МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна технологія розробки модуля системи безпеки та управління будинком»

здобувача групи ІН.м - 23 Грицая Богдана Вікторовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Богдан ГРИЦАЙ

(підпис)

Керівник,

асистент кафедри комп'ютерних наук,

к.ф.-м.н. Ольга ШУТИЛЄВА

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-23 Грицая Богдана Вікторовича

1. Тема роботи: «Інформаційна технологія розробки модуля системи безпеки та управління будинком»
затверджую наказом по СумДУ від *«06» грудня 2023 року №1412-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 18 грудня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для захисту та управлінням будинком. 3) Практична реалізація інформаційної системи. 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для захисту та управління будинком.*		
3	*Практична реалізація інформаційної системи*		
4	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 78 стр., 52 рис., 1 додаток, 30 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки наростає інтерес до створення систем безпеки та управління будинком, які не лише спрощують повсякденне життя, але й забезпечують високий рівень комфорту та безпеки.

Об'єкт дослідження – процес розробки модуля системи безпеки та управління будинком

Мета роботи – розробка інформаційної технології модуля системи безпеки та управління будинком.

Результати – проведено аналіз предметної області, розглянуто існуючі аналоги, сформульовано постановку завдання та обрано технології для його вирішення, а також спроєктовано та реалізовано базу даних, використовуючи систему управління базами даних PostgreSQL, розроблено інформаційну технологію модуля системи безпеки та управління будинком.

ІНФОРМАЦІЙНА СИСТЕМА, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ,
МОНОЛІТНА АРХІТЕКТУРА, JAVA, POSTGRESQL,
SPRING BOOT, SPRING MVC

ЗМІСТ

ВСТУП

Актуальність. Розвиток інформаційних технологій та автоматизації мотивує до появи нових можливостей для створення і вдосконалення систем управління та безпеки в сучасних будинках. Швидкість та ефективність обробки даних, а також доступність передових технологій дозволяють реалізувати інтелектуальні системи, які сприяють зручності та безпеці життя споживачів [1]. Актуальність цього дослідження визначається наростаючим інтересом до створення систем безпеки та управління будинком, які не лише спрощують повсякденне життя, але й забезпечують високий рівень комфорту та безпеки.

Об'єкт дослідження. Інформаційна система, спрямована на ефективне керування та моніторинг різноманітних пристроїв та датчиків у будинку через зрозумілий інтерфейс.

Предмет дослідження. Модуль системи безпеки та управління будинком.

Гіпотеза. Розробка та впровадження модулів систем безпеки та управління будинком, які надають користувачеві можливість ефективного контролю за різними пристроями та датчиками в їхньому домі є однією з перспективних галузей [2]. Саме зростаюча потреба в зручності, безпеці та ефективному використанні ресурсів ставить завдання розробки інтелектуальних систем управління будинком [3]. Однією з ключових проблем, яку спрямована вирішити робота, є необхідність створення інформаційної системи, яка дозволяє користувачам ефективно керувати та моніторити різноманітні пристрої та датчики у будинку через інтуїтивно зрозумілий інтерфейс.

Структура. Кваліфікаційна робота магістра спрямована на аналіз інформаційних технологій для розробки та впровадження інтелектуальних систем управління будинком. А також на реалізацію інформаційної технології розробки модуля системи безпеки та управління будинком.

1. ЛІТЕРАТУРНИЙ ОГЛЯД

1.1 Огляд останніх досліджень та публікацій

Очевидно, що системи автоматизації відіграють важливу роль у сучасності. У попередні роки, інновації зробили стрибок вперед і тому системи автоматизації використовуються в багатьох додатках та людських сферах життя [4].

Модуль системи безпеки та управління будинком це ключовий елемент інтелектуальних систем, призначених для оптимізації функціонування житлових приміщень [5]. Центральна ідея полягає в поєднанні апаратних та програмних компонентів для надання користувачам повного контролю над різноманітними аспектами їхнього будинку [6].

Сутністю модулів безпеки є система виявлення та реагування на різноманітні події, від руху в приміщенні до виявлення небезпеки, які можуть виникнути в будинку. Забезпечення віддаленого доступу через веб-інтерфейс чи мобільний додаток дозволяє користувачам ефективно керувати своїм будинком навіть на відстані [7].

Інформаційні технології модулів управління безпекою та будинком з кожним днем все більш зростають у попиті. Це свідчить про важливість розробки інтегрованих систем, спрямованих на підвищення рівня зручності та безпеки життя користувачів у їхніх домівках [8]. Користувачі прагнуть до максимальної автоматизації та контролю за своїм домом, щоб забезпечити безпеку своїй родині та оптимально використовувати ресурси.

Модулі систем безпеки не лише виявляють небезпеку та реагують на неї, але й надають можливість дистанційного контролю за всіма аспектами життя вдома [9]. Від віддаленого ввімкнення системи опалення до миттєвого перегляду відео з камер безпеки – ці функції стають не лише зручними, але й стратегічно важливими для сучасного способу життя [10].

Можна підсумувати основні переваги використання інформаційних технологій розробки модулю системи безпеки та управління будинком:

– Віддалений моніторинг і контроль: за допомогою системи безпеки розумного будинку є можливість контролювати власну систему безпеки з будь-якої точки світу за допомогою смартфона або планшета [11]. Надається можливість отримувати сповіщення, переглядати відео в реальному часі та дистанційно керувати системою.

– Інтеграція з іншими інтелектуальними пристроями: розумні системи безпеки будинку можна інтегрувати з іншими розумними пристроями у будинку [12].

– Параметри безпеки, що налаштовуються: модуль системи безпеки будинку дозволяють налаштувати параметри безпеки відповідно до конкретних потреб користувача [13].

Отже, модулі систем безпеки та управління будинком сьогодні стають важливою складовою не лише технологічного прогресу, але і зручності, безпеки та збереження ресурсів. А їх поєднання із сучасними тенденціями та визначені характеристики роблять ці модулі не тільки актуальними, але і стратегічно важливими для подальшого розвитку сучасного житла.

1.2 Аналіз аналогів

При створенні власної Інформаційної системи важливо враховувати сильні та слабкі сторони аналогів підтримки діяльності складу, з метою здійснення виваженого врахування кожного критерію. Це сприятиме розробці оптимального власного рішення, у якому будуть враховані всі важливі аспекти та вимоги.

Для аналізу існуючих системи обрані наступні аналоги: Ring [14] , SimpliSafe [15] та Argo [16].

Система Ring є компанією, що спеціалізується на виробництві та розробці систем безпеки для дому, зокрема відеодзвінків, датчиків руху та

камер відеоспостереження. Основний акцент компанія Ring робить на створенні простих у використанні, доступних та ефективних рішень для забезпечення безпеки користувачів та їхніх будинків.

Основні характеристики та функції системи Ring:

– Відеодзвінки. Ring виготовляє відеодзвінки, які можуть бути встановлені на вхідних дверях. Ці пристрої дозволяють користувачам взаємодіяти з відвідувачами через мобільний додаток або веб-інтерфейс.

– Датчики руху. Компанія пропонує датчики руху, які активують камери відеоспостереження, коли виявляють рух навколо будинку. Це дозволяє забезпечити відеозапис подій і попереджати користувачів про потенційні загрози.

– Камери відеоспостереження. Ring пропонує різні моделі камер відеоспостереження для встановлення в різних частинах будинку. Ці камери записують відео високої якості та можуть бути віддалено керовані.

– Доступ через мобільний додаток. Основний контроль над системою Ring здійснюється через мобільний додаток. Користувачі можуть отримувати сповіщення, переглядати відео та керувати налаштуваннями, навіть якщо вони не знаходяться вдома.

Розглянемо докладніше на прикладі налаштування та використання зовнішнього контактного датчика Ring Alarm. Зовнішній контактний датчик Ring Alarm призначений для контролю зовнішніх вікон і дверей, а також воріт. Даний датчик складається з двох частин – датчика та магніту, якщо його встановити на двері чи вікно, він може визначити, відкриті чи закриті двері чи вікно.

Під час налаштування є можливість обрати, чи бажаєте ви, щоб зовнішній контактний датчик надсилав лише про відкриття чи закриття дверей (рис.1.1), чи ви бажаєте, щоб він був під охороною, коли Ring Alarm перебуває в режимі «Вдома» або «Я не вдома» (рис.1.2).

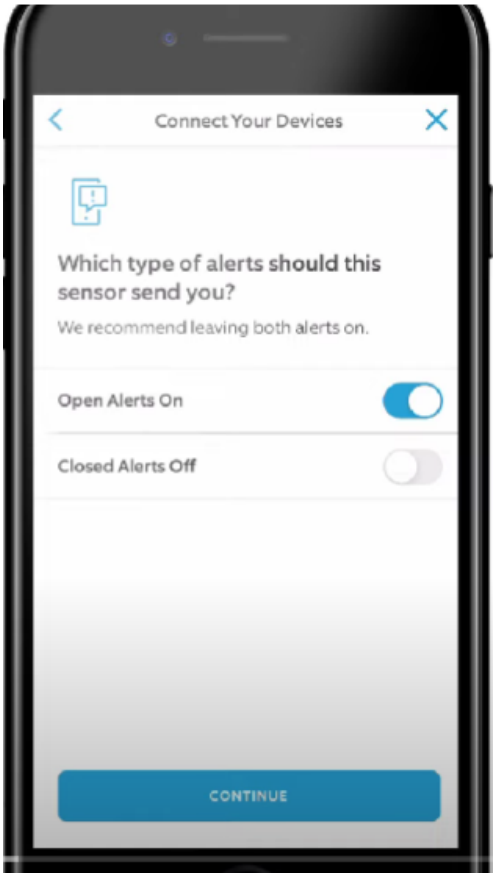


Рисунок 1.1 – Налаштування зовнішнього контактного датчика Ring Alarm

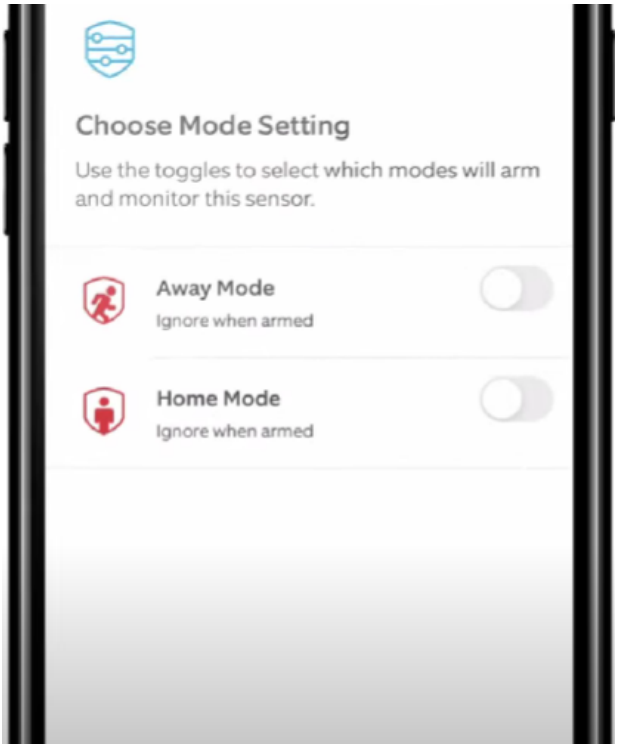


Рисунок 1.2 – Налаштування режиму для контактного датчика Ring Alarm

Хоча система Ring має багато переваг, варто також розглянути недоліки:

– Вартість підписки. Деякі функції, такі як зберігання відеозаписів в хмарі для подовження строку зберігання чи доступ до інших розширених функцій, можуть вимагати платної підписки. Це може збільшити вартість володіння системою з часом.

– Затримка в сповіщеннях. За відгуками деяких користувачів, іноді сповіщення про рух можуть мати затримку. Це є проблемою в ситуаціях, де швидка реакція важлива.

– Система не підтримує всі країни на ринку. Ring може не бути доступним або не має повноцінної підтримки у всіх країнах, що обмежує доступність для деяких користувачів. Зокрема Ring недоступний для користування з України.

SimpliSafe являє собою бездротову систему домашньої безпеки, с проєктовану для простої установки та використання. Дана система надає базові функції безпеки, щоб захистити будинок від потенційних небезпек та надійно повідомляти користувачів про події. Основний функціонал системи SimpliSafe:

– Бездротові датчики. Система використовує бездротові технології, що спрощує процес установки та дозволяє розміщувати датчики та пристрої в різних частинах будинку.

– Датчики руху та відкриття дверей. Включає в себе датчики руху та відкриття дверей, які виявлять будь-які підозрілі події.

– Димовики та датчики витоку газу. SimpliSafe пропонує додаткові функції, такі як датчики диму та витоку газу, для раннього виявлення пожежі або потенційно небезпечних ситуацій.

– Камери відеоспостереження (рис.1.3).

– Сповіщення через мобільний додаток або на електронну пошту. Користувачі отримують сповіщення через мобільний додаток в разі спрацювання сигналізації або виявлення непередбачених подій (рис.1.4).

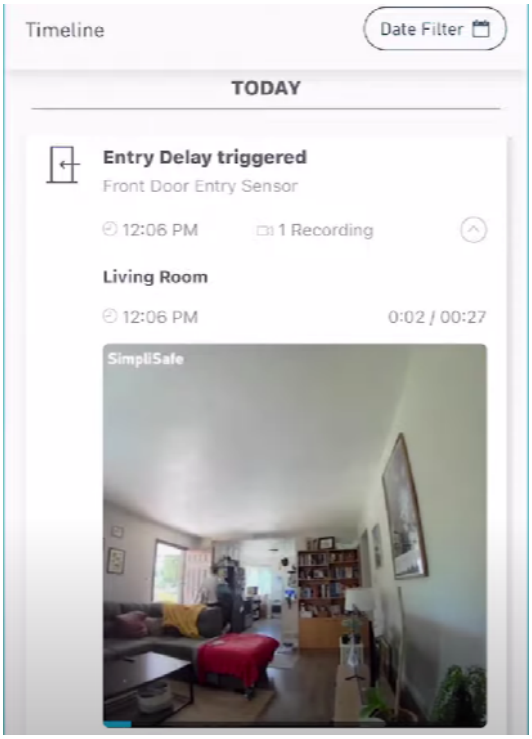


Рисунок 1.3 – Відображення відеозапису з камер спостереження системи SimpliSafe

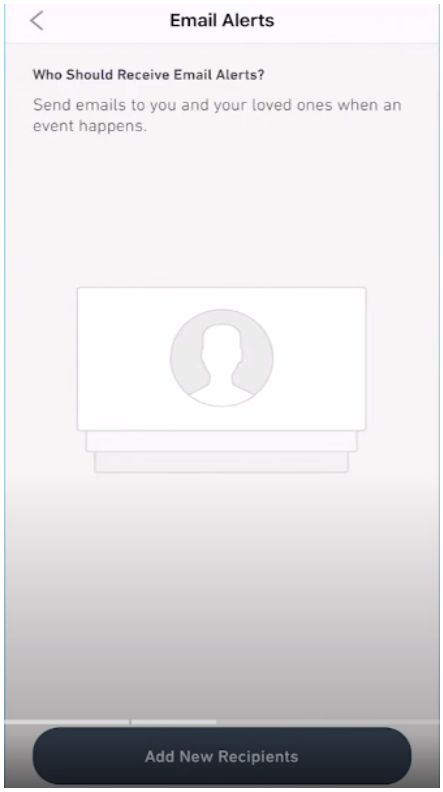


Рисунок 1.4 – Налаштування для надсилання сповіщення на електронну пошту користувача

Розглянемо обмеження системи SimpliSafe:

– Інтеграція з іншими смарт-пристроями може бути обмеженою, порівняно з більш розгалуженими екосистемами.

– Система підтримує лише США. Тобто, SimpliSafe недоступний для користування з України.

Arlo спеціалізується на створенні бездротових систем відеоспостереження та систем домашньої безпеки. Їх пристрої призначені для використання в різних умовах і забезпечують високу якість відеозапису та ряд додаткових функцій.

Основними характеристиками системи Arlo можна виділити. :

– Камери Arlo записують відео в HD-якості та мають нічний режим для зйомки в темряві (рис 1.5).

– Детектор руху та Сповіщення. Камери оснащені детекторами руху, що активують запис відео, і користувачі отримують сповіщення на мобільні пристрої.

– Хмарове зберігання. Відеозаписи можуть зберігатися в хмарі.

– Мобільний додаток та Віддалене керування. Користувачі можуть використовувати мобільний додаток для віддаленого керування системою та перегляду відеозаписів.

Таблиця 1.1 – Порівняльна таблиця аналогів Інформаційних систем

Характеристика/ Система	Ring Alarm	SimpliSafe	Arlo	Sentinel Guard
Зручний інтерфейс	+	+	+	+
Інтеграція з смарт пристроями	+	-	+	+
Сучасний дизайн	+	-	+	+
Розподіл прав	+	+	+	+
Ціна	-	+	-	+

Продовження таблиці 1.1

Мобільний доступ	+	+	+	+
Зона дії в Україні	-	-	-	+

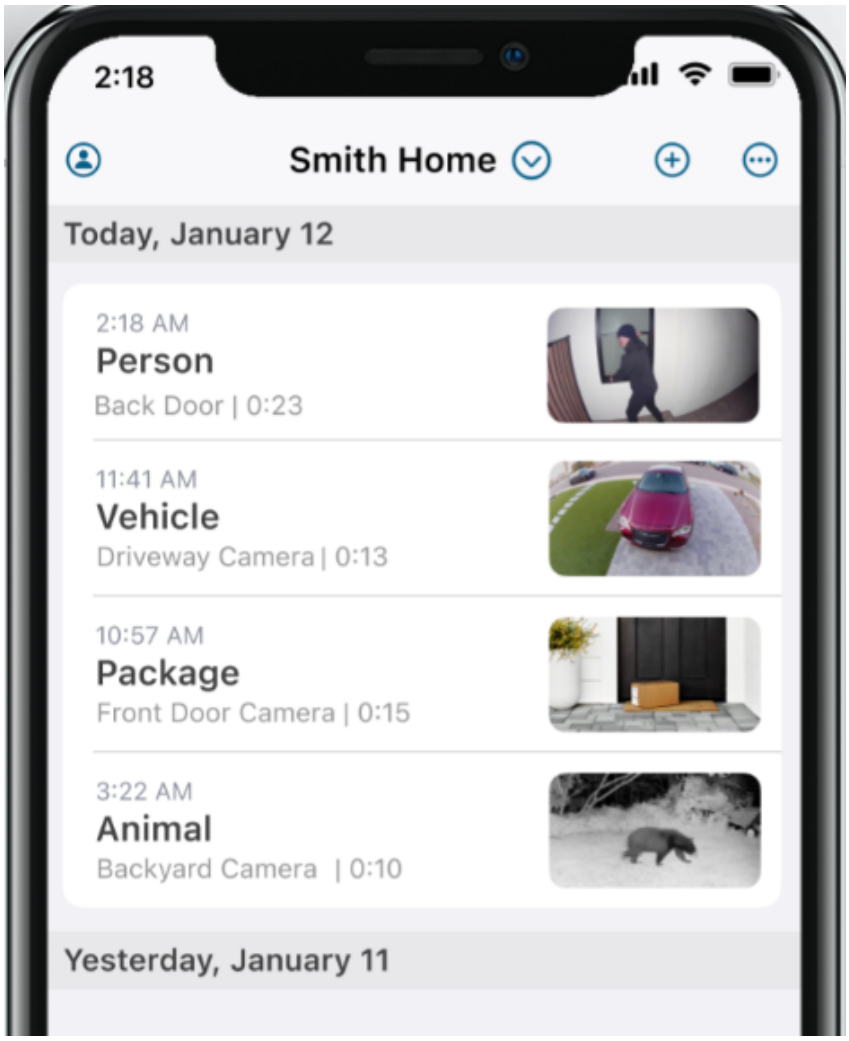


Рисунок 1.5 – Відображення відеозаписів з камер спостереження системи Arlo

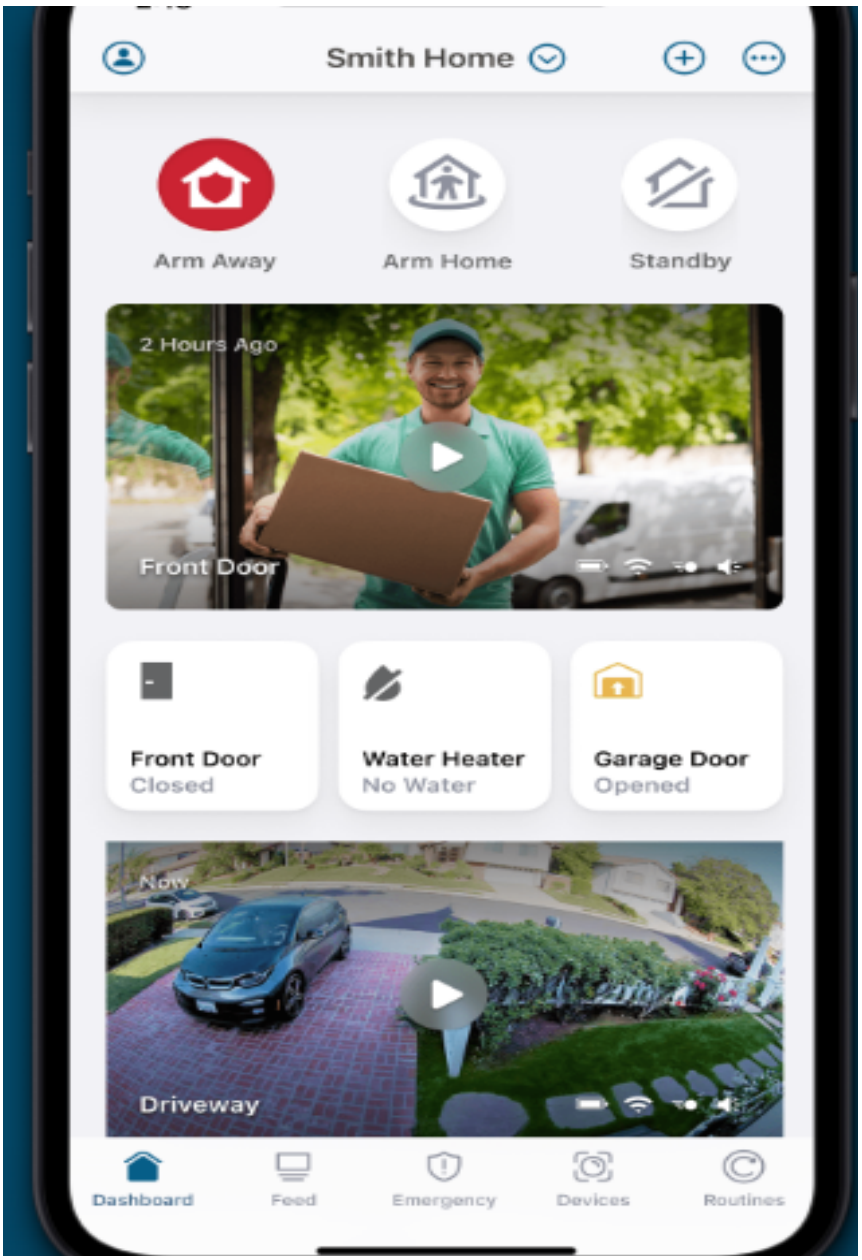


Рисунок 1.6 – Детектор руху та сповіщення Arlo

Із порівняльної таблиці аналогів Інформаційних систем можна побачити, що зручний інтерфейс притаманний усім системам, як і можливість розподілу прав між користувачами.

Інтеграція з іншими смарт-пристроями є більш обмеженою у SimplySafe, порівняно з більш розгалуженими системами, такими як Ring Alarm та Arlo. Можна виділити що SimpliSafe має більш доступну ціну ніж інші аналоги, що робити дану системи більш привабливою для користувачів.

Мобільний доступ доступний для користування всім трьом аналогам, що є важливою характеристикою для Інформаційної системи. Однак, жодна з цих аналогових Інформаційних систем не підтримується та не доступна на території України.

Отже, розробка власної Інформаційної системи безпеки та управління будинком має кілька ключових переваг, що можуть бути визначальними. Розробка власного модуля дозволяє створити систему, що точно відповідає конкретним вимогам користувача або організації, враховуючи їхні унікальні потреби та пріоритети. Власна система дозволяє зберігати контроль над рівнем безпеки та конфіденційності, забезпечуючи надійний захист від зовнішніх загроз [18]. Розробка модулю дозволить інтегрувати його з іншими пристроями та системами, що використовуються у власному будинку.

1.3 Постановка задачі

Метою роботи є проаналізувати та розробити інформаційну технологію розробки модуля системи безпеки та управління будинком, яка відповідає сучасним вимогам безпеки та управління будинком, забезпечуючи високий рівень зручності та персоналізації для кожного користувача. Для реалізації цієї мети необхідно виконати наступні завдання:

– Провести огляд існуючих систем безпеки та управління будинком для визначення їх переваг та недоліків.

– Обрати методи реалізації Інформаційної системи.

– Розробити архітектурну концепцію модулю.

– Створити зручний та інтуїтивний інтерфейс для користувачів, що дозволить легко взаємодіяти з системою.

– Реалізувати можливість реєстрації, авторизації та аутентифікації користувача в системі.

– Реалізувати можливість додавання, видалення та зміни пристроїв.

– Розробити функціонал для ефективного виявлення та реагування на загрози безпеки, такі як виявлення руху та сповіщення.

– Забезпечити можливість віддаленого керування та моніторингу за допомогою будь-якого пристрою.

– Провести тестування та оптимізацію системи. Це потрібно для перевірки ефективності та надійності, щоб у разі необхідності внести необхідні корективи.

Дані завдання визначають напрямок і область досліджень, спрямованих на створення інноваційних інформаційних систем. Актуальність розробки визначається швидким розвитком технологій у галузі розумних систем та підвищеною потребою в персоналізованих рішеннях для безпеки та управління будинком.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Методи дослідження

Визначення архітектурного підходу та технологій є ключовим етапом в розробці інформаційної системи. Для розробки Інформаційної системи була обрана монолітна архітектура.

Монолітна архітектура – це структурний підхід до розробки програмного забезпечення, де весь функціонал додатка вбудований в один цілісний блок [19]. У монолітній архітектурі весь код додатка розташований в одному зведеному компоненті. Це може бути один великий сервер чи веб-додаток, який включає в себе всі необхідні модулі та функції. Усі функції та компоненти розташовані в єдиному кодовому базисі. Розвиток, тестування та підтримка здійснюються в межах цього зведеного додатка. Компоненти взаємодіють один з одним напряму, зазвичай через виклики функцій чи модулів. Зовнішні залежності обробляються в межах єдиного додатка [20].

Для програмної реалізації Інформаційної технології розробки модуля системи безпеки та управління будинком будуть використовуватися наступні технології:

– Java обрана для розробки Інформаційної системи з урахуванням її платформової незалежності, широкої екосистеми та високої продуктивності, особливо в умовах великої кількості користувачів. Вбудовані засоби безпеки, можливості масштабування та об'єктно-орієнтований підхід роблять Java підходящим варіантом для створення стабільних та ефективних веб-додатків [21]. Багатофункціональність мови дозволяють використовувати її для різноманітних завдань в проєкті.

– Spring MVC обрано для веб-контролерів та Spring Boot для швидкого розгортання та налаштування проєкту. Це високопродуктивний та гнучкий стек, який дозволяє легко втілювати кращі практики веб-розробки [22].

– Freemarker – шаблонізатор, який дозволяє швидко реалізувати вигляд сторінок та легко інтегрується з Spring [23]. Вибір обумовлено його гнучкістю та простотою використання в порівнянні з такими аналогами як Thymeleaf.

– PostgreSQL вибрано як систему управління базами даних. Перевагами даної СУБД є надійність, можливість оптимізації та велика спільнота підтримки [24].

– Spring Data надає зручний інтерфейс для роботи з базою даних, відповідаючи сучасним стандартам розробки на основі підходу ORM [25].

Для реалізації естетичного та адаптивного інтерфейсу обрано Bootstrap. Цей фреймворк надає широкий набір готових компонентів, що спрощує розробку інтерфейсу та забезпечує його стандартизацію.

Обрані технології є оптимальним варіантом для розробки Інформаційної технології розробки модуля системи безпеки та управління будинком з врахуванням його функціональних вимог та тенденцій ринку web-розробки.

2.2 Моделювання варіантів використання системи

Діаграма варіантів використання використовуються для візуалізації та моделювання взаємодії між різними акторами та системою. Адже UML діаграма, дозволяє ідентифікувати та відобразити ключові можливості системи, що в свою чергу полегшує комунікацію між замовником та розробником [26]. Діаграми варіантів використання також дозволяють виявити можливі недоліки у дизайні або визначити ситуації, які можуть викликати ризики [27].

Були виділені наступні варіанти використання для Інформаційної технології розробки модуля системи безпеки та управління будинком:

– Реєстрація.

– Авторизація.

– Редагування особистого простору.
– Створення нового домашнього середовища.
– Приєднання до вже існуючого домашнього середовища.
– Додавання до домашнього середовища пристрою.
– Управління доданими пристроями.
– Налаштування пристрою.
– Надсилання сповіщення користувачеві при небезпеці.
– Додавання нових видів пристроїв.
– Перегляд статистики по доданим пристроям.

Переглянути діаграму варіантів використання можна на рисунку 2.1.

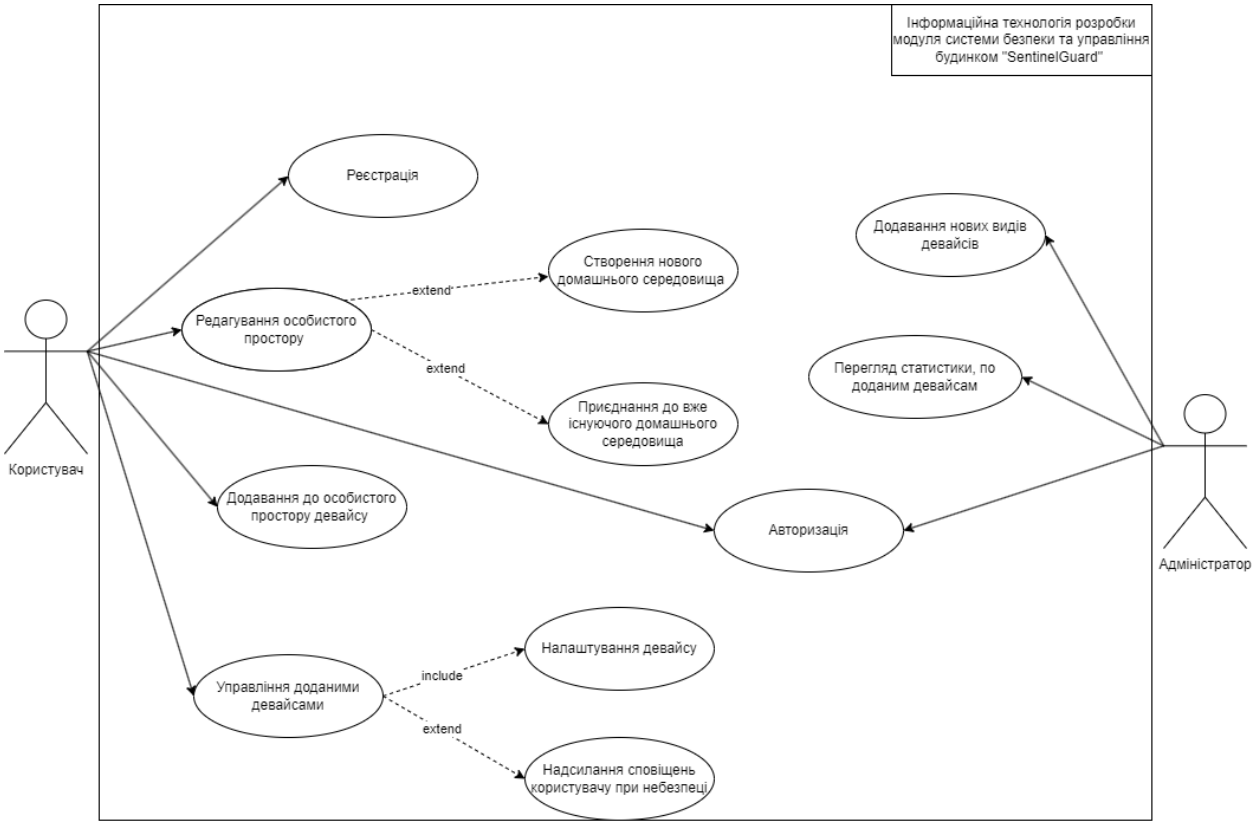


Рисунок 2.1 – Діаграма варіантів використання

Акторами є Користувач та Адміністратор. Адміністратор може додавати нові види пристроїв до системи, які в подальшому будуть використовувати користувачі. Також у Адміністратора є можливість переглядати статистику по

тому скільки разів був доданий кожен пристрій користувачем до особистого простору.

2.3 Проєктування моделі бази даних

Створення фізичної моделі бази даних є кроком до ефективного та структурованого зберігання та управління даними і має важливе значення для успішної розробки та експлуатації інформаційних систем [28].

У ході проєктування Інформаційної системи SentinelGuard було створено фізичну модель бази даних (рис.2.2) та виділено наступні сутності:

– Homes – Інформація про домівку користувача.

– Users – Таблиця містить інформацію про користувачів системи.

– Devices – Інформація про конкретний пристрій присутній у системі дому.

– DeviceParams – Таблиця містить інформацію про параметри пристроїв.

– DeviceTypes – Види пристроїв які присутні у системі.

– DeviceTypeAttributes – Зв’язувальна таблиця для DeviceAttributes та DeviceTypes, для уникнення зв’язку «багато до багатьох».

– DeviceAttributes – Характеристика параметрів для пристроїв.

– Notifications – Інформація про сповіщення користувачу.

– Notification_levels – Таблиця містить інформацію про тип сповіщення яке має надійти користувачу.

– JoinRequests – Запит на приєднання користувача до вже існуючого особистого простору.

– JoinRequestStatus – Статус запиту на приєднання.

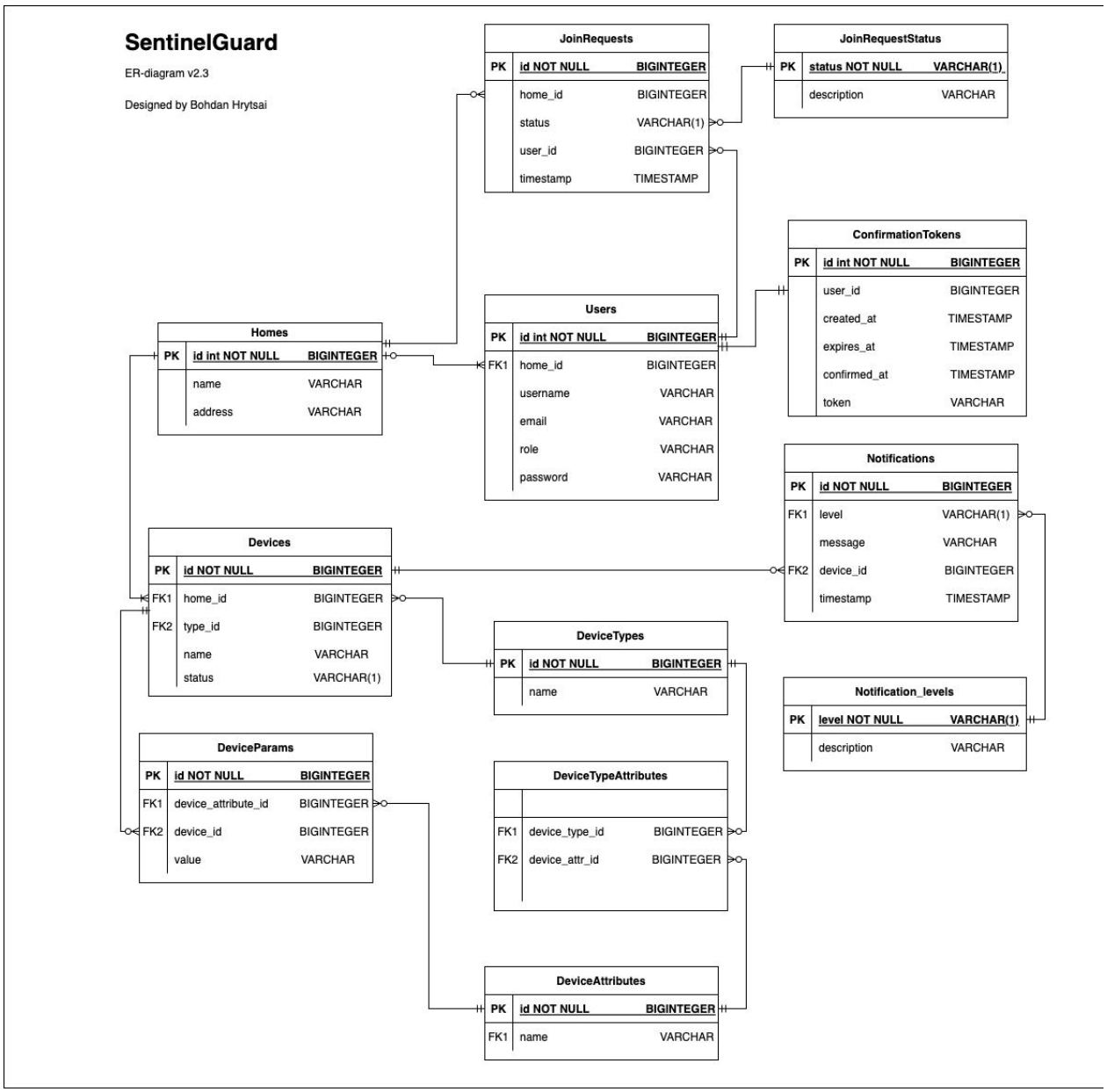


Рисунок 2.2 – Фізична модель бази даних

Даний етап дозволяє реалізувати визначену фізичну модель у конкретному технічному середовищі та стає базовим для подальшого програмування та реалізації функціоналу системи [29].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка структури інформаційної системи

Проєктування структури інформаційної системи має за мету візуалізувати логіку переходів та взаємодію користувачів з різними сторінками [30]. Це спрощує планування та розробку, забезпечуючи зручну навігацію та комунікацію між учасниками проєкту. Переглянути структуру для користувача можна на рисунку 3.1. Структура для адміністратора зображена на рисунку 3.2.

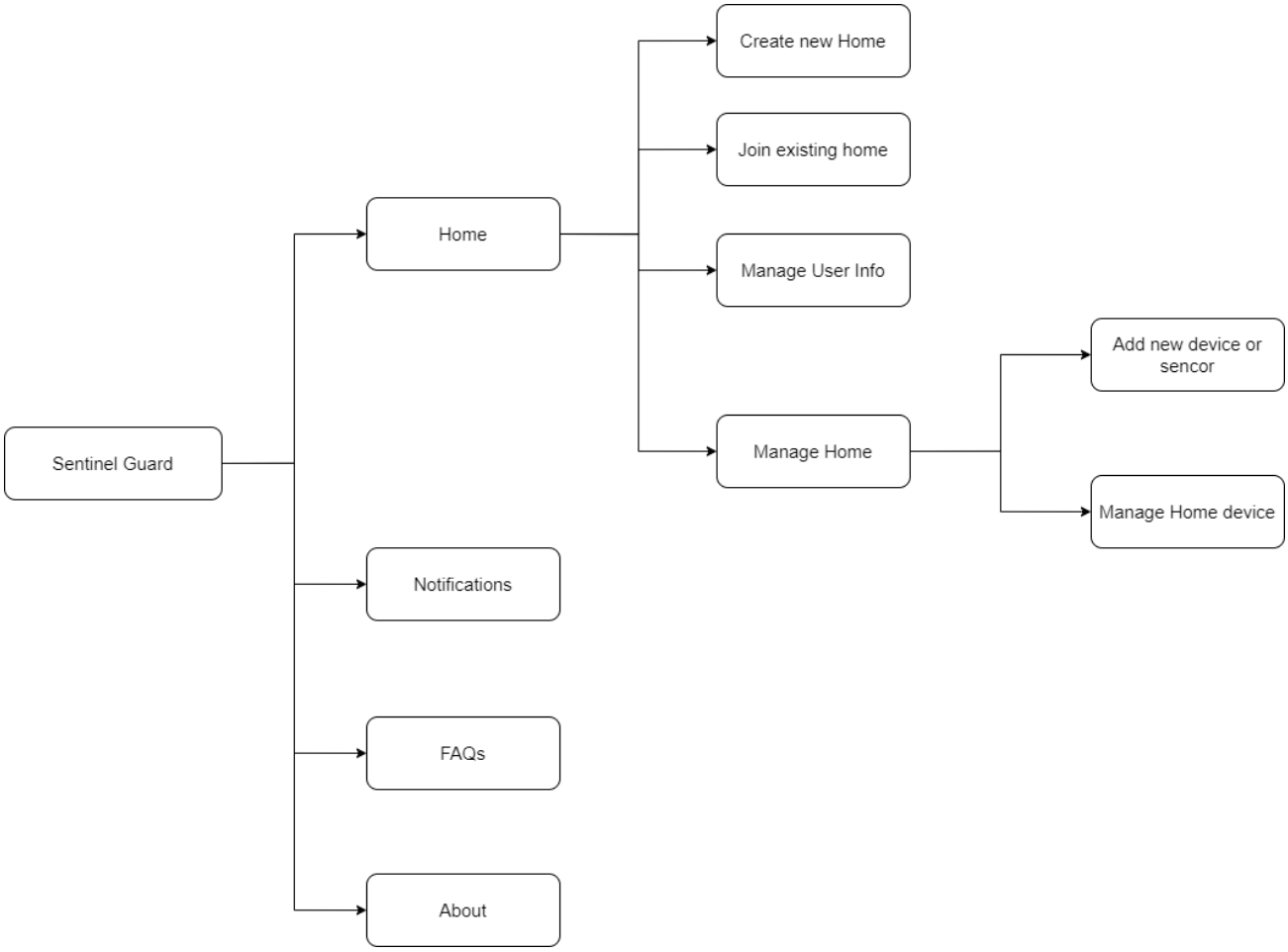


Рисунок 3.1 – Структура Інформаційної системи для користувача

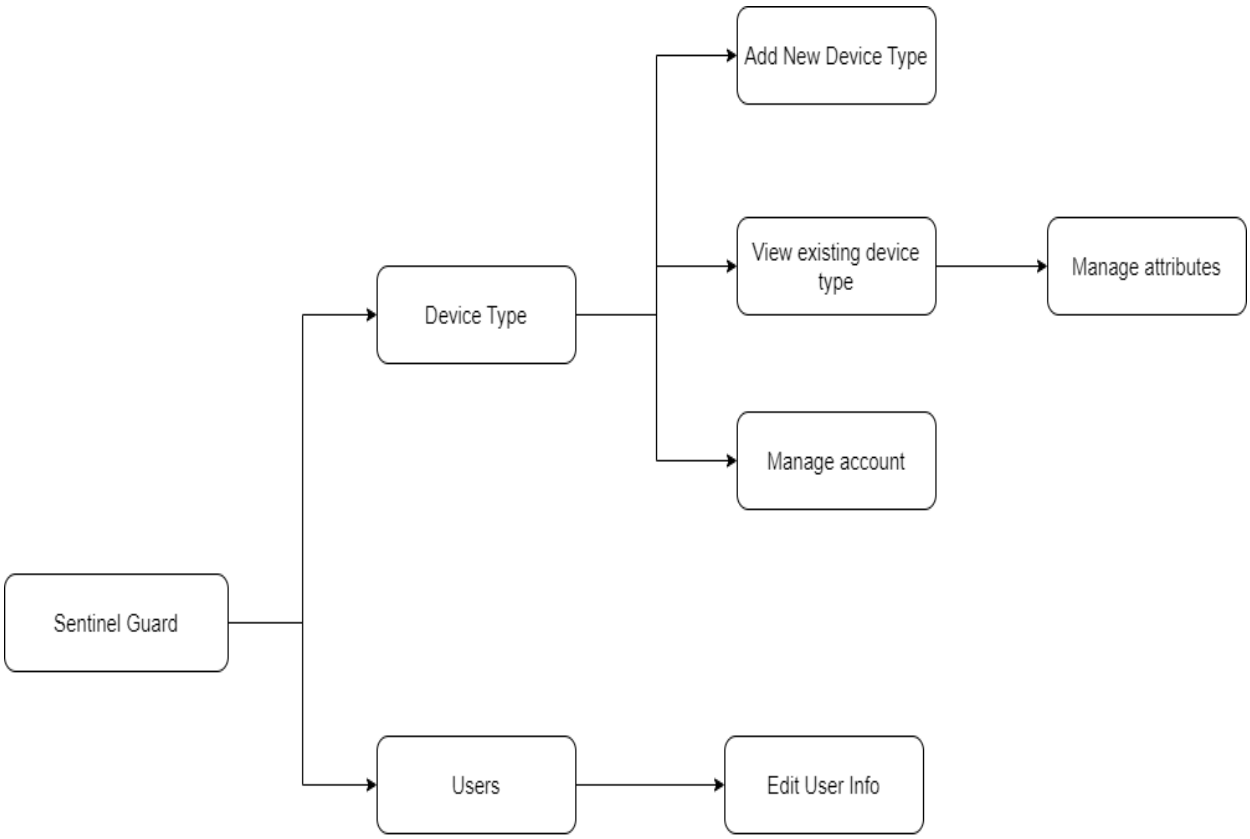


Рисунок 3.2 – Структура Інформаційної системи для адміністратора

Проєктування структури Інформаційної системи домагає полегшити оцінку обсягу роботи та визначити оптимальний функціоналу для кожної частини додатку [31].

3.2 Реалізація системи

3.2.1 Структура проєкту

Для розуміння того, як працює додаток варто розуміти з чого він складається. На рисунку 3.3 показано структуру проєкту.

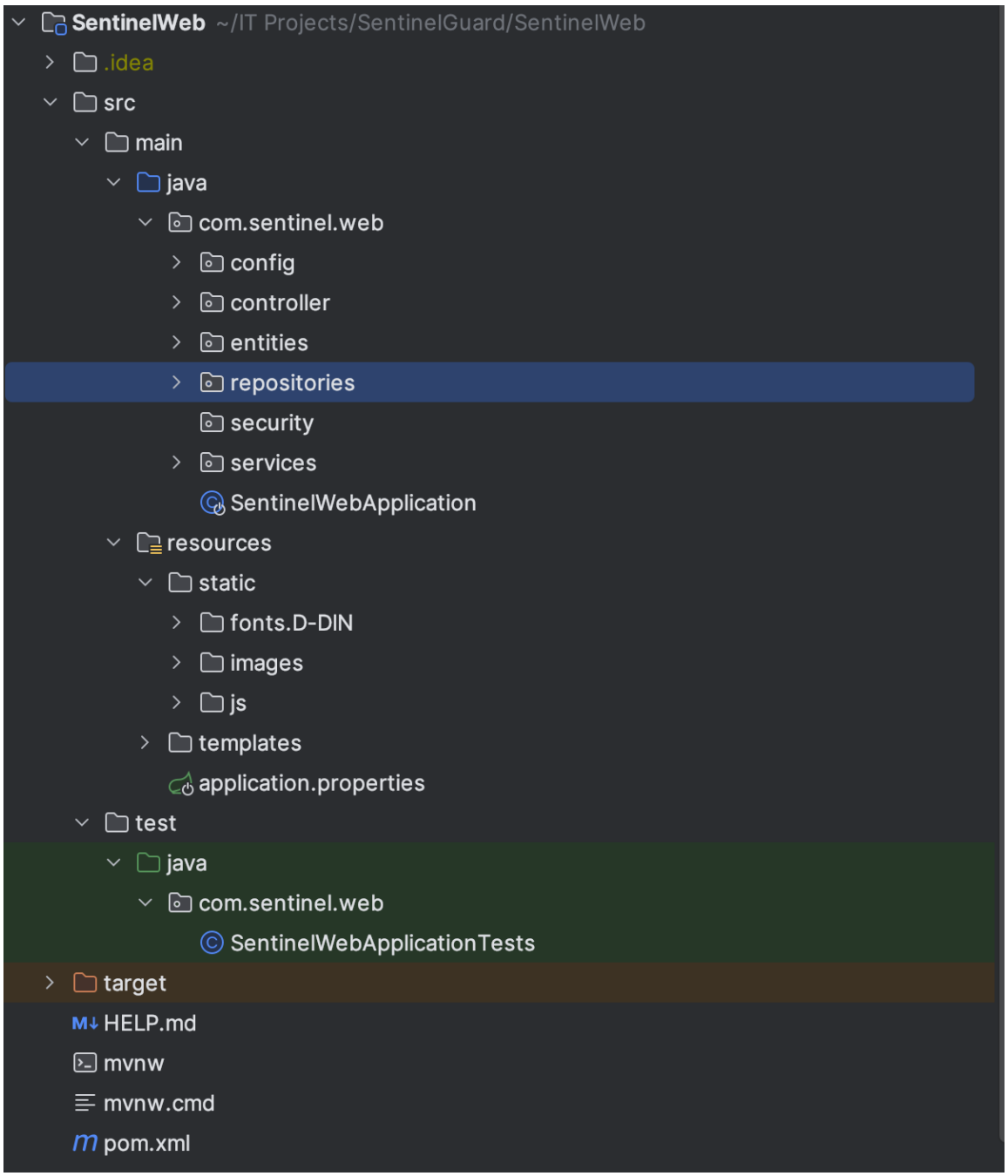


Рисунок 3.3 – Структура проєкту

Директорія SentinelWeb – є батьківською для всього проєкту.

Розглянемо детальніше вміст:

- src – містить весь вихідний код проєкту
  - java – містить Java класи з кодом
    - config – містить класи з Spring Bean конфігураціями;
    - controller – містить класи з контролерами;

        - entities – містить класи з моделями (сутностями) бази даних;
        - repository – містить класи з репозиторіями;
        - services – містить класи сервісів.
    - resources – містить різноманітні ресурси необхідні для роботи додатку
        - static – містить статичні ресурси, такі як JS скрипти, стилі, зображення та шрифти;
        - templates – містить шаблони сторінок, які використовуватимуться для веб-додатку;
        - application.properties – головний конфігураційний файл додатку.
- target – містить сгенеровані під час компіляції та збірки класи, ресурси та бінарні файли;
- pom.xml – Apache Maven файл який містить інформацію про залежності проєкту, допоміжні конфігурації та сценарії збірки.

3.2.2 Реалізація інтеграції з базою даних

Як було зазначено раніше для реалізації можливості використання бази даних, а саме взаємодії системи з базою даних у вигляді ORM (об’єктно-реляційного відображення) було використано можливості фреймворку Spring Data. Використання даного підходу обумовлено необхідністю зменшення кількості коду, а також задля додаткових валідацій, які допоможуть не схибити при створенні сутностей в коді.

Для того щоб відобразити сутність бази даних в Java коді необхідно створити відповідний клас, приклад можна побачити на рисунку 3.4.

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false, unique = true)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "home_id")
    private Home home;
    @Column(name = "username")
    private String username;
    @Column(name = "email", nullable = false)
    private String email;
    @Column(name = "password", nullable = false)
    private String password;
    @Enumerated(EnumType.STRING)
    private Role role;
    private Boolean enabled = false;
```

Рисунок 3.4 – Сутність User

При роботі з будь-якою реалізацією JPA (Spring Data в даному випадку) варто звертати увагу на правильність використання анотацій.

- `@Id` – вказує, що поле є головним ключом в таблиці;
- `@GeneratedValue` - вказує на те, що ідентифікатор буде згенеровано певною, вказаною в дужках, стратегією;
- `@Column` – вказує на те, що поле є відображенням колонки в таблиці (є необов'язковою, якщо назва поля класу та таблиці співпадають)
- `@ManyToOne` – вказує на зв'язок «багато-до-одного» по відношенню до даної сутності;
- `@JoinColumn` – вказує на те, яка саме колонка виористовуватиметься для побудови зв'язку.

Для того щоб почати взаємодію з будь-якою сутністю необхідно налаштувати додаток для отримання доступу до конкретного інстансу бази даних. Це досягається за рахунок конфігурації (рис.3.5).

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=12345

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true
```

Рисунок 3.5 – Конфігурація доступу до бази даних

Варто розглянути кожне налаштування окремо:

- `spring.datasource.url` – рядок підключення до бази даних, який окрім службової інформації СУБД також містить хост, порт та назву бази даних, до якої проводитиметься підключення;
- `spring.datasource.username` – користувач в базі даних;
- `spring.datasource.password` – пароль для вказаного користувача;
- `spring.jpa.database-platform` – діалект бази даних;
- `spring.jpa.hibernate.ddl-auto=update` – вказує на те, що якщо при запуску програму виявлено відмінності між DDL бази даних та класами сутностей, то необхідно оновити базу даних відповідно до класів сутностей;
- `spring.jpa.show-sql=false` – вказує на необхідність відображення виконуваних запитів в консолі;
- `spring.jpa.properties.hibernate.format_sql=true` – вказує на необхідність форматування запитів у зрозумілий для людини формат.

У більшості випадків даних налаштувань достатньо для запуску та роботи системи.

Також для кожної сутності, з якою буде взаємодіяти система необхідно оголосити репозиторії. Варто зазначити, що найбільш популярні операції такі як вибірка всіх значень, вибірка значення по ідентифікатору, збереження чи видалення запису з таблиці – заздалегідь включені та реалізовують автоматично фреймворком і немає необхідності робити це вручну. Нижче наведено декілька прикладів класів з репозиторіями для сутностей. Тут також варто зазначити, що якщо запит не є дуже простим, то можна проставити анотацію @Query над сигнатурою методу і вказати конкретний запит до БД (рис.3.6, рис.3.7).

```
@Repository
public interface ConfirmationTokenRepository extends JpaRepository<ConfirmationToken, Long> {

    1 usage  Bohdan Hrytsai
    Optional<ConfirmationToken> findByToken(String token);

    1 usage  Bohdan Hrytsai
    @Transactional
    @Modifying
    @Query("UPDATE ConfirmationToken c " +
            "SET c.confirmedAt = ?2 " +
            "WHERE c.token = ?1")
    int updateConfirmedAt(String token,
                          LocalDateTime confirmedAt);
}
```

Рисунок 3.6 – Репозиторій для сутності Confirmation Token

```
public interface HomeRepository extends JpaRepository<Home, Long> {
    1 usage  Bohdan Hrytsai
    @Query("SELECT u.home FROM User u WHERE u.id = :userId")
    Home findHomeByUserId(@Param("userId") Long userId);

    1 usage  Bohdan Hrytsai
    @Query("SELECT COUNT(DISTINCT h) FROM Home h JOIN h.devices d WHERE d.type.id = :deviceTypeId")
    Long countDistinctByDeviceTypeId(Long deviceTypeId);
    1 usage  Bohdan Hrytsai
    @Query("SELECT DISTINCT u.home FROM User u WHERE u.email = :email ")
    Optional<Home> findHomeByUserEmail(@Param("email") String email);
}
```

Рисунок 3.7 – Репозиторій для сутності Home

У всіх інших випадках назва методу вказуватиме фреймворку, що саме потрібно виконати (рис. 3.8).

```
public interface NotificationRepository extends JpaRepository<Notification, Long> {
    1 usage  Bohdan Hrytsai
    Optional<List<Notification>> findByDeviceHomeId(Long homeId);
    1 usage  Bohdan Hrytsai
    Optional<List<Notification>> findByDeviceId(Long deviceId);
}
```

Рисунок 3.8 – Репозиторій для сутності Notification

Якщо жодного методу в класі не визначено, то це значить що буде використано лише найпростіші запити до конкретної сутності бази даних (рис.3.9).

```
public interface NotificationLevelRepository extends JpaRepository<NotificationLevel, Long> {

}
```

Рисунок 3.9 – Репозиторій для сутності Notification Level

Кожен метод репозиторію – це абстрактне представлення якогось конкретного запиту до бази даних. Абстрактне тому що запит не містить якихось конкретних значень, наприклад, для фільтрації вибірки.

3.2.3 Реалізація контролерів

Контролери необхідні для того, щоб пов’язувати між собою шар бізнес логіки з моделями та представлення. Кожен контролер (або метод в ньому) повинен мати відносний шлях (url), який опрацьовуватиме. Також для кожного методу необхідно вказати який саме тип реквесту він може опрацювати POST, GET або ж інший. Це досягається завдяки анотаціям `@GetMapping` або `@PostMapping`.

Кожен клас контролеру помічається анотацією `@Controller`, в якій додатково можна вказати загальний мапінг для всіх методів даного контролеру.

У випадку, якщо конкретний контролер працює саме з веб-сторінками, метод контролера має повертати рядкове значення шляху до місцезнаходження шаблону конкретної веб-сторінки. Якщо потрібно відкрити не сторінку, а якийсь шлях то вказується ключове слово redirect.

Приклад реалізації контролеру для мапінгу /user рображено на рисунку 3.10 та рисунку 3.11.

```
@Controller
@RequestMapping("/user")
public class UserController {
    3 usages
    private UserRepository userRepository;
    2 usages
    private BCryptPasswordEncoder passwordEncoder;

    Bohdan Hrytsai
    @Autowired
    public UserController(UserRepository userRepository, BCryptPasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    Bohdan Hrytsai
    @GetMapping("/editUser")
    public String editUser(@AuthenticationPrincipal User user, Model model) {
        model.addAttribute( attributeName: "user", user);

        return "user/editUser";
    }
```

Рисунок 3.10 – Приклад контролеру

```java
@PostMapping("/editUser")
public String editUser(@AuthenticationPrincipal User user,
                       @RequestParam String email,
                       @RequestParam String userName,
                       RedirectAttributes redirectAttributes) {
    user.setEmail(email);
    user.setUsername(userName);
    userRepository.save(user);

    redirectAttributes.addFlashAttribute( attributeName: "alertSuccess",  attributeValue: "User updated successfully");

    if (user.isAdmin()) {
        return "redirect:/admin/users";

    } else {
        return "redirect:/home";
    }

}
```

Рисунок 3.11 – Продовження прикладу контролеру

3.2.4 Реалізація процесу додавання нового типу пристрою

Систему реалізовано таким чином, що новий тип підтримуваного системою пристрою може додати лише користувач з групи адміністраторів. Тому необхідно залогінитись під цим юзером і відкрити сторінку Device Types (рис.3.12).

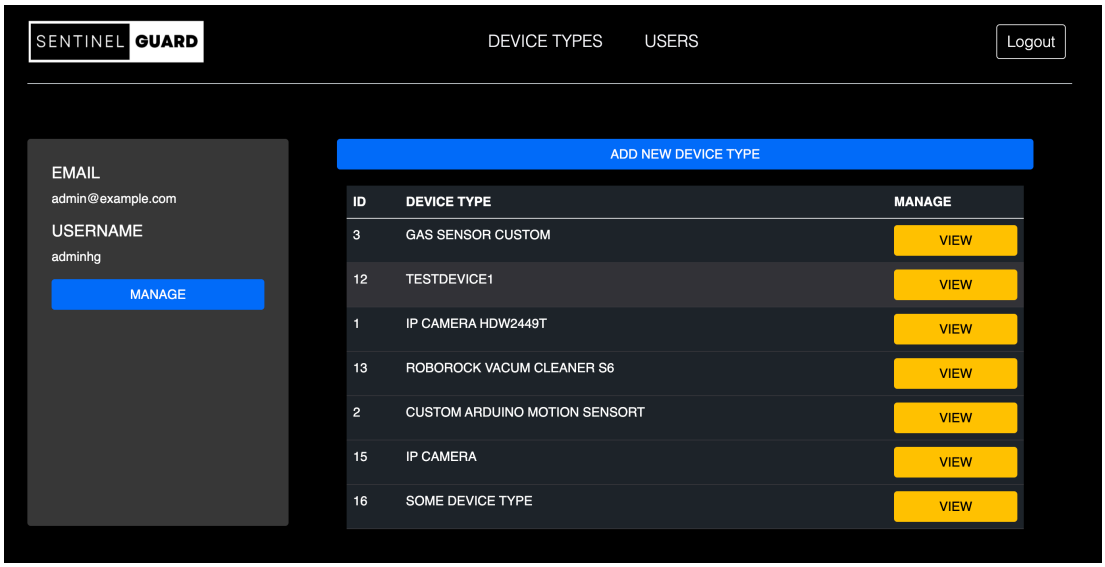


Рисунок 3.12 – Сторінка Device Types

Після натискання кнопки ADD NEW DEVICE TYPE запит потрапляє в Spring Dispatcher Servlet і направляється відповідному контролеру. У даному

випадку за це відповідає клас та метод контролера зображений на рисунку 3.13.

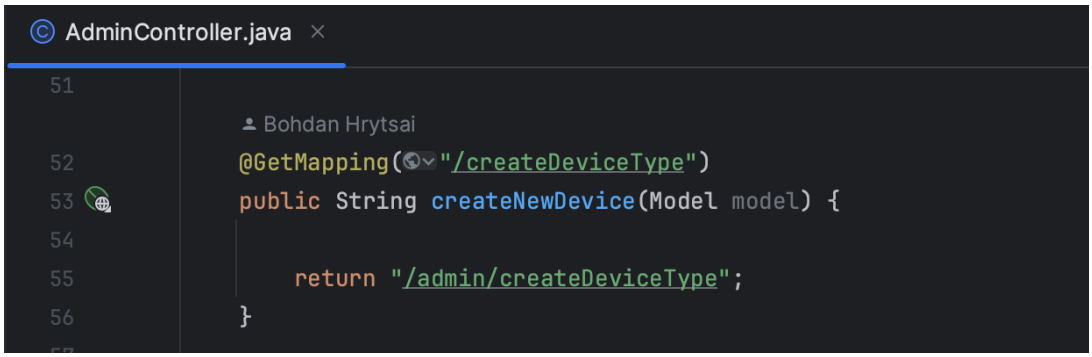


Рисунок 3.13 – Опрацювання контролером після натискання кнопки

У коді не виконується нічого окрім повернення методом шляху до сторінки, яку буде відображено користувачу (рис.3.14).

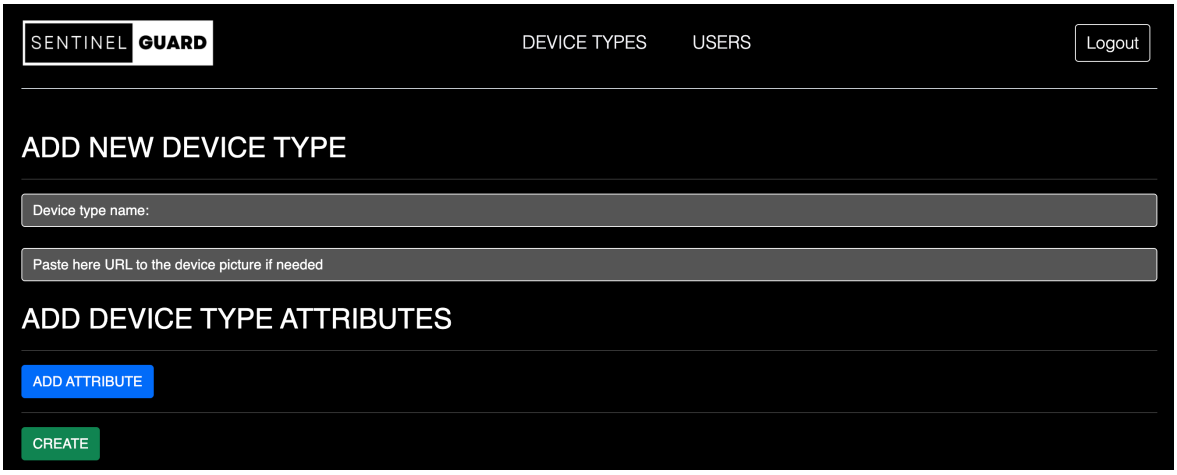


Рисунок 3.14 – Сторінка додавання нового типу пристрою

Вихідний код даної сторінки представлено на рисунках 3.15, 3.16, 3.17. Кожен девайс може мати певну кількість характеристик, які називаються атрибутами.

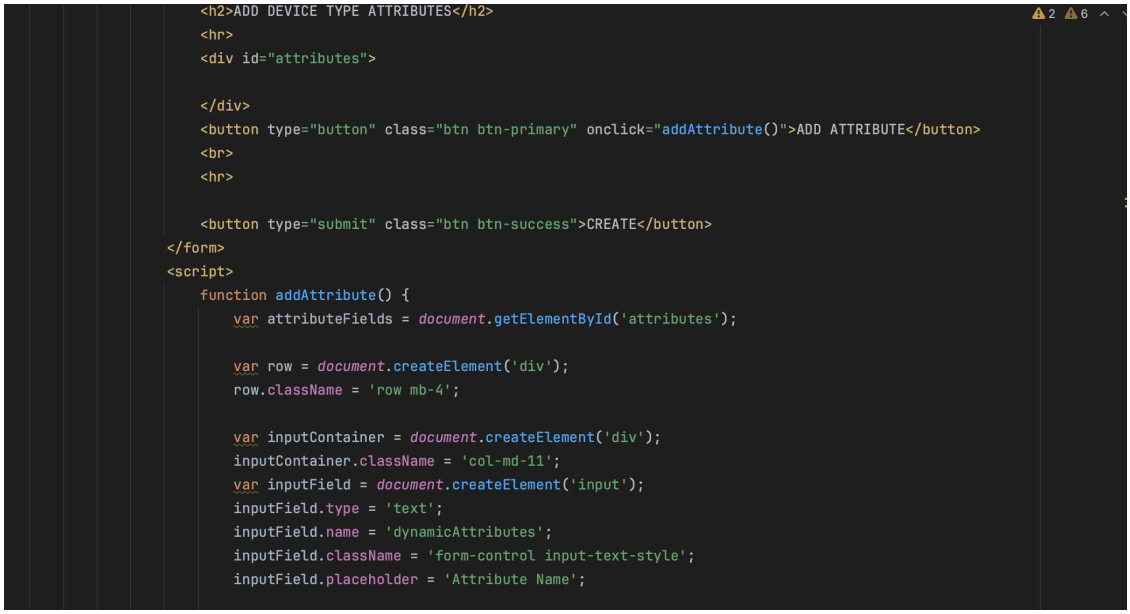
Користувач може додавати будь-яку кількість полей атрибутів натисканням на кнопку ADD ATTRIBUTE (рис. 3.18). Для реалізації можливості динамічного створення необхідної кількості полей було створено JavaScript функцію `addAttribute` ().

```
<#import "../parts/page.ftl" as c>
<#include "../parts/security.ftl">

<@c.page>
    <style>
        .input-text-style {
            background-color: #565656;
            border: 1px solid #fff;
        }

        .input-text-style::placeholder {
            color: #fff;
        }
    </style>
    <div class="row">
        <div class="col-md-12" >
            <#if alertError??>
                <div class="alert alert-danger" role="alert">${alertError}</div>
            </#if>
            <h2>ADD NEW DEVICE TYPE</h2>
            <hr>
            <div class="margin-3"></div>
            <div>
                <form method="post" action="/admin/createDeviceType">
                    <input type="text" name="deviceTypeName" placeholder="Device type name: " class="form-control input-text-style"
                    <input type="url" name="imageUrl" placeholder="Paste here URL to the device picture if needed" class="form-contr
```

Рисунок 3.15 – Вихідний код сторінки «Додавання нового девайсу»



```
                    <h2>ADD DEVICE TYPE ATTRIBUTES</h2>
                    <hr>
                    <div id="attributes">

                    </div>
                    <button type="button" class="btn btn-primary" onclick="addAttribute()">ADD ATTRIBUTE</button>
                    <br>
                    <hr>

                    <button type="submit" class="btn btn-success">CREATE</button>
                </form>
                <script>
                    function addAttribute() {
                        var attributeFields = document.getElementById('attributes');

                        var row = document.createElement('div');
                        row.className = 'row mb-4';

                        var inputContainer = document.createElement('div');
                        inputContainer.className = 'col-md-11';
                        var inputField = document.createElement('input');
                        inputField.type = 'text';
                        inputField.name = 'dynamicAttributes';
                        inputField.className = 'form-control input-text-style';
                        inputField.placeholder = 'Attribute Name';
```

Рисунок 3.16 – Продовження вихідного коду сторінки «Додавання нового пристрою»

```
                var deleteContainer = document.createElement('div');
                deleteContainer.className = 'col-md-1';
                var deleteButton = document.createElement('button');
                deleteButton.type = 'button';
                deleteButton.className = 'btn btn-danger delete-button w-100';
                deleteButton.innerHTML = '&#10006;';
                deleteButton.onclick = function () {
                    attributeFields.removeChild(row);
                };

                inputContainer.appendChild(inputField);
                deleteContainer.appendChild(deleteButton);

                row.appendChild(inputContainer);
                row.appendChild(deleteContainer);

                attributeFields.appendChild(row);
            }
        </script>
    </div>
</div>
</div>

</@c.page>
```

Рисунок 3.17 – Продовження вихідного коду сторінки «Додавання нового пристрою»

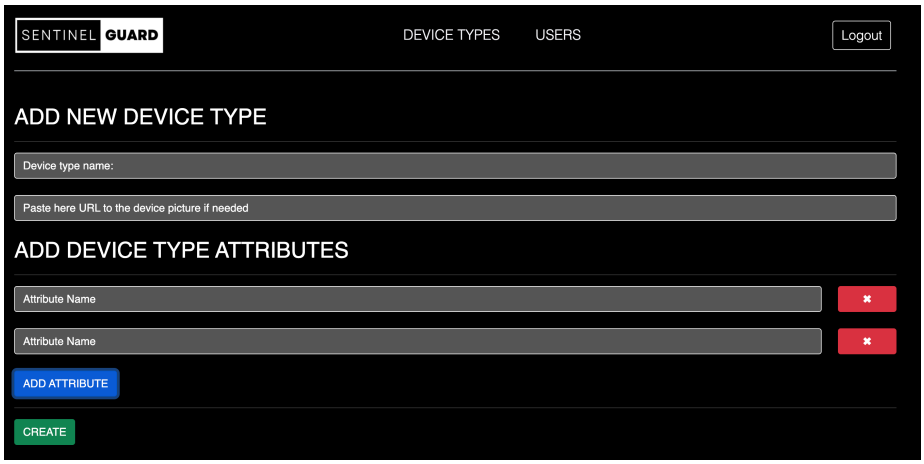


Рисунок 3.18 – Динамічне створення полей атрибутів.

Після заповнення всіх необхідних полей і натискання кнопки CREATE починає виконуватись код контролеру (рис. 3.19, рис.3.20).

```java
@PostMapping("/createDeviceType")
public String createNewDevice(@RequestParam String deviceTypeName,
                               @RequestParam(required = false) String imageUrl,
                               @RequestParam(value = "dynamicAttributes", required = false) List<String> dynamicAttributes,
                               RedirectAttributes redirectAttributes,
                               Model model) {

    if (deviceTypeRepository.findByName(deviceTypeName).isEmpty()) {
        DeviceType deviceType = new DeviceType();
        deviceType.setName(deviceTypeName);
        deviceType.setImageUrl(imageUrl);

        if (dynamicAttributes != null && !dynamicAttributes.isEmpty()) {
            List<DeviceAttribute> deviceAttributes = new ArrayList<>();
            dynamicAttributes.forEach(attrStr -> {
                if (!attrStr.isBlank()) {
                    deviceAttributeRepository.findByName(attrStr)
                            .ifPresentOrElse(
                                    deviceAttributes::add,
                                    () -> {
                                        DeviceAttribute deviceAttribute
                                                = new DeviceAttribute(attrStr);
                                        deviceAttributeRepository.save(deviceAttribute);
                                        deviceAttributes.add(deviceAttribute);
                                    });
                }
            });
```

Рисунок 3.19 – Опрацювання POST реквеста створення типу пристрою

```java
                deviceType.setDeviceAttributes(deviceAttributes);
            }

            deviceTypeRepository.save(deviceType);

            redirectAttributes.addFlashAttribute( attributeName: "alertSuccess",
                    attributeValue: "Device Type " + deviceTypeName + " created successfully");
            return "redirect:/admin/devices";
        } else {
            model.addAttribute(
                    attributeName: "alertError",
                    attributeValue: "Device Type with name " + deviceTypeName +" already exists");
            return "/admin/createDeviceType";

        }
    }
```

Рисунок 3.20 – Продовження опрацювання POST реквеста створення типу девайсу

У випадку успішного створення типу пристрою користувача буде перенаправлено на сторінку з усіма типами пристроїв з додаванням відповідного alertSuccess месседжу.

3.3 Використання інформаційної системи

3.3.1 Використання Інформаційної системи адміністратором

Коли користувач переходить до інформаційної системи Sentinel Guard – першим ділом він потрапляє до домашньої сторінки (рис. 3.21). Щоб

продовжити роботу з інформаційною системою – необхідно авторизуватися або зареєструватися. Переглянути форми сторінок авторизації та реєстрації можна на рисунку 3.22 та рисунку 3.24.

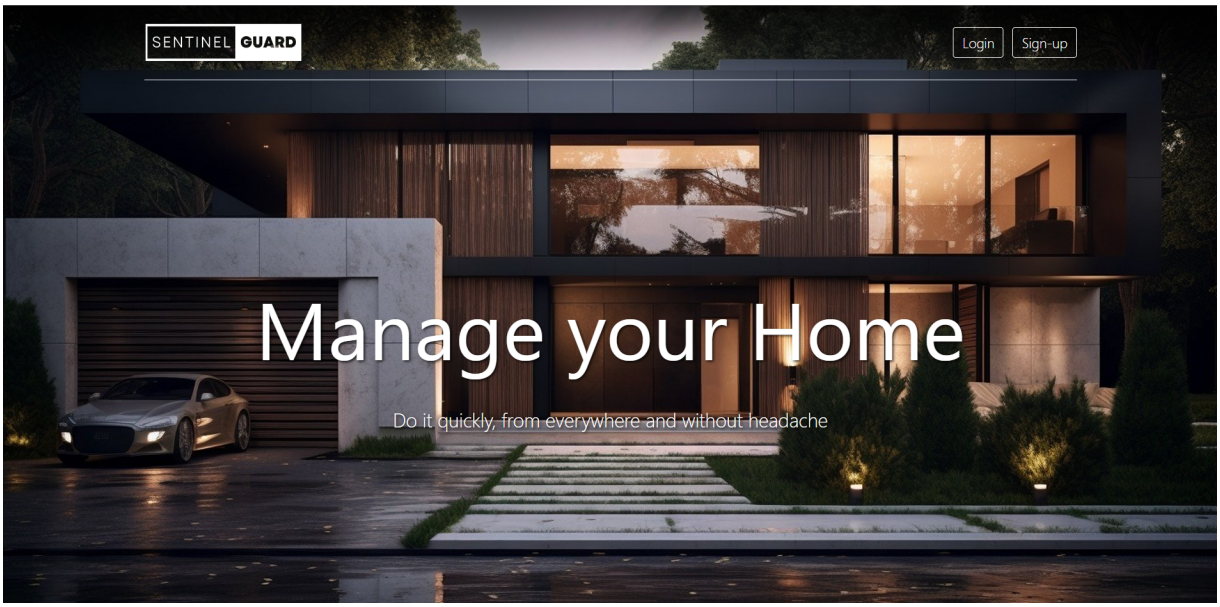


Рисунок 3.21 – Головна сторінка Інформаційної системи Sentinel Guard

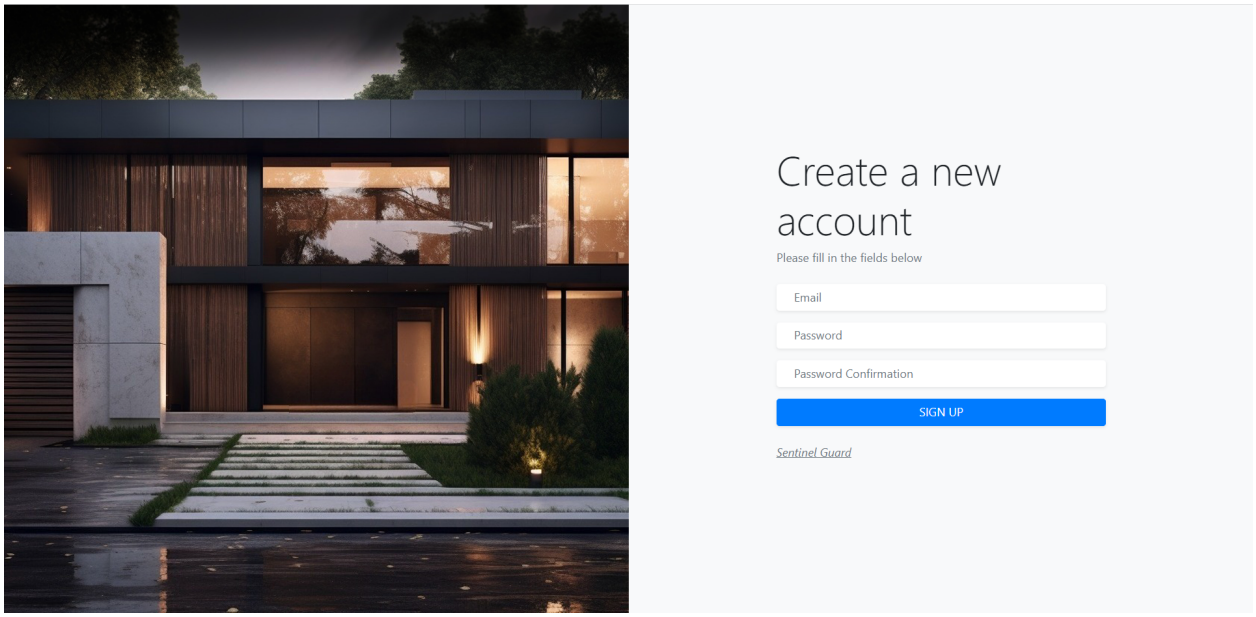


Рисунок 3.22 – Сторінка для реєстрації нового користувача

Після реєстрації, користувачу на електронну пошту надходить посилання, на яке потрібно перейти, щоб успішно завершити процес реєстрації до інформаційної системи (рис.3.23).

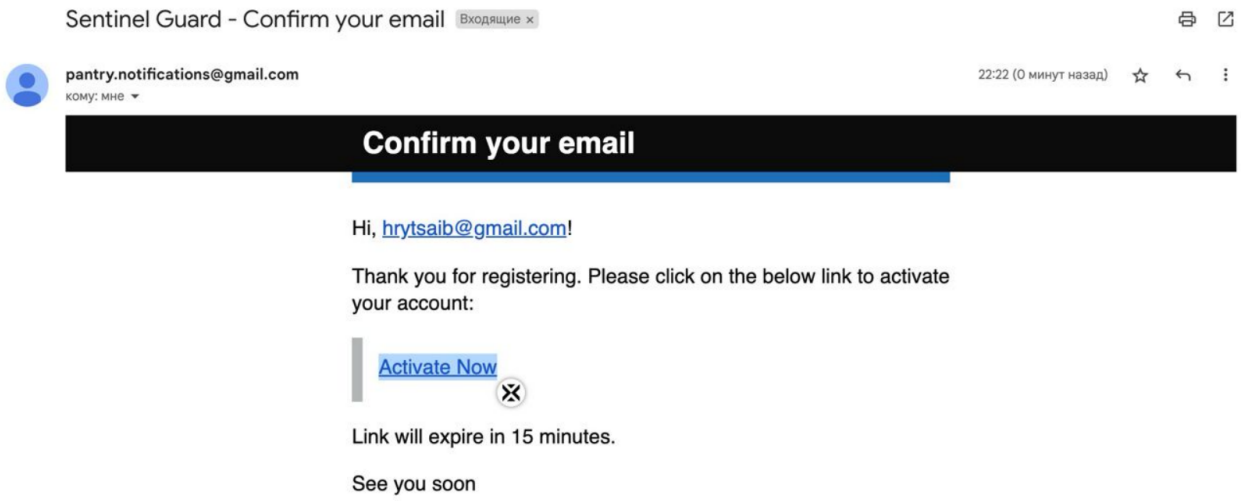


Рисунок 3.23 – Лист надісланий на електронну пошту користувача при реєстрації до інформаційної системи

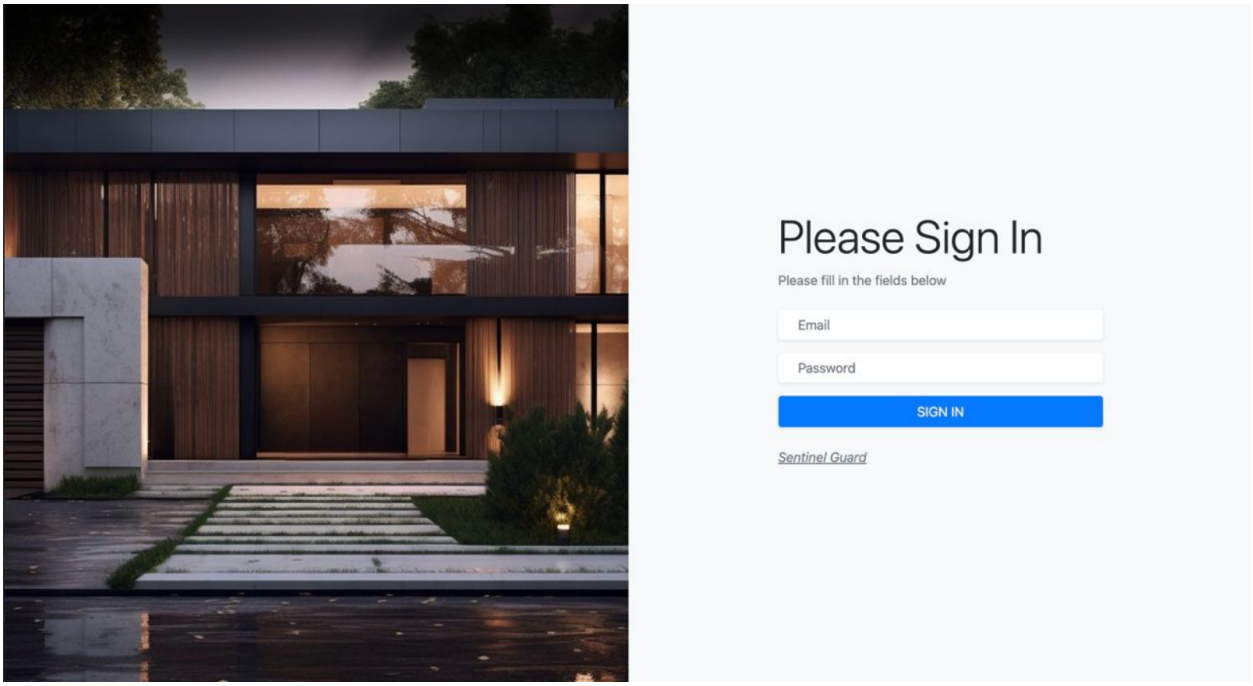


Рисунок 3.24 – Сторінка авторизації

Сторінка «Device Types» (рис.3.25) містить інформацію про типи пристроїв, які доступні для додавання користувачеві в особистий простір у інформаційній системі.

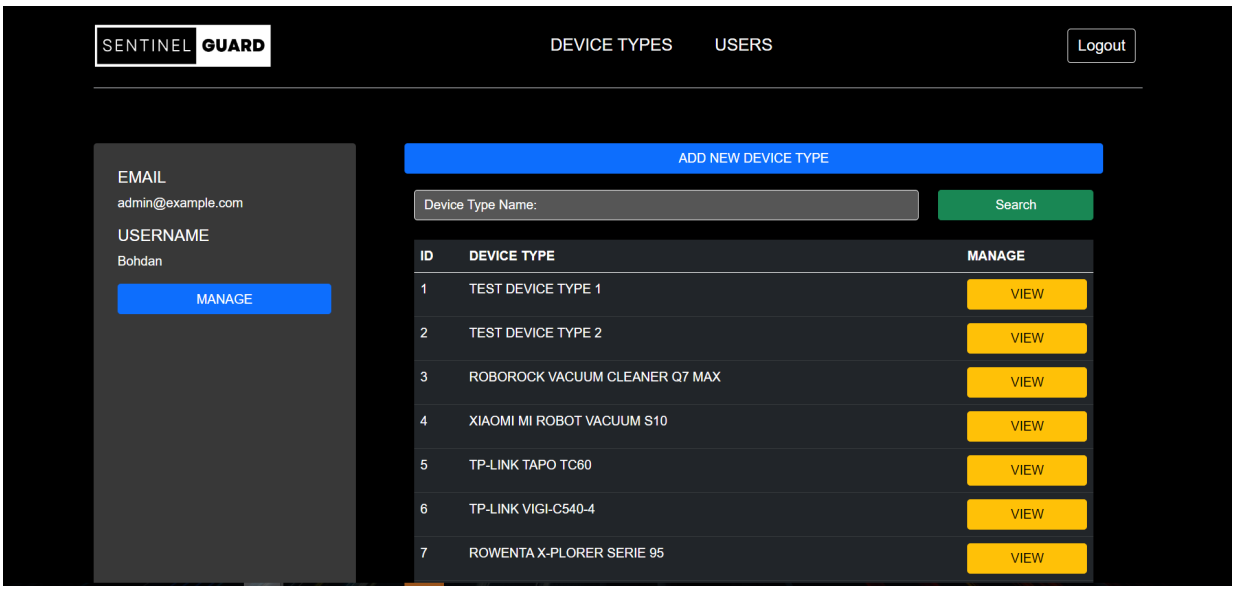


Рисунок 3.25 – Сторінка «Device Types»

Можливо здійснювати пошук по назві пристрою на сторінці Device Types. Здійснений пошук зображений на рисунку 3.25.

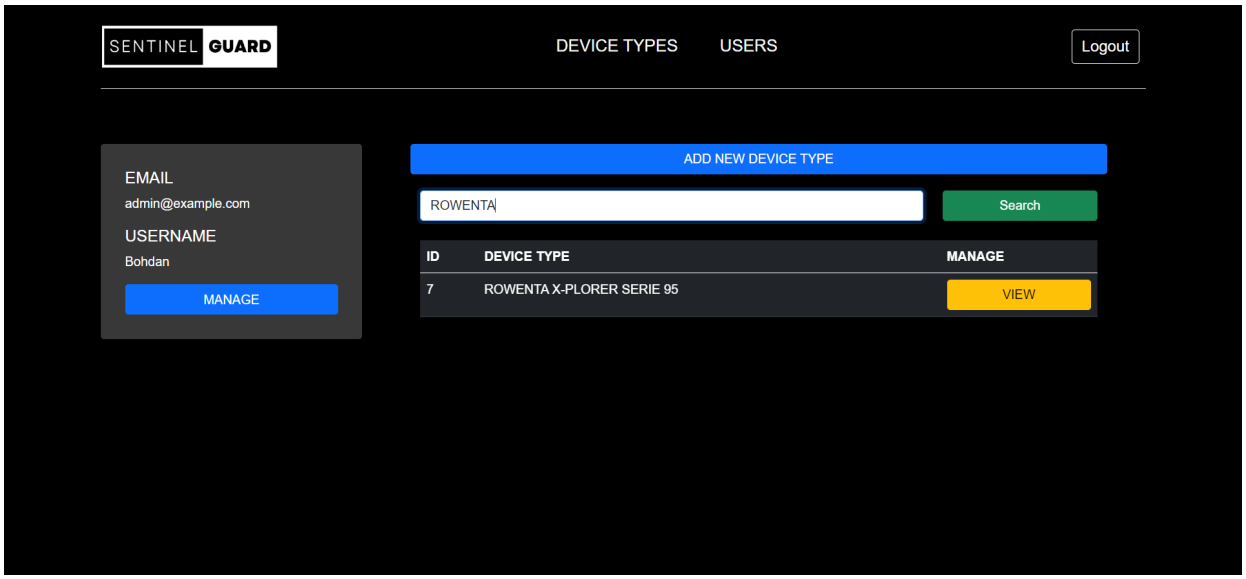


Рисунок 3.26 – Можливість пошуку по по назві пристрою на сторінці Device Types.

Для того, щоб створити новий тип пристрою, необхідно натиснути на кнопку «Add New Device Type». Для додавання нового типу пристрою необхідно ввести назву пристрою та вставити посилання на зображення. Далі для кожного типу пристрою необхідно створити певні атрибути, які

користувачу потрібно буде заповнити при додаванні до особистого простору. Процес додавання нового типу пристрою можна переглянути на рисунку 3.27. Вже створений пристрій зображено на рисунку 3.28.

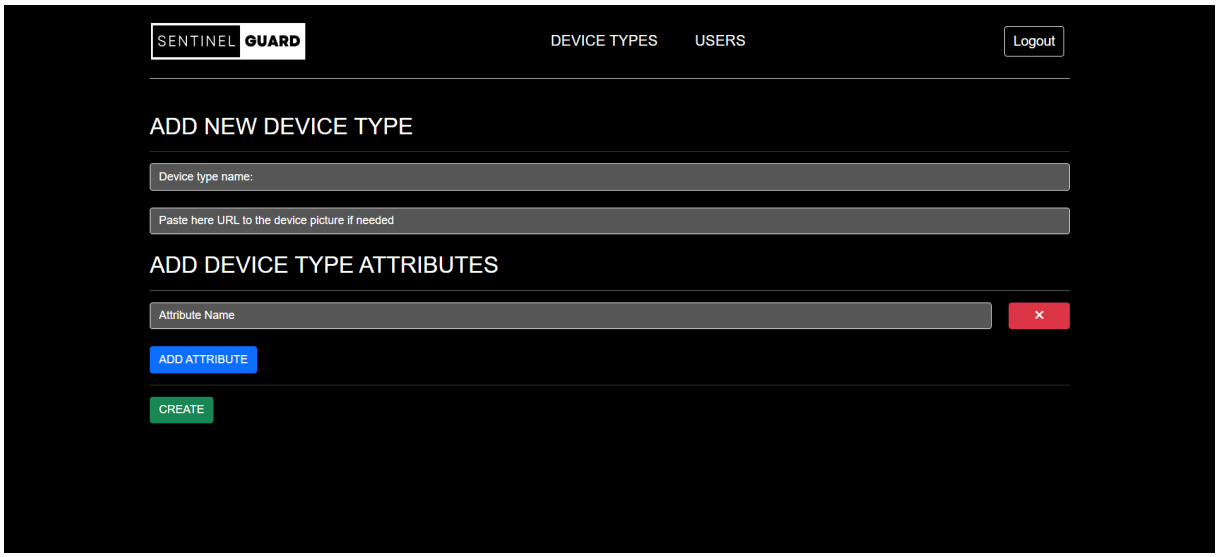


Рисунок 3.27 – Додавання нового типу пристрою.

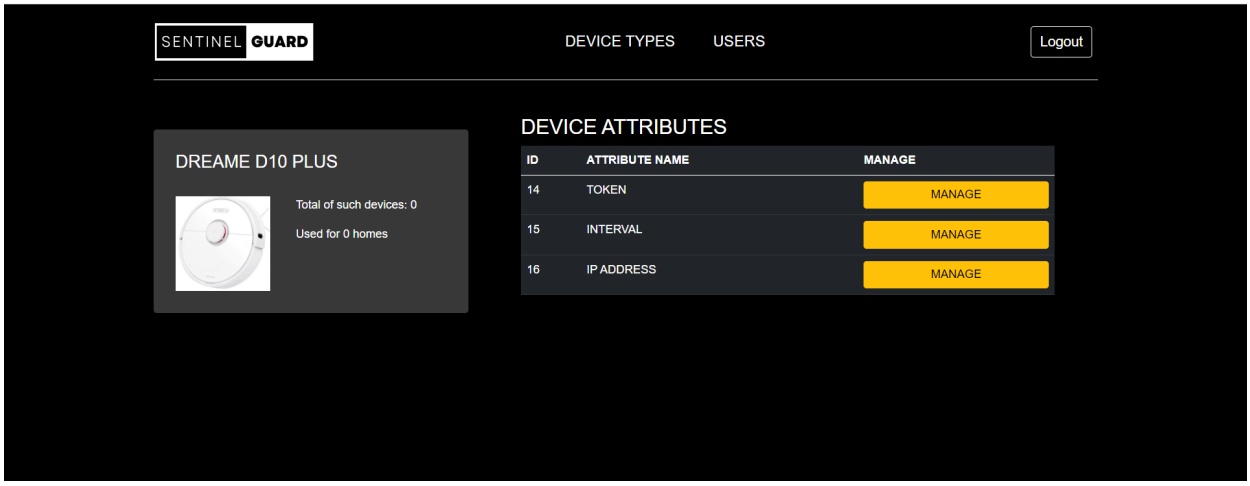


Рисунок 3.28 – Перегляд створеного типу пристрою.

На сторінці "Device Type" також відображена статистика для кожного типу пристрою, включаючи загальну кількість доданих пристроїв і інформацію про те, скільки домівок використовують ці пристрої (рис. 3.29).

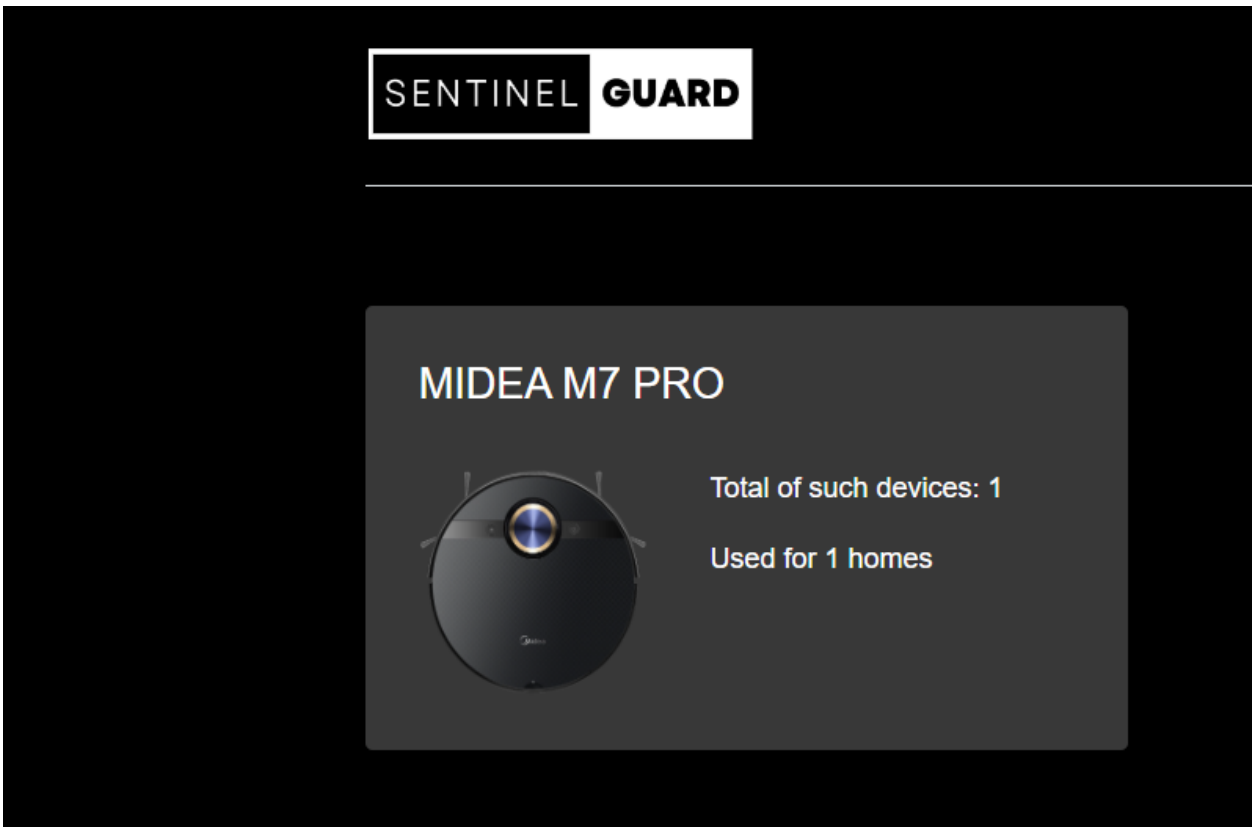


Рисунок 3.29 – Перегляд статистики по доданих пристроях.

Сторінка «Users» (рис.3.30) містить інформацію про всіх користувачів які зареєстровані в інформаційній системі Sentinel Guard. Адміністратор має можливість також редагувати користувацьку інформацію, а саме:

– Email користувача.

– Username користувача.

– Можливість увімкнути або ж вимкнути користувача за допомогою параметра User Enabled. Якщо адміністратор включить параметр User Enabled, то користувач більше не зможе потрапити в систему під своїм обліковим записом.

Редагування користувацької інформації зображено на рисунку 3.31.

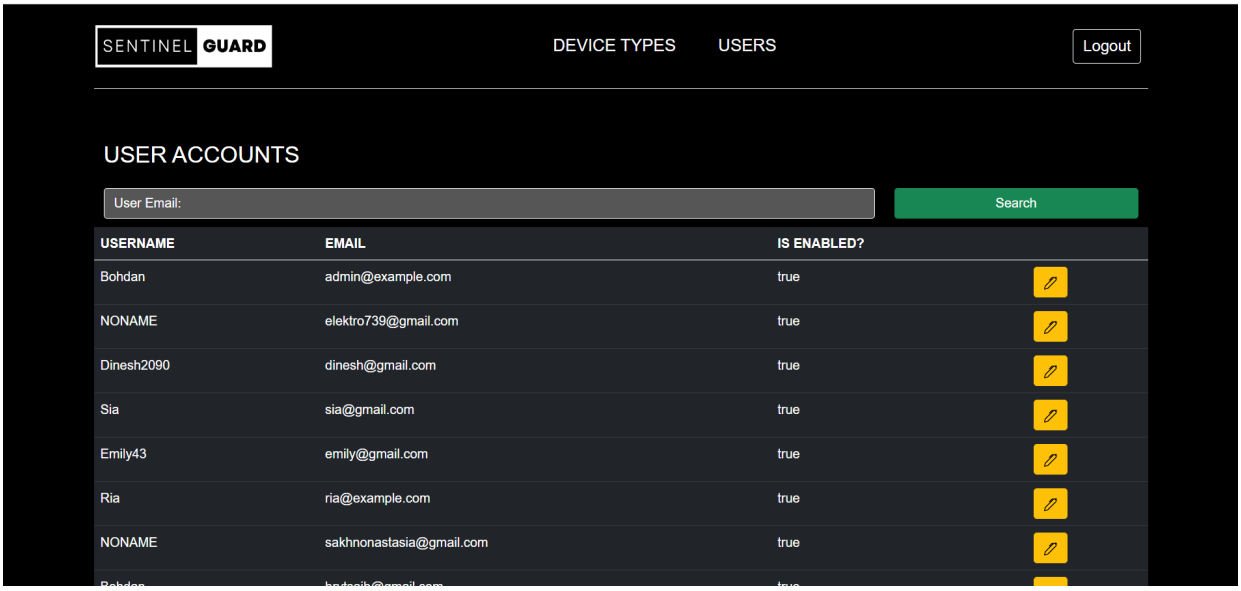


Рисунок 3.30 – Сторінка «Users».

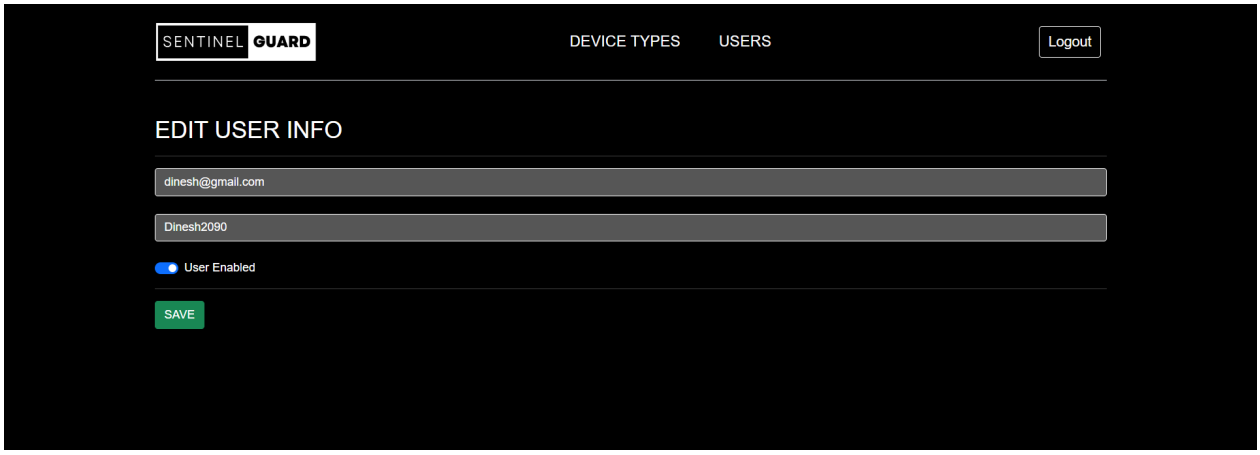


Рисунок 3.31 – Сторінка редагування інформації користувача.

3.3.2 Використання інформаційної системи користувачем

При авторизації до системи, користувач потрапляє на Головну сторінку. Структура інформаційної системи для авторизованого користувача представлена на рисунку 3.32.

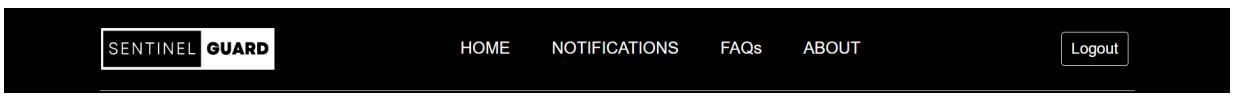


Рисунок 3.32 – Структура інформаційної системи авторизованого користувача.

Сторінка Home (рис.3.33) містить інформацію про особистий простір користувача. А саме, відображає Email, а також забезпечує можливість внесення змін до облікового запису (рис.3.34).

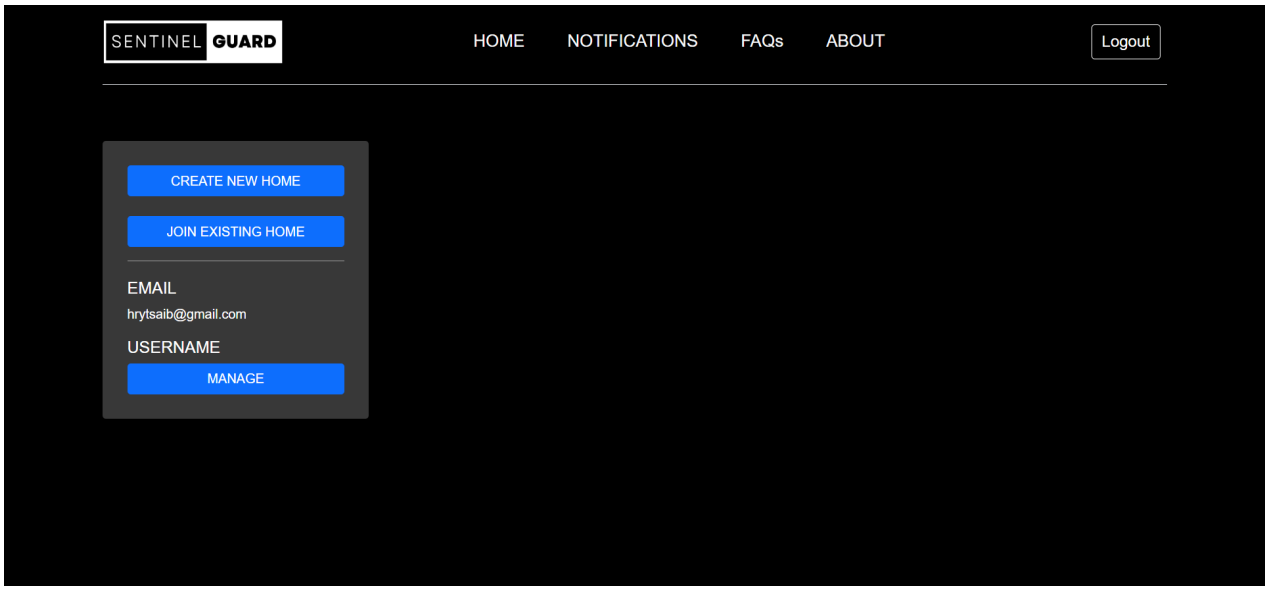


Рисунок 3.33 – Сторінка «Home».

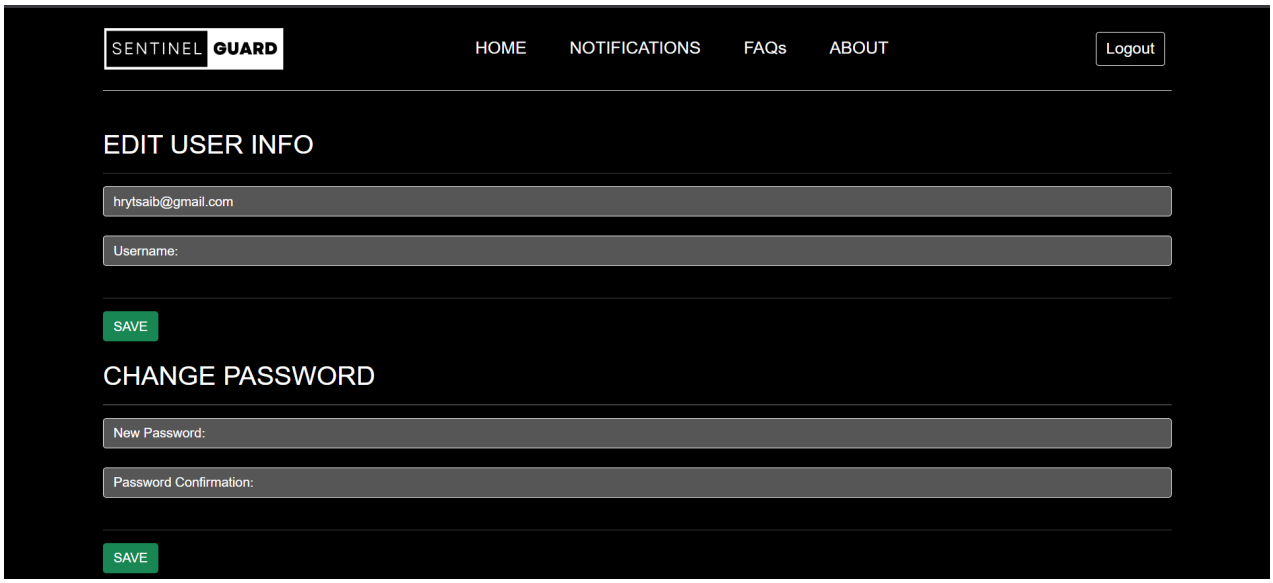


Рисунок 3.34 – Внесення змін до облікового запису

Користувач може створити нове або приєднатися до існуючого домашнього середовища.

Розглянемо випадок якщо користувач хоче створити нове домашнє середовище. Для цього необхідно натиснути кнопку Create New Home. Сторінка створення середовища зображена на рисунку 3.35. Успішне створення зображено на рисунку 3.36.

Рисунок 3.35 – Створення нового домашнього середовища.

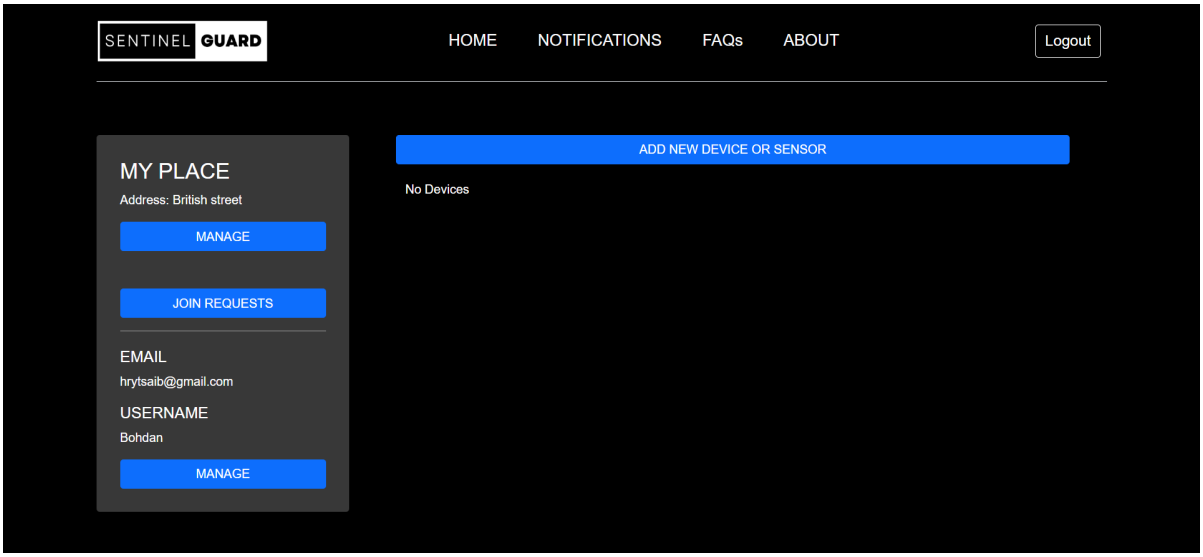


Рисунок 3.36 – Успішне створення нового домашнього середовища.

Наступним кроком необхідно додати новий пристрій до домашнього середовища, це можна зробити натиснувши кнопку Add New Device Or Sensor.

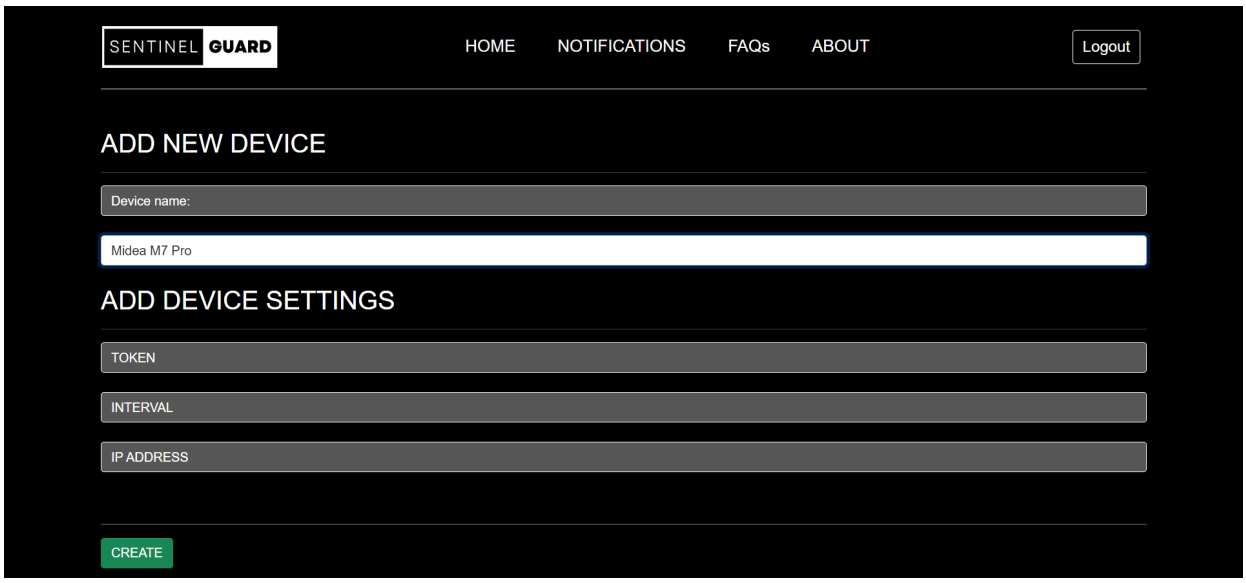


Рисунок 3.37 – Сторінка додавання нового пристрою до домашнього середовища

Доданими пристроями можливо управляти, натиснувши Edit – щоб змінити параметри пристрою або Remove – для того щоб видалити. Сторінка управління доданим пристроєм зображена на рисунку 3.38.

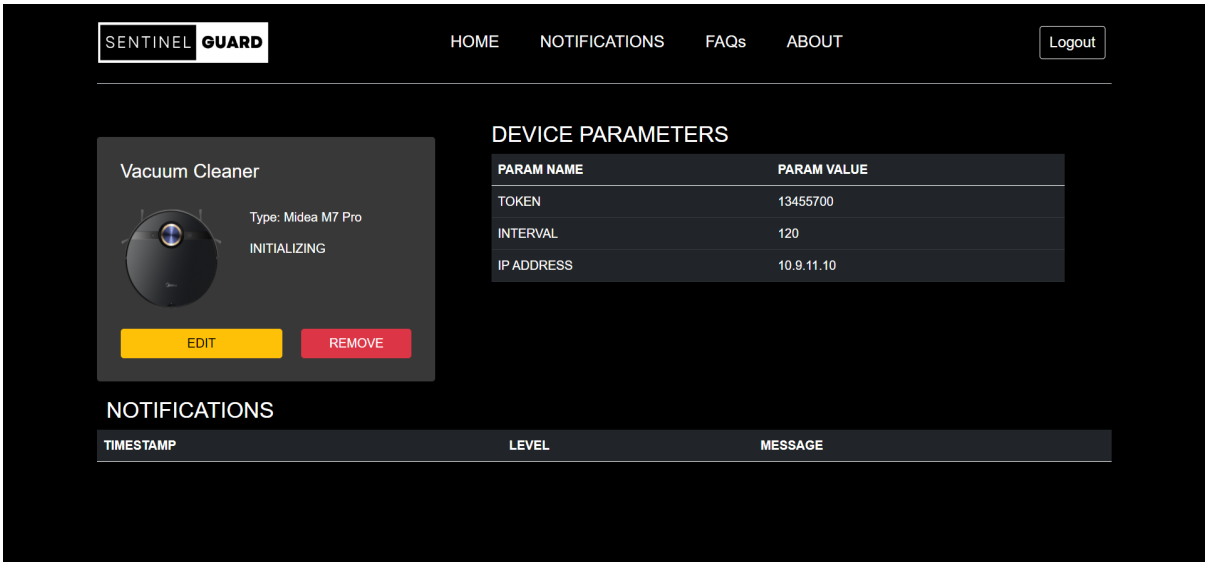


Рисунок 3.38 – Сторінка управління доданим пристроєм

Розглянемо випадок, коли користувач бажає приєднатися до вже існуючого домашнього середовища. Для цього необхідно натиснути Join Existing Home та ввести електронну адресу користувача, котрий вже має Home, до якого

він бажає приєднатися та натиснути Send Join Request. Сторінка під'єднання до існуючого домашнього середовища зображена на рисунку 3.39. Успішно надісланий Join Request можна переглянути на рисунку 3.40.

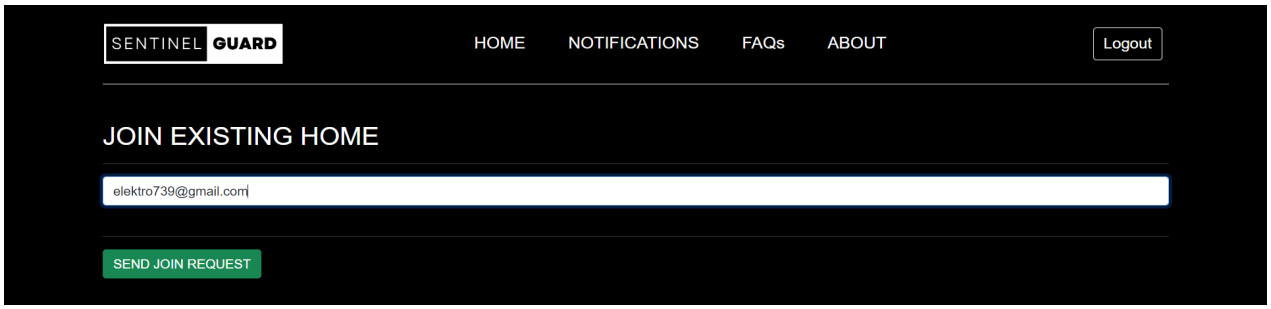


Рисунок 3.39 – Під'єднання до існуючого домашнього середовища.

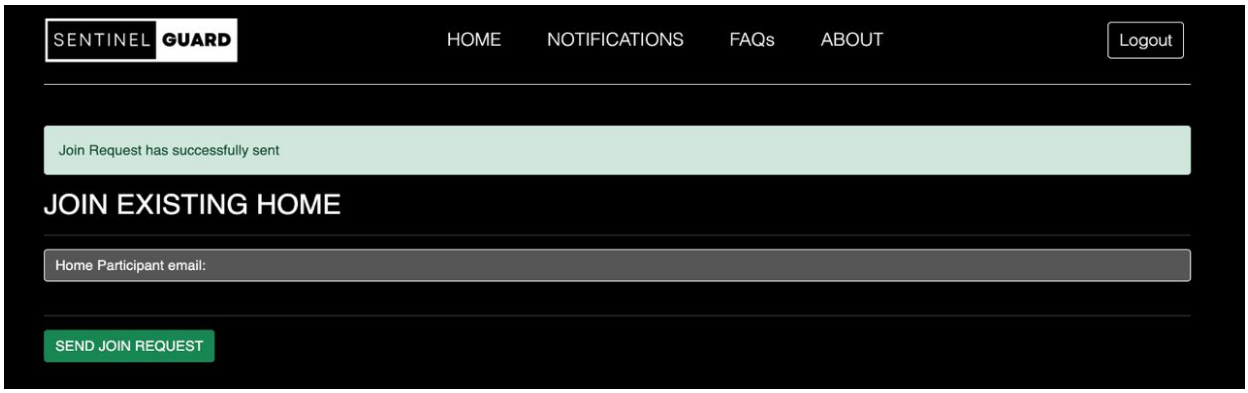


Рисунок 3.40 – Успішно надісланий Join Request.

Після відправки запиту на приєднання, користувачу чия електронна пошта введена в розділі Join Requests надійде сповіщення, що до його домівки бажають приєднатися. Користувач може прийняти запит або відхилити (рис.3.41).

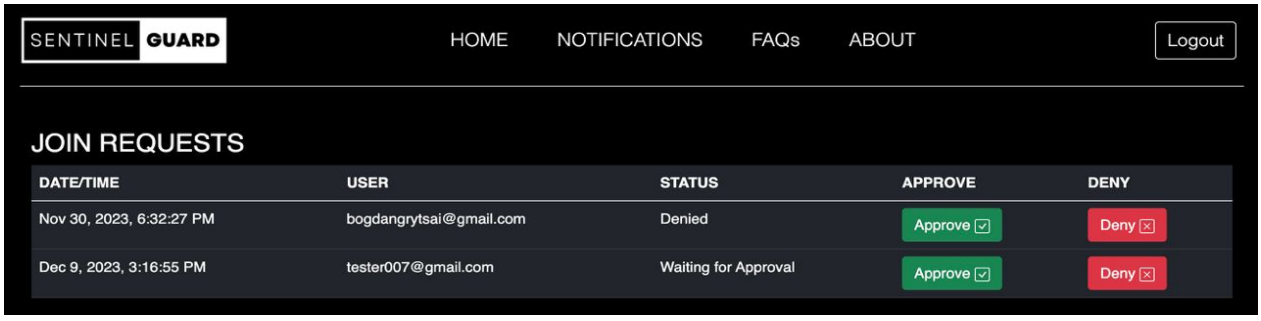


Рисунок 3.41 – Відображення Join Request

Якщо запит прийнято, то на сторінці користувача буде відображено адресу та пристрої домівки. Головну сторінку домівки після приєднання зображено на рисунку 3.42

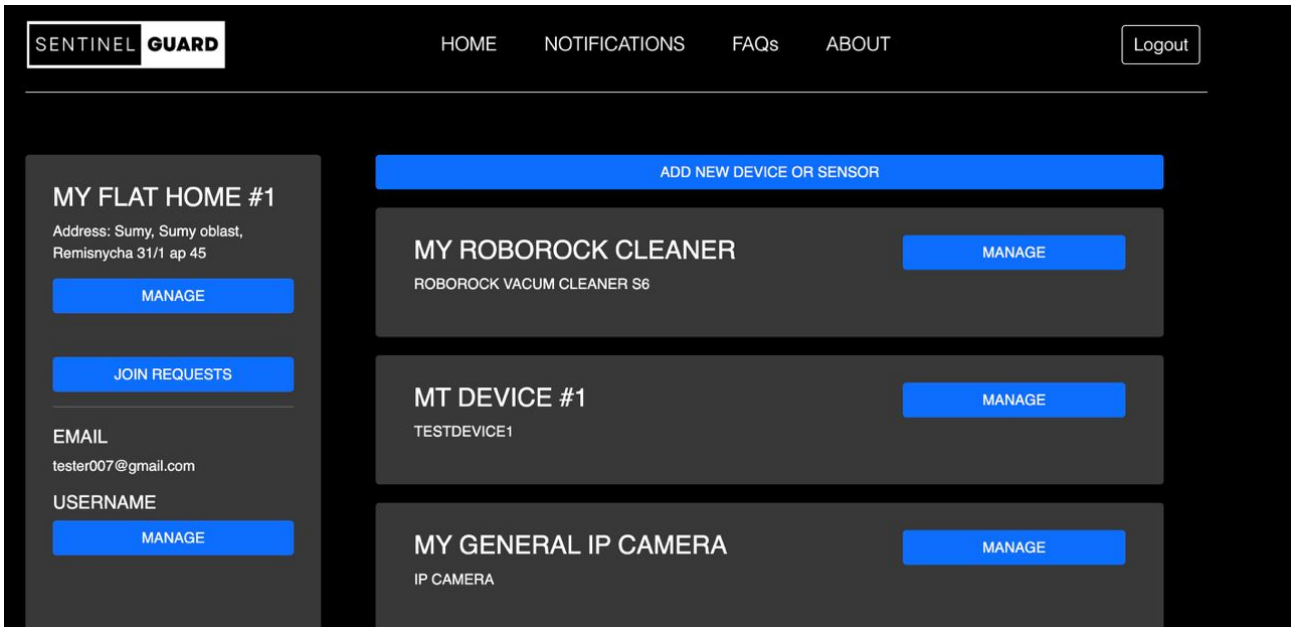


Рисунок 3.42 – Прийнятий Join Request

Сторінка Notifications (рис. 3.43) містить інформацію про всі повідомлення, які були надіслані користувачеві з підключених пристроїв.

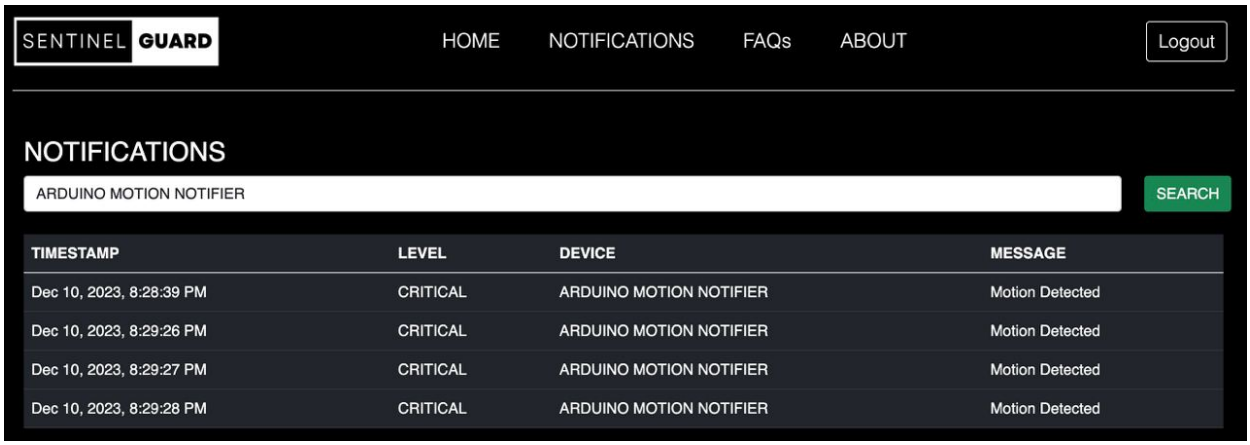


Рисунок 3.43 – Сторінка Notifications

Також переглянути повідомлення, котрі відносяться до конкретного пристрою можна на сторінці Device Parameters (рис.3.44).

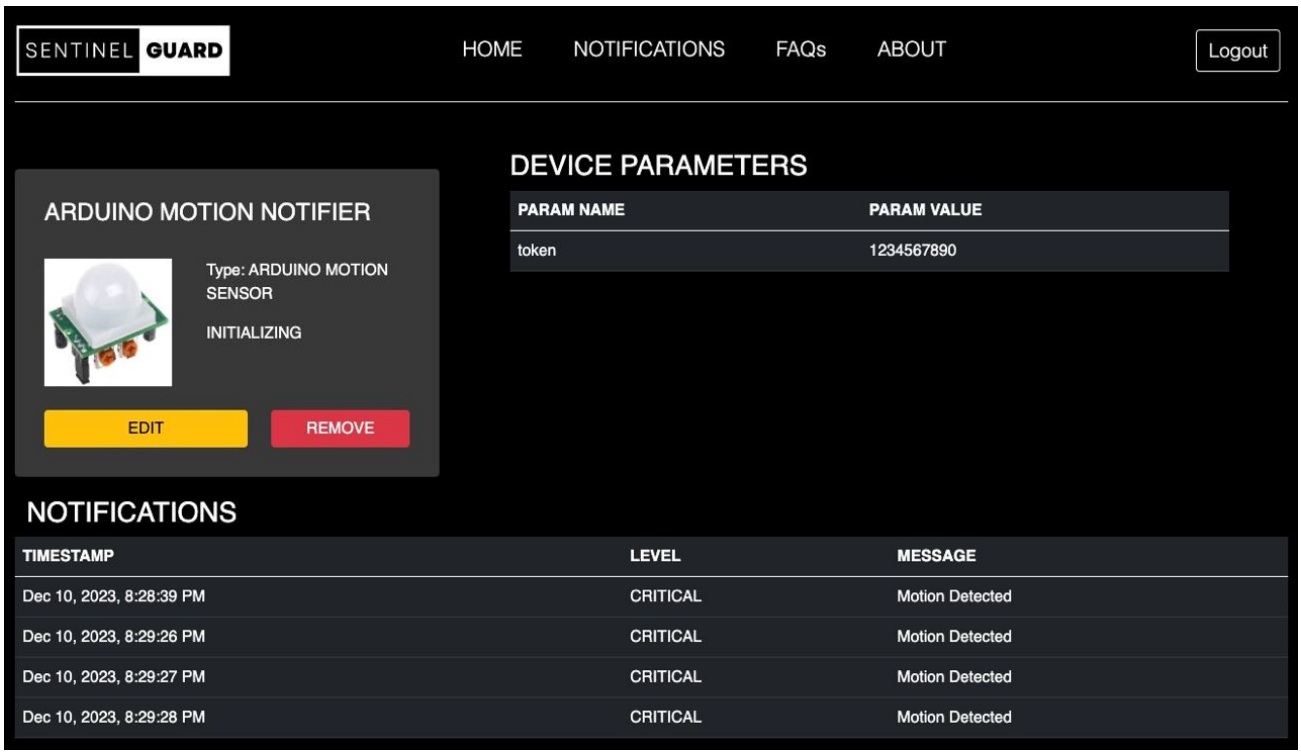


Рисунок 3.44 – Відображення повідомлень на сторінці Device Parameters

На цьому процес розробки можна вважати закінченим.

ВИСНОВКИ

Актуальність інформаційної технології розробки модуля системи безпеки та управління будинком визначається на тлі стрімкого розвитку сучасних технологій та зростання інтересу до "розумного" житла. З поглибленням зв'язку між людьми та технікою з'являється потреба в інтегрованих рішеннях для ефективного контролю за житловим простором.

У ході кваліфікаційної роботи магістра було проведено комплексний аналіз актуальності та необхідності впровадження інформаційної технології для розробки модуля системи безпеки та управління будинком. Літературний огляд включає аналіз останніх досліджень та публікацій у цій галузі, а також вивчення аналогів, що дозволило зазначити прогалини та переваги існуючих рішень.

Методика вирішення поставлених завдань включає в себе докладне вивчення методів досліджень, моделювання варіантів використання системи та проєктування бази даних. Програмна реалізація включає розробку структури інформаційної системи, реалізацію основних компонентів, інтеграцію з базою даних та реалізацію важливих функцій системи.

Отримані результати дозволяють визначити інформаційну технологію розробки модуля системи безпеки та управління будинком як ефективний та перспективний напрямок для поліпшення функціональності та забезпечення безпеки в сучасних будинках. Розроблена система є важливим кроком у напрямку автоматизації та оптимізації процесів управління будинком. Тому, використання системи дозволяє ефективно керувати різноманітними пристроями та забезпечувати високий рівень безпеки для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ian J.Lloyd. Information Technology Law. 2020, p.189
2. Malik NBodwade Y (2018). Literature Review on Home Automation System. Режим доступу до ресурсу:
3. Al-Mutaw, REassa F (2020). A smart home system based on internet of things. Режим доступу до ресурсу: https://thesai.org/Publications/ViewPaper?Volume=11&Issue=2&Code=IJACSA&SerialNo=34
4. Muhammad Asadullah, Ahsan Raza (2017). An overview of home automation systems. Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/7791223
5. Hamsagayatri P, Subash S, Susheel S (2021). Wireless Home Automation System. IOP Conference Series: Materials Science and Engineering (2021) 1084(1) 012075
6. Nan Wang, Aimin Pan and Liexiang Wei (2019). Design and Implementation of Intelligent Home Management System Based on Wireless Control Module A. Режим доступу до ресурсу: https://iopscience.iop.org/article/10.1088/1742-6596/1237/4/042059/pdf
7. Singh S, Verma R, Verma P (2019). Future Home – A Review of Future House or Home with Security and Voice Controller (2019). International Journal of New Technology and Research (2019).
8. Sathesh, Hamdan Y (2021). Smart Home Environment Future Challenges and Issues - A Survey. Режим доступу до ресурсу: https://irojournals.com/iroei/article/view/3/1/1.
9. Batalla J, Gonciarz F (2019). Deployment of smart home management system at the edge: mechanisms and protocols. Neural Computing and Applications (2019) 31(5) 1301-1315
10. Kuo-Lan Su, Song-Hiang Chia, Sheng-Ven Shiau, Jr-Hung Guo. Developing a module-based security system for an intelligent home. Режим

доступу до ресурсу:https://iopscience.iop.org/article/10.1088/1742-6596/1237/4/042059/pdf.

11. Parial P (2022). Home Automation System Using Based on IoT. International Journal for Research in Applied Science and Engineering Technology (2022) 10(7) 80-94.

12. Santoso Budijono, Jeffri Andrianto, Muhammad Axis Novradin Noor (2016). Design and implementation of modular home security system withshort messaging system. Режим доступу до ресурсу: https://d1wqtxts1xzle7.cloudfront.net/55704731/epjconf_icas2013_00025-libre.pdf

13. Jennifer Pattison Tuohy (2022). How Does a Home Security System Work. Режим доступу до ресурсу: https://www.usnews.com/360-reviews/services/home-security/how-does-a-home-security-system-work

14. Ring system. Електронний ресурс: https://ring.com/.

15. SimpliSafe system. Електронний ресурс: https://simplisafe.com/.

16. Argo system. Електронний ресурс: https://www.arlo.com.

17. Adam Hayes (2023). Smart Home: Definition, How They Work, Pros and Cons. Режим доступу до ресурсу: https://www.investopedia.com/terms/s/smart-home.asp#:~:text=Key%20Takeaways,with%20convenience%20and%20cost%20savings.

18. Nithin Thypparambil (2022). Home automation advantages and disadvantages. Режим доступу до ресурсу: https://timesofindia.indiatimes.com/readersblog/nithinsunilthypparampil/home-automation-advantages-and-disadvantages-51585/

19. Rahul AwatiIvy, Wigmor (2022). What is monolithic architecture in software? Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/monolithic-architecture

20. Mehmet Ozkaya (2023). Benefits and Challenges of Monolithic Architecture. Режим доступу до ресурсу: https://medium.com/design-

microservices-architecture-with-patterns/benefits-and-challenges-of-monolithic-architecture-d08906b38354

21. James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley. The Java™ Language, Specification Java SE 7 Edition. с. 128-135.

22. Craig Walls, Spring in Action 5, Fifth edition : Manning Publications Co, 2019 – 498 с.

23. Charles Forsythe (2014). Instant FreeMarker Starter. с 55-60.

24. John C.Worsley, Joshua D.Drake. Practical PostgreSQL. с. 20-35.

25. Adam L. Davis. Spring . Chapter: First Online: 2020. с. 57-59

26. Mule S, Waykar Y (2015). Role of use case diagram in software development. International Journal of Management and Economics (2015).

27. Nishadha (2022). Use Case Diagram Tutorial (Guide with Examples). Режим доступу до ресурсу: https://creately.com/guides/use-case-diagram-tutorial/.

28. Constantine Nalimov (2021). The logical data model explained. Режим доступу до ресурсу: https://www.gleek.io/blog/logical-data-model

29. Sandra Suszterova (2023). Physical Data Model vs. Logical Data Model. Режим доступу до ресурсу: https://www.gooddata.com/blog/physical-vs-logical-data-model/

30. Amani Undru (2023). Designing the structure of the information system. Режим доступу до ресурсу: https://www.crio.do/blog/a-comprehensive-guide-to-system-design/#:~:text=At%20its%20essence%2C%20system%20design,functional%20and%20non%2Dfunctional%20requirements.

ДОДАТОК А

Вміст файлу класу (сутності) Home.java:

```
package com.sentinel.web.entities;

import javax.persistence.*;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import org.hibernate.annotations.Fetch;

import org.hibernate.annotations.FetchMode;

import java.util.List;

@Getter

@Setter

@NoArgsConstructor

@Entity

@Table(name = "homes")

public class Home {

  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)

  @Column(name = "id", nullable = false, unique = true)

  private Long id;

  @Column(name = "name", nullable = false)

  private String name;

  @Column(name = "address")

  private String address;
```

```
    @OneToMany(mappedBy = "home", cascade = CascadeType.ALL,
orphanRemoval = true)
    @Fetch(FetchMode.JOIN)
    private List<Device> devices;
}
```

Вміст файлу класу (сутності) Device.java:

```
package com.sentinel.web.entities;
import javax.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import java.util.List;
@Getter
@Setter
@NoArgsConstructor
@Entity
@Table(name = "devices")
public class Device {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false, unique = true)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "home_id", nullable = false)
```

```
    private Home home;

    @ManyToOne

    @JoinColumn(name = "type_id", nullable = false)

    private DeviceType type;

    @Column(name = "name", nullable = false)

    private String name;

    @Column(name = "status", nullable = false)

    private String status;

    @OneToMany(mappedBy = "device", cascade =
CascadeType.ALL, orphanRemoval = true)

    private List<DeviceParam> params;

}
```

Вміст файлу класу (сутності) DeviceType.java:

```
package com.sentinel.web.entities;

import javax.persistence.*;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import java.util.List;

@Getter

@Setter

@NoArgsConstructor

@Entity

@Table(name = "device_types")
```

```
public class DeviceType {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id", nullable = false, unique = true)
  private Long id;
  @Column(name = "name", nullable = false)
  private String name;
  @Column(name = "imageUrl")
  private String imageUrl;
  @ManyToMany
  @JoinTable(
      name = "device_type_attributes",
      joinColumns = @JoinColumn(name = "device_type_id"),
      inverseJoinColumns = @JoinColumn(name =
"device_attr_id"))
  private List<DeviceAttribute> deviceAttributes;
}
```

Вміст файлу класу (сутності) DeviceAttribute.java:

```
package com.sentinel.web.entities;
import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
```

```
import java.util.List;

@Entity

@Table(name = "device_attributes")

@Getter

@Setter

@NoArgsConstructor

public class DeviceAttribute {

  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)

  @Column(name = "id")

  private Long id;

  @Column(name = "name", nullable = false)

  private String name;

  @ManyToMany(mappedBy = "deviceAttributes")

  @JsonIgnore

  private List<DeviceType> deviceTypes;

  public DeviceAttribute(String name) {

    this.name = name;

  }

}
```

Вміст файлу класу (сутності) DeviceParam.java:

```
package com.sentinel.web.entities;

import javax.persistence.*;

import lombok.Getter;
```

```
import lombok.NoArgsConstructor;
import lombok.Setter;
@Entity
@Table(name = "device_params")
@Getter
@Setter
@NoArgsConstructor
public class DeviceParam {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id")
  private Long id;
  @ManyToOne
  @JoinColumn(name = "device_attribute_id", nullable =
false)
  private DeviceAttribute deviceAttribute;
  @ManyToOne
  @JoinColumn(name = "device_id", nullable = false)
  private Device device;
  @Column(name = "value", nullable = false)
  private String value;
}
```

Вміст файлу класу (сутності) JoinRequest.java:

```
package com.sentinel.web.entities;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import javax.persistence.*;

import java.sql.Timestamp;

@Entity

@Table(name = "join_requests")

@Getter

@Setter

@NoArgsConstructor

public class JoinRequest {

  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)

  @Column(name = "id")

  private Long id;

  @ManyToOne

  @JoinColumn(name = "home_id", nullable = false)

  private Home home;

  @ManyToOne

  @JoinColumn(name = "status", nullable = false)

  private JoinRequestStatus status;
```

```
  @ManyToOne
  @JoinColumn(name = "user_id", nullable = false)
  private User user;
  @Column(name = "timestamp", nullable = false)
  private Timestamp timestamp;
}
```

Вміст файлу класу (сутності) JoinRequestStatus.java:

```
package com.sentinel.web.entities;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name = "join_request_status")
@Getter
@Setter
@NoArgsConstructor
public class JoinRequestStatus {
  @Id
  @Column(name = "status", unique = true, nullable = false)
  private String status;
```

```
  @Column(name = "description", nullable = false)
  private String description;
}
```

Вміст файлу класу (сутності) Notification.java:

```
package com.sentinel.web.entities;
import javax.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import java.sql.Timestamp;
@Entity
@Table(name = "notifications")
@Getter
@Setter
@NoArgsConstructor
public class Notification {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id")
  private Long id;
  @ManyToOne
  @JoinColumn(name = "level", nullable = false)
  private NotificationLevel level;
```

```
  @Column(name = "message", nullable = false)
  private String message;
  @ManyToOne
  @JoinColumn(name = "device_id", nullable = false)
  private Device device;
  @Column(name = "timestamp", nullable = false)
  private Timestamp timestamp;
}
```

Вміст файлу класу (сутності) NotificationLevel.java:

```
package com.sentinel.web.entities;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
@Entity
@Table(name = "notification_levels")
@Getter
@Setter
@NoArgsConstructor
public class NotificationLevel {
  @Id
  @Column(name = "level", unique = true, nullable = false)
  private String level;
```

```
    @Column(name = "description", nullable = false)
    private String description;
}
```

Вміст файлу класу контролеру AdminController.java:

```
package com.sentinel.web.controller;
import com.sentinel.web.entities.*;
import com.sentinel.web.repositories.*;
import
org.springframework.security.core.annotation.AuthenticationPrinc
ipal;
import org.springframework.ui.Model;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import
org.springframework.web.servlet.mvc.support.RedirectAttributes;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
@Controller
@RequestMapping("/home")
```

```
public class HomeController {

private DeviceRepository deviceRepository;

private HomeRepository homeRepository;

private DeviceTypeRepository deviceTypeRepository;

private DeviceParamRepository deviceParamRepository;

private UserRepository userRepository;

private NotificationRepository notificationRepository;

private JoinRequestsRepository joinRequestsRepository;

private JoinRequestStatusRepository
joinRequestStatusRepository;

@Autowired

public HomeController(DeviceRepository deviceRepository,

 HomeRepository homeRepository,

 DeviceTypeRepository deviceTypeRepository,

 DeviceParamRepository deviceParamRepository,

 UserRepository userRepository,

 NotificationRepository notificationRepository,

 JoinRequestsRepository joinRequestsRepository,

 JoinRequestStatusRepository joinRequestStatusRepository

) {

this.deviceRepository = deviceRepository;

this.homeRepository = homeRepository;

this.deviceTypeRepository = deviceTypeRepository;

this.deviceParamRepository = deviceParamRepository;
```

```
        this.userRepository = userRepository;
        this.notificationRepository = notificationRepository;
        this.joinRequestsRepository = joinRequestsRepository;
        this.joinRequestStatusRepository =
joinRequestStatusRepository;
        }
        @GetMapping
        public String home(@AuthenticationPrincipal User user,
          Model model) {
        Home home = homeRepository.findHomeByUserId(user.getId());
        model.addAttribute("user", user);
        model.addAttribute("home", home);
        return "home";
        }
        @GetMapping("/createHome")
        public String createHome(@AuthenticationPrincipal User
user) {
        return "createHome";
        }
        @PostMapping("createHome")
        public String createHome(@AuthenticationPrincipal User
user,
         @RequestParam String homeName,
         @RequestParam String address,
         RedirectAttributes redirectAttributes) {
```

```
        if (user.getHome() == null) {
        Home home = new Home();
        home.setName(homeName);
        home.setAddress(address);
        homeRepository.save(home);
        user.setHome(home);
        userRepository.save(user);
        redirectAttributes.addFlashAttribute("alertSuccess",
        "Home " + homeName + " created successfully");
        } else {
        redirectAttributes.addFlashAttribute("alertError",
        "You already assigned to home " +
user.getHome().getName());
        }
        return "redirect:/home";
        }
        @GetMapping("/editHome")
        public String editHome(@AuthenticationPrincipal User user,
          @RequestParam Long homeId,
          Model model) {
        model.addAttribute("home",
homeRepository.findById(homeId).orElse(null));

        return "editHome";
        }
```

```
    @PostMapping("/editHome")
    public String editHome(@AuthenticationPrincipal User user,
      @RequestParam Long homeId,
      @RequestParam String homeName,
      @RequestParam String address,
      RedirectAttributes redirectAttributes,
      Model model) {
    final String[] returnStr = new String[1];
    homeRepository.findById(homeId).ifPresentOrElse(
    home -> {
    home.setName(homeName);
    home.setAddress(address);
    homeRepository.save(home);
    redirectAttributes.addFlashAttribute("alertSuccess", "Home
is updated!");
    returnStr[0] = "redirect:/home";
    },
    () -> {
    model.addAttribute("alertError", "Home is not found!");
    returnStr[0] = "/home/editHome?homeId=" + homeId;
    });
    return returnStr[0];
    }
    @GetMapping("/addDevice")
```

```
    public String addDevice(@AuthenticationPrincipal User user,
    Model model) {
    List<DeviceType> deviceTypes =
deviceTypeRepository.findAll();
    model.addAttribute("deviceTypes", deviceTypes);
    model.addAttribute("home", user.getHome());
    return "addDevice";
    }

    @PostMapping("/addDevice")
    public String addDevice(@AuthenticationPrincipal User user,
    @RequestParam("deviceName") String deviceName,
    @RequestParam("deviceType") Long deviceTypeId,
    @RequestParam Map<String, String> params) {
    Map<String, String> filteredParams = params.entrySet()
    .stream()
    .filter(entry -> !(entry.getKey().equals("deviceName")
    || entry.getKey().equals("deviceType")))
    .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue));
    List<DeviceParam> paramValues = new ArrayList<>();
    DeviceType deviceType =
deviceTypeRepository.findById(deviceTypeId).orElse(null);
    Device device = new Device();
    device.setName(deviceName);
```

```
        device.setType(deviceType);
        device.setHome(user.getHome());
        device.setStatus("INITIALIZING");
        deviceRepository.save(device);
        List<DeviceAttribute> attributes =
deviceType.getDeviceAttributes();
        attributes.forEach(attr -> {
        filteredParams.forEach((key, value) -> {
        if (attr.getName().equals(key)) {
        DeviceParam param = new DeviceParam();
        param.setDevice(device);
        param.setDeviceAttribute(attr);
        param.setValue(value);
        deviceParamRepository.save(param);
        paramValues.add(param);
        }
        });
        });
        return "redirect:/home";
        }
        @GetMapping("/manageHomeDevice")
        public String manageHomeDevice(@AuthenticationPrincipal
User user,
          @RequestParam Long deviceId,
          Model model) {
```

```
     Device device =
deviceRepository.findById(deviceId).orElse(null);

     List<Notification> notifications =
notificationRepository.findByDeviceId(deviceId).orElse(null);

     model.addAttribute("device", device);

     model.addAttribute("notifications", notifications);

     return "manageHomeDevice";

     }

     @GetMapping("/device/edit")

     public String editHomeDevice(@AuthenticationPrincipal User
user,

      @RequestParam Long deviceId,

      Model model) {

     Device device =
deviceRepository.findById(deviceId).orElse(null);

     model.addAttribute("device", device);

     return "editDevice";

     }

     @PostMapping("/device/edit")

     public String editHomeDevice(@AuthenticationPrincipal User
user,

      @RequestParam Long deviceId,

      @RequestParam String deviceName,

      @RequestParam Map<String, String> params) {

     Device device =
deviceRepository.findById(deviceId).orElse(null);
```

```
        device.setName(deviceName);
        deviceRepository.save(device);
        Map<String, String> filteredParams = params.entrySet()
        .stream()
        .filter(entry -> !(entry.getKey().equals("deviceName")
        || entry.getKey().equals("deviceId")))
        .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue));
        List<DeviceParam> paramValues = device.getParams();
        paramValues.forEach(param -> {
        filteredParams.forEach((key, value) -> {
        if (param.getDeviceAttribute().getName().equals(key)) {
        param.setValue(value);
        deviceParamRepository.save(param);
        }
        });
        });
        return "redirect:/home";
        }
        @PostMapping ("/device/remove")
        public String removeDeviceFromHome(@AuthenticationPrincipal
User user,
          @RequestParam Long deviceId) {
        deviceRepository.deleteById(deviceId);
```

```
    return "redirect:/home";
    }
    @GetMapping("/notifications")
    public String allNotifications(@AuthenticationPrincipal
User user,
      @RequestParam(required = false) Long deviceId,
      Model model) {
    Home home = user.getHome();
    if (home != null) {
    List<Notification> notifications;
    if (deviceId != null) {
    notifications =
notificationRepository.findByDeviceId(deviceId).orElse(null);
    } else {
    notifications =
notificationRepository.findByDeviceHomeId(home.getId()).orElse(n
ull);
    }
    model.addAttribute("devices", home.getDevices());
    model.addAttribute("notifications", notifications);
    }
    return "notifications";
    }
    @GetMapping("/askToJoinHome")
    public String askToJoinHome(@AuthenticationPrincipal User
user,
```

```
    RedirectAttributes redirectAttributes) {
    if (user.getHome() != null) {
    redirectAttributes.addFlashAttribute("alertError", "You are
already assigned to some home");
    return "redirect:/home";
    }
    return "home/askToJoinHome";
    }
    @PostMapping("/askToJoinHome")
    public String askToJoinHome(@AuthenticationPrincipal User
user,
    @RequestParam String homeParticipantEmail,
    RedirectAttributes redirectAttributes) {
    homeRepository.findHomeByUserEmail(homeParticipantEmail).if
PresentOrElse(
      home -> {
      JoinRequest joinRequest = new JoinRequest();
      joinRequest.setUser(user);
      joinRequest.setHome(home);
    joinRequest.setTimestamp(Timestamp.valueOf(LocalDateTime.no
w()));
      joinRequest.setStatus(
     joinRequestStatusRepository.findByStatus("W").orElse(null)
      );
      joinRequestsRepository.save(joinRequest);
```

```
        redirectAttributes.addFlashAttribute("alertSuccess",
"Join Request has successfully sent");

        },

        () -> {

        redirectAttributes.addFlashAttribute("alertError", "Home
is not found for this user email");

        }

        );

      return "redirect:/home/askToJoinHome";

      }

      @GetMapping("/joinRequests")

      public String joinRequests(@AuthenticationPrincipal User
user,

        Model model) {

      if (user.getHome() != null) {

      List<JoinRequest> joinRequests =
joinRequestsRepository.findJoinRequestByStatusStatusAndHomeId("W
", user.getHome().getId()).orElse(null);

      model.addAttribute("joinRequests", joinRequests);

      return "home/joinRequests";

      }

      return "redirect:/home";

      }

      @PostMapping("/manageJoinRequest")

      public String manageJoinRequest(@AuthenticationPrincipal
User user,
```

```
    @RequestParam Long joinRequestId,
    @RequestParam Boolean isApproved) {
    JoinRequest joinRequest =
joinRequestsRepository.findById(joinRequestId).orElse(null);
    if (isApproved) {
    joinRequest.setStatus(
    joinRequestStatusRepository.findByStatus("A").orElse(null)
    );
    User userForHome = joinRequest.getUser();
    userForHome.setHome(joinRequest.getHome());
    joinRequestsRepository.save(joinRequest);
    userRepository.save(userForHome);
    } else {
    joinRequest.setStatus(
    joinRequestStatusRepository.findByStatus("D").orElse(null)
    );
    joinRequestsRepository.save(joinRequest);
    }
    return "home/joinRequests";
    }
    @GetMapping("/manageDevicePage")
    public String manageDevicePage(@RequestParam Long deviceId)
{
    Device device =
deviceRepository.findById(deviceId).orElse(null);
```

```
List<DeviceParam> deviceParams = device.getParams();
for (DeviceParam param : deviceParams) {
if (param.getDeviceAttribute().getName().contains("URL")) {
return "redirect:" + param.getValue();
}
}
return "manageDevicePage";
}
}
```

Код сторінки home.ftl:

```
<#import "parts/page.ftl" as c>
<#include "parts/security.ftl">
<@c.page>
<style>
.gray-block {
background-color: #383838;
padding: 30px;
border-radius: 4px;
margin-top: 20px;
margin-bottom: 20px;
}
</style>
<div class="container-fluid">
<#if alertError??>
```

```
    <div class="alert alert-danger"
role="alert">${alertError}</div>

    </#if>

    <#if alertSuccess??>

    <div class="alert alert-success"
role="alert">${alertSuccess}</div>

    </#if>

    <div class="row">

    <div class="col-md-3 gray-block" style="margin-right:
60px">

    <#if home??>

    <h3 class="text-uppercase">${home.name}</h3>

    <p>Address: ${home.address}</p>

    <form method="get" action="/home/editHome">

    <input type="text" name="homeId" value="${home.id}" hidden>

    <button type="submit" class="btn btn-primary btn-block w-
100">MANAGE</button>

    </form>

    <br><br>

    <a href="/home/joinRequests" class="btn btn-primary btn-
block w-100">JOIN REQUESTS</a>

    <#else>

    <a href="/home/createHome" class="btn btn-primary btn-block
w-100">CREATE NEW HOME</a>

    <br>

    <br>
```

```
    <a href="/home/askToJoinHome" class="btn btn-primary btn-
block w-100">JOIN EXISTING HOME</a>

    </#if>

    <hr>

    <div style="margin-top: 20px">

    <div>

    <h5>EMAIL</h5>

    <p>${user.email}</p>

    </div>

    <div>

    <h5>USERNAME</h5>

    <#if user.username??>

    <p>${user.username}</p>

    </#if>

    </div>

    <form method="get" action="/user/editUser">

    <input type="text" name="userId" value="${user.id}" hidden>

    <button type="submit" class="btn btn-primary btn-block w-
100">MANAGE</button>

    </form></div></div>

    <div class="col-md-8" >

    <#if home??>

    <div class="row" style="margin-top: 20px; margin-bottom:
20px">
```

```
    <a href="/home/addDevice" class="btn btn-primary btn-block
w-100">ADD NEW DEVICE OR SENSOR</a></div>

    <#list home.devices as device>

    <div class="row gray-block">

    <div class="col-md-8">

    <h3 class="text-uppercase">${device.name}</h3>

    <p style="font-size: 1em">${device.type.name}</p></div>

    <div class="col-md-4">

    <form action="/home/manageHomeDevice" method="get">

    <input name="deviceId" type="number" value="${device.id}"
hidden>

    <button class="btn btn-primary w-100 btn-
block">MANAGE</button>

    </form></div></div>

    <#else>

    No Devices

    </#list>

    </#if>

    </div>

    </div>

    </div>

    </@c.page>
```