку 2.4 представлена інформація про входи (під’їзди), квартири та пристрої на дверях.

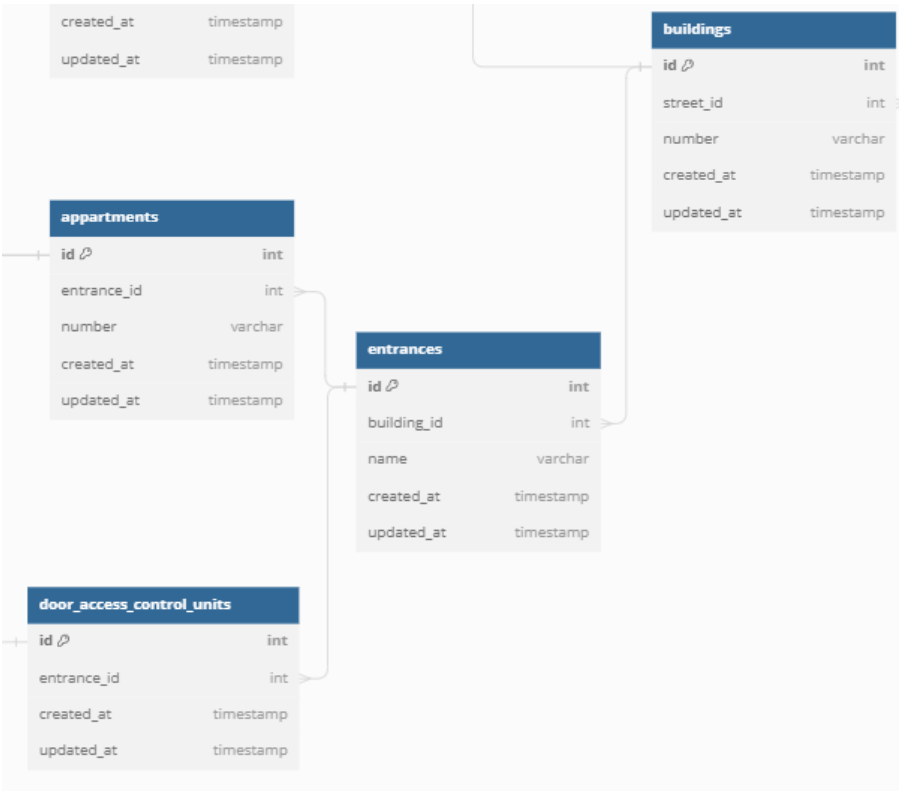



Рисунок 2.4 – ER-діаграма даних про будівлі

На рисунку 2.5 представлена інформація про керівників районів та будинків.

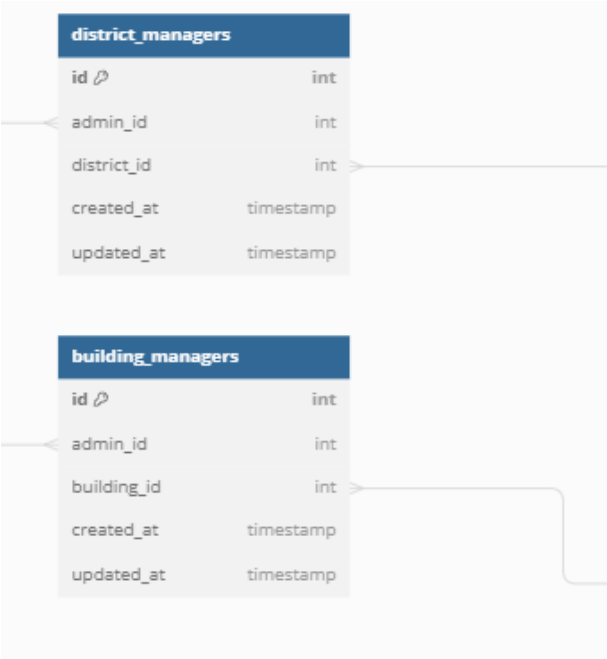



Рисунок 2.5 – ER-діаграма даних про відповідальних осіб

Далі мова буде йти про користувачів. На рисунку 2.6 можна побачити два типи користувачів – адміністраторів та звичайних користувачів. Було вирішено розділити їх дані, так як вони будуть користуватися різними платформами.

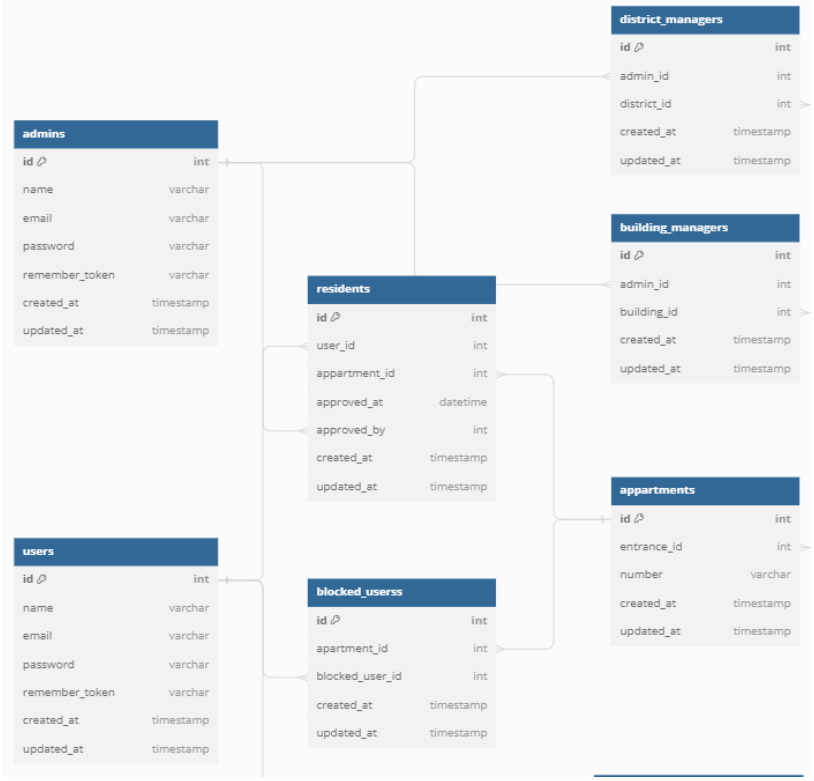



Рисунок 2.6 – ER-діаграма даних користувачів

Також представлено таблиці для розділення адмінів на менеджерів районів та окремих будинків, а також повїязано користувачів з їх квартирами та заблокованими користувачами.

Таким чином, спроєктовано схему даних, що дозволить забезпечити основний функціонал застосункам.

Останнім слід сказати про схему даних сутностей входу, контролеру та його логів (рис. 2.7). Спроєктована схема дозволить відстежувати операції з дверима та їх використання користувачами.

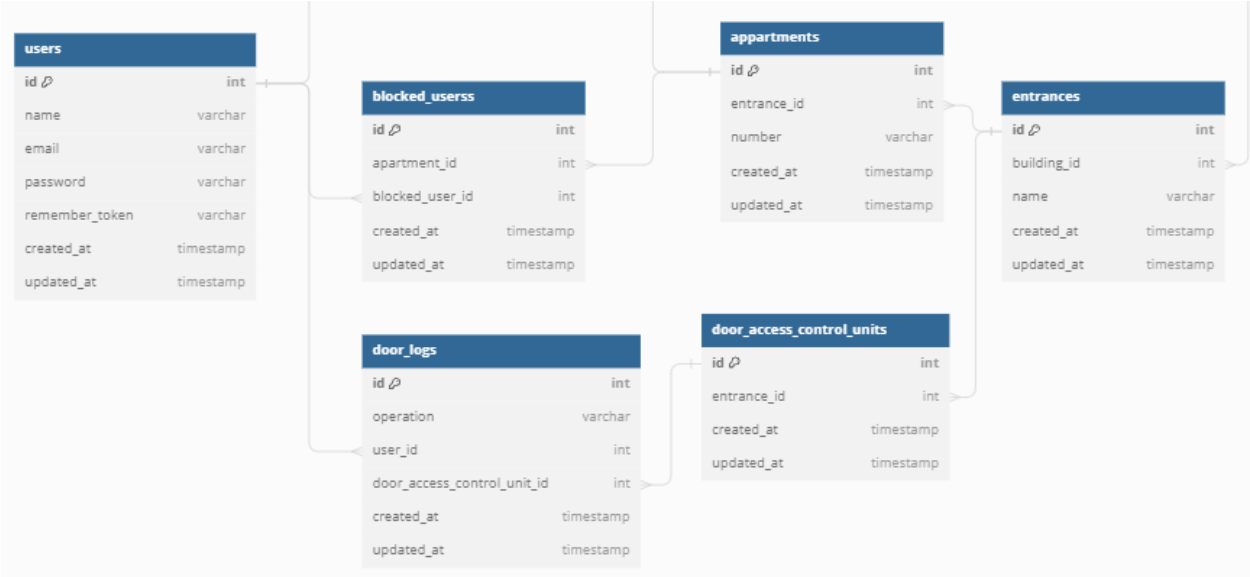



Рисунок 2.7 – ER-діаграма даних входу

Також слід розказати про допоміжні дані – перш за все, логи. Так як досить важливо знати, коли та хто саме здійснив ту чи іншу дію, то було вирішено спроєктувати окрему таблицю для контролю даних (рис. 2.8).

Також важливою частиною функціоналу є контроль доступу користувачів до даних та функціоналу. Для цього виокремлено окрему схему даних, де показано зв’язки щодо ролей та дозволів (рис. 2.9). Спроєктована схема даних визначає механізм контролю доступу, який дозволяє присвоювати ролі користувачам та визначати їхні права доступу до конкретних дій та даних в системі.




Рисунок 2.8 – Схема таблиці для логів

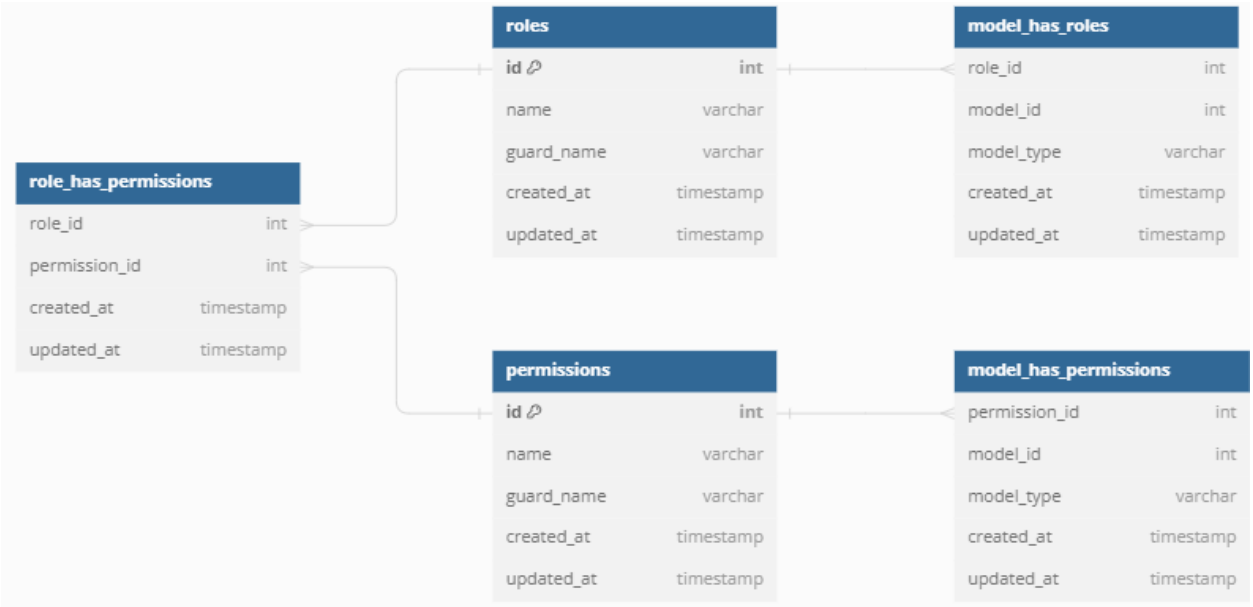



Рисунок 2.9 – ER-діаграма ролей та дозволів

## 2.3 Проєктування додатків

Як було зазначено, рамках даного дипломного проєкту буде розроблено вебдодатки на базі фреймворка Laravel. Laravel є одним із найпопулярніших та широко використовуваних PHP-фреймворків, який забезпечує розробників потужним і елегантним інструментом для створення вебдодатків із використанням архітектурного шаблону MVC (Model-View-Controller).

Архітектурний шаблон MVC розділяє додаток на три основні компоненти: модель (Model), представлення (View) і контролер (Controller). Такий підхід сприяє кращій структуризації коду, полегшує його супровід та масштабування.

Модель відповідає за керування даними, бізнес-логікою та правилами взаємодії з базою даних. У Laravel моделі визначаються за допомогою об’єктів, що наслідують клас «Eloquent». Eloquent ORM (Object-Relational Mapping) надає зручний інтерфейс для роботи з базами даних і дозволяє легко взаємодіяти з даними, використовуючи методи об’єктно-орієнтованого програмування.

Представлення відповідає за відображення даних користувачу. Воно отримує дані від контролера та форматує їх для виведення. У Laravel для створення представлень використовується шаблонізатор Blade, який дозволяє писати чистий і простий код з підтримкою динамічних даних та компонентів. Але, з Laravel можна використовувати і інші засоби для frontenfd розробки, наприклад VueJS.

Контролер діє як посередник між моделлю та представленням. Він отримує запити від користувача, обробляє їх за допомогою бізнес-логіки, визначеної в моделях, і повертає відповідні представлення. У Laravel контролери містять методи, які відповідають різним маршрутам та діям користувача.

Таким чином, архітектура адміністративної панелі та клієнтського застосунку будуть вельми схожі, але деталі ми розглянемо у наступних розділах.

### 2.3.1 Проєктування адміністративної панелі

Адміністративна панель розроблюваної системи буде використовувати Moonshine – бібліотека для створення таких панелей на Laravel. Детальніше про цю бібліотеку буде розказано у розділі про реалізацію, але зараз необхідно

зазначити, що структура проєкту буде дещо простішою за звичайний Laravel проєкт.

На рисунку 2.10 зображено частину діаграми класів, що відноситься до моделей. На ній можна побачити всі численні «один-до-багатьох» та не тільки, відношення між моделями, які були перенесені з ER-діаграми.

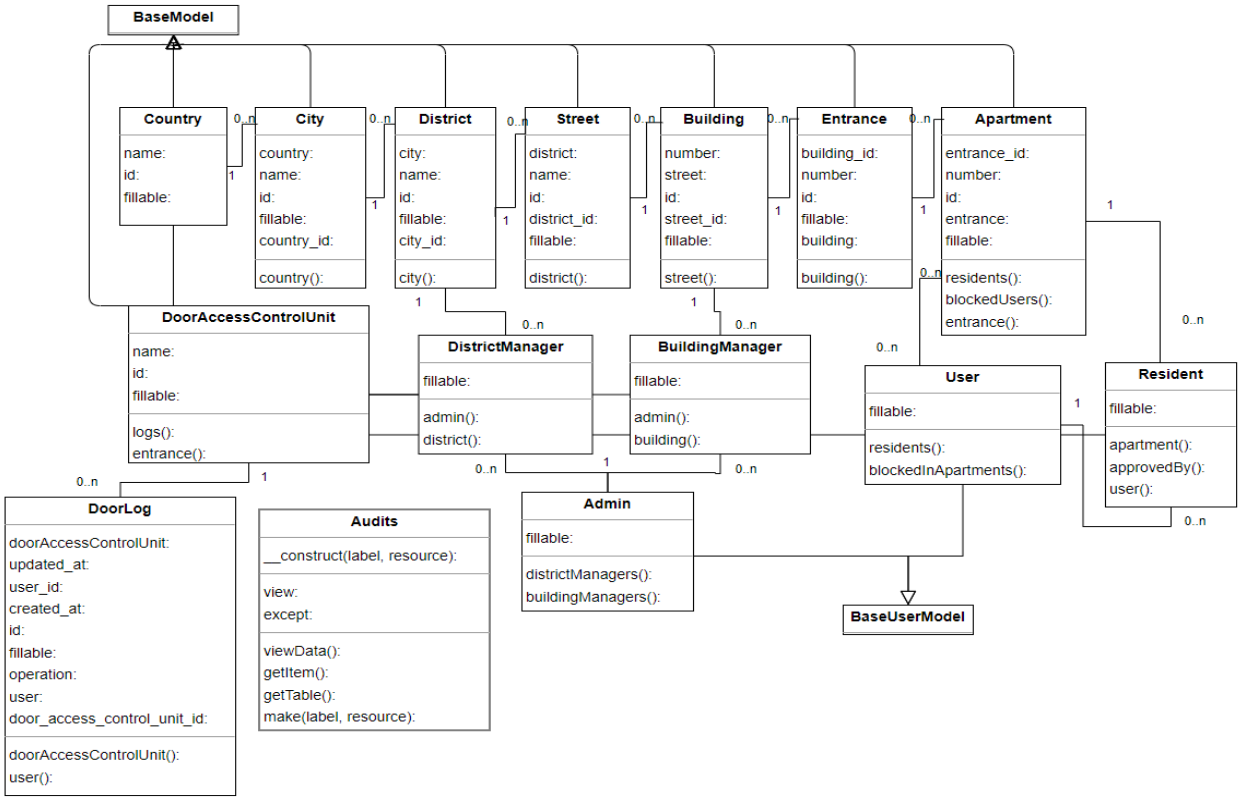



Рисунок 2.10 – Діаграма класів моделей

На рисунку 2.11 зображено ресурси – основний компонент Moonshine додатків, що відповідають за всі CRUD-операції з певною моделлю. Звичайно, було б доречно зобразити і відношення кожного ресурса до відповідної моделі, але тоді б діаграма була б вельми великою, тому для лаконічності зображено лише ресурси.

На рисунку 2.12 представлено діаграму класів Policy, що є основним механізмом контролю доступу до певних ресурсів. Таким чином можна бути впевненим, що дані зможуть бачити та змінювати лише відповідні співробітники, і що відповідальний за будинок не зможе, наприклад, видалити вулицю.

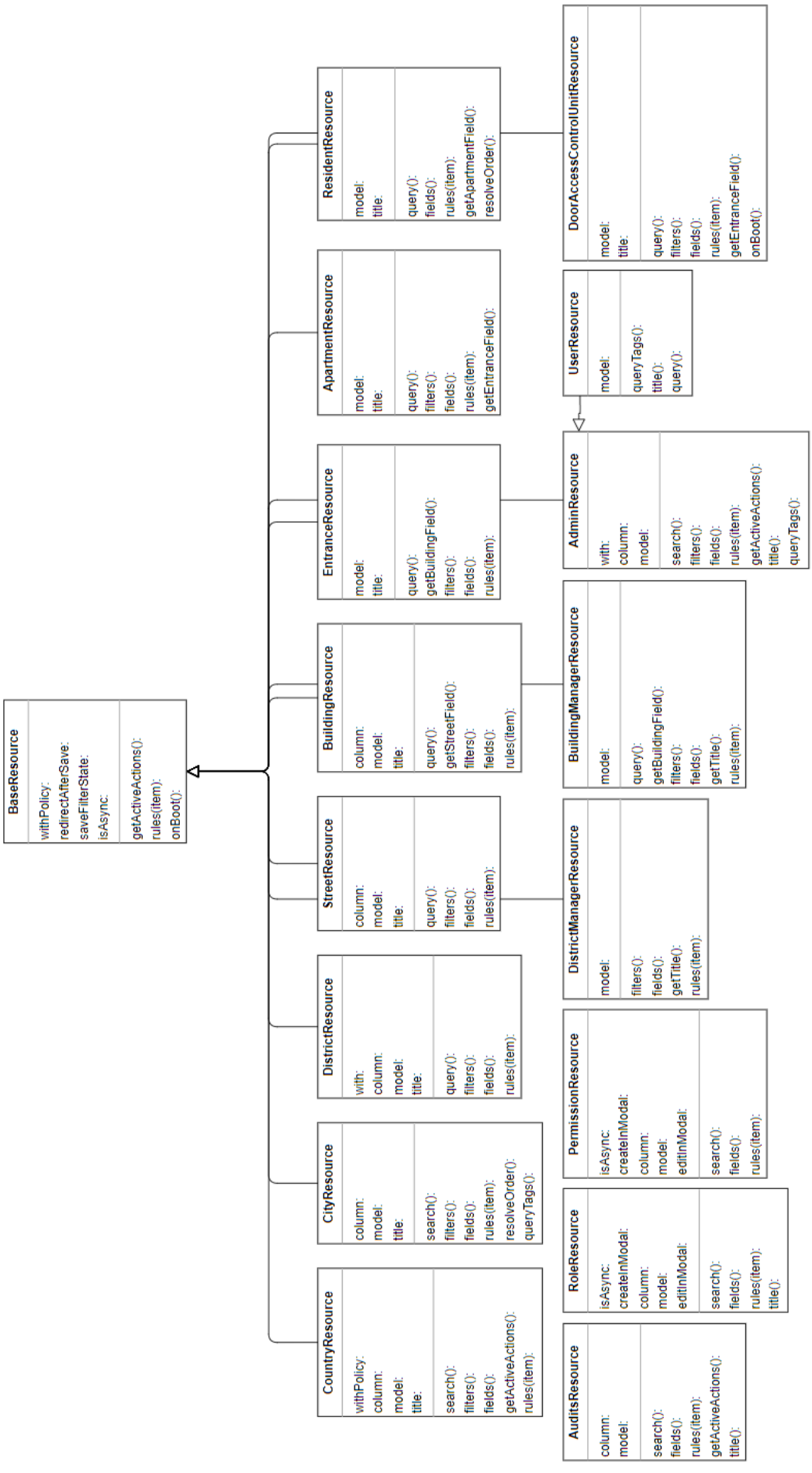



Рисунок 2.11 – Діаграма класів ресурсів

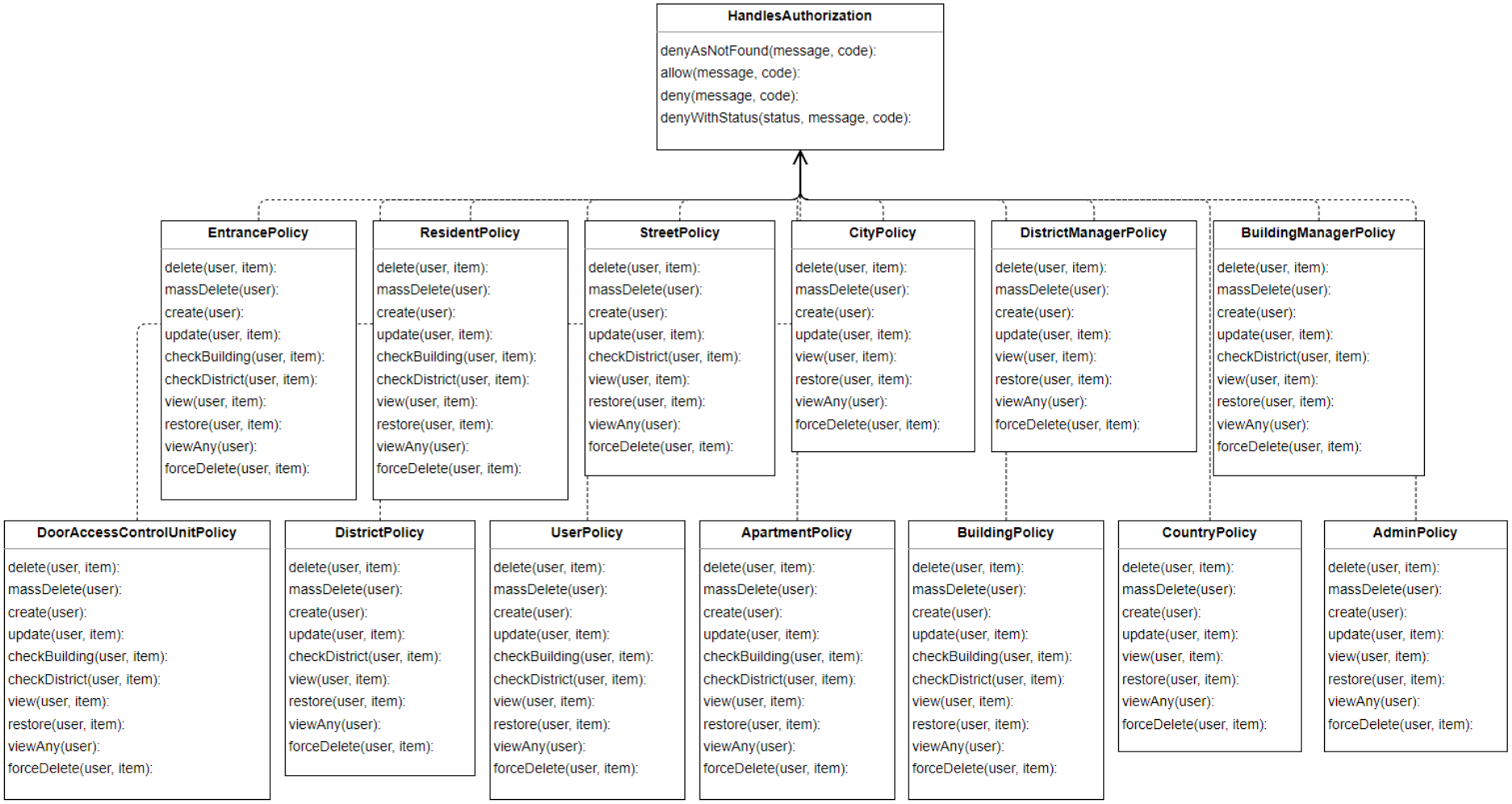



Рисунок 2.12 – Діаграма класів Policy

Таким чином, структура адміністративного застосунку досить лінійна і проста, що вказує на зручність та доцільність використання Moonshine для адміністративних додатків.

### 2.3.2 Проєктування клієнтського застосунку

Клієнтський застосунок також буде використовувати Laravel у якості бекенду, але тут вже можна побачити стандартну архітектуру MVC.

На діаграмі (див. рис. 2.13) представлена взаємодія моделей, контролерів та Vue компонентів. Наприклад, користувач завантажує сторінку з списком квартир. Його запит отримує ApartmentController, робить запит до моделі Apartment та повертає ApartmentIndexComponent з відповідними даними. Потім, коли користувач захоче подзвонити в квартиру, чи зробити іншу дію, то VueComponent створить запит, що Laravel Router передасть до обробки відповідному контролеру і так далі.

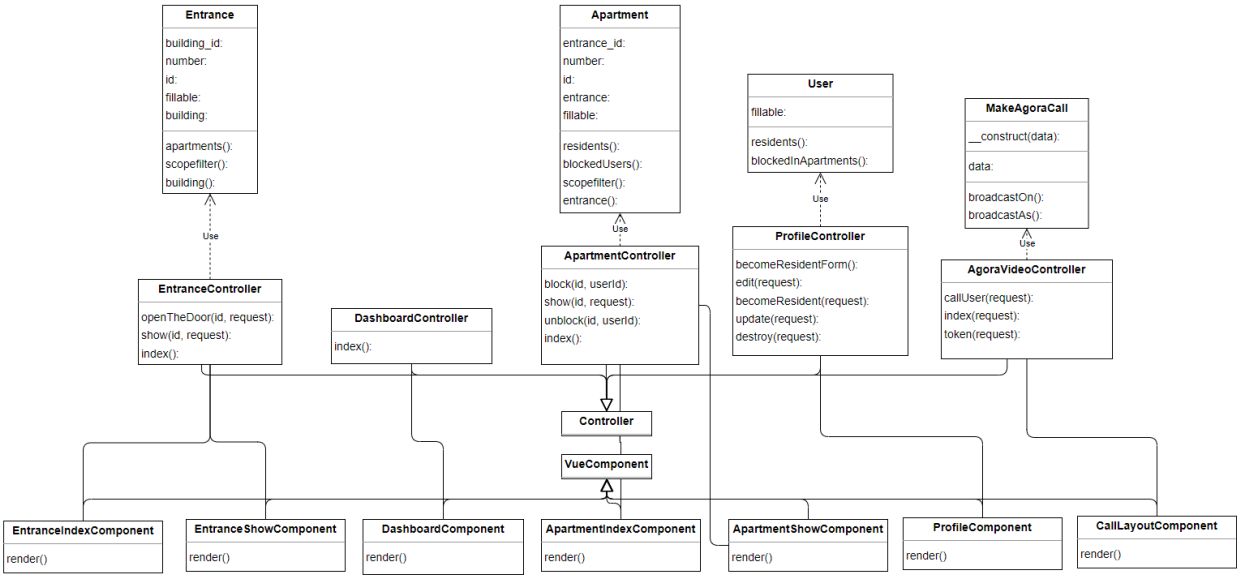



Рисунок 2.13 – Діаграма класів клієнтського застосунку

Звичайно, моделі тут ті ж самі, що і в адміністративній панелі, а тому знову вставляти повну схему класів моделей та їх стосунки не має сенсу.

Найцікавішим тут є створення відеодзвінку між користувачем та мешканцем. Для відображення цього процесу було створено декілька діаграм послідовності, що представлені на рисунках 2.14-15.

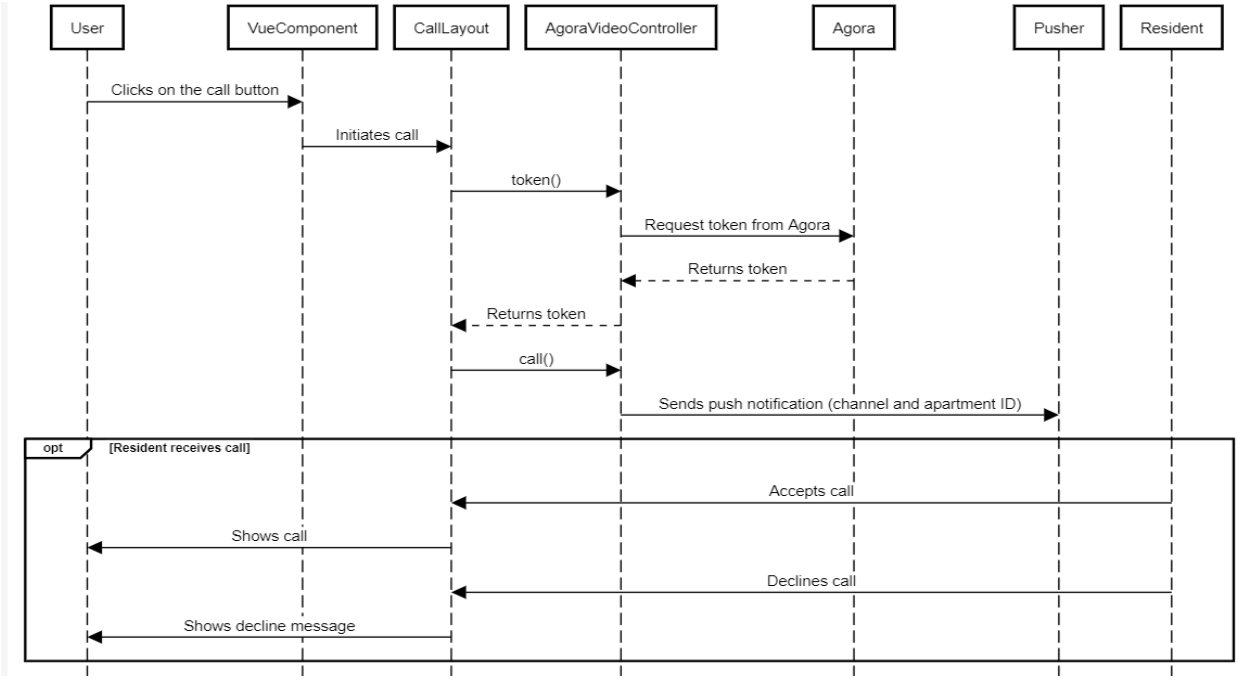



Рисунок 2.14 – Діаграма послідовності для процесу дзвінка в контексті відвідувача

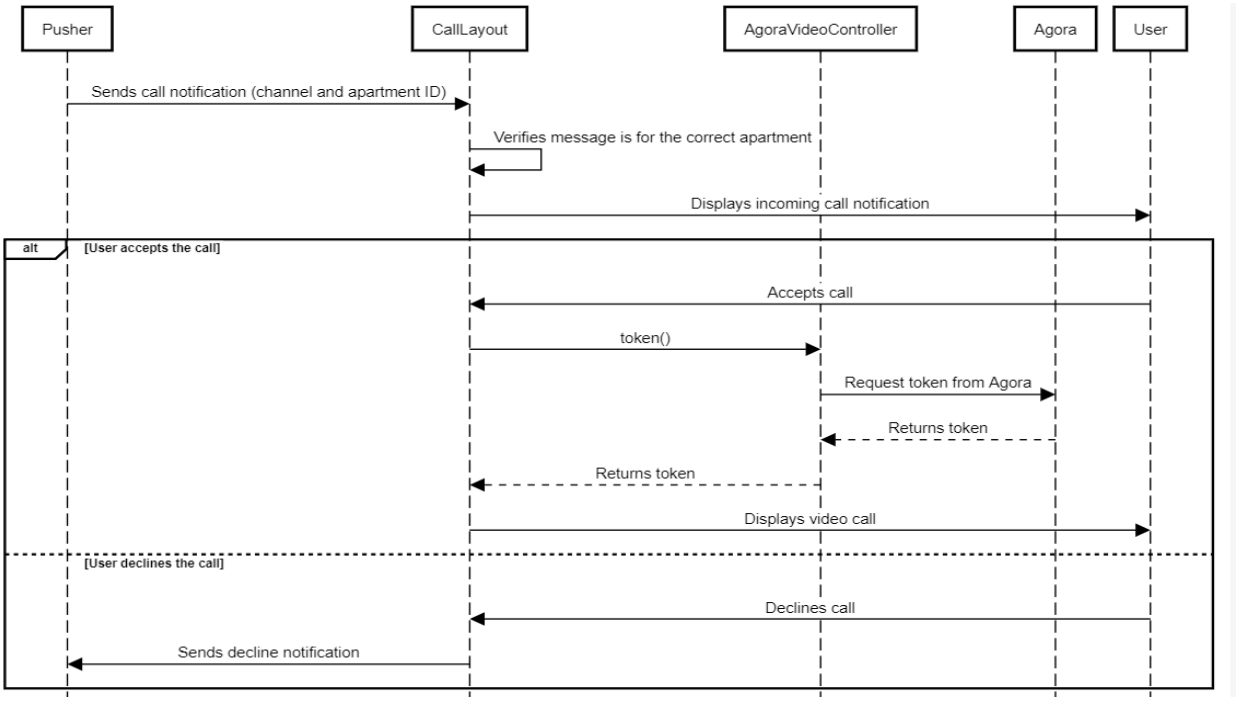



Рисунок 2.15 – Діаграма послідовності для процесу дзвінка в контексті мешканця

На рисунку 2.14 зображено послідовність дій, коли відвідувач вирішує подзвонити у обрану квартиру. На рисунку 2.15 зображено послідовність дій, що відбувається при отриманні дзвінка мешканцем.

## 2.4 Висновки до розділу 2

Таким чином, у другому розділі було визначено прецеденти та створено відповідну діаграму. Також, було визначено схему даних та створено ER-діаграму. Після цього, було почато проєктування системи, вказано особливості проєктування систем з використанням патерну MVC. В результаті, було спроєктовано адміністративну панель, створено діаграми класів, що відображають особливості архітектури проєкту, що використовує Moonshine. Також, було спроєктовано клієнтський застосунок, створено діаграми класів та діаграми послідовності для процесу дзвінка зі сторін відвідувача та мешканця.

# 3 РЕАЛІЗАЦІЯ

## 3.1 Реалізація адміністративної панелі

Як вже було вказано у попередньому розділі, реалізація функціоналу адміністративної панелі в системі «Smart Entry» виконана з використанням фреймворку Laravel та адміністративної панелі Moonshine 2.0. Використання Moonshine спрощує процес розробки та надає можливість швидко впроваджувати функції без написання додаткового коду, таких як контролери, маршрути, та представлення.

Для створення CRUD функціоналу для однієї сутності необхідно:

– написати міграцію: функціонал Laravel дозволяє швидко це зробити та впровадити в дію;

– створити модель, що буде використовуватися для роботи з базою даних;

– створити Policy, якщо це необхідно: у розроблюваній системі важливим є розподіл ролей та дозволів, так що це є важливим кроком;

– створити ресурс та зареєструвати його у системі.

Зупинимося детальніше на останньому кроці. ModelResource є основним компонентом для створення розділу в адмін-панелі під час роботи з базою даних. В класі вказуються всі необхідні параметри для коректних операції з вказаною сутністю, такі як використання policy, назва розділу, назва колонки, що буде використовуватися при відображенні у зв’язках та «хлібних крихтах». Звичнайно, вказуються поля, що будут виводитися як у таблиці на головній сторінці, так і на формах створення та редагування. Також, за необхідності можно кастомізувати ресурс, змінювати запит до БД, додавати свої поля, компоненти тощо. Ще можно не використовувати вбудовані сторінки, а створити свої та підключити їх у спеціальному методі у ресурсі, але так як нам потрібен стандартний CRUD функціонал, то вистачить і ресурсу.

Для прикладу, на рисунку 3.1 представлено код CountryResource.

```php
4 usages
class CountryResource extends BaseResource
{
    public string $column = 'name';
    protected string $model = Country::class;
    protected string $title = 'Countries';
    no usages
    protected bool $withPolicy = true;

    public function fields(): array
    {
        return [
            Block::make([
                ID::make()->sortable(),
                Text::make( label: 'Name')->sortable(),
            ]),
        ];
    }

    public function rules(Model $item): array
    {
        return [
            'name' => ['required', 'string', 'max:255', Rule::unique( table: 'countries')->ignore($item->id)],
        ];
    }

    no usages
    public function getActiveActions(): array
    {
        return ['create', 'view', 'update', 'delete', 'massDelete'];
    }

    public function filters(): array
    {
        return [
            Text::make( label: 'Name'),
        ];
    }

    public function search(): array
    {
        return ['id', 'name'];
    }
}
```

Рисунок 3.1 – Код CountryResource

Тепер необхідно зареєструвати ресурс у MoonShineServiceProvider (рис. 3.2). Таким чином, у меню одразу додається необхідний пункт. Також, можно групувати ці пункти та керувати їх видимістю в залежності від ролей та дозволів користувачів.

```
MenuGroup::make(static fn() => 'Geo', [
    MenuItem::make(
        label: 'Countries',
        new CountryResource()
    )
        ->canSee(fn() => auth()->user()?->hasRole( roles: 'admin')),
    MenuItem::make(
        label: 'Cities',
        new CityResource(),
    )
        ->canSee(fn() => auth()->user()?->hasRole( roles: 'admin')),
    MenuItem::make(
        label: 'Districts',
        new DistrictResource(),
    )
        ->canSee(fn() => auth()->user()?->can( abilities: 'manage_districts') || auth()->user()?->can( abilities: 'manage_own_districts')),
    MenuItem::make(
        label: 'Streets',
        new StreetResource(),
    )
        ->canSee(fn() => auth()->user()?->can( abilities: 'manage_streets') || auth()->user()?->can( abilities: 'manage_own_streets')),
    MenuItem::make(
        label: 'Buildings',
        new BuildingResource(),
    )
        ->canSee(fn() => auth()->user()?->can( abilities: 'manage_buildings') || auth()->user()?->can( abilities: 'manage_own_buildings')),
    MenuItem::make(
        label: 'Entrances',
        new EntranceResource(),
    )
        ->canSee(fn() => auth()->user()?->can( abilities: 'manage_entrances') || auth()->user()?->can( abilities: 'manage_own_entrances')),
    MenuItem::make(
        label: 'Apartments',
        new ApartmentResource(),
    )
        ->canSee(fn() => auth()->user()?->can( abilities: 'manage_apartments') || auth()->user()?->can( abilities: 'manage_own_apartments')),
])
    ->icon( icon: 'heroicons.outline.globe-alt'),
```

Рисунок 3.2 – Код menu в MoonShineServiceProvider

Таким чином, використання Moonshine робить розробку зручною та швидкою, дозволяючи фокусуватися на основній логіці та функціональності, а не на рутинних завданнях налаштування адміністративної панелі.

У Додатку А наведено приклади роботи адміністративної панелі.

### 3.2 Реалізація клієнтського застосунку

Основною особливістю реалізації клієнтського застосунку є забезпечення відвідувачів можливістю робити відеодзвінки до мешканців певної квартири. Як було сказано вище, для цього буде використовуватися Video Calling SDK від Agora, що дозволяє додати індивідуалізовані відео в реальному часі до будь-якого додатка. Це дозволяє розробникам інтегрувати високоякісні відеодзвінки, які можуть бути використані в різних сценаріях,

таких як відеоконференції, дистанційне навчання, телемедицина та соціальні додатки.

Щоб розпочати розробку з використанням Agora Video SDK, спершу необхідно зареєструватися на платформі Agora, створити новий проєкт і отримати унікальний App ID для ідентифікації вашого додатку. Далі потрібно завантажити та інтегрувати SDK, доступний для різних платформ, таких як iOS, Android, Windows, macOS та веб. Після інтеграції SDK слід ініціалізувати його, використовуючи наданий App ID, що дозволить додатку підключатися до сервісів Agora.

Наступний етап – приєднання до каналу, кожен з яких має унікальний ідентифікатор для об’єднання учасників у відеодзвінку. У процесі роботи додаток може обробляти події, такі як підключення або вихід учасників, а також управляти відео- та аудіопотоками. Після завершення відеодзвінка необхідно відключитися від каналу і вивільнити ресурси SDK для коректного завершення сеансу.

На рисунках 3.3 та 3.4 зображено функції для початку та ініціалізації дзвінка. Як можна побачити, згідно з діаграмою послідовності з розділу «Проєктування», спочатку створюється токен, потім ініціалізується дзвінок і відправляється запит до серверу, щоб відправити Pusher-повідомлення до мешканця квартири, що йому дзвонять.

```
const join = async (apartmentId) => {
  try {
    const from = currentUser.value.id;
    const to = apartmentId;
    const channelName = `${from}_${to}`;
    const tokenRes = await axios.post('/agora/token', { channelName });
    const options = {
      appId: process.env.AGORA_APP_ID,
      channel: channelName,
      token: tokenRes.data,
      uid: currentUser.value.id
    };
    await initAgora(options);
    await axios.post('/agora/call-user', {
      apartment_to_call: to,
      channel_name: channelName
    });
  } catch (error) {
    console.error(error);
    ElMessage.error(error.message);
  }
};
```

Рисунок 3.3 – Функція для початку дзвінка

```
const initAgora = async (options) => {
  incomingCall.value = false;
  if (!client) {
    client = AgoraRTC.createClient( config: {
      mode: 'rtc',
      codec: codec
    });
  }

  client.on('user-published', handleUserPublished);
  client.on('user-unpublished', handleUserUnpublished);

  options.uid = await client.join(options.appId, options.channel, options.token, options.uid || null);
  await initTracks();
  const tracks = [audioTrack.value, videoTrack.value];
  await client.publish(tracks);
  showJoinedMessage(options);
  joined.value = true;
};
```

Рисунок 3.4 – Функція для ініціалізації дзвінка

Приклади роботи можна переглянути у Додатку Б.

### 3.3 Висновки до розділу 3

Таким чином, у 3 розділі було реалізовано адміністративну панель та було розказано про особливості реалізації додатків засобами Moonshine. Також, було розказано про особливості реалізації відеодзвінків засобами Agora Video Calling SDK.

# 4 РОЗГОРТАННЯ

## 4.1 Створення Docker образів

Docker контейнери дозволяють створити ізольоване середовище для кожного компоненту системи, що включає всі необхідні залежності для роботи додатку. Контейнеризація спрощує розгортання та забезпечує стабільність, адже Docker гарантує однакову поведінку додатку незалежно від середовища.

Dockerfile – це текстовий файл, який містить інструкції для створення Docker-образу. Кожен рядок у Dockerfile описує крок побудови образу, зокрема, які операції виконувати для налаштування системного середовища. Образ, створений на основі Dockerfile, включає всі необхідні залежності, бібліотеки та файли, що потрібні для запуску програми.

Так як обидва застосунки написані на Laravel, то було визнано доцільним створити один загальний образ (рис. 4.1) і його потім використати безспосередньо для кожного з проєктів (рис. 4.2).

```
FROM php:8.3-fpm-alpine

RUN apk update && apk add --no-cache curl libpng-dev libxml2-dev zip libzip-
dev unzip nginx supervisor nodejs npm \
&& docker-php-ext-install pdo pdo_mysql zip \
    && rm -rf /var/cache/apk/*

COPY --from=composer:latest /usr/bin/composer /usr/local/bin/composer
COPY ./docker/nginx.conf /etc/nginx/nginx.conf
COPY ./docker/supervisord.conf /etc/supervisord.conf

WORKDIR /var/www/app
EXPOSE 80
CMD ["/usr/bin/supervisord", "-c", "/etc/supervisord.conf"]
```

Рисунок 4.1 – Dockerfile laravel-base:8.3

```
FROM xiori007/laravel-base:8.3

COPY . /var/www/app
RUN chmod -R ugo+rw storage/logs storage/framework bootstrap/cache
RUN composer install \
    && npm install \
    && npm run build
RUN php artisan key:generate \
    && php artisan optimize
```

Рисунок 4.2 – Dockerfile для образів застосунків

## 4.2 Створення ресурсів Kubernetes

Об’єкти Kubernetes – це основні елементи, що представляють стан кластеру. Вони описують ресурси, необхідні для розгортання та керування додатками, а також їх конфігурації. Існує кілька видів об’єктів Kubernetes.

Pod – це найменша одиниця обчислень, що створюється та управляється Kubernetes. Pod може містити один або декілька контейнерів, які розгортаються та масштабуються разом. Це базовий об’єкт для запуску контейнеризованих додатків у Kubernetes.

ReplicaSet – об’єкт, який забезпечує визначену кількість запущених Pod у будь-який момент часу. Якщо Pod з якоїсь причини виходить з ладу, ReplicaSet автоматично створює новий Pod для досягнення заданої кількості реплік.

Deployment – об’єкт, що дозволяє автоматизувати процес оновлення додатків, зокрема реалізувати безперервне розгортання та відкат до попередніх версій у разі помилок. Deployment спрощує управління Pod і ReplicaSet, дозволяючи безперервно оновлювати додаток без зупинки сервісу.

Service – об’єкт, який визначає постійну IP-адресу та DNS-ім’я для набору Pod, що надають один і той самий сервіс. Service дозволяє організувати взаємодію між компонентами додатків у кластері, забезпечуючи балансування навантаження між Pod.

ConfigMap та Secret – ці об’єкти використовуються для зберігання конфігураційних даних, які можна динамічно завантажувати в Pod. ConfigMap підходить для зберігання не конфіденційних налаштувань, тоді як Secret призначений для конфіденційних даних, таких як паролі чи ключі доступу.

PersistentVolume (PV) та PersistentVolumeClaim (PVC) – об’єкти, які забезпечують зберігання даних для Pod навіть після їх видалення. PV визначає фізичний ресурс для зберігання, тоді як PVC є запитом на цей ресурс, який Pod може використовувати для збереження інформації.

Ці об’єкти дозволяють Kubernetes ефективно керувати додатками та їхніми ресурсами, забезпечуючи високу доступність, надійність та масштабованість додатків у кластері.

Для розгортання застосунків необхідно створити Deployment (рис. 4.3) та Service (рис. 4.4).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: admin
spec:
  selector:
    matchLabels:
      app: admin
  replicas: 1
  template:
    metadata:
      labels:
        app: admin
    spec:
      containers:
      - name: admin
        image: xiori007/smart-entry-
admin:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app
spec:
  selector:
    matchLabels:
      app: app
  replicas: 1
  template:
    metadata:
      labels:
        app: app
    spec:
      containers:
      - name: admin
        image: xiori007/smart-entry-
app:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

Рисунок 4.3 – Deployment для застосунків

```
apiVersion: v1
kind: Service
metadata:
  name: admin
  labels:
    name: admin
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app: admin
```

```
apiVersion: v1
kind: Service
metadata:
  name: app
  labels:
    name: app
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app: app
```

Рисунок 4.4 – Service для застосунків

У production-середовищі зазвичай використовують хмарні сервіси для баз даних, такі як Amazon RDS, Google Cloud SQL або Azure Database, що забезпечують високу доступність, масштабованість і автоматизоване резервне копіювання. Однак для тестування системи можна використовувати контейнер із MySQL, що дозволяє швидко налаштувати локальне середовище для перевірки функціональності без додаткових витрат на хмарні ресурси. Такий підхід полегшує розробку і тестування, зберігаючи можливість легкого переходу на хмарну базу даних у production-середовищі.

Спочатку необхідно створити PersistentVolume та PersistentVolumeClaim (див. рис. 4.5 та 4.6).

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /d/Docker/mysql
```

Рисунок 4.5 – PersistentVolume

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

Рисунок 4.6 – PersistentVolumeClaim

Також, як і для інших компонентів системи, необхідно створити відповідний Deployment та Service (див. рис. 4.7 та 4.8).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  replicas: 1
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
          imagePullPolicy: Always
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: password
            - name: MYSQL_DATABASE
              value: laraveldb
          ports:
            - containerPort: 3306
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
```

Рисунок 4.7 – Deployment для бази даних

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
    - port: 3306
      targetPort: 3306
  selector:
    app: mysql
```

Рисунок 4.8 – Service для бази даних

Наступним кроком є створення Ingress – ресурс, що визначає правила маршрутизації вхідного трафіку до сервісів, надаючи доступ до додатків, розгорнутих у кластері, ззовні через HTTP та HTTPS протоколи. Зокрема, Ingress надає можливість налаштовувати балансування навантаження, маршрутизацію залежно від URL-підшляхів, переписування заголовків та використання TLS для забезпечення безпеки даних.

В основі Ingress лежить Ingress Controller, який обробляє правила Ingress-об’єктів та відповідає за перенаправлення вхідного трафіку до правильних сервісів. Ingress Controller зазвичай працює як додаток, що запускається у кластері Kubernetes, і може бути реалізований різними способами, такими як NGINX, Traefik або HAProxy. Це дозволяє вибрати оптимальний інструмент для керування вхідним трафіком відповідно до вимог проєкту.

Наприклад, NGINX є одним із найпоширеніших Ingress Controller’ів завдяки своїй високій продуктивності та підтримці великої кількості конфігурацій. Він добре підходить для більшості вебдодатків і має велику кількість налаштувань для оптимізації роботи.

Після обрання та підключення Ingress Controller необхідно створити безпосередньо ресурс Ingress (див. рис. 4.9).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - host: admin.smart-entry.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: admin
                port:
                  number: 80
    - host: app.smart-entry.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: app
                port:
                  number: 80
```

Рисунок 4.9 – Ingress

В результаті було створено таку файлову структуру, як на рисунку 4.10.

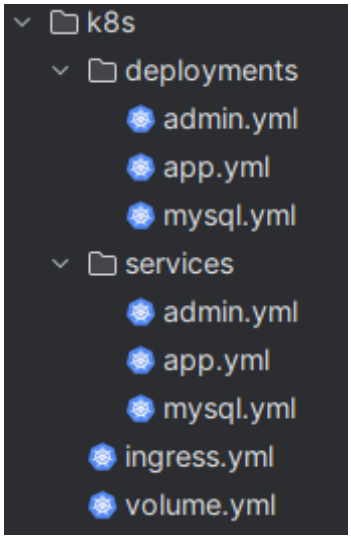



Рисунок 4.10 – Створені конфігураційні файли Kubernetes

Останнім кроком є застосування всіх створених файлів за допомогою команди, де `filename` – це назва необхідного файлу (рис. 4.11).

```
kubectl apply -f {filename}.yml
```

Рисунок 4.11

### 4.3 Висновки до розділу 4

Таким чином, у четвертому розділі було створено базовий Docker-образ для Laravel-додатків, а також образи безпосередньо для адміністративного та клієнтського застосунків. Також було налаштовано ресурси Kubernetes, такі як Deployment і Service, для ефективного управління додатками та балансування навантаження. Крім цього, було налаштовано PersistentVolume для зберігання даних бази MySQL, а також Ingress для доступу до застосунків. У результаті розроблено комплексну архітектуру для надійного розгортання додатків у production-середовищі.

# ВИСНОВКИ

Отже, в ході дослідження була розроблена інноваційна система дистанційного керування доступом до багатоквартирних будинків, яка відповідає сучасним потребам у зручності та безпеці мешканців. Основною метою роботи було створення рішення, яке забезпечує ефективне управління доступом і підвищену зручність для користувачів у міських умовах, де багатоповерхові будівлі є поширеним стандартом.

У першому розділі було проаналізовано існуючі рішення та обрано підхід для створення системи «розумного входу», яка поділяється на адміністративну панель та клієнтський додаток. Було визначено ключові вимоги до функціоналу, що забезпечують відповідність сучасним стандартам, а також обрано технологічний стек для реалізації, включаючи Laravel, VueJS, та інші допоміжні інструменти.

У другому розділі було спроєктовано архітектуру системи, розроблено схему даних та створено ER-діаграму. В межах проєктування адміністративної панелі та клієнтського застосунку були побудовані діаграми класів і послідовностей, що описують основні процеси взаємодії між користувачем та системою.

Третій розділ був присвячений реалізації компонентів системи. Було описано особливості створення адміністративної панелі та клієнтського застосунку, а також інтеграцію функцій відеодзвінків, використовуючи Agora Video SDK.

У четвертому розділі виконано розгортання системи за допомогою Docker та Kubernetes. Створено Docker-образи для адміністративного та клієнтського додатків, налаштовано ресурси Kubernetes, такі як Deployment, Service, PersistentVolume та Ingress, що забезпечило надійне функціонування системи у production-середовищі.

Таким чином, у процесі роботи було отримано корисний практичний досвід у розробці та розгортанні багатокомпонентних систем. Створена система «SmartEntry» відповідає актуальним вимогам безпеки та зручності, що має практичне значення для подальшого розвитку та застосування в багатоквартирних будинках.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Access control and authorization in smart homes: A survey / Z. N. Mohammad etc. *Tsinghua Science and Technology*. 2021. Vol. 26, № 6. P. 906–917. URL: https://doi.org/10.26599/tst.2021.9010001 (дата звернення: 16.03.2024).

2. A review of smart home applications based on Internet of Things / M. Alaa etc. *Journal of Network and Computer Applications*. 2017. Vol. 97. P. 48–65. URL: https://doi.org/10.1016/j.jnca.2017.08.017 (дата звернення: 16.03.2024).

3. Integrated smart door system in apartment room based on internet / M. Husni etc. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*. 2019. Vol. 17, № 6. P. 2747. URL: https://doi.org/10.12928/telkomnika.v17i6.12322 (дата звернення: 16.03.2024).

4. Mas Erwan A. N., Muzaffar Alfian M. N. H., Mohamad Adenan M. S. Smart Door Lock. *International Journal of Recent Technology and Applied Science*. 2021. Vol. 3, № 1. P. 1–15. URL: https://doi.org/10.36079/lamintang.ijortas-0301.194 (дата звернення: 16.03.2024).

5. Principles of Smart Home Control / S. Davidoff etc. *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2006. P. 19–34. URL: https://doi.org/10.1007/11853565_2 (дата звернення: 16.03.2024).

6. Shaik Anwar, D. Kishore. IOT based Smart Home Security System with Alert and Door Access Control using Smart Phone. *International Journal of Engineering Research and*. 2016. Vol. V5, № 12. URL: https://doi.org/10.17577/ijertv5is120325 (дата звернення: 16.03.2024).

7. Shanthini M., Vidya G., Arun R. IoT Enhanced Smart Door Locking System. *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, м. Tirunelveli, India, 20–22 aug. 2020 р. 2020. URL: https://doi.org/10.1109/icssit48917.2020.9214288 (дата звернення: 16.03.2024).

8. Smart Home: Architecture, Technologies and Systems / M. Li etc. *Procedia Computer Science*. 2018. Vol. 131. P. 393–400. URL: https://doi.org/

10.1016/j.procs.2018.04.219 (дата звернення: 16.03.2024).

9. Smart Home Beyond the Home: A Case for Community-Based Access Control / M. Tabassum etc. *CHI ‘20: CHI Conference on Human Factors in Computing Systems*, м. Honolulu HI USA. New York, NY, USA, 2020. URL: https://doi.org/10.1145/3313831.3376255 (дата звернення: 16.03.2024).

10. Wilson C., Hargreaves T., Hauxwell-Baldwin R. Benefits and risks of smart home technologies. *Energy Policy*. 2017. Vol. 103. P. 72–83. URL: https://doi.org/10.1016/j.enpol.2016.12.047 (дата звернення: 16.03.2024).

11. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O’Reilly Media, 2019. 544 p.

12. Laravel Documentation. URL: https://laravel.com/docs/10.x (дата звернення: 08.11.2024).

13. Laravel Auditing Documentation. URL: https://laravel-auditing.com/ (дата звернення: 08.11.2024).

14. Macrae C. Vue.js: Up and Running: Building Accessible and Performant Web Apps. O’Reilly Media, 2018. 174 p.

15. VueJS Documentation. URL: https://vuejs.org/guide/introduction.html (дата звернення: 08.11.2024).

16. Inertia Documentation. URL: https://inertiajs.com (дата звернення: 08.11.2024).

17. Agora Video SDK Documentation. URL: https://docs.agora.io/en/video-calling/ (дата звернення: 08.11.2024).

18. Docker Documentation. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 08.11.2024).

19. Goasguen S. Docker Cookbook: Solutions and Examples for Building Distributed Applications. Sebastopol: O’Reilly Media, Inc., 2015. 366 p.

20. Kubernetes Documentation. URL: https://kubernetes.io/docs/home/ (дата звернення: 08.11.2024).

21. Hohn F. The Book of Kubernetes: A Complete Guide to Container Orchestration. San Francisco: No starch press, Inc., 2022. 384 p.

# ДОДАТОК А

## Приклади роботи адміністративної панелі

На рисунку А.1 представлено головну сторінку, де виведено основну інформацію про продукт та деяку статистику.

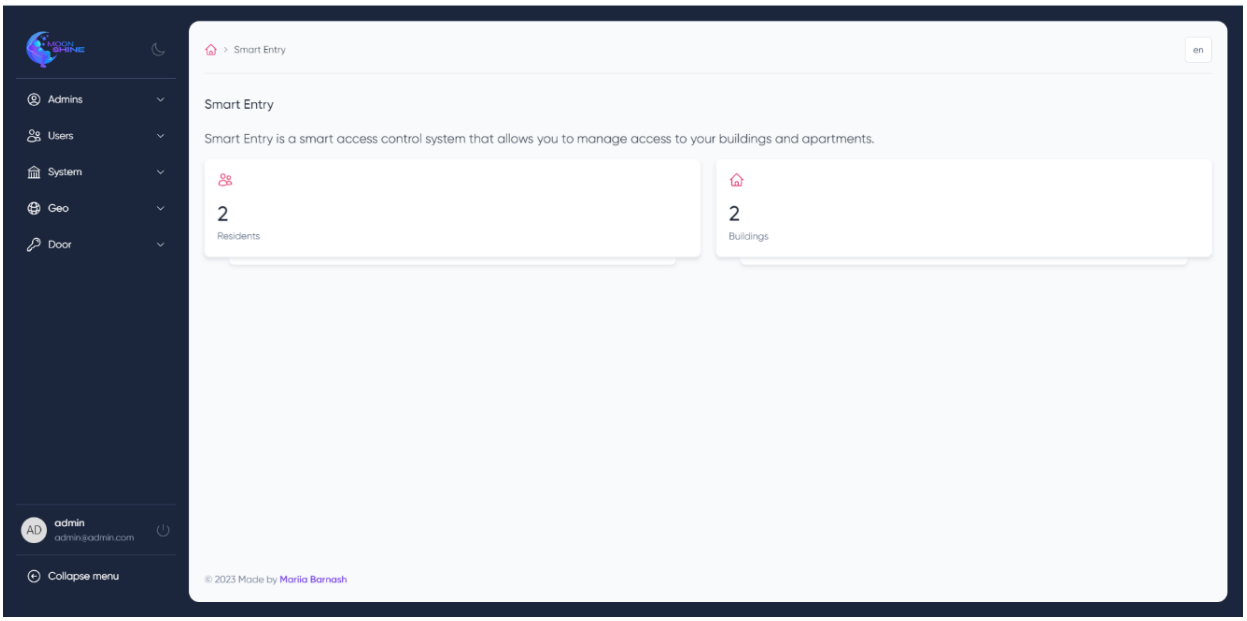



Рисунок А.1 – Dashboard

На рисунку А.2 представлено index сторінку ресурсу адміністраторів. Як можна побачити, тут виводяться всі дані, а також кнопки-посилання на сторінки для детального перегляду та редагування, а також для видалення. Слід зазначити, що ці кнопки виводяться в залежності від policy, тобто лише якщо користувач має право здійснювати ту чи іншу операцію. До того ж, на сторінці представлено так звані QueryTags – швидкі фільтри, які можна додати для популярних чи важливих запитів. Також представлено сам фільтр (рис. А.3).

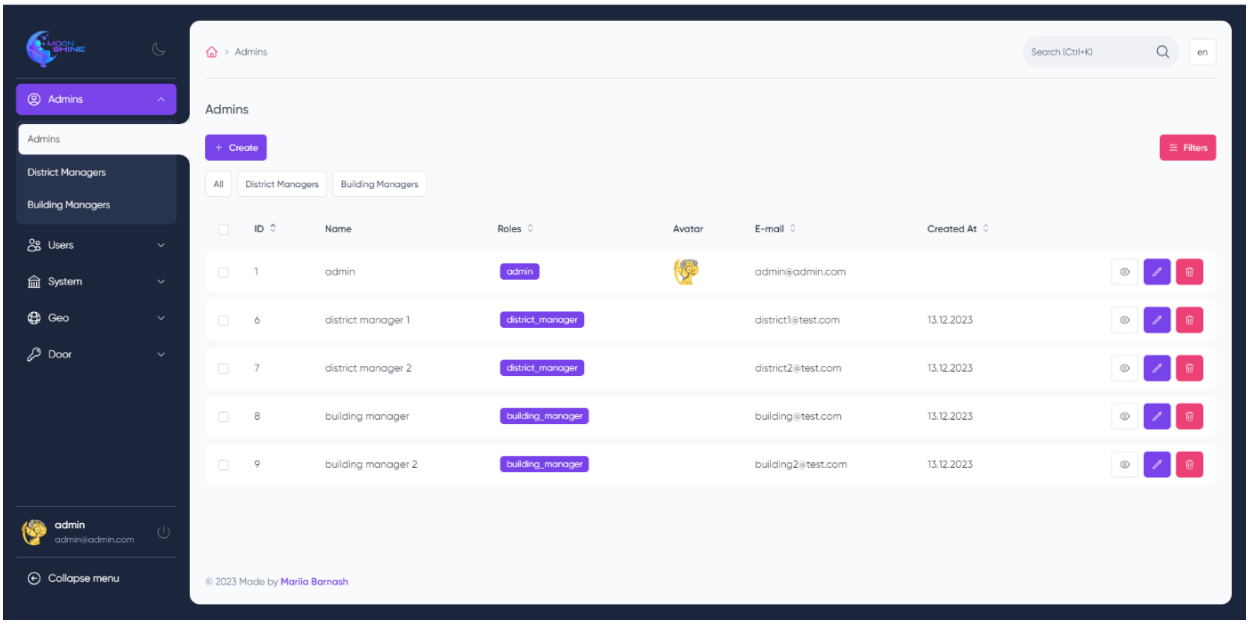



Рисунок А.2 – Головна сторінка ресурсу адміністраторів

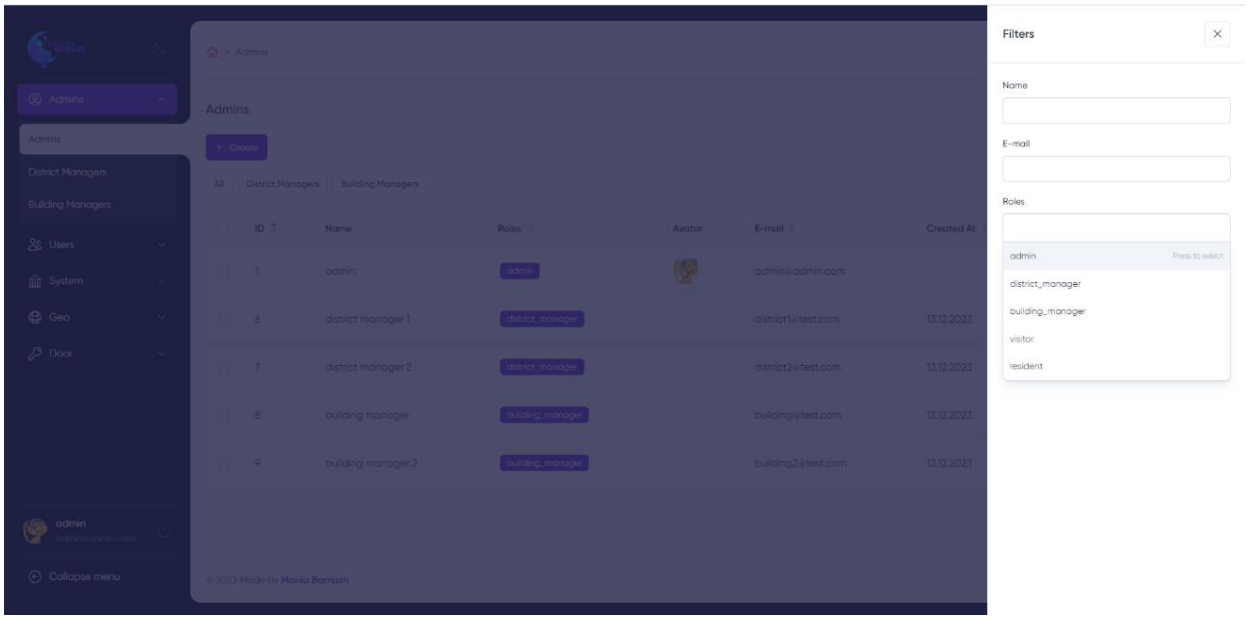



Рисунок А.3 – Фільтр на головній сторінці ресурсу адміністраторів

На рисунку А.4 представлено сторінку для деталей ресурсу адміністраторів. Як можна побачити, тут виводяться всі дані, а також кнопки-посилання для редагування та видалення. Також, нижче є компонент з логами, де можна побачити хто та коли в останнє змінював цю сутність. Слід зазначити, що у ресурсі можна контролювати, які поля будуть виводитися на тій чи інші сторінці, наприклад можна приховати поле id на головній та виодити його тільки у формах, чи навпаки.

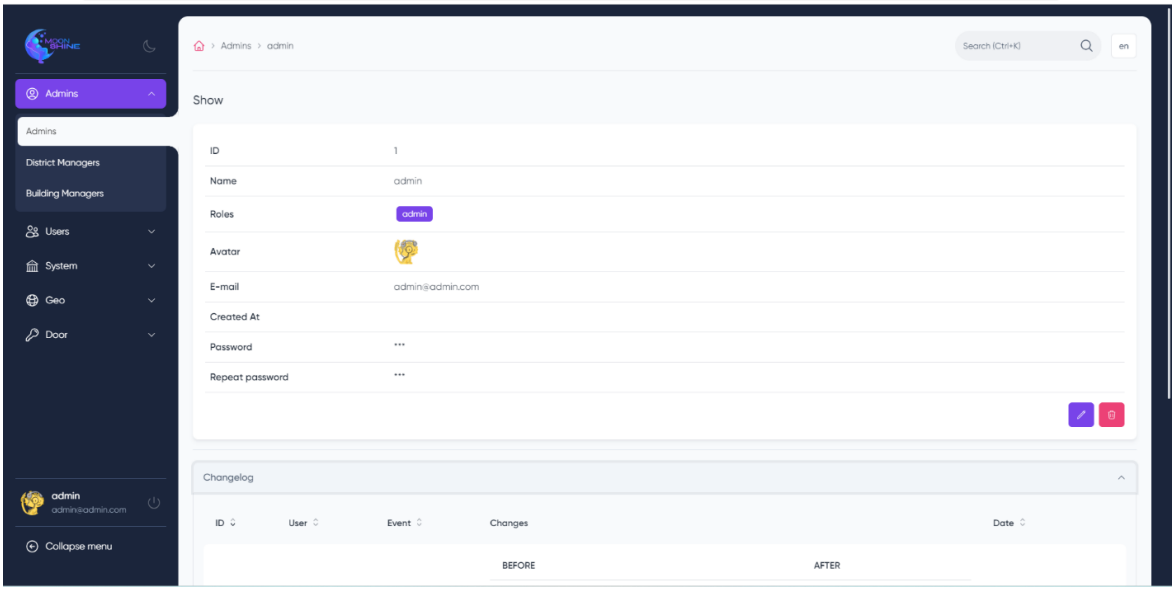



Рисунок А.4 – Сторінка деталей ресурсу адміністраторів

На рисунку А.5 представлено сторінку для редагування ресурсу адміністраторів. Як можна побачити, тут виводяться всі дані, а також кнопки-посилання для детального перегляду та видалення. Слід зазначити, що форма для створення сутності буде мати точно такий вигляд. Також прекрасно видно, яким чином можна кастомізувати сторінки використовуючи вбудовані компоненти Block, Tab та інші. До того ж, кожним окремим компонентам можна надавати власні атрибути чи класи, наприклад використовуючи tailwind-css, на базі якого побудовано UI компоненти Moonshine.

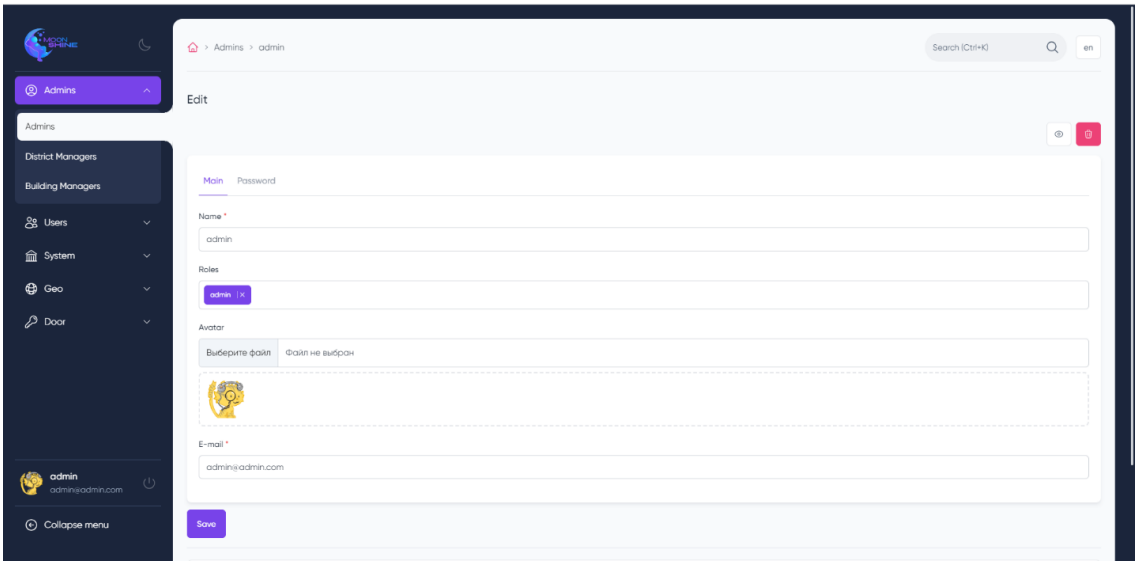
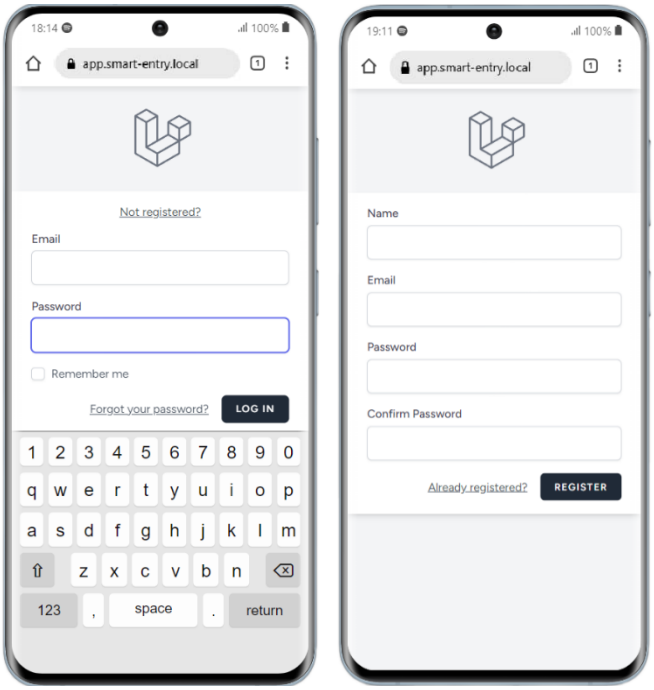



Рисунок А.5 – Сторінка редагування ресурсу адміністраторів

# ДОДАТОК Б

## Приклади роботи клієнтського додатку




а) Логін б) Реєстрація

Рисунок Б.1 – Сторінки автентифікації

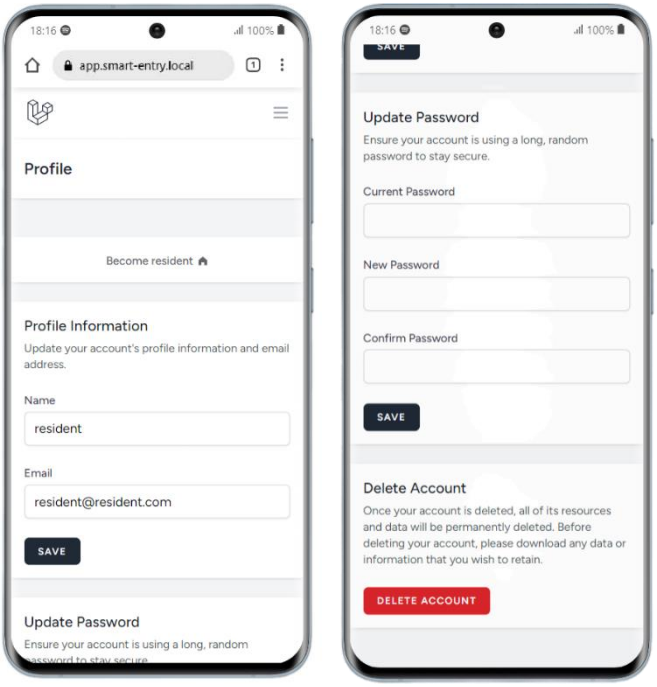



Рисунок Б.2 – Сторінка Profile

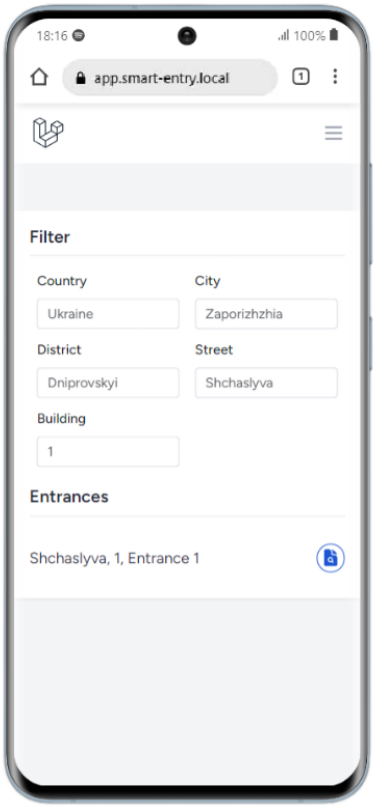



Рисунок Б.3 – Пошук під’їзду

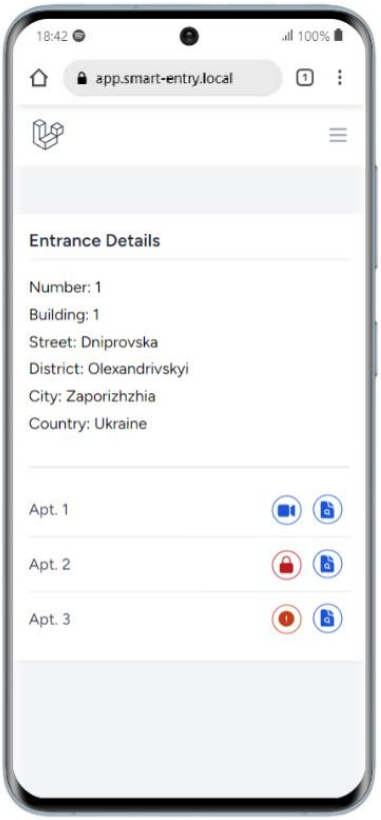



Рисунок Б.4 – Сторінка під’їзду

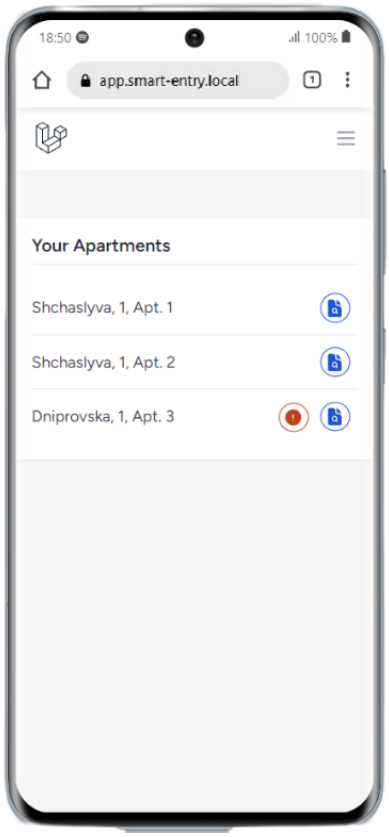
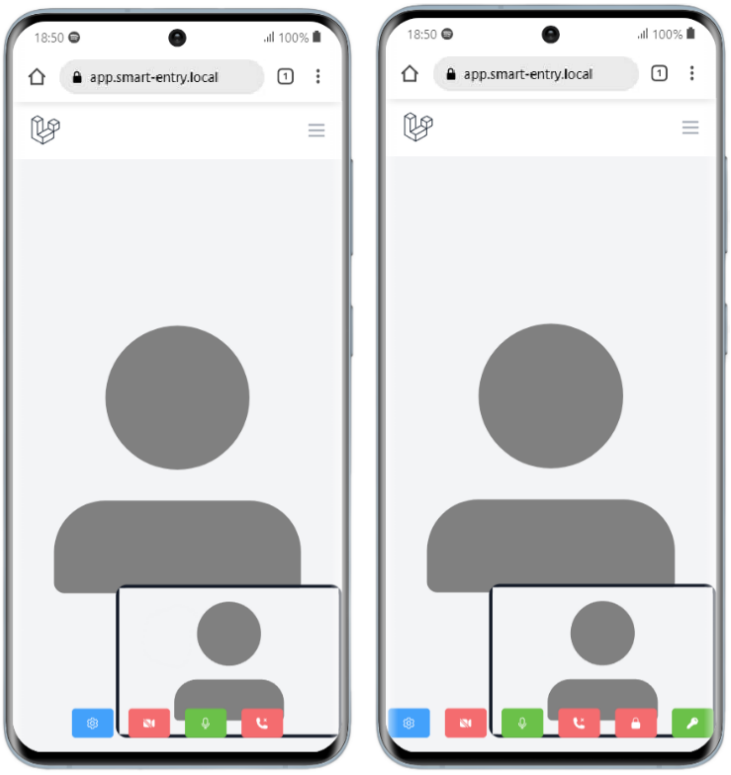



Рисунок Б.5 – Квартири мешанця




а) Відвідувач б) Мешканець

Рисунок Б.6 – Відеодзвінок

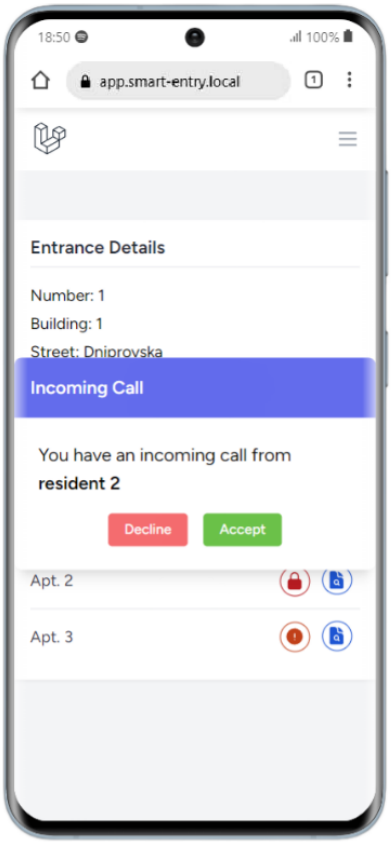



Рисунок Б.7 – Виклик