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

Кафедра обчислювальної техніки

«На правах рукопису»
УДК ___

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
«___»_____2020 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Інженерія програмного забезепечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система "Розумний дім" на базі сучасних веб-технологій»

Виконав:
студент VI курсу, групи ІП-92мп
Горова Марія Анатоліївна _____

Керівник:
доц., к.т.н., доц.,
Русанова Ольга Веніамінівна _____

Консультант з нормоконтролю:
професор, д.т.н., професор,
Жабін Валерій Іванович _____

Рецензент:
доц., к.т.н., доц.,
Орлова Марія Миколаївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Спеціалізація *121. Інженерія програмного забезпечення комп'ютерних систем*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
С.Г. Стіренко
(підпис) (ініціали, прізвище)
« » 20***20*** р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Горовій Марії Анатоліївні
(прізвище, ім'я, по батькові)

1. Тема дисертації Система «Розумний дім» на базі сучасних веб-технологій

Науковий керівник дисертації Русанова О. В., к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «26» жовтня *2020* р. № 3133

2. Строк подання студентом дисертації 25 листопада 2020 р.

3. Об'єкт дослідження Архітектура системи управління розумним будинком

4. Предмет дослідження Архітектурне рішення системи домашньої автоматизації на основі використання екосистеми Espruino

5. Перелік завдань, які потрібно розробити: проаналізувати принцип роботи мікроконтроллерів на ядрі Espruino, проаналізувати способи інтеграції Espruino із NodeJS, MongoDB, розробити КСУ розумним домом, провести UI/UX дослідження та визначити шляхи вдосконалення користувацького інтерфейсу, розробити прототип клієнт-серверного додатку, розрахувати орієнтовну вартість розробки архітектурного рішення на основі Espruino

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормконтроль	Жабін В. І., д.т.н., професор		

7. Дата видачі завдання 18.12.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми роботи	18.12.2019	
2.	Вивчення та аналіз завдання	20.05.2020 – 02.06.2020	
3.	Пошук матеріалів по темі роботи	03.06.2020 – 20.06.2020	
4.	Розробка програмного продукту	3.08.2020 – 31.08.2020	
5.	Оформлення пояснювальної записки	1.09.2020 – 20.11.2020	
6.	Передзахист	25.11.2020	
7.	Захист	17.12.2020	

Студент _____ М. А. Горова
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ О. В. Русанова
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Система «Розумний дім» на базі сучасних веб-технологій

студенткою: Горовою Марією Анатоліївною

Пояснювальна записка складається із вступу та чотирьох розділів. Загальний обсяг роботи: 101 аркуш основного тексту, 18 ілюстрацій, 35 таблиць. При підготовці використовувалася література з 40 різних джерел.

Актуальність. Технології інтернету речей є найбільш пріоритетною тенденцією інноваційного технологічного розвитку у світі. Технологічна консалтингова компанія Gartner, яка робить прогнози щодо майбутнього нових технологічних тенденцій, щороку оголошує очікування цих тенденцій за допомогою графіку під назвою Hype Cycle (Цикл ажіотажу). За результатами 2018 року, технологія IoT знаходилась близько піку кривої очікувань, а плато було прогнозоване через 5-10 років. В 2019 році ажіотаж на технологію спав, оскільки компанії почали забезпечувати попит і для ринку США та західної Європи інтернет речей та системи розумного дому стали більш звичними. Змінилось розуміння комфорту та навіть спосіб життя.

В Україні, де елементи системи «розумний дім» тільки набирають популярності, IoT є одним з найперспективніших напрямків розвитку інформаційних та комунікаційних технологій.

Мета і завдання дослідження. Мета роботи – зниження вартості системи «розумний дім» та підвищення зручності користувацького інтерфейсу.

Для досягнення мети необхідно вирішити наступні задачі:

- Проаналізувати принцип роботи мікроконтроллерів на ядрі Espruino;
- Проаналізувати способи інтеграції Espruino із NodeJS, MongoDB;
- Розробити КСУ розумним домом;

- Провести UI/UX дослідження та визначити шляхи вдосконалення користувацького інтерфейсу;
- Розробити прототип клієнт-серверного додатку;
- Розрахувати орієнтовну вартість розробки архітектурного рішення на основі Espruino.

Об'єкт дослідження – архітектура системи управління розумним будинком.

Предмет дослідження – архітектурне рішення системи домашньої автоматизації на основі використання екосистеми Espruino.

Методи досліджень. Для досягнення поставлених в магістерській роботі задач, використано методи імітаційного моделювання, та юзабіліті тестування (метод UX-дослідження).

Наукова новизна роботи у дослідженні та використанні для рішення існуючої проблеми перспективних нових технологій, які не є широковживаними.

Особистий внесок здобувача. У магістерському дослідженні відображено теоретичні та практичні результати щодо розробки архітектури системи управління розумним будинком на основі можливостей мови JavaScript, NodeJS та Espruino, отримані авторкою самостійно. Мета та завдання дослідження були сформульовані спільно з науковим керівником.

Практична цінність. Практична цінність роботи полягає у створенні прототипу простішого і більш дешевого архітектурного рішення в порівнянні з існуючими.

Публікації:

- «Розумний дім на Node JS». Збірник тез доповідей Міжнародної наукової інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (випуск 54) 10.12.2020.
- «Чому UX-дослідження варті уваги». Збірник тез доповідей Міжнародної наукової інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (випуск 54) 10.12.2020.

Ключові слова: IoT, інтернет речей, система «Розумний Дім».

ABSTRACT

on master's thesis

made on the topic: Smart Home System Based on Modern Web Technologies

by: Mariia Horova

The master thesis consists of introduction and four sections on 101 sheets of the main text, include 18 illustrations, 35 tables and 40 bibliographic titles.

Relevance of the topic. Internet of Things technology is the highest priority trend of innovative technological development in the world. Gartner technology consulting, which predicts the future of new technology trends, annually announces expectations of these trends using a chart called the Hype Cycle. According to the results of 2018, IoT technology was near the peak of the expectation curve, and the plateau was predicted in 5-10 years. In 2019, the excitement over technology subsided as companies began to supply demand and the Internet of Things and smart home systems became more commonplace in the US and Western European markets. The understanding of comfort and even the way of life has changed.

In Ukraine, where the elements of the Smart Home system are only gaining popularity, IoT is one of the most dynamic areas.

The purpose of research. The purpose of the research is to reduce the cost of the Smart Home system and to increase the usability of the user interface.

To achieve this goal it is necessary to solve the following tasks:

- Analyze the principle of Espruino core microcontrollers operation;
- Analyze options of Espruino, NodeJS and MongoDB integration;
- Develop a smart home management control system;
- Conduct UI / UX research and identify ways to improve the user experience;
- Develop prototype of client application;

- Calculate the estimated cost of developing an architectural solution based on Espruino.

Object of research – Smart Home Management System architecture.

Subject of research – home automation system architectural solution, based on the use of the Espruino ecosystem.

Methods of research. To achieve the objectives set in the master's thesis, simulation methods and usability testing (UX-research method) were used.

Scientific novelty of the research is the new technologies usage for the IoT problems resolving.

Personal contribution of the applicant. The research reflects the theoretical and practical results of the smart home management system architecture development with JavaScript, NodeJS and Espruino, obtained by the author herself. The purpose and objectives of the study were formulated in conjunction with the supervisor.

The practical value of the results obtained. Proposed Smart Home Management System architecture is simpler and cheaper compared to existing ones.

Publications:

- "Smart home on Nod JS". Proceedings of the International Scientific Internet Conference "Information Society: Technological, Economic and Technical Aspects of Formation" (issue 54) December 10, 2020.
- "Why UX research is worth noting". Proceedings of the International Scientific Internet Conference "Information Society: Technological, Economic and Technical Aspects of Formation" (issue 54) December 10, 2020.

Key Words: IoT, Internet of Things, Smart Home.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської дисертації

на тему: Система «Розумний дім» на базі сучасних веб-технологій

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CHIP – Connected Home over IP

DB – Data Base

DOM – Document Object Model

HTTPS – Hypertext Transfer Protocol Secure

IDE - Integrated Development Environment

IR – Infra Red

IoT – Internet of Things

IP – Internet Protocol

JS - JavaScript

JSON – JavaScript Object Notation

JWT – JSON Web Token

LAN – Local Area Network

MVC – Model View Controller

MVVM – Model View View Model

NFC – Near field communication

OSI – The Open Systems Interconnection model

RF – Radio Frequency Protocol

SSL – Secure Sockets Layer

SSJS – Server Side JavaScript

SOAP – Simple Object Access Protocol

TCP – Transmission Control Protocol

UI – User Interface

UML – Unified Modeling Language

USB – Universal Serial Bus

UX – User Experience

WSDL – Web Service Definition Language

КСУ – Контрольна система управління

БД – База Даних

ВСТУП

Сучасний світ неможливо уявити без новітніх цифрових технологій, які змінюють не тільки стиль життя людини, але і її уявлення про комфорт. Технології автоматизації розвиваються надшвидко заради створення орієнтованих на людину корисних пристроїв та цифрових послуг.

Наприклад, сайти на початку інтернет-історії створювалися лише для читання. Але вже web 2.0 надав можливість і написання текстів, як, наприклад, у соціальних мережах. Сьогодні маємо поєднання не тільки web-контентів, але й програм, різноманітних служб та data. Майбутнє – за розширенням цих можливостей, за smart web. Йдеться про web-доступ до даних, додатків, тощо. Постійне вдосконалення хмарної інфраструктури дає можливість розвитку централізованої платформи, яку ми називаємо «Інтернет речей».

За даними компанії CBSG, «Інтернет речей» (IoT) – це провідна або бездротова мережа пристроїв з автономним забезпеченням під керуванням інтелектуальних систем, що в свою чергу забезпечені операційною системою, мають автономне інтернет-з'єднання, збирають та аналізують дані [1].

На сьогодні дуже популярним вектором розвитку IoT стала система розумного будинку. Завдяки їй можна використовувати єдиний пристрій для управління різними пристроями в помешканні. Найбільш актуальними стають системи моніторингу навколишнього середовища, енергоефективності та створення максимального комфорту. Це забезпечується наявністю відкритих платформ, які завдяки інтелектуальній сенсорній мережі надають дані про стан помешкання. Доступ до переданих даних є дуже зручним завдяки сенсорним пристроям, PC та смартфонам.

Із розвитком IoT зростає і проблематика – труднощі об'єднання різних пристроїв у таку систему, щоб користувач міг взаємодіяти з будинком як з єдиним цілим (а не окремо з різними пристроями і системами).

Технології на кшталт Apple «Siri», «Alexa» Amazon або «Cortana» від Microsoft вирішують цю проблему за допомогою уніфікації управління розумним

будинком через голосові інтерфейси. Але це також ускладнює систему та підвищує її вартість.

Магістерська дисертація направлена на створення є альтернативного підходу до вирішення проблеми простого інтерфейсу та об'єднання різних пристроїв в одну систему.

Мета роботи – зниження вартості системи «розумний дім» та підвищення зручності користувацького інтерфейсу.

Об'єкт дослідження – архітектура системи.

Предмет дослідження – архітектурне рішення системи домашньої автоматизації на основі використання екосистеми Espruino.

Для досягнення мети необхідно вирішити наступні задачі:

- Проаналізувати принцип роботи мікроконтроллерів на ядрі Espruino та Espruino Web IDE;
- Проаналізувати способи інтеграції Espruino із NodeJS, MongoDB;
- Розробити КСУ розумним домом;
- Провести UI/UX дослідження та визначити шляхи вдосконалення користувацького інтерфейсу.
- Розробити прототип клієнт-серверного додатку;
- Розрахувати орієнтовну вартість розробки архітектурного рішення на основі Espruino.

Espruino – це движок мікроконтролерів, новітня альтернатива Arduino, яка дозволяє програмувати мікроконтролери мовою JavaScript та пропонує інтеграцію з NodeJS.

Espruino легко інтегруються NodeJS, а програмний код реалізується мовою JavaScript. Це дуже зручно, оскільки якщо піти трохи далі, то можна представити NodeJS як об'єднувальну ланку між трьома шарами IoT:

- програмуванням мікросхем Espruino мовою JavaScript,

- програмуванням серверної частини та БД (завдяки стабільній та простій інтеграції з базою даних MongoDB) мовою JavaScript на NodeJS,
- програмуванням клієнтського інтерфейсу (для чого природно використовується мова JavaScript та Node Package Manager);

Таким чином планується створити простий прототип із використанням кількох пристроїв, що об'єднуються в єдину систему та керуються через веб інтерфейс з будь якого пристрою, що підтримує доступ до браузеру (смартфон, смарт ТВ, планшет, ПК, тощо).

Можна прогнозувати, що використання цих сучасних веб технологій, які природньо інтегруються між собою, здатне:

- суттєво спростити архітектуру системи, зробити її такою, що може бути легко модифікована,
- зменшити початкову вартість системи за рахунок оптимізації виконання програмної частини,
- зменшити витрати на підтримку системи.

Наукова новизна роботи у дослідженні та використанні для рішення існуючої проблеми перспективних нових технологій, які не є широковживаними для рішення задач IoT, а її практична цінність – у створенні прототипу простішого і більш дешевого архітектурного рішення в порівнянні з існуючими.

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ СИСТЕМ УПРАВЛІННЯ «РОЗУМНИЙ ДІМ»

1.1. Визначення та застосування систем «Розумний дім»

Об’єкти можна ідентифікувати як об'єкти IoT, якщо вони мають деякі електронні пристрої: датчики, виконавчі пристрої та пристрої управління. Наприклад, датчики вимірюють кілька параметрів (температуру, вологість, місце знаходження об’єкту) та передають зібрані дані через мережу. Сама система працює наступним чином: до мікросхем IoT, на яких розміщуються датчики, додаються ланцюги передачі даних (найчастіше це LAN, Bluetooth та NFC). Персоналізацію передачі даних забезпечує TCP / Ip [1].

Основна ідея системи «Розумний дім» на базі IoT – в створенні системи синхронізованих пристроїв, яка має ряд ключових властивостей [2]:

- Комплексне управління інженерних систем будівлі.
- Функції контролю реалізують підсистеми управління, що керуються алгоритмом, в який закладений набір дій при зміні параметрів датчиків.
- Створення механізму екстренного відключення. Але для контролю за системою в людини також залишається зручний доступ до керування підсистемами.
- В разі відключення управляючої системи, забезпечується коректна робота необхідних підсистем.
- Вартість бслуговування мінімізується, а модернізація систем помешкання стандартизується та автоматизується.
- Можливість використання одразу кількох видів фізичних каналів: слабкострумові лінії, радіоканал, тощо.

Серед переваг систем «Розумний дім» варто окремо відзначити [3, 4]:

- Контроль доступу та функції безпеки;
- Відстежування та модерування ключових параметрів системи та комплексна блискавична реакція на критичну зміну їхніх показників;

- Віддалене управління будівлею.
- Мінімізація людського фактору в забезпеченні коректної життєдіяльності системи.

1.2. Проблеми об’єднання різних пристроїв в одну систему

Існує чимало дротових, гібридних та класичних бездротових протоколів зв'язку, завдяки яким пристрої системи розумного будинку обмінюються інформацією.

1.2.1. Огляд основних бездротових протоколів зв’язку

1. ZigBee. Надає підтримку мереж Mesh, шифрування, тощо. На відміну від Bluetooth та Wi-Fi, має підвищену швидкість взаємодії з датчиками і збільшений час роботи акумулятора. Серед недоліків неможливість підключення датчиків одного постачальника до шлюзу іншого [5].
2. Z-wave. Також використовує радіочастоти та Mesh-мережі. Але через ліцензування частот можна зіштовхнутися з конфліктами в об'єднанні пристроїв, призначених для різних країн. Саме через це протокол є дорожчим за ZigBee, але і більш надійним [6].
3. LoRa та LoRaWAN. Перевага цього протоколу у великій відстані передачі даних та енергоефективності. Частоти, на яких працює LoRa різні в різних країнах. Завдяки потужності передачі даних до 25 мВ можна встановлювати зв'язок на кілометри (від 1 км в завантажених умовах, до 15 км в незавантажених). LoRaWAN (Long Range Wide-Area Network) – більш високорівневий протокол, який ще більше розширює відстань передачі даних [6].
4. Bluetooth. Поширений, але чи не найнезручніший протокол для задач розумного будинку. Серед недоліків невелика дальність доступу до приладів, навіть в самому приміщенні, робота лише на частоті 2,4 ГГц, яка

зазвичай перевантажена іншими пристроями. Проте протокол Bluetooth LE є дешевим та надзвичайно енергоефективним [7].

5. IR. Стара технологія для пристроїв, які не мають альтернативних каналів управління (прилади з пультом дистанційного керування).
6. Протокол Wi-Fi. Найпоширеніший протокол в реалізації систем розумного будинку. Як правило, з'єднує гаджет із системою управління, іноді і для комунікації автономних пристроїв [8].

1.2.2. Огляд основних провідникових технологій зв'язку

1. RS485. Обмін інформацією базується на передачі різниці напруг на кінцях сигнальних проводів від комунікатора до приймача. До однієї лінії можливо під'єднати до 256 пристроїв за максимальної довжини кабелю 1200 метрів.
2. I²C. Протокол з двопровідним з'єднанням пристроїв, є простим способом взаємодії між платами без застосування великої кількості приймачів. Одна пара провідників дозволяє підключити до 128 пристроїв. Передача даних проходить на швидкості від 10 Кбіт/с при включенні в загальну шину пвільних приладів, до 100 Кбіт/с.
3. KNX. Гарантує стабільність функціонування системи, але складний в налаштуванні і дорогий.

Чи не головним недоліком використання провідникових протоколів є необхідність монтажу елементів системи ще на стадії проектування приміщення, а от змінити щось після закінчення робіт буде складно.

1.2.3. Порівняльний аналіз характеристик протоколів зв'язку

На сьогодні частота 2,4ГГц перевантажена, отже ZigBee часто стикається з проблемами доступу. Крім того сигнал на цій частоті згасає набагато швидше, ніж наприклад, на частотах менше 1 ГГц.

Не зважаючи на те, що ZigBee використовує чимало засобів захисту даних (128-бітовий алгоритм AES, три типи ключів для управління безпекою), іноді фіксуються проблеми з безпекою при підключенні нового пристрою [5].

Bluetooth також базується на діапазоні 2,4ГГц. Радіус дії є проблемною зоною цієї технології. Незважаючи на обіцяні в інструкціях «100 метрів в зоні прямої видимості», реальна відстань доступу в сучасних приміщеннях ледь досягає 10 метрів.

Узагальнюючи, підкреслимо що Bluetooth, Wi-Fi і ZigBee мають маленький радіус дії, а Wi-Fi надто енергозатратний. ZigBee та LoRa показують себе дуже енергоефективними у поєднанні. При цьому ZigBee найбільш ефективний на невеликих відстанях, а для більших дистанцій підходить LoRa. На сьогодні лідером є технологія Z-Wave, яка охоплює всі рівні моделі OSI і не використовує діапазон 2,4 ГГц.

Однією з основних проблем, які повинні вирішити постачальники рішень для розумного будинку, вважається підтримка різних типів протоколів. Подолання цієї проблеми має вирішальне значення для правильної інтеграції пристроїв [4].

Користувач не має стикатися з проблемами навіть у випадку, якщо в нього наприклад, IP-камера відеоспостереження, WiFi-телевізор, та Zigbee-освітлення. Наприкінці 2019 року стало відомо, що Amazon, Apple, Google та Zigbee Alliance планують до запуску спільну програму Connected Home over IP (CHIP) – єдиного стандарту для розумного будинку. Таким чином ці компанії планують вирішити проблему сумісності різних пристроїв. Розумні колонки Echo, HomePod та Home зможуть керувати не тільки пристроями Apple, а й пристроями інших виробників, які будуть дотримуватись протоколу.

У CHIP буде відкритий код і виробники не будуть платити роялті за його використання. Проте реалізація задумки наразі тільки у перспективі. У 2021 році планується випуск перших чернеток специфікацій з відкритим кодом, а остаточна реалізація CHIP – справа як найменше 5 років.

Таблиця 1.1.

Протоколи ближнього радіусу дії

Технічні характеристики	Wi-Fi	Bluetooth Low Energy	ZigBee	Z-Wave
Дальність	до 100 м	80 м	100 м/Mesh	30 м/Mesh
Частота	2.4 ГГц, 5 ГГц	2.4 ГГц	915 МГц, 2.4 ГГц	900 МГц
Швидкість передачі	Макс. 7 Гбіт/с	< 1 мбіт/с	250 кбіт/с	10-100 кбіт/с
Споживання енергії	Високе	Понижене	Низьке	Низьке
Аутентифікація	Так	Проблематично	Так	Так
Шифрування	Так	Так	Так	Так
Двонаправленість	Так	Так	Так	Так
Стандарт	IEEE 802.11	Bluetooth 4.0	ZigBee	Z-Wave
Маштабованість	Так	Так	Так	Обмежено

1.3. Аналіз сучасних систем управління «Розумний дім»

1. Mi Home/Aqara Home – недорогий доволі якісний і простий у використанні. Підтримує пртоколи Z-Wave, Zigbee, Wi-Fi, та Bluetooth. Пропонує користувачам для управління системою мобільний додаток. Недоліком є взаємодія з пристроями через китайський хмарний сервер, через що трапляються функціональні збої в системі [9].
2. Rubetek. Спеціалізується на пристроях для систем управління розумним будинком на базі бездротових протоколів RF 433 МГц без зворотного зв'язку, Z-Wave та Wi-Fi. Недорогий, тому широко поширений, але часто

трапляються проблеми з Wi-Fi мережею. Також датчики потребують заміни батарейок кожні 2-3 місяці [9].

3. Vera Plus. Преміальна система працює з протоколами Z-Wave Plus, Wi-Fi 802.11a/b/g/n/ac, Bluetooth 4.0 + BLE, ZigBee та багатьма сторонніми пристрями. Для управління системою пропонується складні та мало зрозумілі web-інтерфейс та мобільний додаток. Вагомим недоліком Vera – неможливість інтеграції в систему пристроїв без шифрування [10].
4. Fibaro Home Cente. Ще одна преміальна, що дозволяє підключення сторонніх пристроїв, проте підтримує тільки протокол Z-Wave. Це призводить до того, що пристрої, розроблені для одного регіону (наприклад, східної Європи) не можуть бути поєднані з пристроями для США, тож при підключенні стороннього пристрою часто виникає збій в роботі всієї системи. Також недоліками Fibaro є складність інтерфейсу, зависання системи та неможливість управління кількома користувачами [9].
5. Wirenboard. Це апаратний комплекс, який не є готовим рішенням, а лише пропонує контролер та модулі. Модулі працюють як з аналоговими, так і бездротовими пристроями для таких протоколів, як RF433, Z-Wave, Zigbee. Для використання Wirenboard користувач повинен мати серйозні технічні знання та навички, тому комплекс скоріше підходить як контролер для іншого готового рішення [11].

Порівняльну характеристику найпоширеніших систем управління розумним будинком наведено у таблиці 1.2.

Таблиця 1.2.

Порівняння конкурентних характеристик існуючих систем управління розумним будинком

Існуюча система	Підтримка усіх основних протоколів	Мультикори-стувацька система	Підтримка різних типів девайсів	Створення та виконання сценаріїв	Цінова категорія
Mi Home/Aqara Home	Так	Ні	Так	Суттєвий лаг у виконанні сценаріїв	Нижче середньої
Rubetek	Ні	Так	Так	Так	Нижче середньої
Vera Plus	Так	Ні	Так	Незрозумілий інтерфейс ускладнює створення сценаріїв	Висока
Fibaro Home Cente	Ні	Ні	Так	Складна та нестабільна система для створення сценаріїв	Висока
Wirenboard	Так	Так	Так	Незрозумілий інтерфейс ускладнює створення сценаріїв	Середня

Спільним моментом існуючих рішень є намагання утримати баланс між різноманітністю пристроїв, які можуть бути інтегровані в систему та вартістю системи. Підтримка різних типів протоколів зв'язку досить вартісна для виробників систем Розумний Дім і не всі системи здатні конкурувати в цьому.

Також більшість виробників приділяють недостатню увагу зручності управління системою для користувача, можливості підключити кількох користувачів до системи.

Оскільки кінцевий споживач зазвичай шукає просте і зручне рішення, однією із задач даної дисертації є дослідження потреб споживачів та підвищення зручності користувацького інтерфейсу, а також створення мультикористувацької системи.

1.4. Вибір напрямку та задач дослідження

Об'єктом управління буде квартира, зображена на рисунку 1.1. Після дослідження проблем об'єднання пристроїв IoT в єдину систему та огляду існуючих пропозицій на ринку, прийнято рішення дослідити можливість зниження вартості розробки систем «Розумний Дім» а також запропонувати варіанти підвищення зручності інтерфейсу.

Основними характеристиками системи будуть:

- Зручність використання;
- Можливість створення власних сценаріїв або модифікації існуючих;
- Можливість управління системою різними користувачами (додаток з модулем авторизації);
- Швидкість передавання розпізнаних команд до контролера;
- Безпека передачі даних між клієнтом та сервером (використання JSON Web Tokens).

Головними завдання контролера повинно бути:

- Управління освітленням;
- Клімат контроль;

- Управління розумними приладами, наприклад кавоваркою (здебільшого для цих девайсів використовується протокол Bluetooth);
- Спостереження за безпекою;
- Обмін інформацією про стан будинка із web-сервісом.

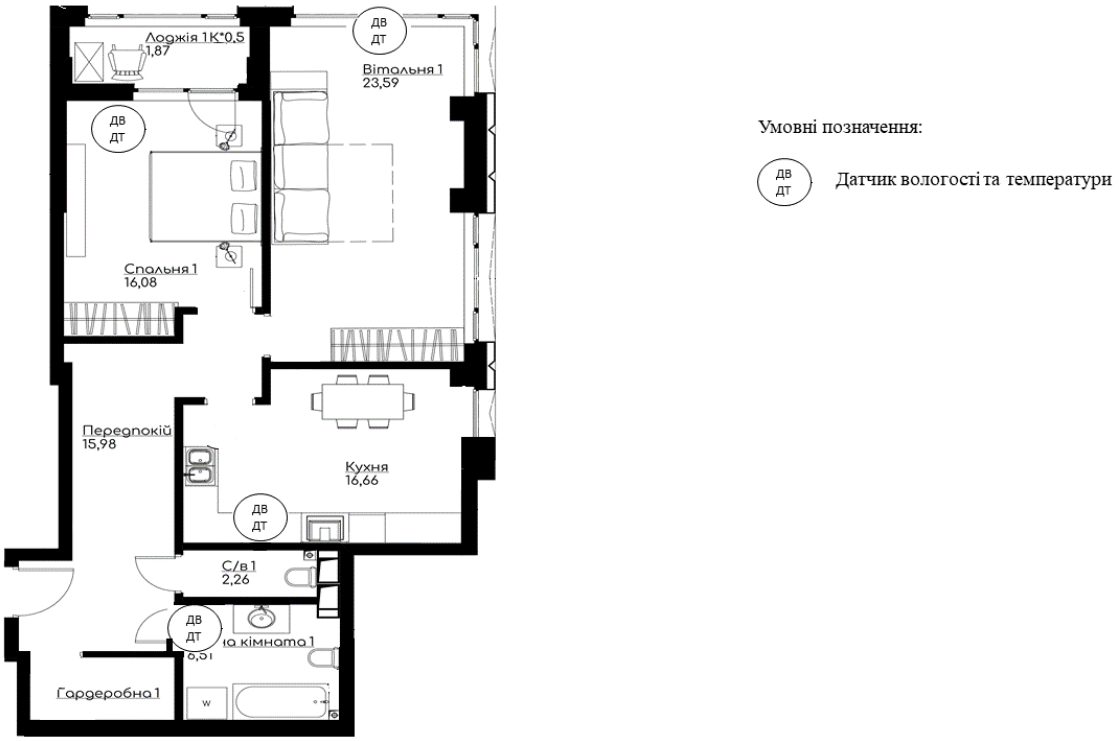


Рис. 1.1. Функціональна схема розумного дому

Контрольованими параметрами розумного помешкання, як відображено в таблиці 1.3, будуть температура та вологість. Визначимо верхні та нижні границі цих параметрів. Якщо показники будуть виходити за граничні межі (наприклад, температура вище 27^0C або нижче 23^0C), увімкнуться відповідно кондиціонер або опалення. Аналогічна схема – із контролюванням вологості повітря.

Система передбачає також створення сценаріїв. Наприклад сценарій «ранок» передбачає: увімкнення світла в спальній о 7.00 та увімкнення кавоварки о 7.15. Крім того управління розумними приладами можливо натисканням кнопки в додатку незалежно від сценаріїв.

Таблиця 1.3.

Керовані параметри

Керовані параметри	Одиниця виміру	Норма	Граничні межі
температура	^{0}C	25^{0}C	мінімальна – 23^{0}C, максимальна – 27^{0}C
вологість	%	55%	мінімальна – 40%, максимальна – 70%

Висновки до розділу 1

Розумний Дім отримав широке поширення в повсякденному житті і динаміка попиту на встановлення таких систем постійно зростає. У розумному будинку всі інженерні системи працюють злагоджено та виконують всі дії за користувача. Ця автоматизація підвищує якість життя в власному будинку.

Не дивлячись на те, що технологія системи Розумного Дому не нова та на ринку існує велика кількість виробників розумних девайсів та систем «під ключ», досі є ряд проблем, які виробникам доведеться вирішити в майбутньому. Одна з найактуальніших проблем полягає у використанні різних протоколів передачі данних. Найпопулярнішими є бездротові протоколи ZigBee, Z-Wave, Wi-Fi та Bluetooth Low Energy. Саме на них необхідно орієнуватись при проектуванні системи Розумний Дім. Об'єднання приладів із різними протоколами зв'язку в одну систему становить досить складну та вартісну задачу. Наразі не всі основні виробники готових систем підтримують усі чотири протоколи. До того ж у деяких виробників існує проблема з локалізацією, привабливістю і простотою інтерфейсу управління, можливістю управління с кількох приладів, тощо.

Після дослідження проблем об'єднання пристроїв IoT в єдину систему та огляду існуючих пропозицій на ринку, прийнято рішення дослідити можливість зниження вартості розробки систем «Розумний Дім» а також запропонувати варіанти підвищення зручності користувацькго інтерфейсу. Було визначено основні задачі, які буде вирішувати система, контрольовані параметри та запропонована функціональна схема розумного дому.

РОЗДІЛ 2

РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ «РОЗУМНИЙ ДІМ»

2.1. Аналіз і обґрунтування вибору програмних засобів для реалізації web-системи

Важливим елементом проектування системи Розумний Дім є програмні засоби її реалізації. Далі ми проаналізуємо мови клієнт-серверного програмування: C#, Java, JavaScript, PHP [12].

1. .NET Core (мова C#). .NET Framework, Visual Studio та ASP.NET дозволяють достатньо просто створити web-службу мовою C#. Компіляція та розгортання web-сервісу легка і швидка, сервером найчастіше служить IIS [12, 13].
2. Java EE. Надає широкі можливості безпеки, аутентифікації, авторизації, тощо. З'єднання з БД забезпечує JNDI API. Java API для обробки XML (JAXP) – частина Java SE [14].
3. PHP. Дозволяє легко створювати web-додатки, підтримує різноманітні БД (MySQL, Informix, Oracle, Sybase, PostgreSQL, Generic ODBC, тощо) та сумісний з більшістю сучасних web-серверів [14].
4. JavaScript. Працює як частина web-браузера, тому мова довгий час використовувалася виключно для програмування клієнта. Завдяки розвитку Node JS, Javascript стала і мовою серверного програмування. Node JS підтримує роботу з БД, файловою системою та іншим необхідним функціоналом. Особливо вдалою є інтеграція Node JS та Mongo DB [15].

Завдяки фрейворкам легко реалізуються основні паттерни програмування. Typescript від Microsoft дозволяє використовувати JavaScript як мову строгої типізації та реалізовувати ООП підхід. Тобто із скриптової мови JavaScript перетворився на об'єктно орієнтовану, дуже близьку до С-подібних мов.

Таблиця 2.1.

Особливості існуючих мов програмування для написання web-додатків [16, 17]

Особливості	C#/ASP.NET Core	Java EE	PHP-скрипти	JavaScript
Динамічні веб сторінки	Так, як код на web-сторінках ASP.NET Core	JSP	Підтримка, код за web-сторінкою php	Використання серверного JavaScript (SSJS)
Web-сервер	Microsoft IIS, Kestrel (Apache, Nginx)	Tomcat, JBoss, GlassFish	Apache, Microsoft IIS	Apache, Microsoft IIS
Web Framework/ бібліотеки	.NET Framework для ОС Windows і ASP.NET Core Linux та Unix	J2EE framework	Yii2, Symfony, тощо	JAXER, Node-red, ioBroker
Управління сесією	Так, управлінням станом ASP.NET Core	Так, вони представлені об'єктом HttpSession	Так, використовуючи $ _SESSION змінну	Так, використовуючи Jaxer.session
Безпека	Так (Архітектура безпеки ASP.NET Core)	Так	Не настільки безпечно	Так

Продовження таблиці 2.1.

Model-View Controller	ASP.NET MVC 5/ Core MVC	J2EE / Spring / Struts	Symfony / Mojavi / CakePHP	Так
Взаємодія бази даних	Так, використовуючи бібліотеку ADO.NET, або Entity Framework	Так (JDBC)	Mysql	Mongo DB, Jaxer.DB
Підтримка рівня безпеки Secure Sockets Layer(SSL)	Так, сертифікат SSL може бути створений та використаний для сервера IIS	Так, JavaSSL	Так (відкрита SSL-функція)	Так, сертифікат SSL може бути створений та використаний для сервера IIS
Web-сервіси (SOAP, WSDL, UDDI)	Так: web-сервери ASPT.NET для Windows і Web Services проекту Core на інших ОС (Linux / Unix, Mac)	Так, за допомогою Apache XML-RPC для бібліотеки Java або Spring Web Services, JAX-WS)	Так (XML-RPC для PHP)	Так, можна використовувати клієнт web-сервісу на основі AJAX / XMLHttpRequest

Продовження таблиці 2.1.

Композиція web-служб	Використання web-сервісів ASP.NET Core. Також можливо, зв'язати два або більше Web Services, які обмінюються даними між собою	Композиція J2EE WS	Так (заявлено PHParray або політичними файлами або вбудовано в WDSL)	Так, завдяки Node JS (веб сервіси будуть написані мовою JavaScript)
Безпека web-служб	Так, шифрування та підпис HTTPS за допомогою SSL стандартів	Так. HTTPS, підтримує "стандарти, основані на XML" (WS-Policy, WSSecurity, WSTransfer стандарти)	Часткова підтримка (обмежена взаємодією SOAP з WSDL)	Так, Ajax можна використовувати з будь-якою динамічною мовою web-програмування, що підтримує протокол HTTPS (SSL)

Порівнюючи основні мови програмування веб додатків можна зробити висновок, що найменш придатна для розробки системи Розумній Дім – мова PHP.

C#/ASP.NET Core, Java EE та JavaScript надають всі необхідні можливості для програмування серверної частини проекту. Клієнтська частина може бути написана тільки за допомогою JavaScript. Програмування мікроконтроллерів можливо будь-якою мовою із перелічених вище за допомогою бібліотек. Серед фреймворків та

бібліотек найпотужнішою є JavaScript фреймворк ioBroker. Вона надає компоненти для роботи із кожним типом бездротових приладів (Zigbee, Wi-Fi, Bluetooth low energy, Z-wave). Усі частини бібліотеки легко інтегруються між собою, що дає можливість об'єднати пристрої з різними протоколами в одну систему. Архітектура фреймфорку є досить простою із можливістю масштабування системи.

Таким чином використання мови JavaScript для створення системи Розумний Дім виглядає досить логічним, оскільки завдяки новітнім бібліотекам та SSJS вона повністю вдовольняє задачам програмування. До того ж, завдяки бібліотекам React Native або Cordova, JavaScript можемо використати для написання інтерфейсу для Android та/ або iOS. Саме її авторка даної роботи обрала для створення власної системи, орієнтуючись в тому числі на перспективу менших витрат на створення системи порівняно з існуючими аналогами.

2.2. Аналіз і обґрунтування вибору мікроконтроллерів з можливістю програмування логіки мовою Java Script

2.2.1. Аналіз принципів роботи платформи Espruino

Платформа Espruino являє собою набір програмних і апаратних засобів для програмування мікроконтролерів [18] мовою Java Script. Вона складається з:

- Прошивки для мікроконтролера з інтерпретатором мови Java Script.
- Середовищи для програмування Espruino Web IDE.
- Плати з мікроконтролером з самою прошивкою для безпосередньої роботи з Espruino.

Щоб писати програми для мікроконтролера на Java Script потрібно встановити Espruino Web IDE. Це програма або розширення для популярного інтернет-браузера Google Chrome, яке встановлюється буквально в один клік [18].

Після натискання на кнопку «Встановити» в меню «сервіси» з'явиться додаток. Потрапити в це меню можна набравши в адресному рядку: chrome://apps/.

При натисканні на піктограму з чашкою кави відкривається саме середовище розробки, що примітно не в браузері, а в окремому вікні [18].

Espruino Web IDE надає два види інтерфейсу: для програмування логіки мовою Java Script (рисунок 2.1) і для програмування за допомогою псевдо коду (рисунок 2.2).

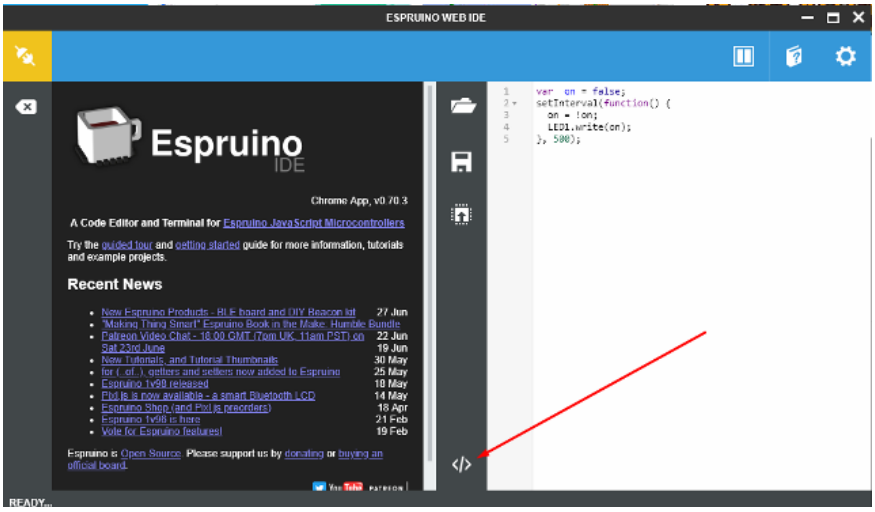


Рис. 2.1. Інтерфейс Espruino Web IDE для програмування логіки мовою Java Script

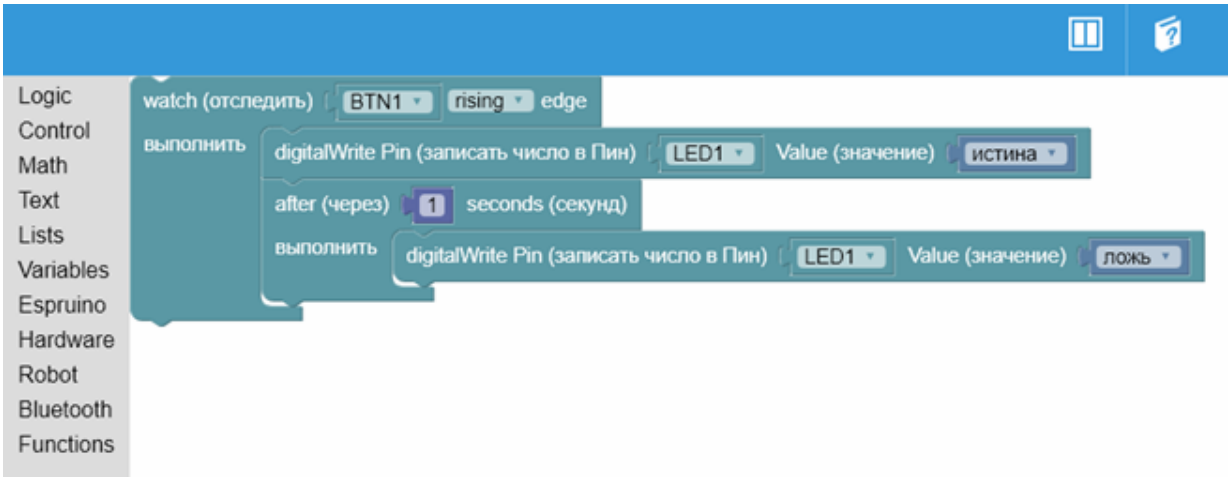


Рис. 2.2. Інтерфейс Espruino Web IDE для програмування за допомогою псевдо коду

Плати «Espruino» і їм подібні являють собою друковану плату з необхідним навісним обладнанням і мікроконтролером, з прошитим інтерпретатором мови Java Script, який обробляє код і переводить його команди в мову одиниць і нулів

безпосередньо під час виконання. У той час як в класичному вигляді програма для мікроконтролерів завантажується в останній вже в вигляді машинного коду [18].

Розглянемо найактуальніші з готових рішень:

Puck.JS - це мініатюрна плата в корпусі, у неї на борту є Bluetooth та інфра червоний передавач [18].

Згідно з інформацією з офіційного сайту, його технічні характеристики і особливості наступні [18, 19]:

- Bluetooth Low Energy;
- Предвстановлений Espruino Java Script інтерпретатор;
- Серцем плати є nRF52832 SoC - 64MHz ARM Cortex M4, 64kB RAM, 512kB Flash;
- 8 x 0.1" GPIO портів (серед них PWM (ШИМ), інтерфейси SPI, I2C, UART, аналоговий вхід);
- 9 x SMD GPIO портів (серед них PWM, SPI, I2C, UART);
- Корпус-шайба з ABS-пластику;
- MAG3110 Magnetometer – вимірювач магнітного поля з трьома вісями;
- IR-передавач;
- Термометр, датчик освітленості і рівня заряду батареї;
- Три світлодіода (червоний, зелений і синій);
- NFC-мітка, що програмується мовою Java Script.
- Піни можуть сприймати дотики (принцип роботи сенсорних екранів смартфонів);
- Вага 14 г;

Рис. 2.3. Зовнішній вигляд плати Puck.JS

Наступна плата - Espruino Wi-Fi, як видно з назви її особливістю є вбудований модуль бездротового зв'язку по мережі Wi-Fi.

Її технічні характеристики [18]:

- Розміри плати: 30x23 мм;
- На платі розміщений роз'єм Micro USB;
- 21 GPIO: 8 аналогових входів, 20 ШИМ, 1 Serial-порт, 3 SPI, 3 I2C;
- На платі встановлено 3 світлодіода, 2 з яких програмуються користувачем, а 1 відображує активність Wi-Fi;
- 1 кнопка;
- Побудована на мікроконтролері STM32F411CEU6 32-bit 100MHz ARM Cortex M4 CPU;
- Пам'ять: 512kb flash, 128kb RAM;
- Зв'язок по Wi-Fi за допомогою ESP8266 (802.11 b/g/n);
- RTC (годинник реального часу із зовнішнім генератором);
- На платі встановлено стабілізатор на 3.3В зі струмом до 250 мА, підтримується напруга від 3.5 до 5 В;

- Споживання струму в сплячому режимі - до 0.05 мА, що дозволяє працювати 2,5 роки від акумулятора ємністю в 2500mAh.

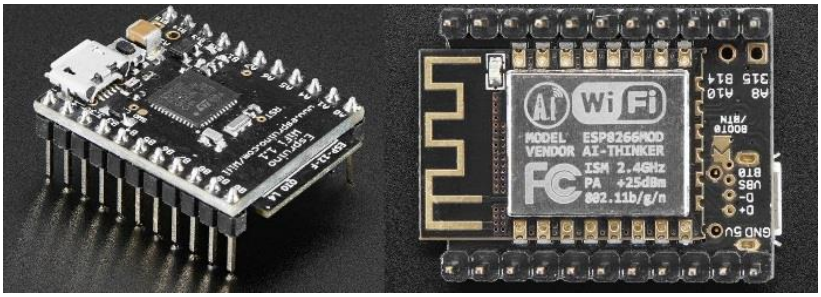


Рис. 2.4. Зовнішній вигляд плати Espruino Wi-Fi

Espruino Pico – мікроконтролер для Java Script в форматі USB-флешки, яка для програмування вставляється напряму в USB-порт комп'ютера.

Рисунок 2.5 Зовнішній вигляд плати Espruino Pico

Технічні характеристики [19]:

- Розміри: 33x15 – із врахуванням USB – штекера;
- 22 GPIO порта, серед яких: 9 аналогових входів, 21 ШИМ, 2 Serial, 3 SPI, 3 I2C;
- USB Type A штекер є частиною плати;
- Два світлодіоди та одна кнопка, що програмуються;
- Побудован на мікроконтролері STM32F401CDU6 32-bit 84MHz ARM Cortex M4 CPU;
- Пам'ять: 384kb flash, 96kb RAM;

- На платі розпаяний стабілізатор напруги на 3.3 В 250mA дозволяє живити її від 3.5 до 16 В;
- У режимі очікування споживає струм до 0.05 мА, виробник і тут заявляє про 2.5 роках роботи від 1 акумулятора на 2500mAh;
- Вбудований польовий транзистор для управління ланцюгами з високим робочим струмом.

2.1.1 Аналіз плати Iskra JS

Iskra JS - плата з вбудованим інтерпретатором JavaScript. Плата є розвитком платформи Espruino, але сумісна з платами / шілд для Arduino. Побудована на мікроконтролері Cortex-M4. 168 МГц, оперативна пам'ять на 5000 змінних Java Script, 1 МБ флеш-пам'яті для зберігання програми. Iskra JS працює швидше, ніж Espruino і за своїми характеристиками не поступається Arduino, який багато хто сприймає практично як золотий стандарт. Порівняльні характеристики двох плат наведені в таблиці 2.2.

Технічні характеристики [18, 20]:

- Мікроконтролер: STM32F405RG (32-бітний ARM Cortex M4);
- Тактова частота: 168 МГц;
- Флеш-пам’ять: 1024 кБ;
- SRAM-пам’ять: 192 кБ;
- Номінальна робоча напруга: 3,3 В;
- Рекомендована вхідна напруга: 7–15 В или 3,6–12 В;
- Максимальний струм з шини 5V: 1000 мА;
- Максимальний струм з шини 3.3V: 300 мА (включно із живленням мікроконтролера);
- Максимальний струм з піна або на пін: 25 мА;
- Максимальний сумарний струм з пінів або на піни: 240 мА;
- Кількість портів вводу-виводу загального призначення: 26;

- Портів з підтримкою ШИМ: 22;
- Портів з АЦП: 12 (12 бит);
- Портів з ЦАП: 2 (12 бит);
- Доступні апаратні інтерфейси: 4× UART/Serial, 3× I²C/TWI, 2× SPI;
- Габарити: 69×53 мм.

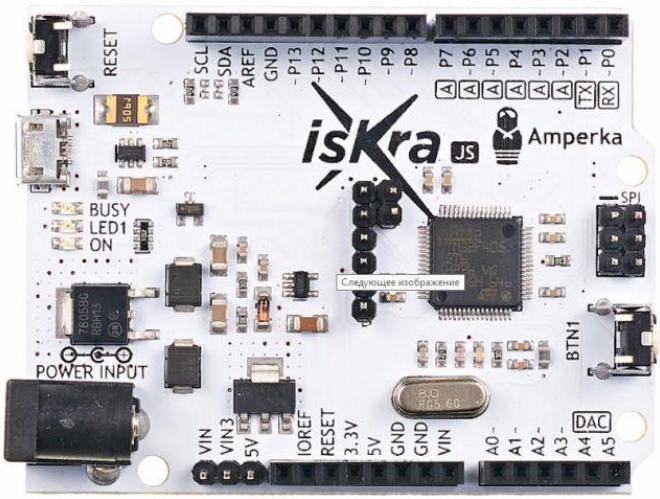


Рис. 2.6. Зовнішній вигляд плати ISKRA JS

Таблиця 2.2.

Порівняння основних технічних характеристик плат Iskra JS та Arduino UNO r3

Характеристика	Iskra JS	Arduino UNO r3
Розрядність МК	32 біт	8 біт
Тактова частота	168 МГц	16 МГц
Об'єм пам'яті	Flash – 1024кб, SRAM – 192 кб, EEPROM – 1кб	Flash – 32кб, SRAM –2 кб, EEPROM – 1кб
Кількість ШИМ-пінів	22, 2 можуть видавати аналоговий сигнал	6

Продовження таблиці 2.2.

Кількість аналогових входів	12	6
Розрядність АЦП	12 біт	10 біт
Напруга логіки входів	3.3 В, але багато з них толерантні до 5В	5В
Струм входів та виходів	25 мА, максимальний сумарний струм – 240 мА	40 мА, максимальний сумарний струм – 150 мА
Доступні інтерфейси	4xUART/Serial, 3xI²C/TWI, 2xSPI	1xUART/Serial, 1xI²C, 1xSPI

Iskra JS програмується мовою JavaScript. Програмування проводиться в середовищі Espruino Web IDE.

2.2.3. Аналіз мікрокомп'ютерів Raspberry PI

Raspberry PI - одноплатний комп'ютер розміром з кредитну карту. Головна перевага Raspberry Pi це 40 контактів введення / виведення загального призначення (GPIO). До них ви сможетможно підключати периферію, виконавчі пристрої, будь-які сенсори і все, що працює від мережі.

Зазвичай плати лінійки Raspberry виходять в різних версіях, відрізняються маркуванням типу «Model A», «Model B» і подібні, відмінності полягають в периферії і потужності. Розглянемо найактуальнішу модель Raspberry Pi в 2020 році - Raspberry Pi 4 Model B.

Технічні характеристики [21]:

- Broadcom №2711, 64-розрядний чотирьохядерний процесор BCM2711 Cortex-A72;

- 1 ГБ / 2 ГБ / 4 ГБ / 8 ГБ ОЗУ LPDDR4 на вибір;
- Порт True Gigabit Ethernet;
- 2 порти USB 3.0 «Super-Speed»;
- 2 x USB 2.0 високошвидкісних порти;
- Бездротова локальна мережа 802.11 b/g/n/ac (2,5 ГГц и 5 ГГц);
- Bluetooth 5.0;
- Подвійні порти micro-HDMI, 4K UHD видео;
- Декодування H.265 (4kp60)
- Декодування H.264 (1080p60);
- OpenGL ES 1.1, 2.0, 3.0 графіка;
- PoE підтримка;

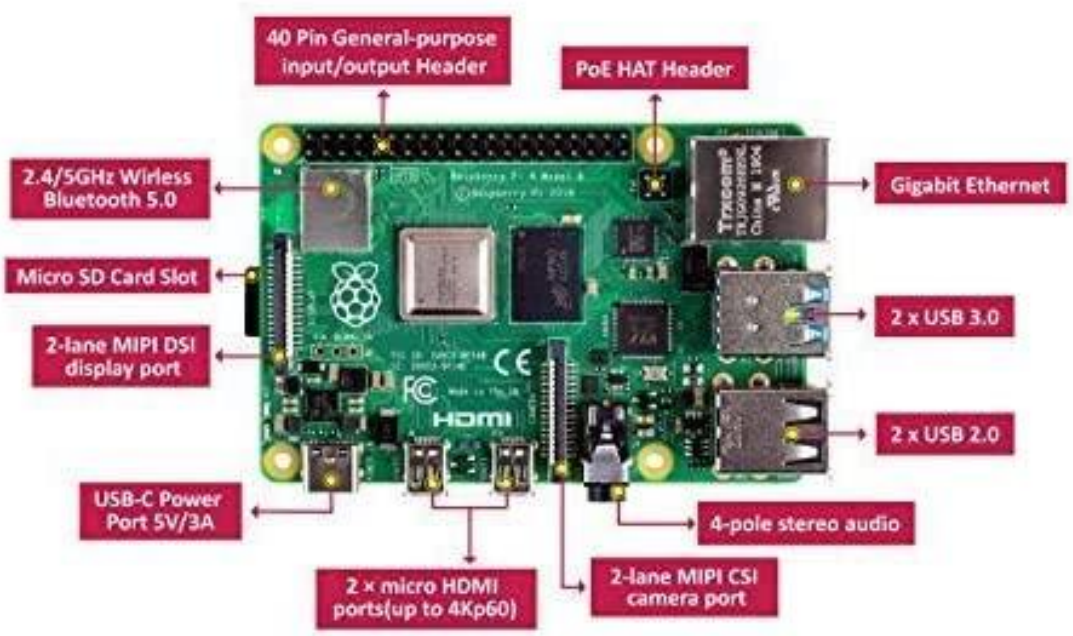


Рис. 2.7. Зовнішній вигляд плати Raspberry Pi 4 Model B

Екосистема Raspberry PI не надає свою IDE для програмування логіки, проте існує досить багато різноманітних IDE, що дозволяють працювати з Raspberry. Наприклад Adafruit – web-орієнтовна IDE. Все, що потрібно зробити для початку роботи – підключити Raspberry Pi до локальної мережі та увійти з нього в Adafruit IDE у web-браузері. Adafruit підтримує розробку мовами Python, Ruby, JavaScript та

ін., дозволяє відправляти різноманітні команди на Raspberry PI через термінал у веб браузері, дозволяє працювати з GIT репозиторіями.

2.2.4. Вибір плати та IDE для програмування логіки системи «Розумний Дім»

На даний момент найбільш сучасною є платформа Espruino. Її основною перевагою є екосистема, що складається із середовища програмування Espruino Web IDE, Espruino Firmware (JavaScript-машина на мікроконтроллері), плати сумісні з Espruino та внутрішні бібліотеки.

Код для Espruino Firmware пишеться у середовищі Node JS. Розробникам не потрібно думати про будь-яку інтеграцію. Для програмування логіки роботи з мікроконтроллерами можна використовувати pure Java Script та пыдключати зовнішні бібліотеки за допомогою npm (Node Package Manager) [22].

Node Package Manager включає як спеціальні бібліотеки для роботи з Espruino, так і бібліотеки для серверного програмування (наприклад FS – бібліотека для роботи з файловою системою), які за потреби можна без обмежень використовувати в розробці системи управління розумним домом.

Node JS із движком V8 під капотом надає потужні можливості для реалізації клієнт-серверної архітекрури та роботи із Mongo DB. Для взаємодії з базою даних через npm підключається пакет mongo-db. Він в свою чергу надає доступ до MongoClient. Команда MongoClient.connect() встановлює з’єднання з базою даних. Запити до бази (insert, update, delete, тощо) виконуються мовою Java Script. Наприклад команда db.collection('notes').insert(note, (err, results) => {}) вставить новий рядок в таблицю «notes» та буде обробляти помилку в разі її виникнення.

Таким чином Espruino дає можливість без будь-яких додатковіх налаштувань працювати в середовищі Node JS та завдяки пакету mongo-db виконувати будь-які запити до бази даних Mongo. Також Espruino надає власну IDE та бібліотеки для програмування. Легка інтеграція з Node JS та Mongo DB є перевагою в порівнянні з

Raspberry PI [23]. Саме тому для реалізації системи управління розумним домом було обрано платформу Espruino.

Окрім значних переваг, обрана платформа має суттєвий недолік: плати лінійки Espruino не сумісні з Arduino. Ця проблема вирішується використанням плати Iskra JS, яка була створена на основі Espruino. Також нам знадобиться плата Puck.JS для управління Bluetooth приладами.

2.3. Вибір UI фреймворку для створення інтерфейсу управління розумним домом

Серед фреймворків для створення інтерактивних інтерфейсів найпопулярнішими є: React, Angular та Vue. Кожен із них декларує компонентний підхід для створення інтерфейсу: розробникам пропонується створювати інкапсульовані компоненти, які керують власним станом, а з них будувати складні інтерфейси. Стан додатку зберігається окремо від DOM.

Також кожен з фреймворків пропонує чітку архітектуру. Наприклад Vue використовує модель MVVM (Model Vue Vue Model), Angular – класичну MVC (Model Vue Controller) модель, React пропонує власну архітектуру Flux. Для створення інтерфейсу системи управління розумним будинком серед переваг React та Vue є швидкість опрацювання даних. В React додатках вона досягається завдяки Virtual DOM (при необхідності оновити елемент DOM оновлюється не вся модель, а тільки той компонент, що змінився, таким чином рендерінг інтерфейсу проходить непомітно для користувача), а також завдяки невеликій вазі бандла, що завантажується браузером [24].

Для проекту Розумний Дім важливою перевагою React є можливість його роботи на сервері, викрористорвуючи Node.js та на на мобільних платформах. Для розробки мобільних додатків існує бібліотека React Native. Архітектура та стиль написання коду додатків React Native майже ідентичні тим, що використовуються для створення веб інтерфейсів за допомогою React. Таким чином зробити

портування додатку з веб на будь яку мобільну платформу буде значно швидшим, простішим і дешевшим, якщо і веб і мобільні версії написані за допомогою React.

Розглянемо детальніше архітекруру та особливості React-додатків.

React - популярна технологія для Frontend-розробки, який прийнято називати фреймворком, хоча базово він не пропонує повноцінну архітектуру. З точки зору MVC моделі все, що він дає - це View (Представлення).

Так як він заснований на компонентному підході, програміст може розробляти додаток за допомогою React просто вказуючи, як би він хотів бачити той чи інший елемент. React буде автоматично оновлювати елемент, коли дані, що лежать в його основі, зміняться. Головні принципи React: гнучкість, ефективність і декларативний код.

Так як React гнучкий, то можна використовувати один і той же код в декількох проектах, створювати на його основі нові додатки і навіть використовувати в уже існуючій базі код без переробок.

Отже, React відповідає за V або View в MVC. За Model (M в MVC) частину відповідає шаблон проектування Flux. Це архітектура, що відповідальна за створення шару даних в JavaScript додатках і розробку серверної сторони в web-додатках. Flux доповнює складові компоненти виду View в React, використовуючи односпрямований потік даних.

Також можна сказати, що Flux більше ніж шаблон, більше ніж фреймворк – це новий підхід до архітектури, який спочатку використовувався для роботи з React додатками, а пізніше став достатньо розповсюдженим в комбінації з іншими фреймворками.

Flux пропонує 4 основних компонента [25]:

- Дія (Action) - помічники, що передають дані в Dispatcher;
- Диспетчер (Dispatcher) - отримує ці дії і передає корисне навантаження зареєстрованим callback-ом;

- Сховища (Stores) - контейнери для стану програми та логіки. Реальна робота додатка відбувається в Stores. Stores, зареєстровані для прослуховування дій Dispatcher, будуть відповідно і оновлювати View;
- Представлення (Views) - React компоненти, що забирають стан (дані) зі Stores, а потім передають дочірнім компонентам;

Це не схоже на звичний MVC, який ми звикли бачити в інших фреймворках. Дійсно, там є Контролер, але здебільшого це контролер, який відповідає за Views. Views знаходяться вгорі ієрархії, і вони передають функціонал і дані елементам-нащадкам.

Flux слідує концепції односпрямованого потоку даних, що робить його простим для пошуку помилок. Дані проходять через прямий потік програми. React і Flux - на даний момент два найпопулярніших фреймворка, які використовують принцип односпрямованого потоку даних [25].

У той час як React використовує віртуальний DOM для відображення змін, Flux робить це трохи інакше. У Flux, взаємодія з призначеним для користувача інтерфейсом викличе ряд дій, які можуть змінити дані програми.

Основна проблема з MVC полягає в тому, що він недостатньо масштабується. Flux довів свою ефективність, тому що все, що пов'язано з Flux, стосується тонкої настройки потоку всередині програми. Розглянемо головні особливості завдяки яким Flux має перевагу над MVC [25]:

1. The Flow - Flux дуже вимогливий до потоку даних в додатку. Dispatcher даних встановлює суворі правила і виключення для управління потоком. У MVC немає такої властивості, потоки реалізуються по-різному.
2. Односпрямований потік в Flux (в той час як MVC двонаправлений в своєму потоці) означає, що всі зміни проходять через один напрямок, через Dispatcher даних. Store не може бути змінено сам по собі. Таким чином структура коду виглядає дуже добре організованою навіть у великих

проектах, які роками підтримуються і допрацьовуються різними командами.

3. Store - в той час як MVC не може моделювати окремі об'єкти, Flux може робити це для того, щоб зберігати будь-які пов'язані з додатком дані.

Наглядно різниця між MVC та Flux представлена на рисунках 2.8 та 2.9 [25].

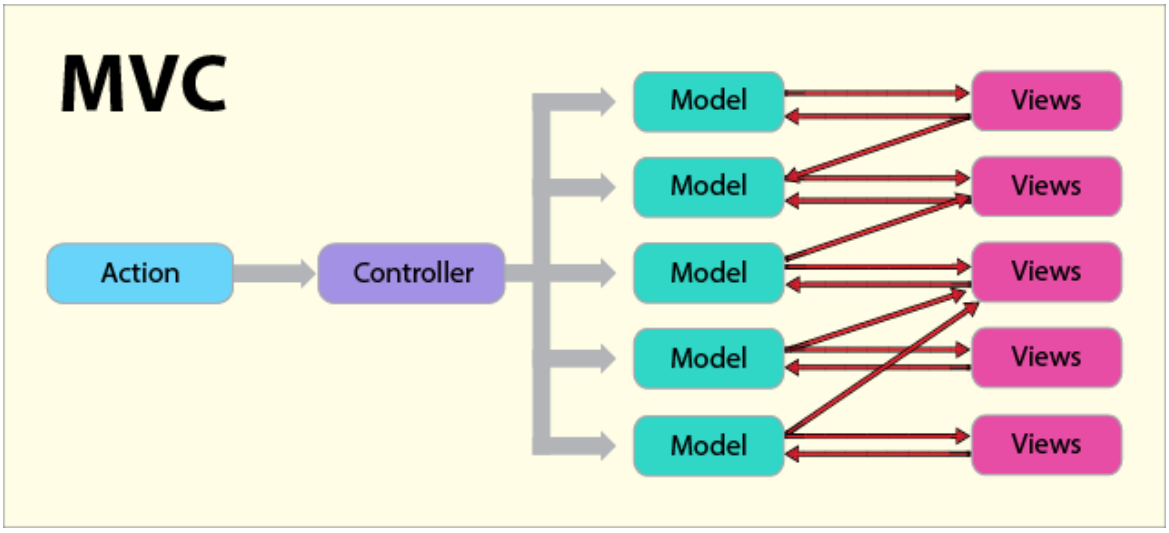


Рис. 2.8. Схематичне представлення направлення потоків даних MVC додатків

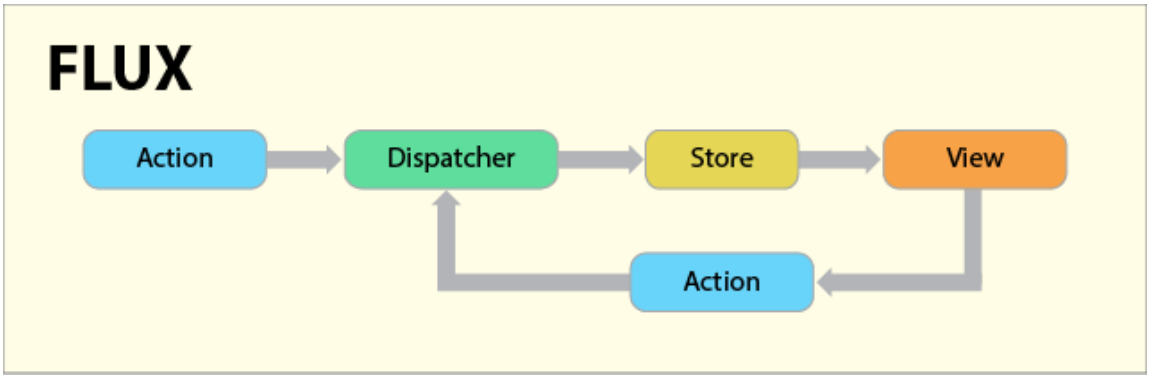


Рис. 2.9. Схематичне представлення направлення потоків даних в додатках з архітектурою Flux

Архітектура Flux реалізована у кількох різних бібліотеках. Найпопулярнішою на даний момент є Redux Saga, яка використовується найчастіше разом із React.

Таким чином фреймворк React в комбінації із бібліотекою Redux Saga повністю задовольняє потреби розробки інтерфейсу системи управління розумним домом.

2.2 Розробка схеми КСУ розумним домом

В першому розділі були сформульовані завдання, які має вирішувати система. Неперервні завдання – це ті, які система відтворює без втручання користувача. Вони включають в себе:

- Регулювання температури: вимірювання температури та порівняння із граничними показниками. Якщо температура перевищує верхній граничний показник, увімкнеться систему охолодження і навпаки – система обігріву увімкнеться при зниженні температури нижче допустимої границі.
- Регулювання рівня вологості: здійснюється за аналогічною схемою.
- Керування освітленням: датчики освітлення відстежують рух у певній зоні. При отриманні сигналу від датчика система вмикає задане реле.
- Відтворення сценаріїв користувача: із заданим інтервалом відтворюється послідовність команд, які вмикають задані реле.

Контрольовані завдання – це ті, які задає користувач за допомогою веб сервісу. Серед них увімкнення або вимкнення світла, кавоварки, термостату, системи охорони, тощо за допомогою кнопки в додатку.

Структурна схема даної системи зображена на рисунку 2.10. UML-діаграма прецедентів наведена на рисунку 2.11 [26]. Акторами в даній схемі є користувач, кліент, сервер та контролер.

Розглянемо яким чином розподіляються задачі між клієнтом, web-сервером та контролером [27]:

1. Клієнт:
    - Містить в собі store, що зберігає актуальний стан додатку;
    - Обробляє команди користувача (JavaScript events): відстежує дії користувача, передає дані на сервер, отримує відгук з сервера та оновлює store;

- Відстежує зміни стану системи (тобто зміни, які приходять з веб сервера) та оновлює store;
- Оновлення store викликає ре-рендерінг того UI компоненту, у якого змінилися дані для відображення.

2. Сервер:
   - Виконує авторизацію користувача;
   - Приймає дані від клієнта та оновлює базу даних;
   - Приймає дані від контролера та оновлює базу даних;
   - Передає дані від клієнта до контролера і навпаки.
3. Контролер:
   - Виконує команди, отримані від сервера;
   - Передає дані до сервера, якщо від останнього надійшов запит;
   - Виконує неперервні завдання (передаючи до сервера дані про зміну стану системи).

У цій схемі важливо тримати БД в актуальному стані для забезпечення можливості мультикористувацького додатку. Чим би не користувався кінцевий споживач (додатком у смартфоні, web-сайтом, тощо), клієнт завжди отримає актуальні дані від сервера.

Безпека схеми досягається за рахунок використання токенів при авторизації користувача. При вході користувача в систему реалізується наступний сценарій:

- Клієнт відправляє на сервер імейл і пароль;
- Сервер шукає користувача в базі даних;
- Якщо користувач існує, сервер хешує пароль та порівнює його з хешем пароля бази даних;
- В разі успіху сервер генерує токен аутентифікації (json web token).

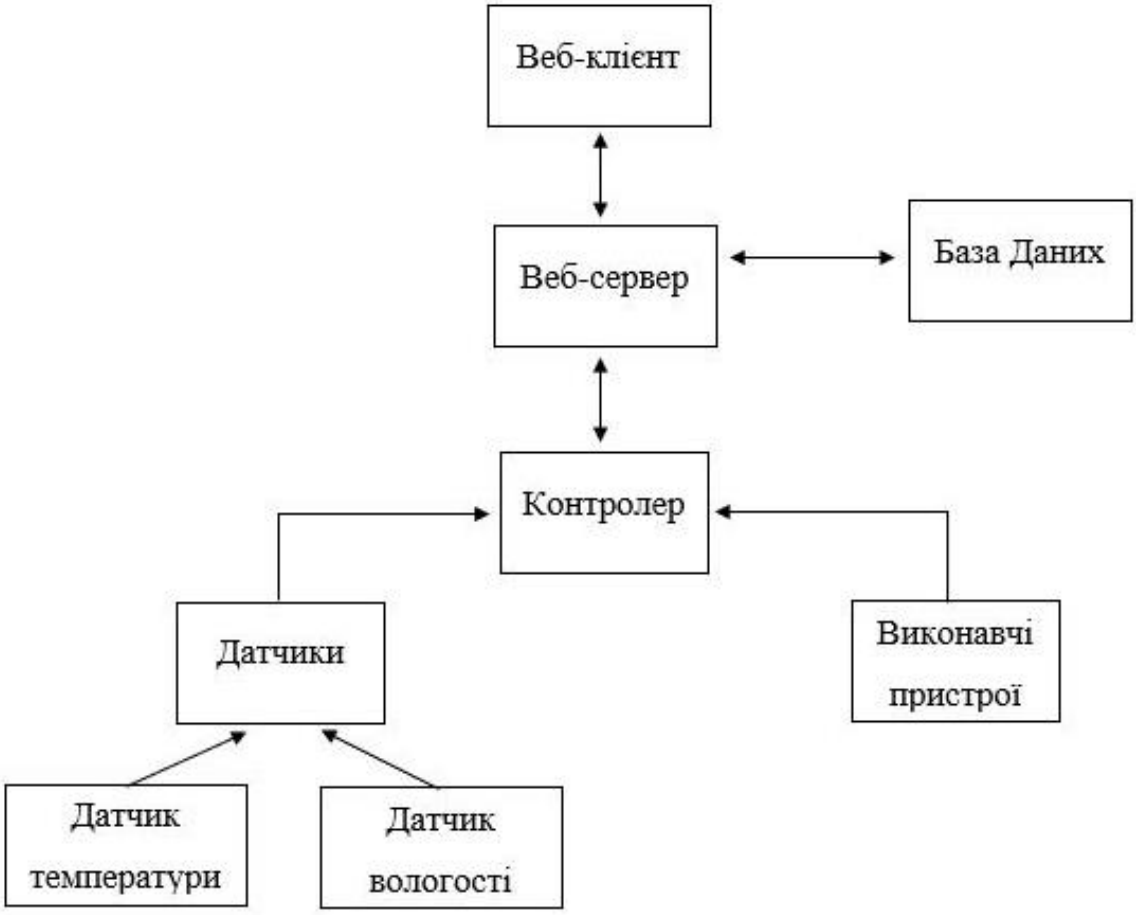
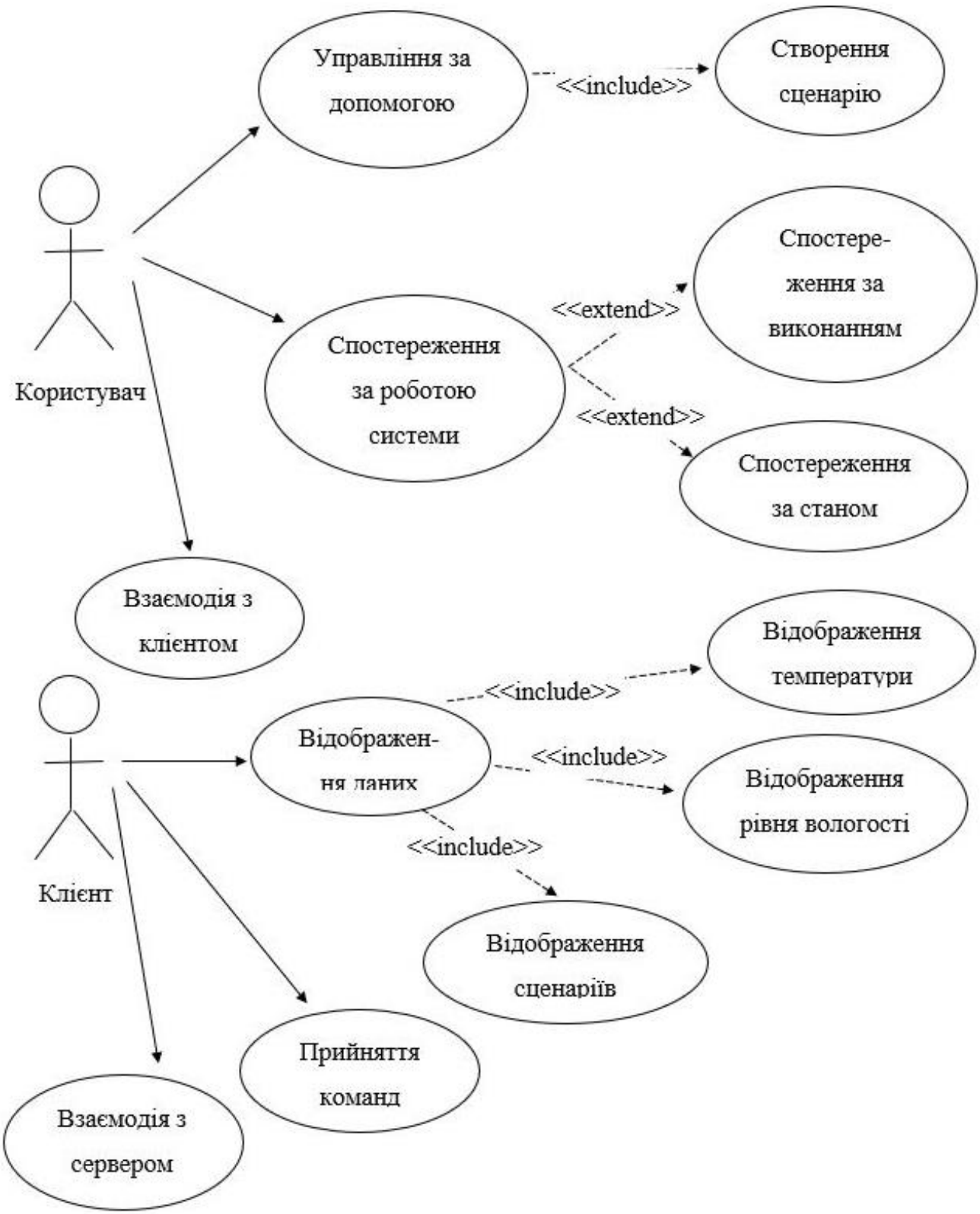


Рис. 2.10. Структурна схема системи «Розумний Дім»


Управління за допомогою
<<include>>
Створення сценарію
Користувач
Спостереження за роботою системи
<<extend>>
Спостереження за виконанням
<<extend>>
Спостереження за станом
Взаємодія з клієнтом
Клієнт
Відображення даних
<<include>>
Відображення температури
<<include>>
Відображення рівня вологості
<<include>>
Відображення сценаріїв
Прийняття команд
Взаємодія з сервером

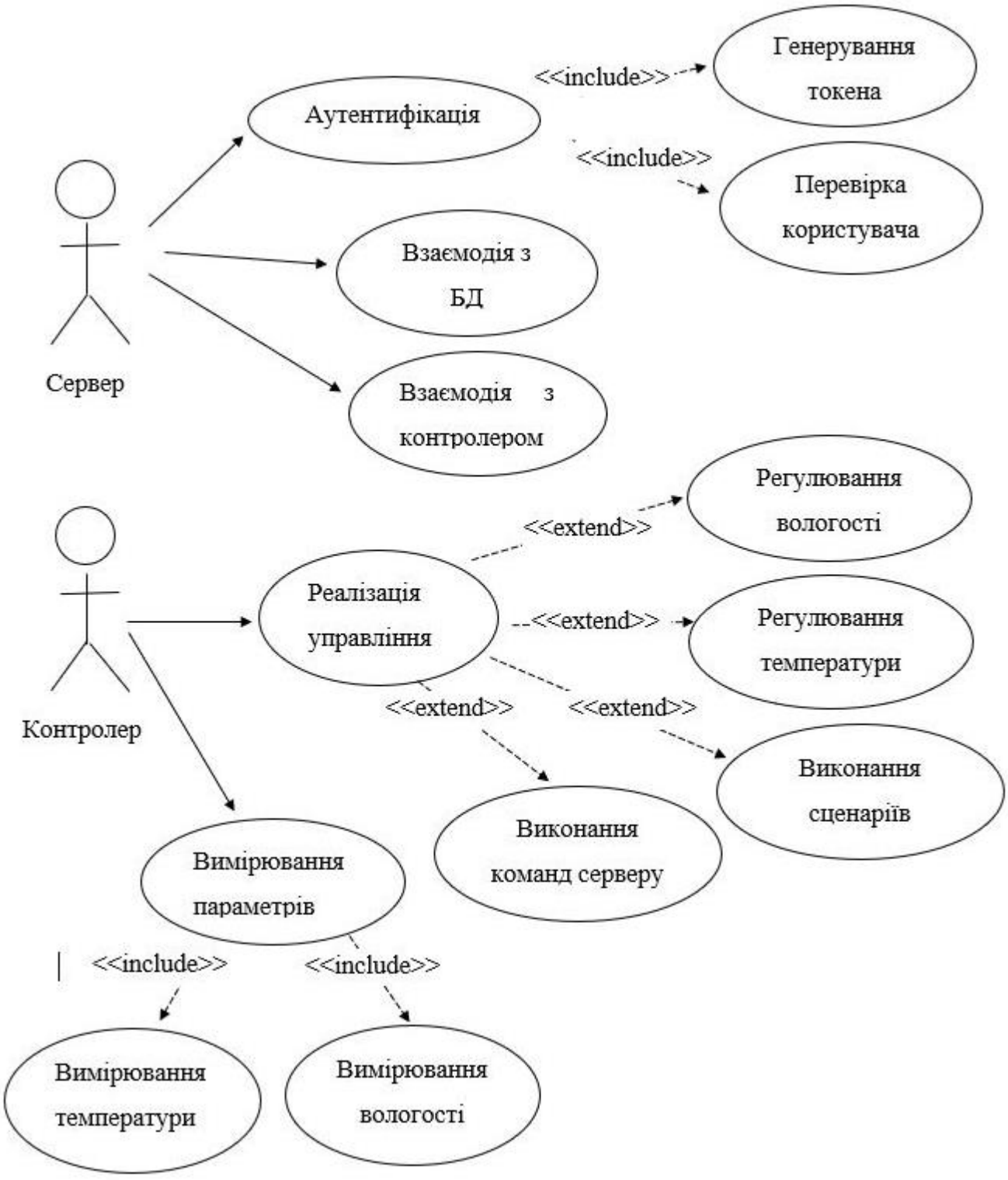


Рис. 2.11. UML-діаграма прецедентів

Для досягнення мети зменшення вартості кінцевого програмного продукту слід приділити увагу архітектурному рішенню. Основну вартість додатку складає оплата праці програмістів, тестувальників, системного архітектора, бізнес аналітиків, тощо. Так, за даними сайту dou.ua, середня місячна заробітна плата системного архітектора складає близько 5 500 у.о., програміста рівня Middle – 2 000 у.о, тестувальника того ж рівня – 1500 у.о. Як буде видно в подальших розрахунках (розділ 4), вартість мікроконтроллерів, модулів та навіть розумних девайсів для тестування системи в рази менша фонду оплати праці. Зрозуміло, що чим складніше архітектурне рішення системи, тим більшу кількість часу буде витрачено на розробку програмного продукту і тим більша команда має бути залучена. Тож, просте архітектурне рішення одразу зменшує кількість спеціалістів, задіяних для його розробки.

Окрім первинної розробки програмного продукту, існує довгий цикл його технічної підтримки і розвитку. Цей період має складати увесь життєвий термін продукту. Чим більш простою та зручною є початква архітектура, тим легше і відповідно швидше та дешевше стає масштабування програмного продукту в майбутньому.

Для спрощення архітектури в даній роботі буде використано єдину мову програмування, що дозволяє відмовитись від проміжних фасадів та мінімізує простой в розробці (за потреби Java Script програміст із досвідом зможе легко переключатись між задачами на будь-якому рівні системи). Обрана Mongo DB не потребує складних архітектурних рішень, база розгортається за годину. В проекті планується використання компонентного підходу в програмуванні клієнта та модульного на рівні сервера та контролера фізичних пристроїв. Такий підхід робить код зрозумілим, масштабованим та інкапсульованим. Додавання нового девайсу не зламає існуючу програму навіть якщо в новому коді будуть помилки. Використання модулів та компонентів дає можливість швидше вникнути в роботу системи новим інженерам та програмістам, а бібліотеки React та Redux диктують стандарти написання коду. Тож можемо прогнозувати, що система в майбутньому при

додаванні нового функціоналу залишатимиться такою ж простою. Всі перечисленні рішення скоротять кількість залучених спеціалістів та зменшать час на розробку системи, а отже мінімізують витрати на впровадження стартапу, розвиток та підтримку продукту в майбутньому.

Висновки до розділу 2

В другому розділі були розглянуті та обрані засоби для реалізації системи управління розумним домом. На думку авторки даної десертації цікавим варіантом для розробки ПО є використання мови Java Script. Останніми роками завдяки розвитку Node JS, Java Script стала мовою програмування і на стороні сервера. Node JS дуже легко комбінується із базою даних Mongo DB, що дозволяє реалізувати клієнт-серверну архітектуру виключно мовою Java Script.

Програмування мікроконтроллерів мовою Java Script не є особливо поширеним. Тим не менш існує ряд мікроконтролерів та мікрокомп'ютерів, середовище розробки яких або орієнтовано на роботу із Node JS, або принаймні підтримує таку можливість. Найпопулярніші мікроконтролери, їх технічні характеристики, переваги та недоліки також були розглянуті у цьому розділі. Найбільш потужним варіантом є використання платформи Espruino, середовище розробки Espruino IDE та мікроконтроллер ISKRA JS, створений на основі ядра Espruino. Програмування логіки мікроконтроллера в такій комбінації можливе через Node JS. До того ж для Espruino існують чисельні бібліотеки, що легко підключаються через менеджер пакетів Node JS.

Також в даному розділі було обрано фреймворк та архітектуру для програмування клієнтської частини (комбінація React та Redux). React реалізує компонентний підхід для створення Vue та може використовуватись як для web-, так і для мобільної розробки. Redux реалізує архітектурний підхід Flux і виконує функції контролера та сховища, в якому міститься поточний стан додатку.

Можливість розробки всієї системи однією мовою та в одному середовищі значно полегшує інтеграцію усіх частин системи (клієнтської, серверної та програми управління мікроконтролерами), що в свою чергу знизить вартість розробки.

На основі обраних технологій була розроблена структура схема КСУ розумним домом та UML-діаграма прецедентів.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ СИСТЕМИ УПРАВЛІННЯ «РОЗУМНИЙ ДІМ»

3.1. Проведення UI/UX дослідження та розробка прототипу системи управління розумним будинком

UX – це досвід користувача, тобто те, який досвід та враження отримує користувач від роботи з інтерфейсом. Чи вдається йому досягти мети і на скільки просто або складно це зробити [28].

User Experience дизайн включає в себе багато дисциплін, такі як [28]:

- інтерактивний дизайн,
- інформаційну архітектуру,
- візуальний дизайн,
- юзабіліті,
- взаємодія між людиною і продуктом.

Іншими словами, UX дизайн – це процес створення корисних, простих і приємних у використанні продуктів (цифрових або фізичних). Основна ціль UX дизайну – це покращений досвід взаємодії з продуктом таким чином, щоб користувачі знаходили в ньому додаткову цінність.

Для розробки дизайну, що враховує потреби кінцевого споживача необхідно провести UX дослідження та на їх основі розробити UI прототип (прототип користувацького інтерфейсу), а також User Story (детальну схему того, які кроки повинен зробити користувач для виконання кожного із завдань системи). Чим менше кроків для досягнення мети робить кінцевий споживач – тим кращим і більш збалансованим вважається інтерфейс.

Оскільки одна із цілей даної дисертації є підвищення зручності користувацького інтерфейсу і набуття таким чином конкурентної переваги, приділимо особливу увагу UX дослідженням та розробці прототипу дизайну інтерфейсу системи управління розумним будинком.

UX дослідження складається з двох частин: збір даних та їх узагальнення. На старті розробки проекту інструментарій дослідження обмежується анкетуванням кінцевих споживачів. В майбутньому на етапі тестування альфа версії програми необхідно передбачити також наступні кількісні та якісні дослідження [29]:

- визначення відсотку користувачів, які одразу знаходять призив до дії (бачать кнопки на які потрібно натиснути для початку роботи с системою/ створення сценарію/ виклику команди: увімкнути світло, тощо);
- замір середнього часу, який витрачають користувачі на кожен із сценаріїв User Story (наприклад скыльки часу потрібно корисувачу щоб увімкнути чайник, змінити смценарій, створити свій сценарій);
- проведення опитування щодо зручності дизайну: чи достатнього розміру кнопки і чи немає у користувачів проблеми щоб натискати їх на мобільному пристрої, си легко може користувач відрізнити основні кнопки від додаткових, активні від неактивних, чи легко в меню знайти потрібний розділ та ін.

Оскільки подібні дослідження можуть бути проведені лише після створення альфа версії продукту, в даній роботі вони будуть закладені у календарний план та кошторис.

Для розробки прототипу проведемо анкетування користувачів. Анкета складається з наступних блоків [29]:

- Дані користувача (вік, стать, освіта, рівень доходу). Ці дані важливі для подальшого узагальнення відповідей, а також для прийняття спірних рішень. Наприклад якщо 30% респондентів не хочуть створювати власні сценарії для управління системою і майже всі вони – жінки похилого віку, слід прийняти рішення наскільки вони відповідають цільовій аудиторії продукту і чи є їх думка вирішальною.
- Тестування гіпотези. Гіпотеза – це базова User Story, в яку закладені основні дії користувача. Наприклад: увімкнення/вимкнення будь-якого

приладу, спостереження за будь-яким процесом (у нашому випадку чи увімкнена сигналізація в будинку на даний моммент), створення сценарію «Ранок» із доступними пристроями «кавоварка», освітлення, музична колонка. Користувачу пропонується по черзі реалізувати кожен із сценаріїв на клікабельному прототипі. Інтерв'юер спостерігає за діями користувача та відповідає на відкриті питання: наскільки зручно було користувачу виконати кожен із сценаріїв та скільки приблизно часу це зайняло.

- Відкритий відгук споживача: тут він може сказати що йому сподобалось, а що ні.

Зразок анкети наведений у Додатку А до даної дисертації. Розглянемо гіпотетичні сценарії та гіпотези до них.

Ви вийшли з дому та напівдорозі засумнівались чи увімкнута система охорони. Перевірте чи будинок поставлений на сигналізацію та за потреби увімкніть її.

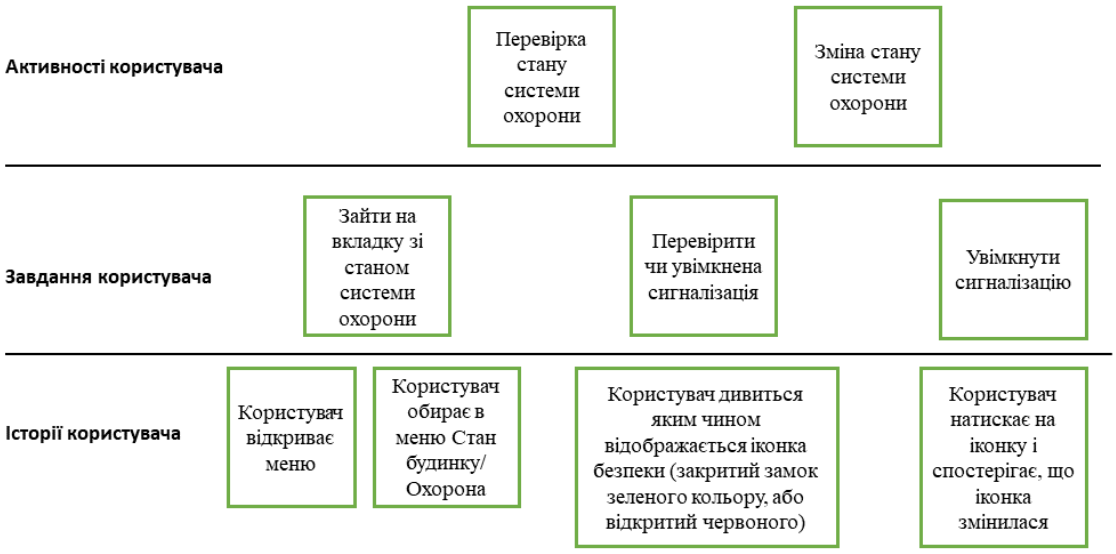


Рис. 3.1. Гіпотеза до сценарію перевірки стану системи охорони та увімкнення сигналізації

Закінчилось літо і вам стало важко прокидатись вранці. Створіть собі сценарій «приємний ранок», за яким ви будете прокидатись під легку музику та запах свіжої кави.

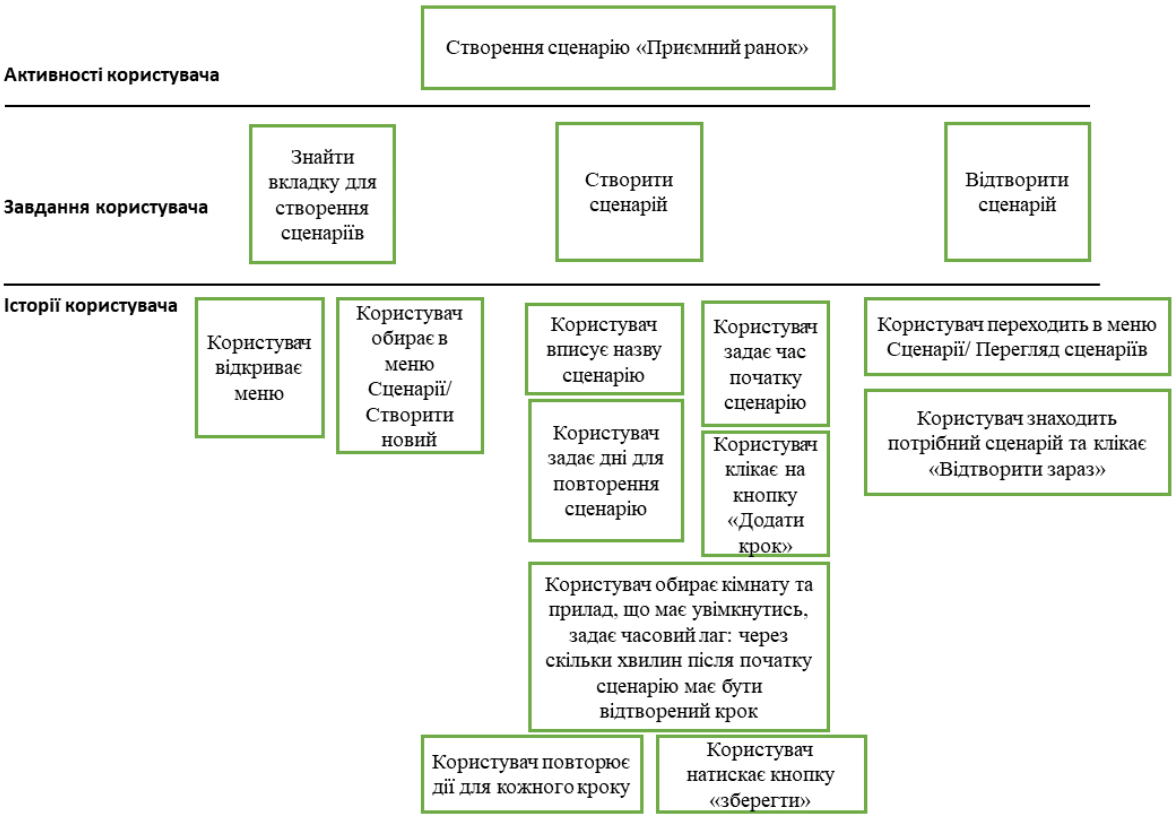


Рис. 3.2. Гіпотеза до сценарію створення унікального користувацького сценарію «Приємний ранок»

На вулиці дощ і ви поспішаєте додому із мрією про гарячий чай. Налаштуйте систему таким чином щоб чайник закипів як раз до вашого прибуття, яке планується через 15 хвилин.

Рис. 3.3. Гіпотеза до сценарію увімкнення чайника

Базовий прототип інтерфейсу для проведення UX дослідження наведений у додатку Б.

До початку дослідження виділимо цільову аудиторію. Це дозволить зробити більш релевантну вибірку. Оскільки метою даної роботи є також створення системи дешевшої ніж у більшості конкурентів, її потенційними користувачами можуть бути люди із доходом вище середнього та високим. За особистими характеристиками потенційні споживачі маєть бути схильними пробувати нове, високо цінити комфорт та безпеку. Щодо віку обмежень майже немає, респондент має бути хіба що повнолітнім. Важливо, щоб додатком було зручно користуватись як молоді, яка звикла до різноманітних додатків і гаджетів та дуже швидко сприймає нову інформацію, так і людям похилого віку, яким часто важко розібратись в інтерфейсах веб додатків.

Попереднє дослідження було проведено на вибірці з 35 респондентів, з яких:

- 10 респондентів віком 20-30 років із доходом вище середнього, вищою або неповною вищою освітою (респондент навчається у ВУЗі) та перспективною спеціальністю;
- 15 респондентів віком 30-50 років із вищою освітою та високим доходом;

- 10 респондентів віком 50-70 років, щомісячні витрати яких вище середнього та включають в сбе предмети розкоші.

Зведені результати дослідження наведені у таблиці 3.1.

Таблиця 3.1.

Зведені результати UX дослідження гіпотези на основі трьох базових сценаріїв

Сценарій	Середня оцінка (від 0 до 1)	Середній час виконання
Ви вийшли з дому та напівдорозі засумнівались чи увімкнута система охорони. Перевірте чи будинок поставлений на сигналізацію та за потреби увімкніть її.	0,97	14 секунд
Закінчилось літо і вам стало важко прокидатись вранці. Створіть собі сценарій «Приємний ранок», за яким ви будете прокидатись під легку музику та запах свіжої кави.	0,86	2 хвилини 10 секунд
На вулиці дощ і ви поспішаєте додому із мрією про гарячий чай. Налаштуйте систему таким чином щоб чайник закипів як раз до вашого прибуття, яке планується через 15 хвилин.	1	18 секунд

Проаналізуймо результати дослідження:

1. 2 респонденти не зрозуміли чи був реалізован сценарій увімкнення охорони. За їх словами зміни іконки не достатньо щоб бути впевненими у тому що сигналізація була увімкнена;
2. Деякі респонденти не змогли реалізувати сценарій «Приємний ранок» оскільки не здогадались про необхідність зайти на сторінку з переліком

сценаріїв та натиснути на кнопку «Відтворити». Незручність також викликала неможливість легко замінити кроки місцями. Побажанням 17 з 35 респондентів є можливість вибору музики у додатку, або увімкнення обраного треку з телефону;

3. Усі респонденти реалізували сценарій увімкнення чайника без перешкод.

На основі результатів дослідження можна зробити наступні висновки:

1. При розробці дизайну прототипу необхідно врахувати текстові підказки, що супроводжують дії користувача. Наприклад, коли користувач вмикає/ вимикає систему охорони, має не тільки змінюватись іконка, а також з'являтись вспливаюча підказка «Система охорони увімкнута» або «Система охорони вимкнена»
2. Для того, щоб користувачу було інтуітивно легше розуміти систему, має бути запроваджено єдине правило для всіх дій: вимкнений прилад/ сценарій має візуалізуватись червоною іконкою/кнопкою, а увімкнений прилад/ сценарій – зеленою іконкою/кнопкою. Написи на кнопках мають змінюватись відповідно.
3. Власний сценарій після збереження має одразу бути активним. Натискання кнопки відтворити не має бути обов'язковою умовою. Проте слід передбачити можливість видалення сценарію та його тимчасового вимкнення.
4. Зробити вибір часового лагу не обов'язковим
5. В одну з наступних ітерацій розробки системи включити реалізацію вибору музики з власного телефону для роботи із приладом «колонка» та можливість змінювати місцями етапи сценарію через «drag and drop».

Перед створенням UX-прототипу системи сформулюємо які User Story мають бути реалізовані в першому прототипі системи:

1. Авторизація.
2. Увімкнення єдиничного приладу одразу, або через заданий час.

3. Увімкнення єдиничного приладу в заданому інтервалі (наприклад щодня о 7 ранку).
4. Модуль «Безпека»: Відстежування стану охорони будинку (він або поставлений на охорону, або ні).
5. Модуль «Безпека»: дистанційне увімкнення/ вимкнення системи охорони.
6. Відстеження стану будинку: рівня вологості, температури.
7. Створення сценаріїв користувачем.
8. Можливість зробити сценарій активним або неактивним.
9. Видалення сценаріїв.

В проектному плані закладемо також User Story для майбутніх ітерацій:

1. Додавання та видалення додаткових приладів.
2. Редагування сценаріїв.
3. Можливість самостійно задати та змінити норму вологості та температури.
4. Використання музики з телефону.
5. Використання сценаріїв «з коробки», що створені розробниками та не можуть бути змінені. Наприклад, сценарій «Відпустка», за яким вимикаються усі прилади, окрім системи охорони, а освітлення налаштовується таким чином, щоб щодня у вечірній час в рандомній кімнаті вмикалось світло на 3 години. Таким чином створюється ефект присутності господарів.
6. Відправка сценарію на чужий прилад із встановленою аналогічною системою (тобто можливість поділитися своїм сценарієм із другом, який використовує таку ж систему).
7. Створення web-адмінки для тих користувачів, які хочуть максимально кастомізувати систему.
8. Зміна даних користувача для авторицації.
9. Відновлення пароля.

UX-прототип та User Story до нього представлені у додатку В.

3.1. Розробка клієнта для системи «Розумний дім»

Для розробки клієнта системи управління розумним будинком використовується компонентний підхід та бібліотеки React JS та Redux. Додаток поділений на компоненти, які можна перевикористовувати та сторінки, що створені із використанням цих компонентів.

Проект реалізован як SPA (single page application) таким чином, щоб при переході на іншу сторінку система не перезавантажувалась, а оновлювалась тільки частина компонентів, для відображення нової сторінки. Для цього реалізован роутінг.

Компоненти відображають тільки View із паттерну MVC. Контроллером служить сервіс Redux, що відповідальний за коммунікацію із сервером та передачу стану додатку компонентам. В Redux виділені наступні модулі (редьюсери):

1. Device – відповідальний за коммунікацію з сервером та отримання даних з колекції devices БД, а також за зберігання, оновлення та передачу стану (дані, отримані від сервера) до компонентів. Зворотньо при зміні стану приладу (увімкнення або вимкнення) користувачем, компонент передає оновлені дані редьюсеру, а той в свою чергу оновлює стан додатку та пересилає оновлені дані на сервер.
2. Modal – редьюсер, що відповідає за стан модального вікна (компонент, що перевикористовується на різних сторінках додатку та має власний стан).
3. Scene - відповідальний за коммунікацію з сервером та отримання даних з колекції scenes БД, а також за зберігання, оновлення та передачу стану (дані, отримані від сервера) до компонентів. При створенні нового сценарію, зміні його стану, або видаленні компонент передає оновлені дані редьюсеру, а той в свою чергу оновлює стан додатку та пересилає оновлені дані на сервер.
4. Rooms – редьюсер, що відповідає за отримання інформації від серверу про кімнати будинку. Оскільки у демо версії системи управління не передбачена можливість повної конфігурації будинку користувачем, тут немає операцій

додавання, видалення або зміни кімнат. Редьюсер лише отримує дані з колекції rooms та передає їх до компонентів.

5. User – відповідальний за збереження інформації про авторизованого користувача та передачі цієї інформації до компоненту авторизації, а також за комунікацію із сервером (отримання даних про користувача за вказаним імейлом та паролем, надсилання даних нового користувача при реєстрації).

Розглянемо алгоритми, що були реалізовані на клієнтському рівні згідно гіпотезам дій користувача, розробленим на попередньому етапі. Лістинг програми наведено у додатку Г.

3.1.1. Алгоритм авторизації

1. Користувач заходить в додаток та потрапляє на сторінку авторизації (комонент Authorization). Компонент отримує від user.reducer state користувача. Початковий стан – null (авторизований користувач відсутній).
2. Користувач вписує імейл в поле «email».
3. Зміна значення поля викликає обробник, який валідує імейл за правилами: значення має включати символ «@», до та після символу «@» має бути мінімум один символ, значення має включати точку після символу «@», але не раніше ніж через один символ після «@». Після точки має бути хоча б один символ.
4. Якщо значення імейлу не валідне – з'являється підказка про невідповідність імейлу. В іншому випадку підказка не з’являється/зникає після введення валідного значення.
5. Користувач вписує пароль в поле «пароль».
6. Якщо поле пароль заповнене та значення поля імейл валідне – кнопка «Увійти» активна, в іншому випадку – вона не активна.

7. Після натискання кнопки, компонент викликає метод user.reducer, який в свою чергу через Api сервіс робить GET запит на сервер.
8. Сервер повертає або об’єкт користувача (імейл та id без паролю), або об’єкт помилки.
9. Якщо сервер передав дані користувача, редьюсер оновлює стейт, що також оновлюється для компонента. Якщо в стейті компонента існує користувач, компонент перестає відображатись і користувач потрапляє на стартову сторінку додатку.
10. Якщо сервер повернув об’єкт помилки, редьюсер не оновлює стан. Всередині компонента відображається помилка авторизації. Користувач знову має ввести дані, або зареєструватись.

3.1.2. Алгоритм реєстрації

1. Користувач заходить в додаток та потрапляє на сторінку авторизації (комонент Authorization). Компонент отримує від user.reducer state користувача. Початковий стан – null (авторизований користувач відсутній).
2. Якщо користувач не був зареєстрований в системі, він заповнює форму реєстрації. Користувач вписує імейл в поле email.
3. Зміна значення поля викликає обробник, який валідує імейл за правилами: значення має включати символ «@», до та після символу «@» має бути мінімум один символ, значення має включати точку після символу «@», але не раніше ніж через один символ після «@». Після точки має бути хоча б один символ.
4. Якщо значення імейлу не валідне – з'являється підказка про невідповідність імейлу. В іншому випадку підказка не з’являється/зникає після введення валідного значення.
5. Користувач вписує пароль в поле «пароль».

6. Користувач вписує пароль повторно в поле «повторіть пароль». Якщо паролі не співпадають – з’являється відповідне сповіщення.
7. Якщо імейл валідний, поля з паролем заповнені та паролі співпадають, кнопка «Зареєструватись» активна. Інакше – ні.
8. Користувач натискає на кнопку «Зареєструватись». Компонент викликає метод редьюсера, що робить через Api сервіс POST запит на сервер.
9. Сервер повертає об’єкт створеного користувача (імейл та id без паролю), або об’єкт помилки.
10. Якщо сервер передав дані користувача, редьюсер оновлює стейт, що також оновлюється для компонента. Якщо в стейті компонента існує користувач, компонент перестає відображатись і користувач потрапляє на стартову сторінку додатку.
11. Якщо сервер повернув об’єкт помилки, редьюсер не оновлює стан. Всередині компонента відображається помилка реєстрації. Користувач знову має ввести дані.

3.1.3. Алгоритм відображення стану приладу

1. Користувач переходить на сторінку «Охорона», «Управління приладами» або «Клімат контроль». Компонент перед рендерингом викликає метод device.reducer та room.reducer, що через Api сервіс робить GET запит на сервер, оновлює стейт отриманою інформацією та передає стейт компоненту.
2. Компонент отримує масив об’єктів із описом кімнат та масив об’єктів із описом приладів. У кожного прилада існує поле “room”, за допомогою якого для кожної кімнати проводиться вибірка доступних приладів для відображення.
3. Згруповані дані відображаються у вигляді «аккордеону»: при натисканні на назву кімнати відображається список приладів.

4. Якщо статус приладу true – прилад увімкнений, інакше – вимкнений. В залежності від статусу відображається відповідний текст, колір та назва кнопки. Кнопка – це окремий копмонент, CSS клас, стан та текст якої передається батьківським компонентом.

3.1.4. Алгоритм керування станом приладу

1. Користувач натискає на кнопку. Компонент викликає метод device.reducer та передає в нього об'єкт приладу із новим станом.
2. Відповідний метод device.reducer робить через Api сервіс PUT запит на сервер, отримує від сервера позитивний статус та оновлює стейт.
3. Після оновлення стейту в редьюсері, компонент отримує також оновлений стейт. При зміні статусу приладу змінюється CSS клас та відповідно відображення кнопки, а також тексту, що супроводжує відображення стану приладу.

3.1.5. Алгоритм створення сценарію

1. Користувач заходить на сторінку для створення сценарію. Компонент містить в собі наступні компоненти:
2. DateTimePicker (компонент із модифікацією, оснований на React бібліотеці), який відображує календар для вибору дати або годинник для вибору часу початку роботи сценарію,
3. форма для додавання нової сцени,
4. модальне вікно,
5. кнопка.
6. До початку рендерингу батьківський компонент викликає метод device.reducer та room.reducer, що через Api сервіс робить GET запит на сервер, оновлює стейт отриманою інформацією та передає стейт компоненту.

7. Компонент отримує масив об’єктів із описом кімнат та масив об’єктів із описом приладів. У кожного прилада існує поле “room”, за допомогою якого для кожної кімнати проводиться вибірка доступних приладів для відображення. Згруповані дані передаються компоненту форми, що в свою чергу передається компоненту модального вікна як контент.
8. Батьківський компонент містить власний стейт (стан) – об'єкт із полями, які мають бути збережені та передані на сервер. Це – масив сцен, дата початку роботи сценарію, час, чи слід повторювати сценарій щотижня/щодня (Boolean), назва, статус (за замовченням активний) та id, що генерується.
9. Користувач обирає час та дату. Дані від компонентів DateTimePicker передаються батьківському компоненту та зберігаються у стейті.
10. Користувач вписує назву сценарію. При зміні значення в інпуті оновлюється відповідне поле у стейті.
11. Користувач натискає на кнопку «Додати сцену». По кліку на кнопку викликаєтьс метод modal.reducer, що змінює стан модального вікна із невидимого на видимий. Відкривається модальне вікно, в яке ми раніше передали форму додавання нової сцени.
12. Форма відображує список кімнат та список приладів у вигляді select. Список приладів залежний від того, яку кімнату обрав користувач. При вибирі нової кімнати він оновиться. Це регулюється стейтом форми.
13. Користувач обирає кімнату, прилад та за бажанням вводить час затримки в необов’язкове поле. Зміна значень інпутів оновлює стейт форми.
14. Користувач натискає кнопку «Додати». Викликається метод форми, що закриває модальне вікно через modal.reducer, а також додає в масив сцен батьківського компонента нову сцену.
15. Користувач повторює кроки 7-10. Щоразу до стейту батьківського компоненту додається нова сцена. Вісі сцени, що існують в стейті

батьківського компоненту відображаються у вигляді текстового сценарію з кнопкою «видалити».

16. Користувач натискає на кнопку «видалити». Це викликає метод батьківського компоненту, який видяляє сцену із списка сцен в стейті. Автоматично оновлюється відображення сцен.
17. Користувач змінює положення кнопки «Повторювати щодня» або «Повторювати щотижня». Це викликає метод компоненту, що змінює відповідне поле в стейті. Зміна стейту автоматично впливає на зміну відображення. До кнопки додається CSS клас, що робить її зеленою.
18. Якщо користувач заповнив усі необхідні дані (в стейті компоненту не порожнє поле дати та часу початку роботи сценарія, назви та масив сцен не порожній), кнопка «Зберегти сценарій» активна. Інакше – не активна.
19. Користувач натискає кнопку «Зберегти сценарій». Це викликає метод scene.reducer, в який передається стейт компоненту (об'єкт сценарію із згенерованим id).
20. Метод scene.reducer через Api сервіс робить POST запит на сервер. Сервер передає позитивний статус, стейт редьюсера оновлюється.
21. Стейт компоненти очищується і таким чином відображення повертається в початковий стан. Користувач може додати ще один сценарій.

3.1.6. Алгоритм редагування сценарію

1. Користувач переходить на сторінку перегляду сценаріїв. Компонент перед рендерингом викликає метод scene.reducer, що через Api сервіс робить GET запит на сервер, оновлює стейт отриманою інформацією та передає стейт компоненту.
2. Компонент отримує масив об'єктів сценаріїв, кожен з яких містить масив сцен.
3. Дані відображаються у вигляді «аккордеону»: при натисканні на назву сценарію відображається його деталі.

4. Деталі сценарію відображаються у текстовому вигляді із двома керуючими кнопками: «вимкнути/увімкнути» сценарій та «видалити сценарій».
5. Користувач змінює стан кнопки «вимкнути/увімкнути» сценарій. Це викликає метод scene.reducer, що що через Api сервіс робить PUT запит і передає на сервер сценарій з оновленим статусом. Сервер повертає оновлений сценарій. Редьюсер оновлює стейт.
6. Компонент реагує на оновлення стейту редьюсера та змінює відображення кнопки «вимкнути/увімкнути» сценарій.

3.1.7. Алгоритм видалення сценарію

1. Користувач натискає на кнопку «видалити сценарій». Це викликає метод scene.reducer, що що через Api сервіс робить DELETE запит і передає на сервер id сценарію. Сервер повертає позитивний статус. Редьюсер оновлює стейт.
2. Компонент реагує на оновлення стейту редьюсера та змінює відображення: рендерить оновлений список сценаріїв.

3.2. Розробка сервера для системи «Розумний дім» та інтеграція із Mongo DB

Node JS сервер був розроблений із використанням бібліотек Express та Mongoose. Остання дає можливість створювати зручні моделі, схема яких наслідується від класу Schema бібліотеки Mongoose. При створенні моделі містять в собі методи прототипу для роботи с Mongo DB. Наприклад model.find() робить вібірку всієї таблиці з БД, model.findById(id) – вибірку лише того об’єкту таблиці, _id якого збігається з переданим в методі, тощо.

MongoDB - документно-орієнтованою NoSQL база даних. Вона складається з баз даних, які зберігають в собі колекції. Кожна колекція складається з документів, які в свою чергу складаються з полів (пари ключ-значення). Тобто колекція – це не звична SQL таблиця, а JSON-подібний об’єкт. Колекції не пов'язані між собою

ключами. Організація БД «smart_home» наведено на рисунку 3.4. Для проекту прототипу системи управління розумним будинком була створена БД у хмарному кластері.

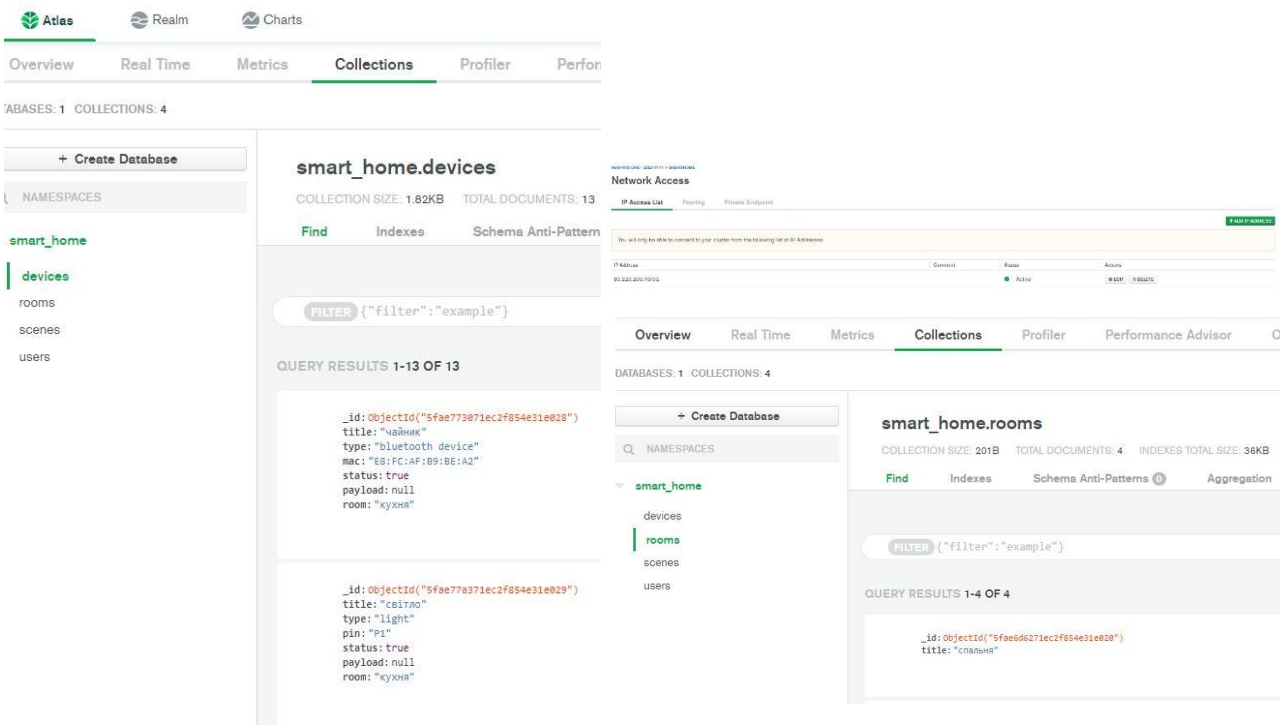


Рис. 3.4. Схема бази даних «smart_home»

Актуальний стан БД можна також відстежувати на сервері, що використовує порт 3003. Дані доступні за шляхами (роутами) прописаними в конфінурації сервера. Розглянемо алгоритми, що були реалізовані на рівні сервера для забезпечення роботи системи. Лістинг програми наведено у додатку Г.

3.2.1. Алгоритм реєстрації

1. Сервер отримує від клієнта дані нового користувача (імейл, пароль, id).
2. Сервер хешує пароль за допомогою бібліотеки argon2.
3. Сервер створює новий об’єкт користувача як екземпляр моделі User та передає в конструктор дані із хешованим паролем.
4. Сервер додає в БД нового користувача. Таким чином справжній пароль не зберігається ані на клієнті, ані на сервері, ані в базі даних.

5. Сервер генерує JSON Web Token (тут і далі jwt), в якому закодовані дані користувача (його id) з використанням секретного ключа [30].
6. Сервер відправляє клієнту дані нового користувача (імейл та jwt). Клієнт зберігає jwt та надалі підставлятиме його в усі запити до сервера в хедер Authorization [31].

3.2.2. Алгоритм авторизації

1. Сервер отримує імейл та пароль користувача.
2. Сервер робить запит до БД та отримує список користувачів.
3. Сервер шукає у списку користувачів такого, в якого імейл співпадає з імейлом, що прийшов у запиті.
4. Якщо користувач існує, сервер декодує його пароль та порівнює із паролем, що прийшов у запиті за допомогою бібліотеки argon2.
5. Якщо користувач із заданим імейлом не знайдений, сервер відправляє статус 404.
6. Якщо перевірка пароля була успішною, сервер генерує jwt та відправляє його клієнту [30].
7. В іншому випадку сервер відправляє статус 404. Клієнт зберігає jwt (або оновлює існуючий) та надалі підставлятиме його в усі запити до сервера в хедер Authorization.

3.2.3. Алгоритм увімкнення та вимкнення приладів

1. Сервер отримує від клієнта запит із об’єктом прилада.
2. Сервер перевіряє хедер Authorization. Якщо jwt не був переданий, або jwt не відповідає секретному ключу, сервер розриває з’єднання та відправляє відповідну помилку клієнту [30, 31].
3. Якщо jwt існує і він коректний, сервер робить запит до БД в колекцію devices за переданим йому id.
4. Якщо прилада не існує, сервер відправляє клієнту статус 404.

5. Якщо прилад існує, сервер передає дані оновленого приладу до DeviceController, що відповідає за програмування фізичних приладів, та викликає його метод switchDevice().
6. Сервер оновлює відповідний прилад у БД.
7. Сервер відправляє клієнту об’єкт оновленого приладу.

3.2.4. Алгоритм додавання сценарію

1. Сервер отримує від клієнта запит із об’єктом сценарію.
2. Сервер перевіряє хедер Authorization. Якщо jwt не був переданий, або jwt не відповідає секретному ключу, сервер розриває з’єднання та відправляє відповідну помилку клієнту [30, 31].
3. Якщо jwt існує і він коректний, сервер створює нову модель сценарію (екземпляр класу Scene).
4. Сервер додає новий сценарій в колекцію scenes БД.
5. Сервер відправляє статус 200 клієнту.

3.2.5. Алгоритм редагування сценарію

1. Сервер отримує від клієнта запит із об’єктом сценарію.
2. Сервер перевіряє хедер Authorization. Якщо jwt не був переданий, або jwt не відповідає секретному ключу, сервер розриває з’єднання та відправляє відповідну помилку клієнту [30, 31].
3. Якщо jwt існує і він коректний, сервер створює новий сценарій як екземпляр моделі Scene.
4. Сервер робить запит на сервер за id переданого сценарію.
5. Якщо колекція scenes не містить сценарій з таким id, сервер відправляє клієнту статус 404.
6. Сервер оновлює сценарій в колекції scenes БД.
7. Сервер відправляє клієнту оновлений сценарій.

3.2.6. Алгоритм видалення сценарію

1. Сервер отримує від клієнта запит із об’єктом сценарію.
2. Сервер перевіряє хедер Authorization. Якщо jwt не був переданий, або jwt не відповідає секретному ключу, сервер розриває з’єднання та відправляє відповідну помилку клієнту [30, 31].
3. Якщо jwt існує і він коректний, сервер створює новий сценарій як екземпляр моделі Scene.
4. Сервер робить запит на сервер за id переданого сценарію.
5. Якщо колекція scenes не містить сценарій з таким id, сервер відправляє клієнту статус 404.
6. Якщо сценарій був знайдений в колекції, сервер видаляє сценарій з колекції scenes БД.
7. Сервер відправляє клієнту статус 200.

3.3. Програмування контролерів

Для програмування фізичних приладів було використано бібліотеку @amperka, розробника Iskra JS. Були використані модулі «@amperka/led» [32] для програмування лед світильників, «@amperka/pid» [33] для керування метео станцією за допомогою ПІД регулятора, «@amperka/meteo-sensor» для зчитування показників датчиків термогідрометра, «@amperka/power-control» [34] для вмикання/вимикання простих приладів (в даному випадку сигналізації).

Пропорційно-інтегрально-диференціюючий (ПІД) регулятор потрібен для підтримки однієї характеристики приладу на потрібному рівні за допомогою зміни іншої характеристики. Наприклад він може витримувати рівень вологості повітря задану в двох граничних межах, регулюючи потужність зволожувача повітря [33].

Для управління приладами необхідно передати відповідним класам бібліотеки пін, до якого в платі Iskra JS був підключений прилад та за необхідності додаткові дані (наприклад граничні показники для ПІД). Після цього можна

викликати потрібний метод для зчитування інформації про стан приладу/ його увімкнення або вимкнення [36].

Bluetooth прилади під'єднані до Espruino Puck та управляються за допомогою методів глобального класу NRF. З'єднання з приладом відбувається за допомогою виклику методу NRF.connect(), в який передається mac адреса приладу [35].

Прилади, що входять в систему розумного дому, розділені на наступні класи:

1. Led: приймає в конструкторі id лед приладу, за цим ідентифікатором знаходить пін, до якого підключений прилад та встановлює з'єднання. Всередині класу є два методи: switchOn (вмикає прилад через заданий час, за замовченням 0 мілісекунд), та switchOff (відповідно вимикає прилад через заданий час, за замовченням 0 мілісекунд).
2. BluetoothDevice: приймає в конструкторі mac адресу приладу (зберігається в БД), встановлює з'єднання (NRF.connect()). Всередині класу є два методи: switchOn (вмикає прилад через заданий час, за замовченням 0 мілісекунд), та switchOff (відповідно вимикає прилад через заданий час, за замовченням 0 мілісекунд).
3. MeteoControl (для управління ПІД присторями): приймає в конструкторі тип приладу регулювання (зволожувач або кондиціонер), пін датчика та пін приладу регулювання. В залежності від типу приладу регулювання, клас встановлює параметри регулювання (граничні та нормальний показник вологості або температури) та створює на основі цих даних pid об'єкт бібліотеки «@amperka/pid». В середині класу є єдиний метод act, який в свою чергу викликає метод pid.run() та передає дані сенсора та приладу регулювання. Метод передбачає встановлення з'єднання щохвилини, тобто кожні 60 секунд ПІД пристрій порівнює реальні показники сенсора із граничними та за необхідності збільшує або зменшує потужність приладу регулювання.
4. MeteoSensor: приймає в конструкторі id та pin, встановлює з'єднання із приладом, що підключений до даного піну. Метод класу act() кожні

тридцять секунд зчитує дані приладу, звертається до серверу із GET запитом приладу за id, оновлює об'єкт, що повернув сервер отриманими з приладу даними та робить PUT запит на сервер, передаючи оновлений об'єкт приладу. Таким чином стан рівня температури та вологості в БД оновлюються приблизно кожні тридцять секунд.

5. Security: встановлює з'єднання із пристроєм за піном P0 та містить в собі два методи: switchOn (вмикає прилад через заданий час, за замовченням 0 мілісекунд), та switchOff (відповідно вимикає прилад через заданий час, за замовченням 0 мілісекунд).
6. DeviceController: проміжний клас, який приймає команди від сервера. Він в конструкторі отримує об'єкт приладу, та в залежності від типу приладу створює екземпляр одного із перелічених вище класів. Метод switchDevice() викликає метод екземпляру: switchOn() якщо статус об'єкту «true» та switchOff() в іншому випадку.
7. SceneControl: проміжний клас, який приймає команди від сервера. Він приймає в конструкторі сценарій, якщо він активний, вираховує затримку (час у мілісекундах через який має бути відтворений сценарій). Основний метод класу – метод play(). Якщо сцена активна, всередині методу для кожної сцени сценарію створюється екземпляр класу DeviceController та через задану затримку викликається метод switchDevice() контроллеру кожного приладу сценарію.

Оскільки для створення прототипу не передбачена закупівля приладів та проведення інженерних робіт з їх монтажу, в проекті були реалізовані алгоритми з використанням фейкових даних.

Розглянемо алгоритми, що були реалізовані на рівні програмування фізичних пристроїв для забезпечення роботи системи. Лістинг програми наведено у додатку Г.

3.3.1. Алгоритм регулювання температури та рівня вологості

1. Для кожного термогідрометра створюється екземпляр класу MeteoSensor та викликається його метод act().
2. Приблизно щотридцять секунд кожен із об'єктів зчитує дані свого приладу.
3. Об'єкт робить запит до сервера для оновлення відповідного документу колекції devices даними, отриманими з приладу.
4. Для кожної пари термогідрометр-прилад керування створюється екземпляр класу MeteoControl.
5. Приблизно щохвилини кожен із створених об'єктів встановлює з'єднання. ПІД пристрій порівнює реальні показники сенсора із граничними та за необхідності збільшує або зменшує потужність приладу регулювання.

3.3.2. Алгоритм увімкнення та вимкнення приладів

1. На сервері для приладу, який потрібно увімкнути або вимкнути створюється екземпляр класу DeviceController.
2. Завдяки логіці класу одразу створюється інкапсульований екземпляр класу Led, BluetoothDevice або Security та встановлюється з'єднання.
3. На сервері викликається його метод DeviceController.switchDevice(), що в свою чергу аналізує статус об'єкту та викликає метод switchOn() або switchOff() інкапсульованого екземпляра.
4. Відпрацьовує логіка даного приладу, наведена в описі класів. Прилад вмикається або вимикається.

3.3.3. Алгоритм виконання сценаріїв

1. Щохвилини з рівня програми управління фізичними приладами йде GET запит на сервер та отримує список актуальних сценаріїв.
2. Для кожного сценарію створюється екземпляр класу SceneControl та викликається метод об'єкту play().

3. Всередині методу калькулюється дата та час відтворення сценарію. Якщо дата старту сценарію в мілісекундах від 1970 року менша за поточну, до неї додається інтервал в мілісекундах до того часу, поки вона не буде більшою за поточну. Час затримки калькулюється як різниця між датою та часом відтворення сценарію у мілісекундах та поточною датою у мілісекундах.
4. Якщо сцена активна, для кожної сцени сценарію створюється екземпляр класу DeviceController та через задану затримку викликається метод switchDevice() контроллеру кожного приладу сценарію. В даному методі свою чергу викликається метод switchOn() інкапсульованого екземпляра.
5. В циклі відпрацьовує логіка кожного приладу, наведена в описі класів. Прилад вмикається через задану затримку.
6. Кроки 1-6 повторюються щохвилини.

Висновки до розділу 3

В цьому розділі описано проведення UX-дослідження, на основі якого був розроблений дизайн прототипу та сформульовані User Stories, що в свою чергу послужили базою для написання та програмної реалізації алгоритмів роботи системи.

Для програмування клієнтської частини були використаны бібліотеки React JS та Redux, застосований паттерн програмування flux та компонентний підхід. Сервер був запрограмований на Node.JS з використанням бібліотек бібліотек Express та Mongoose. База даних була створена у хмарному кластері Mongo DB. Також було розроблено та запрограмовано алгоритми управління фізичними приладами для плати Iskra.JS та мікроконтроллера Espruino Puck. Для цього були використані необхіді модулі з бібліотеки @amperka.

Реалізація системи управління розумним домом основана на делегуванні функціоналу незалежним модулям на кожному з трьох програмних рівнів (клієнтському, рівні сервера та рівні управління фізичними приладами). На клієнтському рівні це – React компоненти, що відповідають за відображення, та Redux редьюсери, що відповідають за спілкування з сервером. На рівні сервера – окламі моделі та роути для приладів, користувачів, кімнат та сценаріїв. На рівні програмування фізичних пристроїв – окремі класи для кожного з типів пристроїв, що інкапсулюються в класі контролеру, відповідальному за спілкування з сервером.

Мета захисту даних користувача при створенні багатокористувацького додатку була досягнута завдяки алгоритму реєстрації та авторизації, що включає в себе хешування паролів та використання JSON Web Token.

Завдяки проведенню UX-дослідження був реалізований зручний та інтуітивно зрозумілий для усіх основних категорій користувачів інтерфейс прототипу, що є однією із цілей даної дисертації.

РОЗДІЛ 4

РОЗРОБКА СТАРТАП-ПРОЕКТУ

Стартап — це тимчасова структура, яка займається пошуком рентабельної бізнес-моделі, що масштабується та відтворюється.

Основними складовими стартапу є [37]:

1. Ідея. Ідея – це початкова точка, з якої починається стартап. Вона має бути або зовсім свіжою і новою для обраного ринку, або включати в себе певні нововведення, що потенційно можуть стати конкурентною перевагою майбутнього продукту. Саме ідея перш за все має привабити потенційних інвесторів.
2. Команда. Команда стартапу зазвичай складається з кількох ентузіастів, які готові певний час працювати безкоштовно заради втілення ідеї. У кожного із членів команди має бути своя зона відповідальності, свій вклад в стартап і відповідно до вкладу – відсоток акцій.
3. Стартовий капітал. Зазвичай організатори стартапу намагаються залучити інвестиції для втілення своєї ідеї навіть на етапі розробки прототипу. Пізніше інвестори отримують свою частку дивідендів від продажу акцій стартапу. На ранніх етапах все ж можливі невеликі інвестиції з боку учасників стартапу, а не сторонніх інвесторів.

На етапі планування стартапу слід приділити увагу розробці інформаційної карти проекту, розподілу обов'язків між учасниками стартапу, визначення їх вкладу в проект, побудові морфологічної карти, побудувати бізнес модель, розробити ринкову стратегію проекту та маркетингову програму, проаналізувати ринкові можливості запуску стартап проекту, розробити виробничий та організаційний план. Усі ці кроки будуть реалізовані в підрозділах даного розділу.

4.1. **Розробка інформаційної карти проекту**

Інформаційна карта стартап проекту – це по суті дорожна карта, якою керуються учасники стартапу в подальшій роботі. Інформаційна карта проекту «Система "Розумний дім" на базі сучасних web-технологій» наведена в таблиці 4.1.

Таблиця 4.1.

Інформаційна карта проекту

Назва проекту	«Система "Розумний дім" на базі сучасних web-технологій»
Автори проекту	Горова М. А.
Анотація	Розробка системи управління розумним будинком, використовуючи можливості сучасних веб технологій. Система управління повинна мати привабливий та інтуітивно зрозумілий дизайн, бути легко масштабованою, відносно недорогою в реалізації, підтримувати різні типи приладів. Алгоритм має легко адаптуватись до задач нових клієнтів. Система має надавати користувачу можливість відчувати максимальну незалежність від виробника: не тільки самостійно управляти приладами в будинку, перевіряти стан будинку, але й легко створювати власні сценарії, додавати нових користувачів системи, використовувати її на будь якому мобільному девайсі або як через веб додаток.
Термін реалізації проекту	5 місяців

Продовження таблиці 4.1.

Необхідні ресурси	Інтелектуальні ресурси: • Команда (3 чол – генератор ідей, спеціаліст, дипломат) Фінансові ресурси: • Залучені спеціалісти: інженер – 1 чол на 5 місяців, JavaScript розробник – 2 чол на 5 місяців, QA – 1 чол на 5 місяців, UI/UX дизайнер – 1 чол на 1 місяць. • Технічне забезпечення: Ліцензія на програмний пакет Atlassian, мікроконтроллер Espruino Puck, Iskra JS, набір перехідників, набір модулів, зарядний пристрій та набір кабелів, тестовий прилад Bluetooth чайник Xiaomi, тестовий набір датчиків, тестовий прилад термогідрометр, тестовий прилад терморегулятор, тестовий прилад розумний зволожувач повітря. • Залучення сторонніх сервісів: проведення повторного UX дослідження, послуги рекламної агенції.
Опис проблеми, яку вирішує проект	Сучасні системи управління розумним домом складні та незручні для кінцевого споживача, часто не підтримують можливість використання кількома людьми з різних девайсів, не надають можливості реєстрації та встановлення додатку на інший девайс (наприклад у випадку, коли користувач загубив телефон). Інтерфейси системи управління не локалізовані для українського користувача.

Продовження таблиці 4.1.

Головні цілі та завдання проекту	Ціль: надати широкому колу українців доступну технологію «Розумний будинок». Завдання: • Забезпечення зручного та привабливого інтерфейсу українською мовою з можливісю локалізації. • Забезпечення мультиплатформенної та мультикористувацької системи. • Зниження собівартості системи та відповідно її вартості для споживача. • Забезпечення захисту особистих даних користувачів.
Очікувані результати	• Перехід більшості українських домогосподарств, що використовують окремі розумні прилади, на повноцінну систему «Розумний дім». • Залучення нових домогосподарств до використання системи. • Зацікавленість системою інвесторів з країн східної Європи, вихід на ринок регіону.

4.2. Визначення команди та розподіл задач

Засновнниками стартапу будуть 3 людини, ролі яких розподіляються наступним чином: генератор ідей (IT спеціаліст), спеціаліст (IT спеціаліст) та дипломат (юрист).

Розподілимо задачі та розробимо схему взаємодії проектної команди. В таблиці 4.2 наведено орієнтовний графік реалізації проекту.

Таблиця 4.2.

Орієнтовний графік реалізації проекту

Задача		Місяці				
		1	2	3	4	5
1	Розподіл обов'язків	1 тиждень				
2	Технічне дослідження	3 тижні				
3	Формування виробничої команди та видача ТЗ	2 тижні				
4	Розробка дизайну		2 тижні			
5	Розробка програмного забезпечення		4 місяці			
6	Розробка інженерної схеми та монтаж обладнання		1 місяць			
7	Проведення повторного UX дослідження та корегування дизайну інтерфейсу, User Stories				1 місяць	
8	Тестування системи			3 місяці		
9	Пошук інвесторів		4 місяці			
10	Аналітика			1 місяць		
11	Юридичне забезпечення		4 місяці			
12	Рекламна кампанія			3 місяці		

Розподілимо задачі між участниками стартапу. Задачі генератора ідей – розробка дизайну, проведення UX дослідження, участь в технічному дослідженні, тестування системи та проведення рекламної кампанії. В даному випадку генератор ідей виступає більше як product owner та бізнес аналітик, а також координує роботу рекламної агенції. Задачі спеціаліста – формування технічної команди, розробка програмного забезпечення та одночасно управління технічною командою. Тож в його обов'язки входить і технічне дослідження, і розробка, і тестування системи, і розробка інженерної схеми та монтаж тестового обладнання. Задачею дипломата є пошук інвесторів, аналітика, юридичне забезпечення, участь у формуванні рекламної стратегії та узгодження пропозицій рекламної агенції.

Нижче наведена схема взаємодії учасників стартапу.

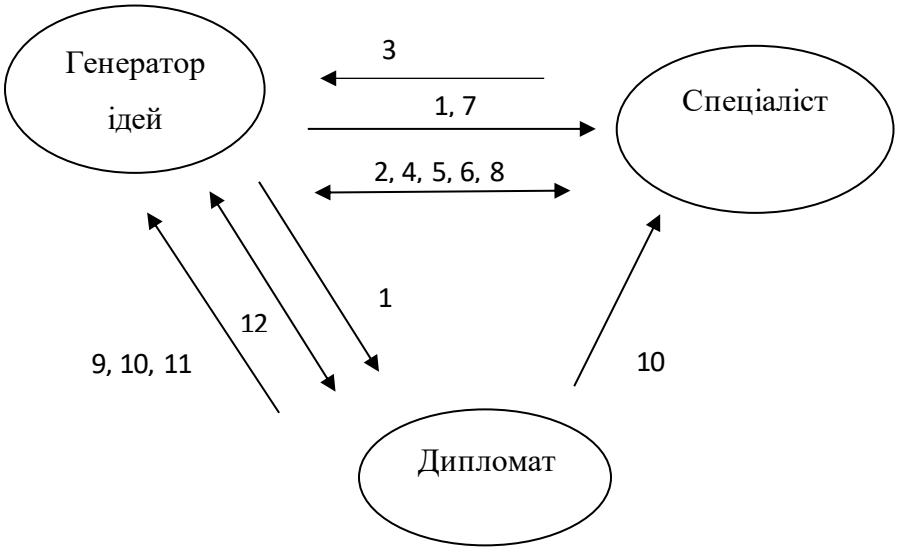


Рис. 4.1. Схема взаємодії учасників стартапу

Визначимо важливість владу кожного з учасників стартапу. Ключові фактори оцінки та їх вага наведені у таблиці 4.3.

Таблиця 4.3.

Відносна важливість факторів у реалізації проекту

Фактор	Вага
Ідея	4
Аналітика	5
Компетентність	6
Залученість і ризики	7
Обов'язки	6
Пошук інвесторів та покупців	8

Визначимо особистий внесок кожного партнера у створення стартап проекту.

Таблиця 4.4.

Оцінка особистого внеску учасників стартапу

Фактор	Генератор ідей	Спецаіліст	Дипломат
Ідея	6	4	0
Аналітика	2	0	8
Компетентність	5	7	2
Залученість і ризики	2	2	2
Обов'язки	8	8	6
Пошук інвесторів та покупців	2	0	10

Наведемо зведену таблицю оцінки внеску учасників та ваги кожного з факторів.

Таблиця 4.5.

Зведена таблиця оцінки особистого внеску учасників стартапу та ваги факторів

Фактор	Вага фактору	Генератор ідей	Спецаіліст	Дипломат
Ідея	4	6	4	0
Аналітика	5	2	0	8
Компетентність	6	5	7	2
Залученість і ризики	7	2	2	2
Обов'язки	6	8	8	6
Пошук інвесторів та покупців	8	2	0	10

Розрахуємо дольової участі кожного партнера у стартапі.

Таблиця 4.6.

Розподіл дольової участі партнерів у прибутку стартапу

Фактор	Вага	Генератор ідей	Спецаіліст	Дипломат	
Ідея	4	6x4=24	4x4=16	0	
Аналітика	5	2x5=10	0	8x5=40	
Компетентність	6	5x6=30	7x6=42	2x6=12	
Залученість і ризики	7	2x7=14	2x7=14	2x7=14	
Обов'язки	6	8x6=48	8x6=48	6x6=36	
Пошук інвесторів та покупців	8	2x8=16	0	10x8=80	
Разом		142	120	146	408
Дольовий відсоток у прибутку стартапу		35%	29%	36%	100%

4.3. Побудова морфологічної карти проекту

Першим кроком побудови морфологічної карти проекту є визначення продукту та його основних функцій. Продуктом є додаток, завдяки якому користувач управляє системою «Розумний дім». Визначимо його основні функції:

- Додаток має шифрувати дані користувачів для запобігання витоку даних.
- Додаток має бути візуально привабливим, зручним та зрозумілим для кінцевого користувача.
- Система управління має бути горизонтально масштабованою.
- Додаток має бути мультикористувацьким.
- Додаток повинен надавати користувачу можливість максимальної кастомізації за рахунок створення, редагування та видалення власних сценаріїв, реєстрації нових приладів або видалення старих, налаштування граничних кліматичних показників.
- Компанія має надавати проектні послуги щодо вибору приладів на основі побажань замовника, розробки інженерної схеми, монтажу та підключення.

Таблиця 4.7.

Морфологічна карта проекту

Основні параметри	Проміжні рішення			
	1	2	3	4
Шифрування даних	Симетричні алгоритми (AES, CAST, ГОСТ, Blowfish, DES)	Асиметричні алгоритми (El-Gamal, RSA)	Використання JSON web token та хешування даних	Інші
Дизайн	UI заснований UX досідженні	Без проведення досліджень		

Продовження таблиці 4.7.

Масштабованість платформи: вибір технології роботи з БД	Використання SQL БД на окремому сервері	Використання NoSQL БД на окремому сервері	Використання SQL БД у хмарі	Використання NoSQL БД у хмарі
Система управління: вибір технологічного стеку	.Net	Java SDK	Node JS	Інші
Система управління: вибір мікроконтролерів	Arduino	Raspberry	Espruino	Інші
Інтерфейс системи управління: вибір бібліотеки/ фреймворку	Angular	React	View	Інші
Адмінка для користувача для кастомізації	Надається	Не надається		
Надання проектних послуг	Надаються	Не надаються		
Надання послуг монтажу обладнання	Надаються	Не надаються		

Товар за задумом передбачає:

- Використання алгоритмічної системи шифрування даних.
- UI заснований на власних UX досідженнях.
- Використання NoSQL БД у хмарі (Mongo DB).
- Використання Node JS для розробки серверу.
- Використання мікроконтролерів Espruino.
- Використання React для розробки інтерфейсу.
- Користувач не може повністю кастомізувати систему (реєструвати або видаляти прилади, додавати кімнати, тощо), але може через основний інтерфейс реєструвати нових користувачів, створювати, редагувати та видаляти сценарії).
- Проектні послуги включають в себе розробку інженерної схеми, вибір приладів та підключення їх до системи. Також можлива розробка інженерної схеми та підключення існуючих приладів та пост обслуговування: підключення нових приладів в існуючу систему.

Товар у реальному виконанні відрізняється:

- Вибором JSON web token та хешування даних.
- Використання мікроконтролерів різних виробників, створених на ядрі Espruino.

Товар з підкріпленням:

- Вдосконалення мікроконтролеру для розширення можливостей системи.
- Надання адмінки для повної кастомізації системи користувачем.
- Кастомізація інтерфейсу за побажанням клієнта (в рамках надання проектних послуг).
- Надання повного спектру інженерних та монтажних послуг, в тому числі співробітництво із студіями інтер'єрного дизайну при розробці проекту будинку/квартири.

4.4. Побудова бізнес моделі стартапу та розробка ринкової стратегії проекту

<table>
<tr><td colspan="2">Фора:
Зрозумілий локалізований інтерфейс, невисока ціна, більша варіативність функцій</td><td colspan="2">Рішення:
Розробка додатку із простим інтерфейсом українською мовою та опціями локалізації, зниження вартості</td><td colspan="2">Проблема:
Складність та дороговизна існуючих систем управління для споживача, відсутність локалізації відсікає велику кількість домогосподарств</td></tr>
<tr><td>Ключові партнери:
Виробники розумних приладів, виробники мікроконтролерів, ком’юніті програмістів (забезпечують розвиток нових технологій)</td><td>Ключові види діяльності:
Розробка ПЗ, інженерне проектування, консультування

Ключові ресурси:
Інтелектуальні, фінансові</td><td>Ціннісна пропозиція:
Зручний та привабливий інтерфейс, легке масштабування та налаштування системи управління, легка кастомізація</td><td>Взаємодія з клієнтами:
Реклама в інтернеті (пряма та не пряма через блогерів)

Канали збуту:
Прямий продаж, через забудовників</td><td>Сегменти споживачів:
Домогосподарства, дизайнерські бюро, забудовники, архітектурні бюро

Гіпотези:
Привабливий, зручний та зрозумілий додаток для управління системою «Розумний дім», недорога варіативна система</td><td>Кроки:
Підготовчий етап: грудень 2020
Дизайн, розробка, тестування: січень-квітень 2021
Пошук інвесторів: лютий-квітень 2021
Рекламна кампанія: березень-травень 2021</td></tr>
<tr><td colspan="2">Потоки доходів:
Прямий продаж, подальше обслуговування
Метрики:
30 проектів/ місяць, вартістю в середньому 7 тис грн.</td><td colspan="4">Структура витрат
Зарплатний фонд – близько 608 тис грн.
Проведення маркетингових досліджень – 50 тис грн.
Матеріальні та нематеріальні активи, прилади – близько 20 тис грн.
Рекламний бюджет на запуск – 200 тис грн.</td></tr>
</table>

Рис. 4.2. Бізнес модель стартапу станом на перший рік роботи

Для розробки ринкової стратегії продукту визначимо цільові групи потенційних споживачів, після чого визначимо базову стратегію розвитку.

Таблиця 4.8.

Вибір цільових груп потенційних споживачів

п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Конкурен-ція в сегменті	Простота входу у сегмент
1	Домогосподарства середнього + та преміум класу	Висока, якщо цінова пропозиція буде для них доступною	Орієнтовно в Україні можна продати близько 20 тис проектів/міс	Висока	Середня: конкуренція висока, але ринок постійно зростає, немає законодавчих блоків
2	Дизайнерські бюро, архітектурні бюро	Висока, оскільки надання подібних послуг розширить можливості їх бізнесу	Орієнтовно в Україні можна продати близько 1тис проектів/рік	Висока	

Продовження таблиці 4.8.

3	Забудовники, що спеціалізуються на житлі преміум класу	Висока, оскільки система буде їх конкурентною перевагою	Орієнтовно 500 проектів/рік	Помірна конкуренція, оскільки не багато компаній орієнтовані на даний сегмент	Середня: конкуренція висока, але ринок постійно зростає, немає законодавчих блоків
Які цільові групи обрано: Домогосподарства середнього класу, дизайнерські бюро, архітектурні бюро. Співпраця із забудовниками має бути наступним кроком, коли стартап виросте в достатньо велику компанію, що спроможна забезпечити проектування та обслуговування масштабних об'єктів.					

Таблиця 4.9.

Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспромо-жні позиції	Базова стратегія розвитку
Переорієнтація проекту в бік B2B (співпраця із забудовниками, дизайнерськими та архітектурними бюро)	Стратегія концентрованого зростання	Цінова пропозиція, user-friendly інтерфейс	Стратегія інтегрованого зростання

Далі необхідно визначити базову стратегію конкурентної поведінки та на її основі – стратегію позиціонування продукту. Стратегія конкурентної поведінки визначена у таблиці 4.10, а стратегія позиціонування – у таблиці 4.11.

Таблиця 4.10.

Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Шукати нових споживаців та забирати існуючих	Так: базову функціональність системи (можливість віддаленого управління приладами з додатка)	Стратегія «загарбник»

Таблиця 4.11.

Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Зручність інтерфейсу Отримання розширених можливостей управління станом будинку Легкість управління приладами Захищеність даних Надійність	Стратегія інтегрова-ного зростання	Зручний та зрозумілий інтерфейс Легкість управління приладами Розширення можливостей за рахунок створення сценаріїв, мультиплатіорменності та можливості підключити кількох користувачів Доступна цінова пропозиція Захищеність даних	Комфорт Доступність Надійність

Наступним кроком є визначення ключових переваг потенційного товару (таблиця 4.12).

Таблиця 4.12.

Визначення ключових переваг потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
Комфортне житло	Доступність вищого рівня комфорту Захищеність житла	Цінова пропозиція Зручність інтерфейсу Розширені можливості додатку

Сформулюємо рівень цін на проектування системи у таблиці 4.13.

Таблиця 4.13

Визначення меж встановлення ціни на продукт

Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Від 30 000 грн.	Від 50 000 грн/ місяць	Від 15 000 грн (за підключення стандартних приладів споживача в єдину систему без додаткового програмування та інженерних робіт) до 100 000 грн (за проектування системи для великого будинку, підбір та узгодження приладів, проведення інженерних робіт та кастомізацію додатку).

На основі визначення ключових переваг концепції товару, розробимо трирівневу модель товару в таблиці 4.14.

Таблиця 4.14

Трирівнева маркетингова модель товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Мобільний додаток для управління системою «Розумний дім»</td></tr>
<tr><td rowspan="11">II. Товар у реальному виконанні</td><td colspan="3">Властивості/характеристики</td></tr>
<tr><td></td><td>НМ, НН</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>Помірна вартість</td><td>НМ</td><td>Вр</td></tr>
<tr><td>Зручний та привабливий інтерфейс</td><td>НМ</td><td>Ор</td></tr>
<tr><td>Захист даних споживачів</td><td>НМ</td><td>Тх</td></tr>
<tr><td>Можливість керування приладами</td><td>НМ</td><td>Ор, Тх</td></tr>
<tr><td>Можливість моніторингу стану системи</td><td>НМ</td><td>Ор, Тх</td></tr>
<tr><td>Можливість створення сценаріїв</td><td>НМ</td><td>Ор, Тх</td></tr>
<tr><td>Мультиплатформенність</td><td>НМ</td><td>Тх</td></tr>
<tr><td colspan="3">Якість: UI/ UX стандарти, високий code coverage, ефективне шифрування данних споживачів, швидкодія системи, висока якість технічної підтримки</td></tr>
<tr><td colspan="3">Пакування: відсутнє – це мобільний та онлайн додаток</td></tr>
<tr><td colspan="3">Марка: «Доступний Розумний Дім»</td></tr>
</table>

Продовження таблиці 4.14

III. Товар із підкріпленням	До продажу: • можливість протестувати демо додатку в офісі компанії • надання інженерного проекту • підбір розумних приладів, які інтегруються в систему за запитом споживача
	Після продажу: • підключення додаткових приладів, • надання адмінки для максимальної кастомізації системи, • створення та оновлення бібліотек приладів • оновлення додатку
За рахунок чого потенційний товар буде захищено від копіювання: за рахунок реєстрації торгової марки, патентування розробленого методу управління системою розумного дому.	

Описавши ключові вигоди і переваги товару, ринкову стратегію, цільову аудиторію та межі цінової пропозиції ми можемо визначити канали збуту та концепцію маркетингових комунікацій. Оскільки в перший період планується продаж додатку і проектних послуг кінцевому споживачу, основним каналом комунікації буде пряма та непряма реклама, а також рекомендації дизайнерів інтер'єрів, архітекторів будинків, тощо (мереживний маркетинг).

Продаж планується на перших етапах без посередників в офісі команії, в квартирі/ будинку замовника, в офісі дизайнерської компанії, тощо. Докладніше канали збуту представлені в таблиці 4.15.

Таблиця 4.15.

Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Прийняття рішення про купівлю на основі реклами та рекомендацій. Часто – імпульсивне рішення про купівлю (тому між першим контактом та встановленням системи має пройти якомога менше часу).	Доставка та монтаж мікроконтроллерів. Підключення приладів. Надання доступу до завантаження додатку з віддаленого серверу.	0/1 Між стартап компанією та замовником – немає посередників. Між стартап компанією та замовником можливий посередник (архітектурне або дизайнерське бюро, тощо)	Прямий продаж плюс Appstore, Google Play

4.5. Аналіз ринкових можливостей запуску стартап проекту

Для оцінки ринкових можливостей запуску стартап проекту та визначення своєчасності його створення, слід проаналізувати рентабельність ринку, фактори загроз і можливостей, визначити фактори конкурентосмроможності. Тільки на основі цих даних можливо прийняти рішення про те слід запускати стартап в його

поточному вигляді, слід почекати, або слід модифікувати ідею, обрати іншу ринкову стратегію тощо [38].

Спочатку складемо попередню характеристику потенційного ринку проекту.

Таблиця 4.16

Попередня характеристика потенційного ринку стартап-проекту

п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	Безліч
2	Загальний обсяг продаж, грн/ум.од	-
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Отримання патенту на розроблений метод управління системою «Розумний дім»
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	40%

Як вже згадувалось раніше, ринок конкурентний, однак при цьому він постійно зростає, що при відсутності особливих обмежень, дає можливість зайняти свою нішу.

Таблиця 4.17

Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Комфортне житло	Домогосподарства середнього + та преміум класу	Розмір доходів домогосподарства	Зручність інтерфейсу Отримання розширених можливостей управління станом будинку Легкість управління приладами Захищеність даних Надійність

Тепер сформулюємо фактори загроз та можливостей для розвитку стартап-проекту.

Таблиця 4.18

Фактори загроз виходу стартап-проекту на ринок

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Копіювання конкурентами	Конкуренти можуть скопіювати технологію та створити аналогічний продукт (можливо більш дешевий або кращої якості	Патентування розробленого методу управління системою «Розумний дім»
2	Пандемія	Пандемія негативно вплинула на доходи більшості домогосподарств	Донесення до споживача інформації про те як система дозволить заощадити в майбутньому за рахунок економії електроенергії, тощо. Донесення до споживача інформації про те, що в умовах самоізоляції комфортне житло та автоматизація домашніх процесів набуває набагато більшого значення ніж раніше
3	Невідома стартап компанія	Недовіра з боку споживачів	Залучитись рекомендаціями з боку лідерів думок та професіоналів

Таблиця 4.19.

Фактори можливостей виходу стартап-проекту на ринок

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Пандемія	В умовах вимушеної самоізоляції комфорт вдома став надзвичайно важливим і люди готові витрачати більше коштів на девайси та послуги, які зроблять самоізоляцію більш приємною	Закладення можливості швидкого розширення та пришвидшення виходу на ринок
2	Розвиток НТП	Можуть з'явитись нові технології які дозволять скоротити бюджети/модернізувати систему, переорієнтувати її тощо	Готовність оперативно користуватися наданими можливостями
3	Цікавість з боку інших ринків	Споживачі на інших ринках можуть зацікавитись ідеєю та замовити адаптовану систему	Створити альтернативний бізнес план та стратегію розвитку

Проаналізувавши фактори загроз та можливостей, можна зробити висновок що пандемія є не стільки ризиком, скільки чудовою можливістю для стартапу. Люди, які вимушені знаходитись в самоізоляції, прагнуть підвищити рівень

домашнього комфорту і готові платити за нові девайси і сервіси, про які раніше навіть не думали.

Таблиця 4.20.

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
Тип конкуренції – чиста конкуренція	Безліч компаній пропонують схожі за функціоналом системи управління	Робота над розширенням можливостей та покращенням технічних характеристик продукту
За рівнем конкурентної боротьби – міжнародне конкурентне середовище	Ринок наповнений закордонними аналогами та товарами замінниками	Приділення уваги локалізації, можливість виходу на міжнародний ринок
За галузевою ознакою – міжгалузева	Система автоматизації потенційно може використовуватись в інших сферах	Можливість вийти на інші ринки із адаптованою системою
Конкуренція за видами товарів – товарно-видова	Існують різні технології побудови та управління системою	Розширення функціоналу, підвищення рівня безпеки
За характером конкурентних переваг – цінова конкуренція	Є безліч подібних систем, та розумних пристроїв зі своїм управлінням.	Необхідно зробити систему не тільки зрозумілою для користувача, але й конкурентною за ціною.

Маючи дані ступеневого аналізу, проведемо ще аналіз конкуренції в галузі за Портером (таблиця 4.21) [39].

Таблиця 4.21.

Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	В Україні існує кілька десятків компаній, що надають повний спектр послуг по проектуванню та підключення системи під ключ, а також онлайн платформи, що продають схожі рішення	Вітчизняні та іноземні компанії, що займаються розробкою ПЗ	Виробники мікроконтролерів та розумних пристроїв	Клієнти мають обмежений бюджет. Клієнти ставляться до систем «Розумний дім» як до чогось мало доступного і незвичного	Як таких замінників немає

Продовження таблиці 4.21.

Висновки:	Конкурентна перевага за рахунок того, що виробник системи і є його постачальником. Це дозволяє конкурувати по ціні.	Є можливість виходу на ринок та потенційні конкуренти	Постачальники диктують умови роботи на ринку, оскільки необхідно орієнтуватись на існуючі прилади, протоколи, які вони використовують і особливості їх роботи	Клієнти поки що не диктує умов роботи ринку, оскільки не знає до кінця можливостей систем «Розумний дім». Тим не менш, вони вже дають відгуки, які варто врахувати	Як таких замінників немає

Можна зробити висновок, що проект не має поступатись якістю конкурентам, а також має і може конкурувати за ціною. Крім того, на українському ринку склалась цікава ситуація, коли розумні гаджети і прилади вже користуються попитом, а система «Розумний дім» досі є новинкою та існує міф про її складність та недоступність. В цьому є можливість для стартапу, оскільки компанія може стати популяризатором та експертом та за відносно короткий час заслужити довіру споживачів.

Обгрунтуємо фактори конкурентоспроможності в таблиці 4.22.

Таблиця 4.22.

Обгрунтування факторів конкурентоспроможності стартапу

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Нижча ціна ніж у конкурентів	Систему зможе дозволити собі не тільки супер преміум клас, а й середній + клас.
2	Розширені можливості системи	Продукт адаптується під потреби споживача
3	Привабливий та зрозумілий інтерфейс	Споживачу не потрібно окремо вчитися використовувати систему управління та перекладати з англійської. Простіший порог входу робить систему більш привабливою в порівнянні з конкурентами

За допомогою факторів конкурентоспроможності проведемо аналіз слабких та сильних сторін стартапу.

Таблиця 4.23.

Порівняльний аналіз слабких та сильних сторін стартап-проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з проектом						
			3	2	1	0	1	2	3
1	Нижча ціна ніж у конкурентів	20							
2	Розширені можливості системи	10							
3	Привабливий та зрозумілий інтерфейс	10							

На основі виявлених факторів конкурентоспроможності, а також аналізі слабких та сильних сторін в порівнінні з конкурентами, проведемо SWOT-аналіз.

Таблиця 4.24.

SWOT-аналіз стартап проекту

Сильні сторони: Ціна Функціональність системи Інтерфейс	Слабкі сторони:
Можливості: Пандемія Цікавість з боку інших ринків Розвиток НТП	Загрози: Копіювання конкурентами Пандемія та зниження доходів споживачів Поява нових конкурентів

На основі SWOT-аналізу розробимо також альтернативну стратегію виходу проекту на ринок.

Таблиця 4.25.

Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Залучення додаткових інвестицій	Середня	3-4 місяці
2	Впровадження акцій, відтерменувань платежів	Висока	1-2 місяці
3	Продаж через забудовників, дизайнерські та архітектурні бюро. В т.ч. включення системи в ремонт «під ключ» від забудовника.	Висока	2-3 місяці

Виходячи з аналізу альтернативного впровадження проекту на ринок, можна зробити висновок, що найбільш ефективною альтернативною стратегією при відсутності великих рекламних бюджетів є переорієнтація в бік співпраці з забудовниками та компаніями що надають послуги ремонту, інтер'єрного дизайну тощо. Також за необхідності можна підключити акції та розсрочки.

4.6. Розробка виробничого плану та розрахунок витрат на запуск проекту

Мета створення виробничого плану – проаналізувати можливості ефективного організаційного та ресурсного забезпечення проекту. Цей план також необхідний для того, щоб продемонструвати потенційним інвесторам, що запропоновані рішення дозволять ефективно запустити стартап проект.

Виробничий план складається з: календарного плану та розрахунку планової потреби у виробничих площах, устаткуванні, витрат на персонал, матеріальні та нематеріальні активи, тощо. В цьому підрозділі наведено розрахунки для нульового року проекту (тобо витрати, необхідні для його запуску) [40].

Таблиця 4.26.

Календарний план-графік реалізації стартап-проекту

№ з/п	Етапи реалізації	Період реалізації проекту						
		0-й рік				1-й рік	2-й рік	3-й рік
		1-й кв.	2-й кв.	3-й кв.	4-й кв.			
1	Проведення НДДКР	х						
2	Розробка проектних матеріалів	х						
3	Створення компанії	х						
4	Придбання нематеріальних активів, отримання дозвільних документів тощо	х						

Продовження таблиці 4.26.

5	Передвиробничі маркетингові дослідження	x						
6	Придбання матеріальних ресурсів	x						
7	Розробка та тестування системи	x	x					
8	Проведення рекламної компанії		x					
9	Початок продажів системи			x				

Для запуску стартапу немає потреби в оренді або придбанні виробничих площ, офісу тощо, особливо в умовах пандемії. Робота над проектом планується в увіддаленому режимі (онлайн) з власних комп'ютерів та ноут-буків. Так само немає потреби у придбанні матеріальних активів. Щодо обладнання, то для розробки та тестування системи необхідно придбати мікроконтролери, датчики та тестові розумні прилади. Також необхідно залучити спеціалістів з дизайну, інженерії та розробника ПЗ для швидкої підготовки системи з прототипу. Структура витрат на придбання девайсів, нематеріальних активів та оплату роботи працівників наведена у таблицях 4.27, 4.28 та 4.29.

Таблиця 4.27.

Планова потреба у виробничому обладнанні та устаткуванні

№ з/п	Вид обладнання (устаткування, пристрою)	Терміни постачання	Вартість, грн.
1	Мікроконтролер Iskra JS ARM Cortex-M4	4 дні	3540
2	Мікроконтролер Espruino Puck	2 тижня	1128
3	Трійка модуль Troyka Shield	4 дні	310
4	Цифровий датчик температури та вологості	4 дні	150
5	Кабель USB - MICROUSB	1 день	120

Продовження таблиці 4.27.

6	Блок живлення Ginzzu GA-3311UW	4 дні	300
7	Кабель HDMI 2.0	1 день	100
8	Комплект перемичок	4 дні	210
9	Силовий ключ	4 дні	150
10	Troyka Mega Tail Shield для додаткових пінів	4 дні	370
11	Потенціометр для регулювання напруги	4 дні	130
12	TV вихід	1 день	100
13	Світлодіодний модуль, 4 шт	4 дні	480
14	Wi-Fi (Troyka-модуль)	4 дні	420
15	Bluetooth чайник Xiaomi	1-2 дні	1200
16	Терморегулятор	1-2 дні	945
17	Розумний зволожувач повітря	1-2 дні	1600
18	Xiaomi Mi TV HD 4A 32	1-2 дні	5400
Разом:			16 653

Таблиця 4.28.

Планова потреба у нематеріальних активах

№ з/п	Вид активів	Активи, що можуть бути віднесені до даного виду	Вартість, грн.
1	Ліцензії	Ліцензія на програмний пакет Atlassian	3 000
Разом:			3 000

Таблиця 4.29.

Планова потреба витрат на персонал

№ з/п	Категорія персоналу	Чисельність	Заробітна плата, грн. на місяць	Відрахування на соціальні заходи, грн. на місяць	Витрати на оплату праці за період (5 міс), грн
1	Інженер	1	20 000	7 480	137 400
2	JavaScript розробник	2	25 000	9 350	343 500
3	QA	1	15 000	5 610	103 050
4	UI/UX дизайнер	1	18 000	6 372	24 372
Разом:					608 322

Зведемо наведені та інші витрати в єдину таблицю загальних початкових витрат для запуску проекту.

Таблиця 4.30

Загальні початкові витрати на запуск проекту

№ з/п	Стаття витрат	Сума, грн
1	Зарплатний фонд	608 322
2	Витрати на придбання приладів	16 653
3	Вартість нематеріальних активів	3 000
4	Проведення маркетингових досліджень	50 000
5	Витрати на ректаму в інтернеті	200 000
Разом:		877 975

Отже, загальна потреба в інвестиціях для запуску проекту – 877 975 грн.

Висновки до розділу 4

В даному розділі було визначено конкурентні переваги та недоліки стартапу, запропоновано основний та альтернативні шлях його розвитку, проаналізовано конкурентне середовище, загрози та можливості, проведена клькуляція витрат на запуск стартапу, а також розроблений бізнес план з урахуванням каналів збуту, ціни, календарного плану підготовки проекту, розподілу обов'язків між засновниками, тощо.

Із проведенного аналізу можна зробити висновок, що вихід на ринок стартапу можливий, більш того в умовах пандемії можна прогнозувати вищий попит на систему та проектно-монтажні послуги.

Цільовою аудиторією було обрано домогосподарства середнього плюс класу та тих, хто вважається багатими. Комунікація зі споживачами планується через пряму та непряму рекламу в інтернеті, а також через архітектурні та дизайнерські бюро, компанії-забудовники (наприклад включення системи «Розумний дім» в опції ремонту від забудовника.

Напрямками подальшого розвитку стартапу є: розробка великих комерційних проектів (бізнес центри, ТРЦ, житлові комплекси), створення адмінки в пакеті із власними модулями-бібліотеками та надання можливості користувачу без технічних знань та навичок легко підключати нові прилади до системи.

Конкурентними перевагами системи є її зручність, багатофункціональність та приваблива ціна. Саме останній пункт дозволяє включити в сегмент споживачів середнього плюс класу в цільову аудиторію стартапу.

Було прораховано плановий термін реалізації першої версії системи на основі розробленого прототипу, а також розраховано суму інвестицій для виходу на ринок.

ВИСНОВКИ

В магістерській дисертації була розроблена система управління розумним домом на базі ядра Espruino, реалізована її архітектура та додаток для управління приладами системи.

Архітекрура системи управління «Розумний дім» складається з:

- Клієнта
- Web-сервера
- Бази даних
- Контролера для управління фізичними пристроями

Система повністю реалізована мовою програмування JavaScript із використанням сучасного паттерну програмування Flux, бібліотек React, Redux, Express та Mongoose, що є її унікальною особливістю. За рахунок використання NodeJS та єдиної мови програмування, архітектурне рішення вийшло простим та зручним.

Для програмування клієнта було використано компонентний підхід, а сервер і контролер управління фізичними пристроями складається з модулів. Такий підхід дає можливість легко масштабувати проект. Додавання нових приладів, модулів та компонентів не впливає на роботу старих.

Для роботи із NodeJS було обрано базу даних Mongo, що розміщується у хмарному кластері, оскільки Mongo легко інтегрується в проект. Для спілкування з базою не потрібні додаткові налаштування серверу, проте для реалізації модульного підходу була використана бібліотека Mongoose.

Контролер для управління фізичними пристроями був розроблений для плати IskraJS та мікроконтроллера Espruino Puck на основі модулей для спілкування з фізичними пристроями @amperka.

Інтерфейс додатку був реалізований на основі проведенних UX-досліджень, тому є інтуітивно простим та виконує усі функції, необхідні для управління розумним будинком.

В магістерській дисертації було також проведено маркетинговий аналіз, на основі якого розроблений стартап-проект. В рамках четвертого розділу дослідження було розраховано орієнтовний бюджет запуску стартап проекту та наведено орієнтовний графік організації стартапу, розробки програмного продукту та проведення рекламної кампанії.

Було визначено конкурентні переваги системи (ціна та зручний інтерфейс), досяжні завдяки розробленому архітектурному рішенню. Відносно невисока ціна дає можливість розширення цільової аудиторії, а компонентний та модульний підхід природньо визначає шляхи подальшого розвитку програмного продукту (створення власних бібліотек приладів, що складаються із модуля фізичного приладу, серверного модуля та інтерфейсного компонента, та надання можливості користувачам підключати нові прилади через адмінку завдяки цим бібліотекам).

Із проведенного маркетингового аналізу можна зробити висновок, що вихід на ринок стартапу можливий, вірогідність успіху висока, особливо в умовах пандемії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Менахем Домб. Системи розумного дому, заснованого на основі інтернет речей [Електронний ресурс]. Режим доступу: https://www.intechopen.com/books/internet-of-things-iot-for-automated-and-smart-applications/smart-home-systems-based-on-internet-of-things

2. Система розумний дім: [Електронний ресурс]: // Ecotown – Режим доступу: https://ecotown.com.ua/news/Systema-Rozumnyy-dim-zmenshuye-vytraty-na-komposluhy-do-30/

3. Переваги «Розумного» будинку: [Електронний ресурс]: // klyuch – Режим доступу: http://klyuch.com.ua/m/articles/economy/yaki-perevagy-rozumnogo-domu-/

4. Ера цифрових помічників в домашній автоматизації: що нового? – 2019 [Електронний ресурс]. Режим доступу: https://worldvision.com.ua/ua/era-tsifrovykh-pomoshchnikov-v-domashney-avtomatizatsii-chto-novogo/

5. Сети ZigBee – зачем и почему [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/155037/

6. Бибель В. П., Глухов В. С., Пристопюк О. В. Вибір бездротової технології передавання даних для обладнання навчальних лабораторій/ В. П. Бибель, В. С. Глухов, О.В. Пристопюк.// Analysis of data wireless technologies for educational process modernization – 2016.

7. Горбенко І. Д., Замула О. А. Інформаційні технологій. Оцінка показників захищеності сучасних бездротових систем зв'язку широкосмугового доступу на основі врахування особливостей технологій/ І.Д. Горбенко, О.А. Замула – OFDM. ХНУ, 2012 – 67-75 с.

8. S. Folea, D. Bordencea, C. Hotea and H. Valean, "Smart home automation system using Wi-Fi low power devices," Automation Quality and Testing Robotics (AQTR), 2012 IEEE International Conference on, Cluj-Napoca, 2012, pp. 569-574.

9. Огляд готових рішень систем «Розумний дім». [Електронний ресурс]. Режим доступу до ресурсу: https://sprut.ai/client/article/1544

10. С верой в автоматизацию. Обзор линейки контроллеров Vera. [Електронний ресурс]. Режим доступу до ресурсу: https://www.ferra.ru/review/smarthome/smartHome-Vera.htm

11. IntraHouse + Wirenboard. О контроллере Wiren Board 6. [Електронний ресурс]. Режим доступу до ресурсу: https://forum.ih-systems.com/topic/169/intrahouse-wirenboard

12. R. O. Duda and P. E. Hart. Pattern Classification and Scene Analysis. Wiley & Sons, New York, 1973.

13. C. Felix and I. Jacob Raglend, "Home automation using GSM," Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on, Thuckafay, 2011, pp. 15-19.

14. R. A. Ramlee, M. A. Othman, M. H. Leong, M. M. Ismail and S. S. S. Ranjit, "Smart home system using android application," Information and Communication Technology (ICoICT), 2013 International Conference of, Bandung, 2013, pp. 277-280.

15. IoT архитектура – первый взгляд под капот [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/420173/

16. N. Dickey, D. Banks and S. Sukittanon, "Home automation using Cloud Network and mobile devices," Southeastcon, 2012 Proceedings of IEEE, Orlando, FL, 2012, pp. 1-4.

17. M. A. Ullah, A. R. Celik, “An Effective Approach to Build Smart Building Based on Internet of Things (IoT)”, Journal of Basic and Applied Scientific Research, issues 6, 2016, pp. 56-62

18. Мікроконтролери програмовані на javascript: який вибрати, характеристики і можливості [Електронний ресурс] – Режим доступу: https://blox.com.ua/mikrokontrolery-prohramovani-na-javascript-iakyj-vybraty-kharakterystyky-i-mozhlyvosti.html

19. Espruino Pico [Електронний ресурс] – Режим доступу: https://www.espruino.com/Pico

20. Iskra JS: подключение, настройка, распиновка и схемы [Електронний ресурс] – Режим доступу: http://wiki.amperka.ru/js:iskra_js

21. Raspberry Pi 4 [Електронний ресурс] – Режим доступу: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/?resellerType=home

22. Как создать и подключить свою библиотеку в Espruino Web IDE [Електронний ресурс] – Режим доступу: http://wiki.amperka.ru/js:modules:connection

23. Top 8 IDEs for Programmers, Coders and Beginners on the Raspberry Pi [Електронний ресурс] – Режим доступу: https://blog.idrsolutions.com/2014/12/top-8-ides-programmers-coders-beginners-raspberry-pi/

24. HTML/CSS/JS + Node.js + node-webkit = Кроссархитектурные приложения [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/181670/

25. Flux: Архитектура приложений на React.js — всестороннее исследование [Електронний ресурс] – Режим доступу: https://medium.com/@marina.kovalyova/flux-the-react-js-application-architecture-773f515d068d

26. Елементи UML [Електронний ресурс]: // KDE – Режим доступу: http://docs.kde.org/stable/uk/kdesdk/umbrello/uml-elements.html

27. Керуючі пристрої «Розумного» будинку: [Електронний ресурс]: // sutem – Режим доступу до ресурсу: http://sutem.com.ua/7121smartbus.php

28. The Complete Guide to UX Research Methods [Електронний ресурс] – Режим доступу: https://www.toptal.com/designers/user-research/guide-to-ux-research-methods

29. How To Conduct UX Research For Maximum Value [Електронний ресурс] – Режим доступу: https://usabilitygeek.com/how-to-conduct-ux-research/

30. Authentication and Authorization in Node JS [Електронний ресурс] – Режим доступу: https://www.codementor.io/@manashkumarchakrobortty/authentication-and-authorization-in-node-js-19brdvhsyw

31. React Authentication: How to Store JWT in a Cookie [Електронний ресурс] – Режим доступу: https://medium.com/@ryanchenkie_40935/react-authentication-how-to-store-jwt-in-a-cookie-346519310e81

32. Библиотека управления @amperka/led [Електронний ресурс] – Режим доступу: http://wiki.amperka.ru/js:led

33. Библиотека управления @amperka/pid [Електронний ресурс] – Режим доступу: http://wiki.amperka.ru/js:pid

34. Библиотека управления @amperka/ power-control [Електронний ресурс] – Режим доступу: http://wiki.amperka.ru/js:power-control

35. Bluetooth LE Security and Access Control [Електронний ресурс] – Режим доступу: https://www.espruino.com/BLE+Security

36. Сам себе инженер: домашняя автоматика и роботы на JavaScript [Електронний ресурс] – Режим доступу: https://geekbrains.ru/posts/js_diy_robots

37. Стартап (Startup): що це таке простими словами + 5 ідей та порад щодо створення start-up проекту з мінімальними вкладеннями [Електронний ресурс] – Режим доступу: https://biznescat.com/biznes/78-startup-shcho-tse-take.html

38. Іщенко М. І., Нусінов В. Я. «Аналіз зовнішнього середовища як складова стратегічного аналізу» / М. І. Іщенко, В. Я. Нусінов – М.: Ефективна економіка № 6, 2014.

39. Модель п'яти сил конкуренції М. Портера [Електронний ресурс] – Режим доступу: https://pidru4niki.com/12980108/marketing/analiz_konkurentiv

40. Зозуля Д. М. «Бизнес-планирование в современной предпринимательской среде: теоретико-прикладной аспект» // Науково-методичний журнал «Концепт». – 2017. – Т. 31. – С. 566–570.

ДОДАТКИ

ДОДАТОК А

Анкета для проведення UX дослідження

1. Дані користувача

Вік: _____

Стать: _____

Рівень щомісячного доходу (для респондентів, що працюють):

☐ вище середнього (20-50 тис грн);

☐ високий (50 тис грн і вище);

Рівень щомісячних витрат (для респондентів, що не працюють): _____

Структура витрат (на що респондент витрачає гроші та приблизно в яких долях):

☐ їжа_____

☐ одяг_____

☐ розваги (кіно, ресторани, театри, концерти, тощо)_____

☐ освіта_____

☐ техніка_____

☐ подорожі_____

☐ інше_____

Освіта:

☐ вища/неповна вища (навчається)

спеціальність: _____

☐ середня спеціальна/ неповна середня спеціальна (навчається)

спеціальність: _____

☐ середня

2. Тестування гіпотези

Сценарій	Приблизний час виконання	Оцінка виконання (1 – виконав, 0 – не виконав)
Ви вийшли з дому та напівдорозі засумнівались чи увімкнута система охорони. Перевірте чи будинок поставлений на сигналізацію та за потреби увімкніть її.		
Закінчилось літо і вам стало важко прокидатись вранці. Створіть собі сценарій «приємний ранок», за яким ви будете прокидатись під легку музику та запах свіжої кави.		
На вулиці дощ і ви поспішаєте додому із мрією про гарячий чай. Налаштуйте систему таким чином щоб чайник закипів як раз до вашого прибуття, яке планується через 15 хвилин.		

3. Відгук респондента

ДОДАТОК Б

Прототип інтерфейсу для проведення UX дослідження

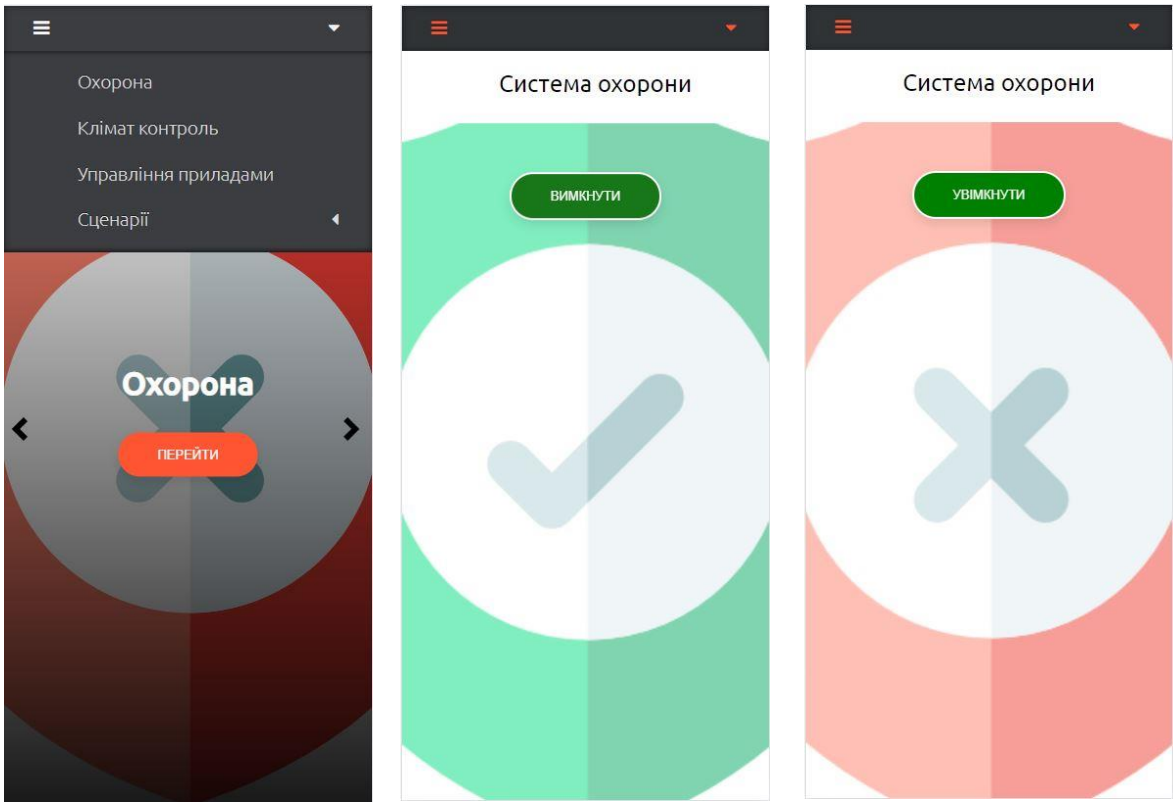


Рис. Б.1. Дизайн прототипу до сценарію управляння системою охорони

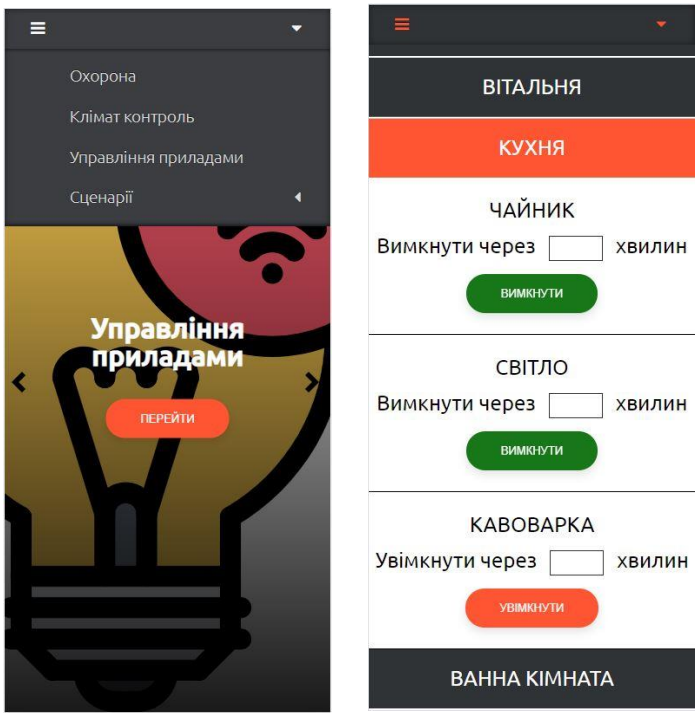


Рис. Б.2. Дизайн прототипу до сценарію увімкнення чайника

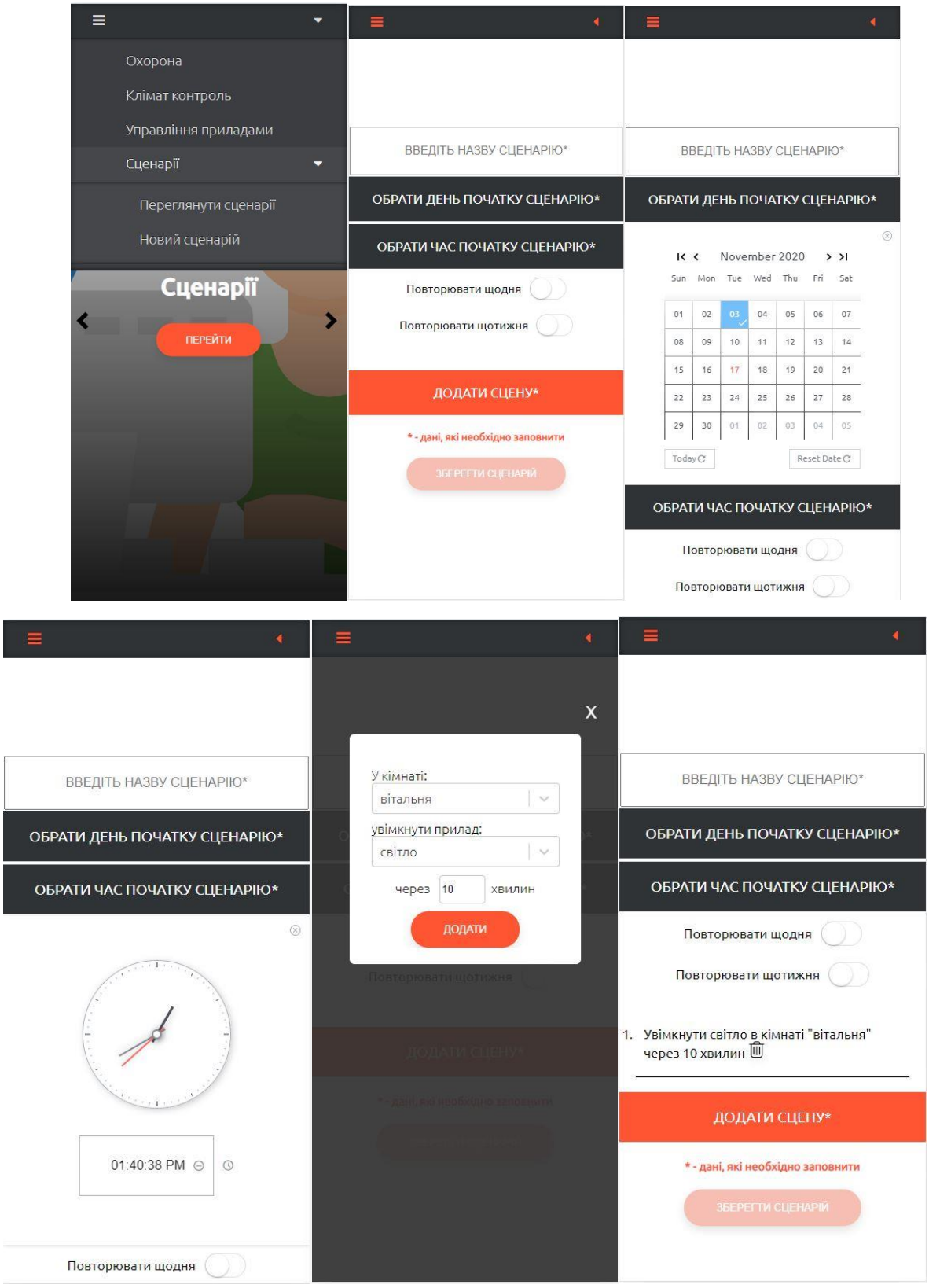


Рис. Б.3. Дизайн прототипу до сценарію створення унікального користувацького сценарію «Приємний ранок»

ДОДАТОК В

Гіпотеза щодо дій користувача та прототип клієнтської частини системи управління розумним домом

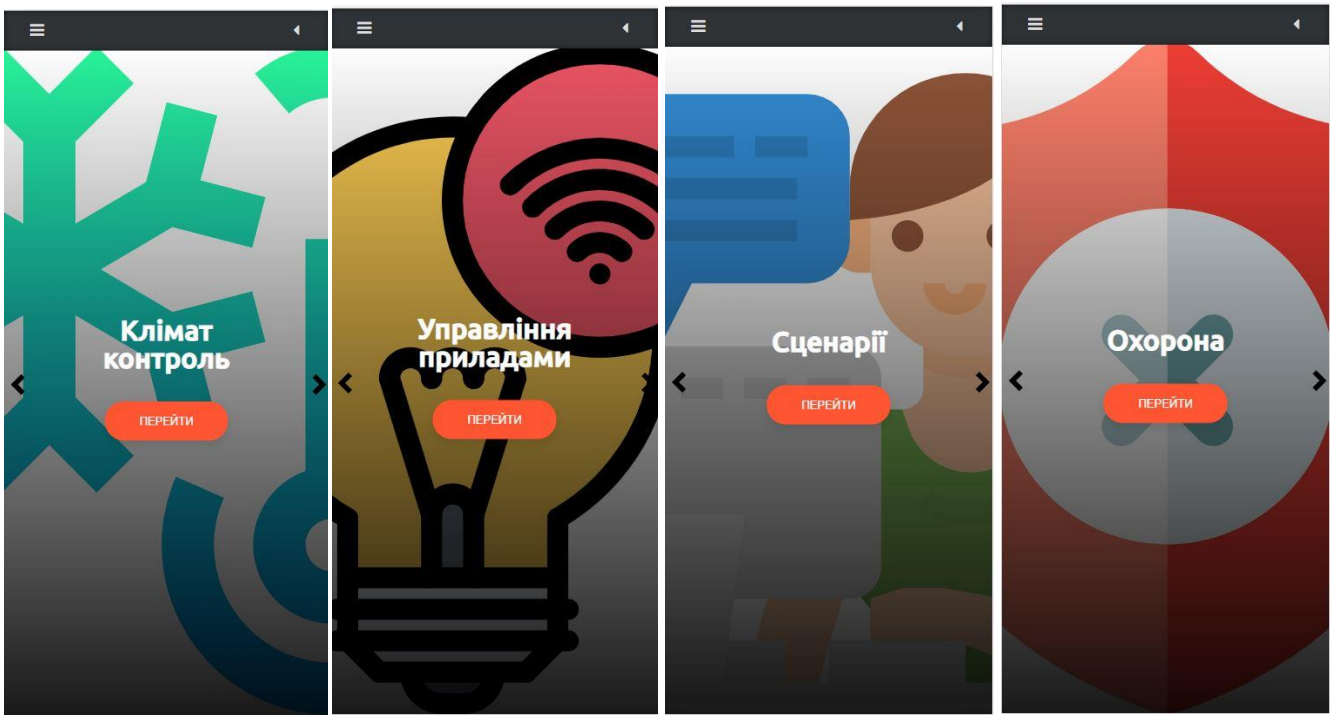


Рис. В.1. Прототип головного екрану додатку

Активності користувача		Перевірка стану системи охорони		Зміна стану системи охорони
Завдання користувача	Зайти на вкладку зі станом системи охорони		Перевірити чи увімкнена сигналізація	Увімкнути сигналізацію
Історії користувача	Користувач відкриває меню	Користувач обирає в меню «Охорона»	Користувач дивиться яким чином відображається іконка безпеки (закритий замок зеленого кольору, або відкритий червоного)	Користувач натискає на іконку і спостерігає, що змінилася іконка, кнопка та підказка

Рис. В.2. Гіпотеза перевірки та зміни стану системи охорони

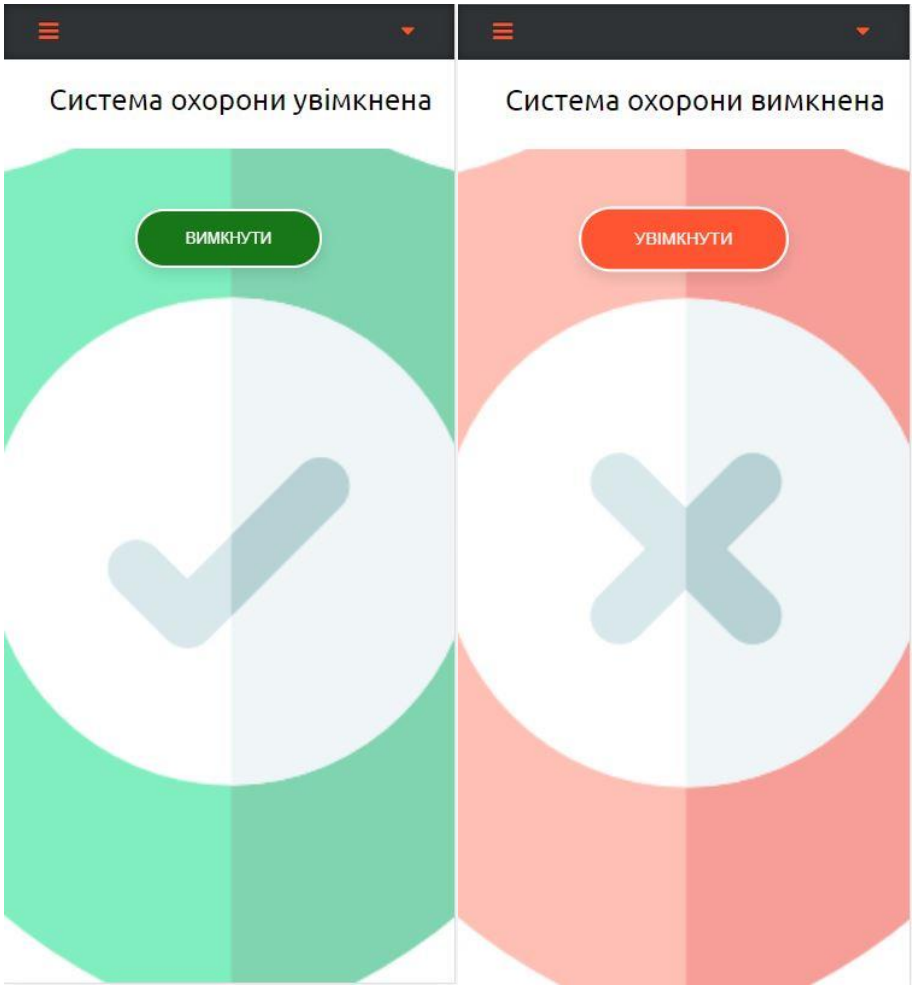


Рис.В.3. Прототип сторінки «Охорона»

Рис. В.4. Гіпотеза перевірки та стану вологості та температури

Рис. В.5. Прототип сторінки «Клімат контроль»

Рис. В.6. Гіпотеза увімкнення/вимкнення приладу

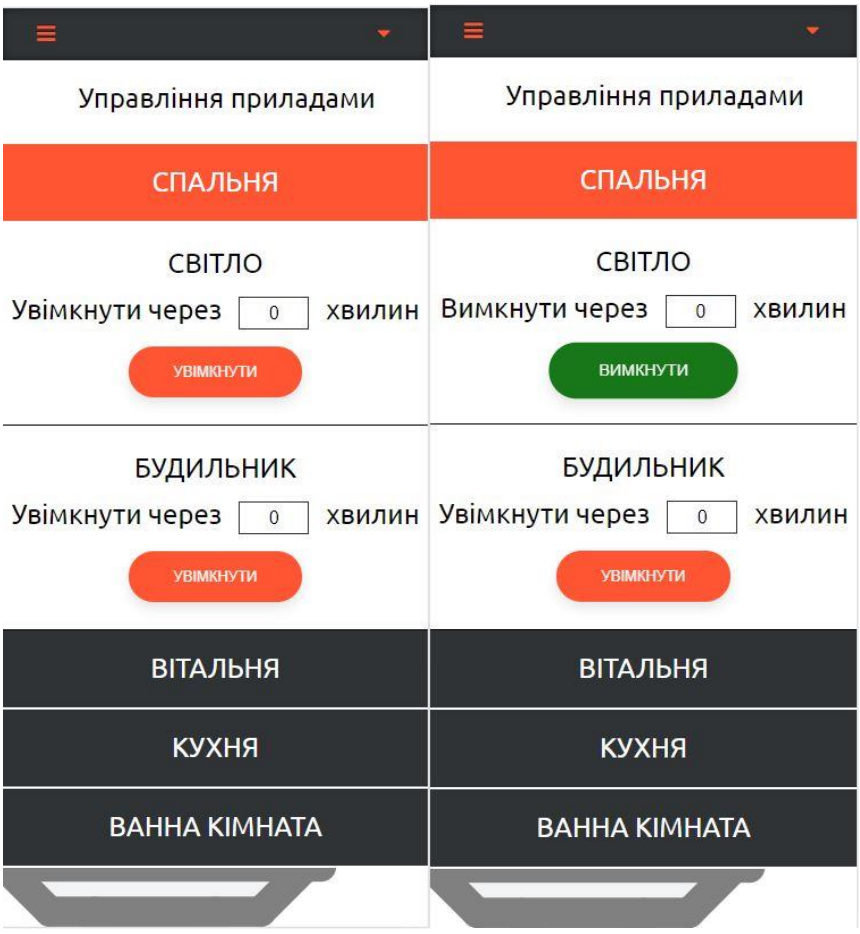


Рис. В.5. Прототип сторінки «Управління приладами»

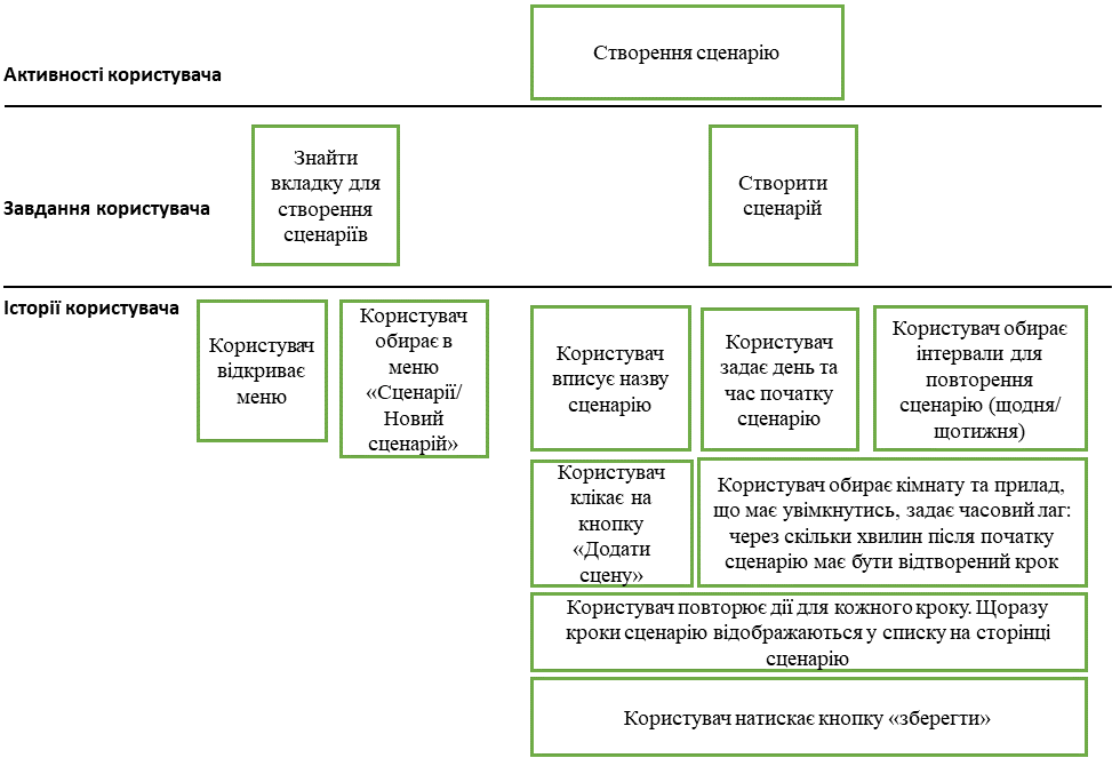
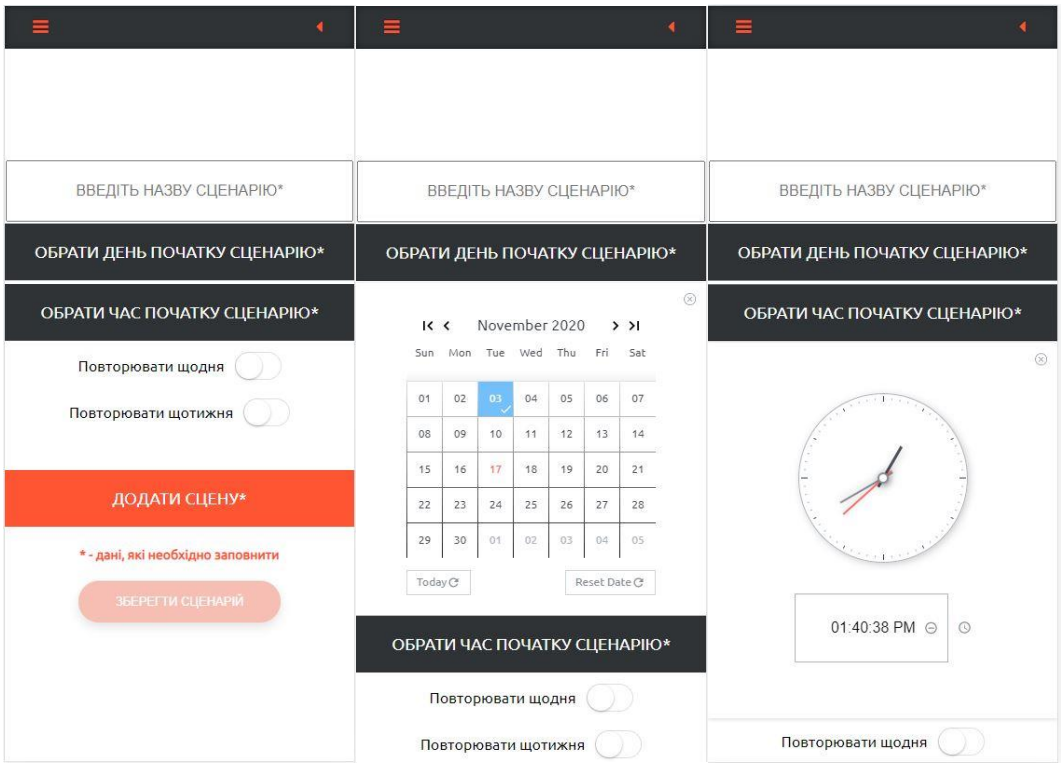


Рис. В.6. Гіпотеза створення сценарію



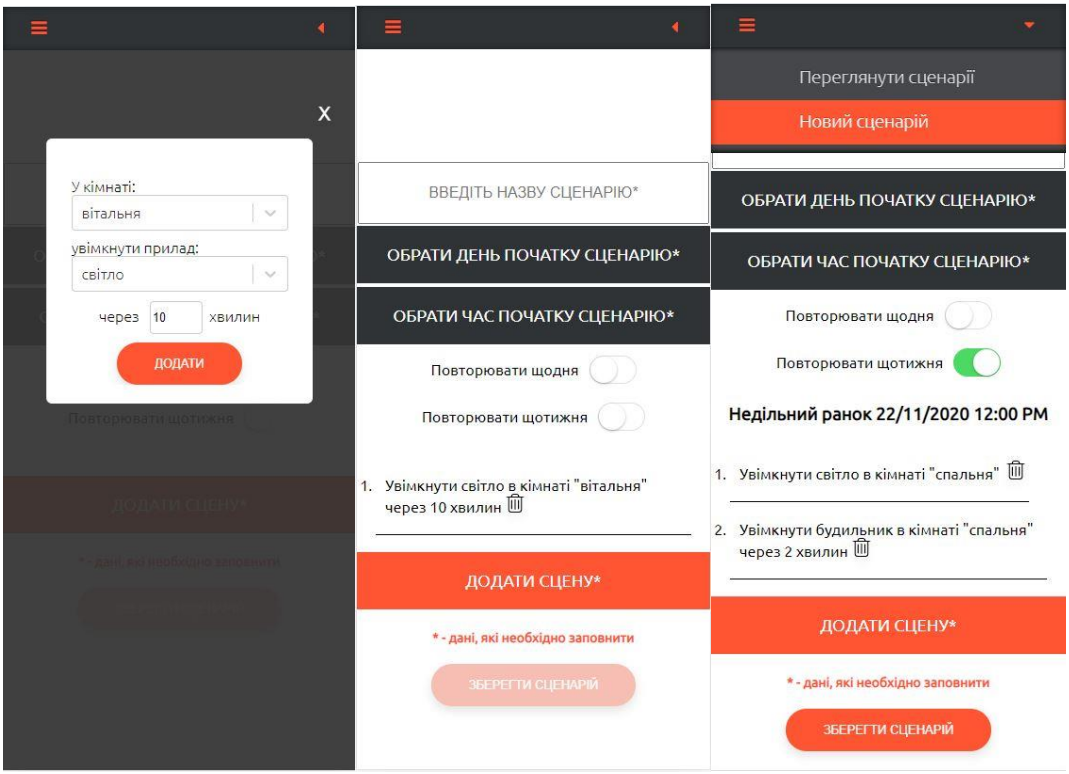


Рис.В.7. Прототип сторінки «Новий сценарій»

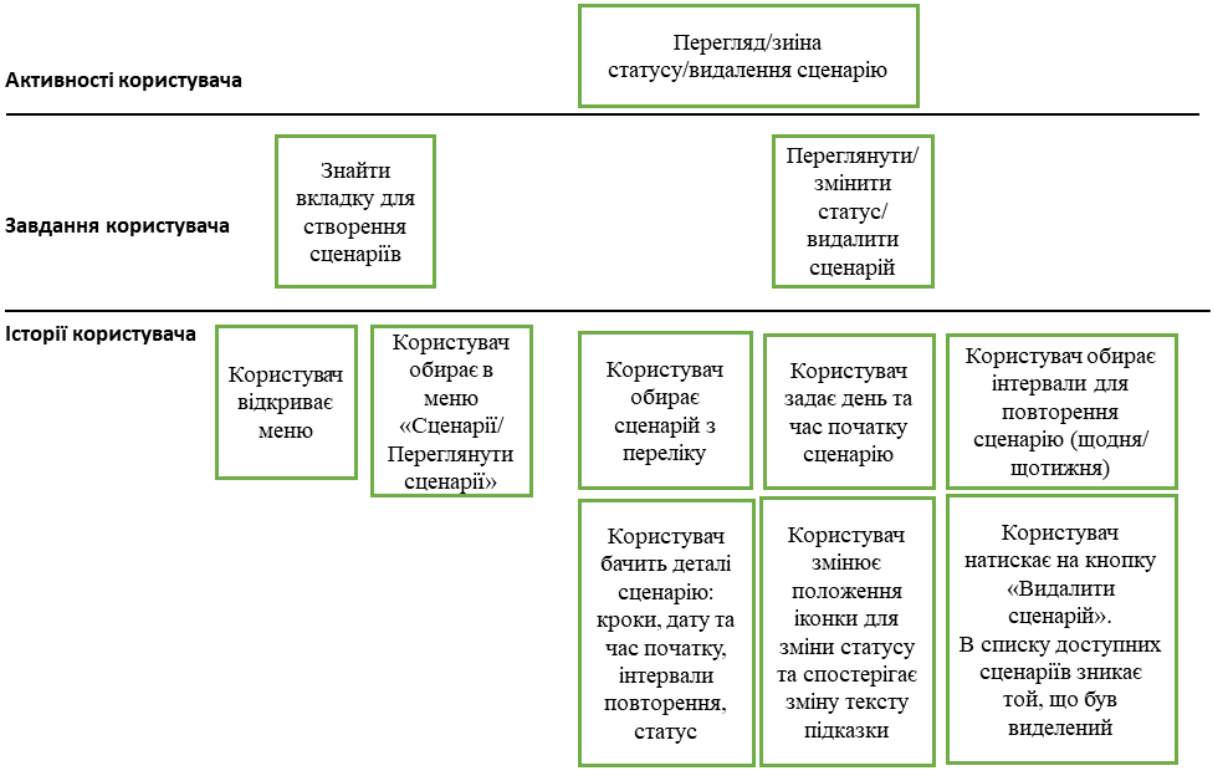


Рис. В.8. Гіпотеза перегляду, увімкнення/вимкнення та видалення сценарію

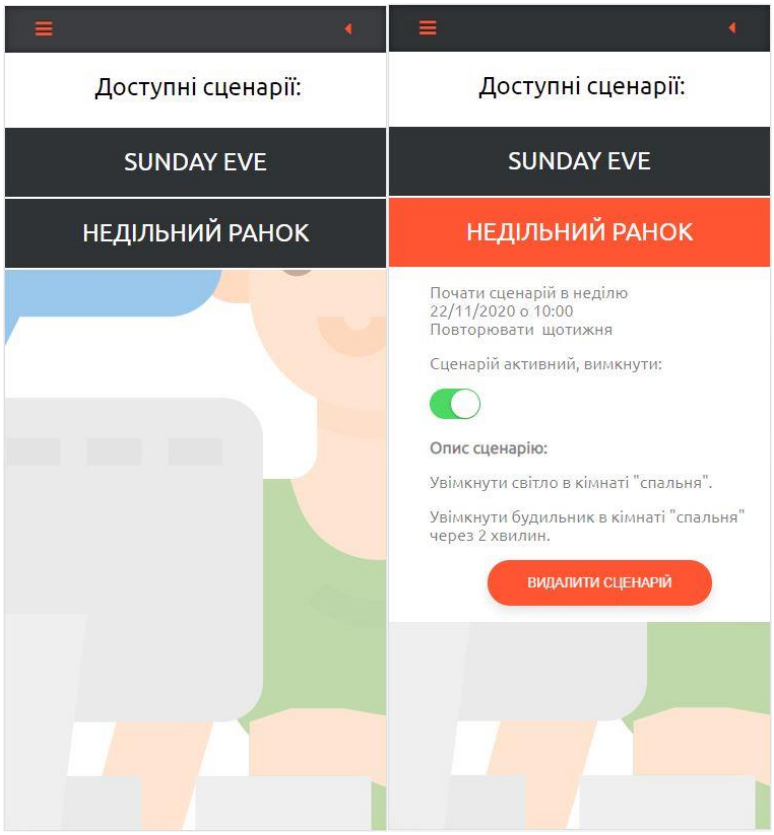


Рис. В.9. Прототип сторінки «Доступні сценарії»

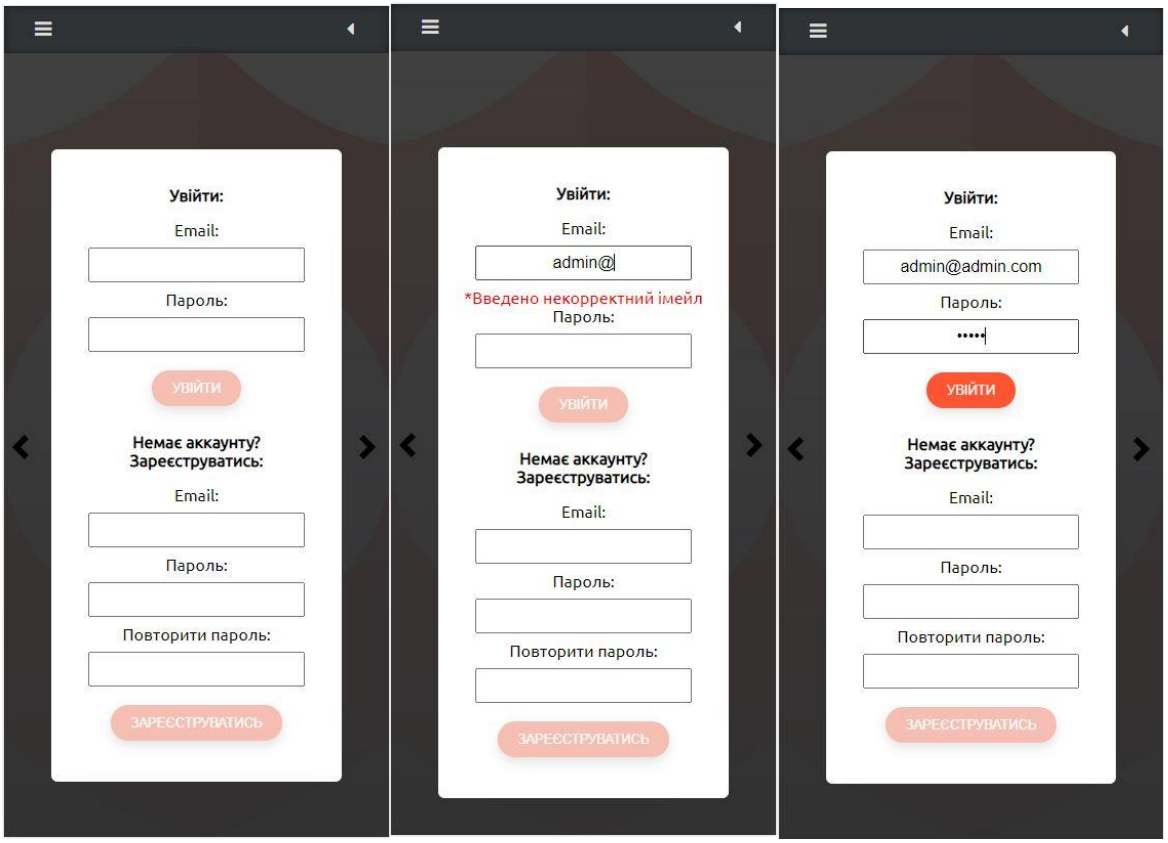


Рис. В.10. Прототип сторінки авторизації/реєстрації

ДОДАТОК Г

Лістинг програми системи управління розумним домом

1. Клієнтська частина

1.1. Коммунікація з сервером

```
import axios from 'axios';
import Cookie from "js-cookie";
const api = axios.create({
  baseURL: 'http://localhost:3003/',
});
const token =  Cookie.get("token") ? Cookie.get("token") : null;

class Apis {
  getDevice() {
    return api.get('devices', { headers: {'Authorization': token}});
  }
  editDevice(device) {
    device.state = device.on;
    return api.put(`devices/${device._id}`, device, { headers: {'Authorization': token}});
  }
  getScene() {
    return api.get('scenes', { headers: {'Authorization': token}});
  }
  editScene(scene) {
    return api.put(`scenes/${scene._id}`, scene, { headers: {'Authorization': token}});
  }
  addScene(scene) {
    return api.post('scenes', scene, { headers: {'Authorization': token}});
  }
  removeScene(id) {
    return api.delete(`scenes/${id}`, { headers: {'Authorization': token}})
  }
  getRooms() {
    return api.get('rooms', { headers: {'Authorization': token}});
  }
  getUser(user) {
    return api.get(`users/${user.email}&${user.password}`);
  }
  addUser(userData) {
    return api.post('users', userData)
  }
};
export default Apis;
```

1.2. Redux

```
import { combineReducers } from 'redux';
import { routerReducer } from 'react-router-redux';
import { reducer as device } from './device.reducer.js';
import { modalReducer as modal} from './modals.reducer';
import {reducer as scene} from './scene.reducer';
import {reducer as rooms} from './rooms.reducer';
import {reducer as user} from './user.reducer';

const rootReducer = combineReducers({
  routing: routerReducer,
  device,
  modal,
  scene,
  rooms,
  user
});
export default rootReducer;

const initialState = {
  isVisible: false,
  content: null,
};
```

1.3. modalReducer

```
const OPEN_MODAL =  'OPEN_MODAL';
const CLOSE_MODAL =  'CLOSE_MODAL';
export function modalReducer(state = initialState, action = {}) {
  switch (action.type) {
    case OPEN_MODAL: {
      return {...state, content: action.content, isVisible: true}
    }
    case CLOSE_MODAL: {
      return {...state, isVisible: false}
    }
    default: return state;
  }
}
export function openModal(content){
  return {
    type: OPEN_MODAL,
    content,
  }
}
export function closeModal(){
  return {
    type: CLOSE_MODAL,
  }
```

```
}
```

1.4. scene.reducer

```
import Apis from '../../api/api';
const apis = new Apis();

const MODULE_PREFIX = '/scene';
const LOAD_SCENE_STARTED = `${MODULE_PREFIX}/LOAD_SCENE_STARTED`;
const LOAD_SCENE_FINISHED = `${MODULE_PREFIX}/LOAD_SCENE_FINISHED`;
const EDIT_SCENE_STARTED = `${MODULE_PREFIX}/EDIT_SCENE_STARTED`;
const EDIT_SCENE_FINISHED = `${MODULE_PREFIX}/EDIT_SCENE_FINISHED`;
const ADD_SCENE_STARTED = `${MODULE_PREFIX}/ADD_SCENE_STARTED`;
const ADD_SCENE_FINISHED = `${MODULE_PREFIX}/ADD_SCENE_FINISHED`;
const REMOVE_SCENE_STARTED = `${MODULE_PREFIX}/REMOVE_SCENE_STARTED`;
const REMOVE_SCENE_FINISHED = `${MODULE_PREFIX}/REMOVE_SCENE_FINISHED`;

const initialState = {
 loading: false,
 scene: [],
};

export function reducer(state = initialState, action = {}) {
 switch (action.type) {
   case LOAD_SCENE_STARTED:
   case EDIT_SCENE_STARTED:
   case ADD_SCENE_STARTED:
   case REMOVE_SCENE_STARTED:
   {
    return {...state, loading: true};
   }
   case LOAD_SCENE_FINISHED: {
    return {...state, loading: false, scene: action.payload};
   }
   case EDIT_SCENE_FINISHED: {
    const idxToChange = state.scene.findIndex(s => s._id === action.payload._id);
    state.scene[idxToChange] = action.payload;
    return {...state, loading: false, scene: state.scene};

   }
   case ADD_SCENE_FINISHED: {
    const newScene = action.payload;
    return {...state, loading: false, scene:[...state.scene, newScene]};
   }
   case REMOVE_SCENE_FINISHED: {
    const idxToRemove = state.scene.findIndex(s => s._id === action.payload._id);
    state.scene.splice(idxToRemove, 1);
    return {...state, loading: false, scene: state.scene};
   }
   default: return state;
 }
```

```
}

export function loadScenes() {
  return (dispatch) => {
    dispatch({
      type: LOAD_SCENE_STARTED,
    });
    apis.getScene()
      .then((resp) => {
        dispatch({
          type: LOAD_SCENE_FINISHED,
          payload: resp.data,
        });
      })
      .catch((e) => {
        dispatch({
          type: LOAD_SCENE_FINISHED,
          payload: [],
        });
      });
  };
}
export function editScene(scene) {
  return (dispatch) => {
    dispatch({
      type: EDIT_SCENE_STARTED,
    });
    apis.editScene(scene)
      .then((resp) => {
        dispatch({
          type: EDIT_SCENE_FINISHED,
          payload: resp.data,
        });
      })
      .catch((e) => {
        dispatch({
          type: EDIT_SCENE_FINISHED,
          payload: null,
        });
      });
  };
}

export function addScene(scene) {
  return (dispatch) => {
    dispatch({
      type: ADD_SCENE_STARTED,
    });
    apis.addScene(scene)
      .then((resp) => {
```

```
      dispatch({
        type: ADD_SCENE_FINISHED,
        payload: resp.data,
      });
    })
    .catch((e) => {
      dispatch({
        type: ADD_SCENE_FINISHED,
        payload: null,
      });
    });
 };
}

export function removeScene(scene) {
 return (dispatch) => {
   dispatch({
     type: REMOVE_SCENE_STARTED,
   });
   apis.removeScene(scene._id)
     .then((resp) => {
       dispatch({
         type: REMOVE_SCENE_FINISHED,
         payload: resp.data,
       });
     })
     .catch((e) => {
       dispatch({
         type: REMOVE_SCENE_FINISHED,
         payload: null,
       });
     });
 };
}
```

1.5. rooms.reducer

```
import Apis from '../../api/api';
const apis = new Apis();

const MODULE_PREFIX = '/rooms';
const LOAD_ROOMS_STARTED = `${MODULE_PREFIX}/LOAD_ROOMS_STARTED`;
const LOAD_ROOMS_FINISHED = `${MODULE_PREFIX}/LOAD_ROOMS_FINISHED`;

const initialState = {
 loading: false,
 rooms: [],
};

export function reducer(state = initialState, action = {}) {
```

```
 switch (action.type) {
   case LOAD_ROOMS_STARTED:
   {
    return {...state, loading: true};
   }
   case LOAD_ROOMS_FINISHED: {
    return {...state, loading: false, rooms: action.payload}
   }
   default: return state;
 }
}

export function loadRooms() {
 return (dispatch) => {
   dispatch({
    type: LOAD_ROOMS_STARTED,
   });
   apis.getRooms()
    .then((resp) => {
     dispatch({
       type: LOAD_ROOMS_FINISHED,
       payload: resp.data,
     });
    })
    .catch((e) => {
     dispatch({
       type: LOAD_ROOMS_FINISHED,
       payload: [],
     });
    });
 };
}
```

1.6. device.reducer

```
import Apis from '../../api/api';
const apis = new Apis();
const MODULE_PREFIX = '/devices';
const LOAD_DEVICE_STARTED = `${MODULE_PREFIX}/LOAD_DEVICE_STARTED`;
const LOAD_DEVICE_FINISHED = `${MODULE_PREFIX}/LOAD_DEVICE_FINISHED`;
const EDIT_DEVICE_STARTED = `${MODULE_PREFIX}/EDIT_DEVICE_STARTED`;
const EDIT_DEVICE_FINISHED = `${MODULE_PREFIX}/EDIT_DEVICE_FINISHED`;

const initialState = {
 loading: false,
 device: [],
};
export function reducer(state = initialState, action = {}) {
 switch (action.type) {
   case LOAD_DEVICE_STARTED:
```

```
    case EDIT_DEVICE_STARTED:
    {
     return {...state, loading: true};
    }
    case LOAD_DEVICE_FINISHED: {
     return {...state, loading: false, device: action.payload}
    }
    case EDIT_DEVICE_FINISHED: {
     const idxToChange = state.device.findIndex(d => d._id === action.payload._id);
     state.device[idxToChange] = action.payload;
     return {...state, loading: false, device: state.device}
    }
    default: return state;
  }
}
export function loadDevices() {
  return (dispatch) => {
    dispatch({
     type: LOAD_DEVICE_STARTED,
    });
    apis.getDevice()
     .then((resp) => {
      dispatch({
       type: LOAD_DEVICE_FINISHED,
       payload: resp.data,
      });
     })
     .catch((e) => {
      dispatch({
       type: LOAD_DEVICE_FINISHED,
       payload: [],
      });
     });
  };
}
export function editDevice(device, timeOut=0) {
  return (dispatch) => {
    dispatch({
     type: EDIT_DEVICE_STARTED,
    });
    setTimeout(()=> {
     apis.editDevice(device)
     .then((resp) => {
      dispatch({
       type: EDIT_DEVICE_FINISHED,
       payload: resp.data,
      });
     })
     .catch((e) => {
      dispatch({
```

```
        type: EDIT_DEVICE_FINISHED,
        payload: null,
       });
     });
    }, timeOut*60000)
  };
}
```

1.7. user.reducer

```
import Apis from '../../api/api';
import Cookie from "js-cookie";
const apis = new Apis();
const MODULE_PREFIX = '/scene';
const LOAD_USER_STARTED = `${MODULE_PREFIX}/LOAD_USER_STARTED`;
const LOAD_USER_FINISHED = `${MODULE_PREFIX}/LOAD_USER_FINISHED`;
const ADD_USER_STARTED = `${MODULE_PREFIX}/ADD_USER_STARTED`;
const ADD_USER_FINISHED = `${MODULE_PREFIX}/ADD_USER_FINISHED`;

const initialState = {
  loading: false,
  user: null,
};
export function reducer(state = initialState, action = {}) {
  switch (action.type) {
    case LOAD_USER_STARTED:
    case ADD_USER_STARTED:
    {
      return {...state, loading: true};
    }
    case ADD_USER_FINISHED:
    case LOAD_USER_FINISHED: {
      Cookie.set("token", action.payload.token);
      return {...state, loading: false, user: {id: action.payload._id, email: action.payload.userEmail}};
    }
    default: return state;
  }
}
export function logIn(userData) {
  return (dispatch) => {
    dispatch({
      type: LOAD_USER_STARTED,
    });
    apis.getUser(userData)
      .then((resp) => {
        dispatch({
          type: LOAD_USER_FINISHED,
          payload: resp.data,
        });
      })
      .catch((e) => {
```

```
        dispatch({
          type: LOAD_USER_FINISHED,
          payload: null,
        });
      });
  };
}
export function addUser(userData) {
  return (dispatch) => {
    dispatch({
      type: ADD_USER_STARTED,
    });
    apis.addUser(userData)
      .then((resp) => {
        dispatch({
          type: ADD_USER_FINISHED,
          payload: resp.data,
        });
      })
      .catch((e) => {
        dispatch({
          type: ADD_USER_FINISHED,
          payload: null,
        });
      });
  };
}
```

1.2. React компоненти та відображення

1.2.1. index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <title>SmartHome</title>
    <link rel="stylesheet" type="text/css" href="https://cdn.rawgit.com/alpertuna/react-metismenu/master/dist/react-metismenu-standart.min.css"
/>
    <link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.5.0/css/font-awesome.min.css" />
    <link rel="stylesheet" type="text/css" href="./slider.css" />
    <link rel="stylesheet" type="text/css" href="./slider-animations.css" />
    <link rel="stylesheet" type="text/css" href="./react-picky-date-time.min.css" />
  </head>
  <body>
    <div id="root" style="overflow: hidden;"></div>
  </body>
</html>
```

1.2.2. index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import { Provider } from 'react-redux';
import { ConnectedRouter } from 'react-router-redux';
import configureStore from './redux/configureStore';
import createHistory from 'history/createBrowserHistory';
import App from './components/App';

const history = createHistory();
const store = configureStore({ history });
const Root = () => (
  <Provider store={store}>
    <ConnectedRouter key="ConnectedRouter" history={history}>
      <App/>
    </ConnectedRouter>
  </Provider>
);
ReactDOM.render(<Root />, document.getElementById('root'));
```

1.2.3. app.js

```
import React, { Component } from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Menu from './BurgerMenu';
import NotFound from './Pages/NotFound';
import HomePage from './Pages/Home';
import DeviceControl from './Pages/DeviceControl';
import SceneCreation from './Pages/SceneCreation';
import SceneObservation from './Pages/SceneObservation';
import Security from './Pages/Security';
import Climat from './Pages/Climat';
import ModalWindow from './Modal';
import 'react-dropdown/style.css';

class App extends Component {
  render() {
    return (
      <Router>
        <div>
          <div>
            <Menu menuProps={{ skewY: -5, bottomOffset: -30 }}/>
            <ModalWindow />
            <Switch>
              <Route path="/" exact component={HomePage}/>
              <Route path="/devices" exact component={DeviceControl}/>
              <Route path="/new-scene" exact component={SceneCreation}/>
              <Route path="/scenes" exact component={SceneObservation}/>
```

```
                <Route path="/security" exact component={Security}/>
                <Route path="/temp" exact component={Climat}/>
                <Route path='/*' component={NotFound} />
              </Switch>
            </div>
          </div>
        </Router>
      );
    }
  }

  export default App;
```

1.2.4. Authentication page

```
import React, { Component } from 'react';
import { logIn, addUser } from '../../../redux/modules/user.reducer';
import { connect } from 'react-redux';
import style from './style.css';

class UserAuthentication extends Component {
  constructor(props) {
    super(props);
    this.state = {
      password: '',
      passwordRepeat: '',
      email: '',
      showEmailError: false,
      showPasswordError: false,
      showSuccessMessage: false,
      isSignIn: false,
      isSignUp: false
    }
  }
  shouldComponentUpdate() {
    return true;
  };
  componentWillUpdate(nextProps) {
    if(nextProps.user) return nextProps.user;
  };
  onPasswordChange = (event) => {
    const password = event.target.value;
    this.setState(password === this.state.passwordRepeat ? {password: event.target.value, showPasswordError: false} : {password: event.target.value, showPasswordError: true});
  }
  onRepeatPesswordChange = (event) => {
     const pass = event.target.value;
     this.setState(pass === this.state.password ? {passwordRepeat: pass, showPasswordError: false} : {passwordRepeat: '', showPasswordError: true});
  }
```

```
    onEmailChange = (event) => {
      const userEmail = event.target.value;
      this.setState(this.isValid(userEmail) ? {email: userEmail, showEmailError: false} : {email: this.state.email, showEmailError: true});
    }
    isValid = (email) => {
      const re = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
      return re.test(String(email).toLowerCase());
    }
    isSighnInButtonActive = () => this.isValid(this.state.email) && this.state.password && this.state.isSignIn;
    isSighnUpButtonActive = () => this.isValid(this.state.email) && this.state.password && this.state.passwordRepeat && this.state.isSignUp;
    handleSighnIn = () => {
      this.setState({
       showSuccessMessage: true,
       password: '',
       passwordRepeat: '',
       email: '',
      });
      this.props.logIn({email: this.state.email, password: this.state.password});
    }
    handleSighnUp = () => {
      this.setState({
       showSuccessMessage: true,
       password: '',
       passwordRepeat: '',
       email: '',
      });
      this.props.addUser({_id: `_${Math.random().toString(36).substr(1, 8)}`, userEmail: this.state.email, password: this.state.password});
    }
    render() {
      const user = this.props.user;
      return (
        <div className={ style.cont }>
          <p>Увійти:</p>
          <form className={style.form} onChange={() => this.setState({isSignIn: true})}>
            <span>Email:</span>
            <input type='text' onChange={this.onEmailChange}/>
            <div className={style.error}>{this.state.showEmailError && this.state.isSignIn ? '*Введено некорректний імейл' : ''}</div>
            <span>Пароль:</span>
            <input type='password' onChange={this.onPasswordChange}/>
            <button onClick = {this.handleSighnIn} disabled = {!this.isSighnInButtonActive()}>Увійти</button>
            <div className={style.success}>{this.state.showSuccessMessage && this.state.isSignIn && user ? 'Ви успішно пройшли авторизацію' : ''}</div>
            <div className={style.error}>{this.state.showSuccessMessage && this.state.isSignIn && !user ? 'Ви не авторизовані в системі' : ''}</div>
          </form>
          <p>Немає аккаунту? Зареєструватись:</p>
          <form className={style.form} onChange={() => this.setState({isSignUp: true})}>
            <span>Email:</span>
            <input type='text' onChange={this.onEmailChange}/>
```

```
                <div className={style.error}>{this.state.showEmailError && this.state.isSignUp ? '*Введено некорректний імейл' : ''}</div>
                <span>Пароль:</span>
                <input type='password' onChange={this.onPasswordChange}/>
                <span>Повторити пароль:</span>
                <input type='password' onChange={this.onRepeatPesswordChange}/>
                <div className={style.error}>{this.state.showPasswordError && this.state.isSignUp ? '*Введені паролі не співпадають' : ''}</div>
                <button onClick = {this.handleSighnUp} disabled = {!this.isSighnUpButtonActive()}>Зареєструватись</button>
                <div className={style.success}>{this.state.showSuccessMessage && this.state.isSignUp && user ? 'Ви успішно пройшли реєстрацію' : ''}</div>
                <div className={style.error}>{this.state.showSuccessMessage && this.state.isSignUp && !user ? 'Виникла помилка, спробуйте ще раз' : ''}</div>
              </form>
            </div>
          )
        }
      }
      const mapStateToProps = (state) => {
        const {user, loading} = state.user;
        return {
          user,
          loading
        };
      };
      const mapDispatchToProps = {
        logIn, addUser
      };
      export default connect(mapStateToProps, mapDispatchToProps)(UserAuthentication);
```

1.2.5 Climat control page

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { loadDevices } from '../../../redux/modules/device.reducer';
import { loadRooms } from '../../../redux/modules/rooms.reducer';
import Collapsible from 'react-collapsible';
import style from './style.css';

class Climat extends Component {
  componentWillMount (){
    this.props.loadDevices();
    this.props.loadRooms();
  };
  render() {
    const device = this.props.device ? this.props.device : [];
    const rooms = this.props.rooms ? this.props.rooms : [];
    return (
      <div className={ style.cont }>
        <div className={style.cover}></div>
        <div className={style.header}>Рівень температури та вологості</div>
        {
```

```
          rooms.map((room) => {
            const roomDevice = device.filter(d => d.room === room.title).find(d=> d.type === 'temp');
            return (
              <Collapsible className={style.collapsible} trigger={room.title} key={room._id}>
                <div key={roomDevice._id} className={style.deviceContainer}>
                  <div className={style.deviceName}>
                    Рівень температури: {roomDevice.payload_T}C
                  </div>
                  <div className={style.deviceName}>
                    Рівень волості: {roomDevice.payload_W}%
                  </div>
                </div>
              </Collapsible>
            )
          })
        }
      </div>
    )
  }
}
const mapDispatchToProps = {
  loadDevices, loadRooms
};
const mapStateToProps = (state) => {
const {device, loading} = state.device;
const {rooms} = state.rooms;
return {
  device,
  loading,
  rooms
};
};
export default connect(mapStateToProps, mapDispatchToProps)(Climat);
```

1.2.6. DeviceControl page

```
import React, { Component } from 'react';
import style from './style.css';
import DeviceList from '../../DeviceList';
export default class DeviceControl extends Component {
  render() {
    return (
      <div className={ style.cont }>
        <div className={ style.cover }></div>
        <div className={ style.header }>Управління приладами</div>
        <DeviceList />
      </div>
    )
  }
}
```

1.2.7. HomePage

```
import React from 'react';
import Slider from 'react-animated-slider';
import AuthenticationWindow from '../../AuthWindow';

const content = [
  {
    name: 'Охорона',
    url: '/security',
    button: 'Перейти',
    text: '',
    image: 'https://www.flaticon.com/svg/static/icons/svg/1161/1161389.svg',
  },
  {
    name: 'Клімат контроль',
    url: '/temp',
    button: 'Перейти',
    text: '',
    image: 'https://www.flaticon.com/svg/static/icons/svg/1287/1287963.svg',
  },
  {
    name: 'Управління приладами',
    url: '/devices',
    button: 'Перейти',
    text: '',
    image: 'https://www.flaticon.com/svg/static/icons/svg/2005/2005391.svg',
  },
  {
    name: 'Сценарії',
    url: '/scenes',
    button: 'Перейти',
    text: '',
    image: 'https://www.flaticon.com/svg/static/icons/svg/3063/3063816.svg',
  },
]
const HomePage = () => (
  <div className="wrapper">
    <AuthenticationWindow />
    <Slider className="slider-wrapper">
      {content.map((item, index) => (
        <div
          key={index}
          className="slider-content"
          style={{ background: `url('${item.image}') no-repeat center center` }}
        >
          <div className="inner">
            <h1>{item.name}</h1>
            <p>{item.text}</p>
```

```
          <a href={item.url}><button>{item.button}</button></a>
        </div>
      </div>
    ))}
  </Slider>
 </div>
)

export default HomePage;
```

1.2.8. SceneCreation page

```
import React, { Component } from 'react';
import { openModal } from '../../../redux/modules/modals.reducer';
import { addScene } from '../../../redux/modules/scene.reducer';
import { connect } from 'react-redux';
import style from './style.css';
import SceneForm from '../../SceneForm';
import DatePicker from '../../DateTimePicker/DatePicker';
import TimePicker from '../../DateTimePicker/TimePicker';
import RemoveButton from '../../RemoveBtn';
import StatefullButton from '../../StatefullButton';
import Switch from 'react-ios-switch';

class SceneCreation extends Component {
 constructor(props) {
  super(props);
  this.state = {
   title: '',
   startDay: '',
   startTime: '',
   actions: [],
   repeatDaily: false,
   repeatWeekly: false,
  }
 }
 openModalWindow = () => this.props.openModal(<SceneForm onActionCreate={this.showAction}/>)
 changeTitle = (event) => this.setState({title: event.target.value})
 changeStartDate = (event) => this.setState({startDay: event.target.value})
 changeStartTime = (event) => this.setState({startTime: event.target.value})
 showAction = (newAction) => this.setState({actions: [... this.state.actions, newAction]})
 addDate = (date) => this.setState({startDay: date})

 addTime = (time) => this.setState({startTime: time})
 removeAction = (event) => {
  const idx = event.target.value;
  const actions = this.state.actions;
  actions.splice(idx, 1);
  this.setState({actions: actions});
 }
```

```
isSaveActionDisabled = () =>
    this.state.actions.length === 0 ||
    this.state.startDay === '' ||
    this.state.startTime === '' ||
    this.state.title === '';
saveScene = () => {
  const date = this.state.startDay.split('/');
  const UTCDateTime = `${date[1]}/${date[0]}/${date[2]} ${this.state.startTime}`;
  const payload = {
    _id: `_${Math.random().toString(36).substr(2, 9)}`,
    name: this.state.title,
    startDate: new Date(UTCDateTime),
    isActive: true,
    actions: this.state.actions,
    repeatDaily: this.state.repeatDaily,
    repeatWeekly: this.state.repeatWeekly
  };
  this.props.addScene(payload);
  this.setState({
    title: '',
    startDay: '',
    startTime: '',
    actions: [],
    repeatDaily: false,
    repeatWeekly: false,
  })
}
render() {
  return (
    <div className={ style.cont }>
      <input type='text' placeholder='Введіть назву сценарію*' value = {this.state.title} onChange = {this.changeTitle} />
      <DatePicker onDateSelect = {this.addDate}/>
      <TimePicker onTimeSelect = {this.addTime} />
      <div className={style.switchContainer}>
        <span>Повторювати щодня   </span>
        <Switch
          checked={this.state.repeatDaily}
          onChange={() => {this.setState({repeatDaily: !this.state.repeatDaily})}}
        />
      </div>
      <div className={style.switchContainer}>
        <span>Повторювати щотижня   </span>
        <Switch
          checked={this.state.repeatWeekly}
          onChange={() => {this.setState({repeatWeekly: !this.state.repeatWeekly})}}
        />
      </div>
      <h3>{this.state.title} {this.state.startDay} {this.state.startTime}</h3>
      <ol className={style.sceneList}>
        {
```

```
          this.state.actions.map((action, index) => {
            const timeoutMassege = action.timeOut > 0 ? `через ${action.timeOut} хвилин` : '';
            return (
              <li key={index}>
                Увімкнути {action.device.title.toLowerCase()} в кімнаті "{action.device.room}" {timeoutMassege}
                <RemoveButton value={index} onClick = {this.removeAction}/>
              </li>
            )
          })
        }
      </ol>
      <div className={style.addBtn} onClick = {this.openModalWindow}> Додати дію* </div>
      <div className={style.note}>* - дані, які необхідно заповнити</div>
      <StatefullButton disabled={this.isSaveActionDisabled()} onClick={this.saveScene} name='Зберегти сценарій' />
    </div>
  )
 }
}
const mapDispatchToProps = {
 openModal, addScene
};
export default connect(null, mapDispatchToProps)(SceneCreation);
```

1.2.9. SceneObservation page

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { loadScenes, editScene, removeScene } from '../../../redux/modules/scene.reducer';
import Collapsible from 'react-collapsible';
import Switch from 'react-ios-switch';
import style from './style.css';

class SceneObservation extends Component {
 componentWillMount (){
  this.props.loadScenes();
 };
 shouldComponentUpdate() {
  return true;
 };
 componentWillUpdate(nextProps) {
  if(nextProps.scene) return nextProps.scene;
 };
 sceneToggle(scene) {
  scene.isActive = !scene.isActive;
  this.props.editScene(scene);
 }
 deleteScene(scene) {
```

```
    this.props.removeScene(scene);
  }
  render() {
    const scenes = this.props.scene ? this.props.scene : [];
    return (
      <div className={ style.cont }>
        <div className={style.cover}></div>
        <div className={style.header}>Доступні сценарії:</div>
        {
          scenes.map(scene => {
            const startDate = new Date(scene.startDate);
            const startDay = () => {
              switch(startDate.getDay()) {
                case 0: return "неділю";
                case 1: return "понеділок";
                case 2: return "вівторок";
                case 3: return "середу";
                case 4: return "четвер";
                case 5: return "п'ятницю";
                case 6: return "суботу";
              }
            };
            const sceneCount = scene.actions.length;
            return (
              <Collapsible className={style.collapsible} trigger={scene.name} key={scene._id}>
                <div className={style.sceneContainer}>
                  <ul>
                    <li>
                      Почати сценарій в {startDay()}  
{startDate.getDate()}/{startDate.getUTCMonth()+1}/{startDate.getUTCFullYear()}
                       o {startDate.getUTCHours()}:{startDate.getUTCMinutes().length > 1 ? startDate.getUTCMinutes() :
startDate.getUTCMinutes()+'0'}
                    </li>
                    <li>
                      {scene.repeatDaily || scene.repeatWeekly ? 'Повторювати' : ''} {scene.repeatDaily ? 'щодня' :
''} {scene.repeatWeekly ? 'щотижня' : ''}
                    </li>
                    <li>
                      <p>{scene.isActive ? 'Сценарій активний, вимкнути:' : 'Сценарій неактивний, увімкнути:'}</p>
                      <Switch
                        checked={scene.isActive}
                        onChange={() => this.sceneToggle(scene)}
                      />
                    </li>
                    <li>
```

```
                    <p>
                      <strong>Опис сценарію:</strong>
                    </p>
                    <div>
                      {
                        scene.actions.map((action,  i) => {
                          return (
                            <p key={i}>
                              Увімкнути {action.device.title.toLowerCase()}  в кімнаті "{action.device.room}"{action.timeOut  > 0 ? 'через
' + action.timeOut  + ' хвилин.' : '.'}
                            </p>
                          )
                        })
                      }
                    </div>
                  </li>
                  <li>
                    <button className={style.removeBtn}  onClick={() => this.deleteScene(scene)}>Видалити  сценарій</button>
                  </li>
                </ul>
              </div>
            </Collapsible>
          )
        })
      }
    </div>
  )
 }
}
const mapDispatchToProps = {
 loadScenes, editScene, removeScene
};
const mapStateToProps  = (state) => {
 const {scene, loading}  = state.scene;
 return {
   scene,
   loading,
 };
};
export default connect(mapStateToProps,  mapDispatchToProps)(SceneObservation);
```

1.2.10 Security page

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
```

```
import { loadDevices, editDevice } from '../../../redux/modules/device.reducer';
import style from './style.css';
import StatusButton from '../../StatusButton';

class Security extends Component {
 componentWillMount (){
   this.props.loadDevices();
 };
 shouldComponentUpdate() {
   return true;
 };
 componentWillUpdate(nextProps) {
   if(nextProps.device) return nextProps.device;
 };
 deviceSwitch = (device) => {
   device.status = !device.status;
   this.props.editDevice(device);
 };
 render() {
   const device = this.props.device ? this.props.device : [];
   const securitySystem = device.filter(d => d.type==='security');
   return (
       securitySystem.map((device, idx) => {
       const deviceStateMessage = device.status ? 'Система охорони увімкнена' : 'Система охорони вимкнена';
       const buttonName = device.status ? 'Вимкнути' : 'Увімкнути';
       const buttonClassName = device.status ? style.btnOn : style.btnOff;
       const iconClassName = device.status ? style.securityIcon : style.noSecureIcon;
       const switchDevice = () => {this.deviceSwitch(device)};
       return(
         <div className={ style.cont } key={idx}>
           <div className={iconClassName}>
             <div className={style.cover}></div>
             <div className={style.header}>{deviceStateMessage}</div>
             <StatusButton className={buttonClassName} onClick = {switchDevice} name={buttonName}></StatusButton>
           </div>
         </div>
       )
      })
   )
 }
}
const mapDispatchToProps = {
 loadDevices, editDevice
};
const mapStateToProps = (state) => {
const {device, loading} = state.device;
return {
 device,
 loading,
};
```

```
};
export default connect(mapStateToProps, mapDispatchToProps)(Security);
```

1.2.11. Authentication Modal Window

```
import React, {Component} from 'react';
import { connect } from 'react-redux';
import CloseButton from '../CloseButton';
import style from './style.css';
import Modal from 'react-modal';
import UserAuthentication from '../Pages/Authentication';

class AuthenticationWindow extends Component{
 shouldComponentUpdate() {
   return true;
 };
 componentWillUpdate(nextProps) {
   if(nextProps.user) return nextProps.user;
 };
 render(){
   let isVisible = this.props.user ? false : true;
   const user = this.props.user;
   return (
     <Modal
       isOpen={isVisible}
       onRequestClose={() => isVisible = false}
       className = {style.popup}>
       <div className = {style.popupContent}>
         <div className = {style.popupForm}>
           <UserAuthentication/>
         </div>
       </div>
     </Modal>
   )
 }
}
const mapStateToProps = (state) => {
 const {user, loading} = state.user;
 return {
   user,
   loading
 };
};
export default connect(mapStateToProps, null)(AuthenticationWindow);
```

1.2.12. Menu

```
import React, { Component } from 'react';
import MetisMenu from 'react-metismenu';
```

```
import style from './style.css';

const content = [
    {
      icon: 'bars',
      label: '',
      id: 1,
      content: [
        {
          label: 'Охорона',
          to: '/security',
          id: 2,
          parentId: 1,
        },
        {
         label: 'Клімат контроль',
         to: '/temp',
         id: 3,
         parentId: 1,
        },
        {
         label: 'Управління приладами',
         id: 4,
         parentId: 1,
         to: '/devices',
        },
        {
         label: 'Сценарії',
         id: 8,
         parentId: 1,
         content: [
          {
           label: 'Переглянути сценарії',
           to: '/scenes',
           id: 9,
           parentId: 8,
          },
          {
           label: 'Новий сценарій',
           to: '/new-scene',
           id: 10,
           parentId: 8,
          },
         ]
        },
      ],
    },
]
class Menu extends Component {
  render() {
```

```
    return <MetisMenu
     content={content}
     activeLinkFromLocation
     className={style.menu}
     clasNameLink={style.link}
    />
  }
};
export default Menu;
```

1.2.13. DatePicker

```
import PickyDateTime from 'react-picky-date-time';
import React, { Component }  from 'react';
import style from '../style.css';

class DatePicker extends Component {
   constructor(props) {
     super(props);
     this.state = {
       showPickyDateTime: false,
       date: new Date().getDay() + 1,
       month: new Date().getMonth() + 1,
       year: new Date().getFullYear(),
     };
   }
   onDatePicked = (res) => {
     const { date, month, year } = res;
     this.setState({ year: year, month: month, date: date });
     this.props.onDateSelect(`${date}/${month}/${year}`);
   }
   onClose = () => {
     this.setState({showPickyDateTime: false});
   }
   toggleCalendar = () => {
     const isVisible = !this.state.showPickyDateTime;
     this.setState({showPickyDateTime: isVisible});
   }
   render() {
     const {
       showPickyDateTime,
       date,
       month,
       year,
     } = this.state;

     return(
       <div>
         <span className = {style.pickerToggle} onClick = {this.toggleCalendar}>Обрати день початку сценарію*</span>
         <PickyDateTime
```

```
          size="xs"
          mode={0}
          locale={`en-us`}
          show={showPickyDateTime}
          onClose={() => this.onClose()}
          defaultDate={`${month}/${date}/${year}`}
          onDatePicked={res => this.onDatePicked(res)}
          onResetDate={res => this.onDatePicked(res)}
          onResetDefaultDate={res => this.onDatePicked(res)}
        />
      </div>
    );
  }
}
export default DatePicker;
```

1.2.14. Time Picker

```
import PickyDateTime from 'react-picky-date-time';
import React, { Component } from 'react';
import style from '../style.css';

class TimePicker extends Component {
 constructor(props) {
   super(props);
   this.state = {
     showPickyDateTime: false,
     hour: '12',
     minute: '00',
     second: '00',
     meridiem: 'PM'
   };
 }
 onSecondChange = (res) => {
   this.setState({ second: res.value });
   this.props.onTimeSelect(`${this.state.hour}:${this.state.minute} ${this.state.meridiem}`);
 }
 onMinuteChange = (res) => {
   this.setState({ minute: res.value });
   this.props.onTimeSelect(`${this.state.hour}:${this.state.minute} ${this.state.meridiem}`);
 }
 onHourChange = (res) => {
   this.setState({ hour: res.value });
   this.props.onTimeSelect(`${this.state.hour}:${this.state.minute} ${this.state.meridiem}`);
 }
 onMeridiemChange = (res) => {
   this.setState({ meridiem: res });
   this.props.onTimeSelect(`${this.state.hour}:${this.state.minute} ${this.state.meridiem}`);
 }
 onSetTime = (res) => {
```

```
    this.setState({
      showPickyDateTime: false,
      second: res.clockHandSecond.value,
      minute: res.clockHandMinute.value,
      hour: res.clockHandHour.value
    });
    this.props.onTimeSelect(`${this.state.hour}:${this.state.minute} ${this.state.meridiem}`);
  }
  toggleClock = () => {
    const isVisible = !this.state.showPickyDateTime;
    this.setState({showPickyDateTime: isVisible});
  }
  onClose = () => {
    this.setState({showPickyDateTime: false});
  }
  render() {
    const {
      showPickyDateTime,
      second,
      minute,
      hour,
    } = this.state;
    return(
      <div>
        <span className = {style.pickerToggle} onClick = {this.toggleClock}>Обрати час початку сценарію*</span>
        <PickyDateTime
          size="xs"
          mode={2}
          locale="en-us"
          show={this.state.showPickyDateTime}
          onClose={() => this.onClose()}
          onSecondChange={res => this.onSecondChange(res)}
          onMinuteChange={res => this.onMinuteChange(res)}
          onHourChange={res => this.onHourChange(res)}
          onMeridiemChange={res => this.onMeridiemChange(res)}
          onResetTime={res => this.onSetTime(res)}
          onResetDefaultTime={res => this.onSetTime(res)}
          onClearTime={res => this.onSetTime(res)}
        />
      </div>
    );
  }
}
export default TimePicker;
```

1.2.15. Device List

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { loadDevices, editDevice } from '../../redux/modules/device.reducer';
```

```
import { loadRooms } from '../../redux/modules/rooms.reducer';
import Collapsible from 'react-collapsible';
import StatusButton from '../StatusButton';
import style from './style.css';

class DeviceList extends Component {
 componentWillMount (){
  this.props.loadDevices();
  this.props.loadRooms();
 };
 shouldComponentUpdate() {
  return true;
 };

 componentWillUpdate(nextProps) {
  if(nextProps.device) return nextProps.device;
 };
 deviceSwitch = (device, timeOut) => {
  device.status = !device.status;
  this.props.editDevice(device, timeOut);
 };
 render() {
  const device = this.props.device ? this.props.device : [];
  const rooms = this.props.rooms ? this.props.rooms : [];
  return (
    <div >
      {
        rooms.map((room) => {
          const roomDevices = device.filter(d => d.room===room.title).filter(d=> d.type !== 'temp');
          return (
            <Collapsible className={style.collapsible} trigger={room.title} key={room._id}>
              {
                roomDevices.map(device => {
                  const switchDevice = () => {this.deviceSwitch(device, state.switchTimeOut)};
                  const buttonName = device.status ? 'Вимкнути' : 'Увімкнути';
                  const buttonClassName = device.status ? style.btnOn : style.btnOff;
                  const state = {
                    switchTimeOut: 0
                  };
                  const handleTimeOutChange = (event) => {
                    state.switchTimeOut = event.target.value;
                  };
                  return(
                    <div key={device._id} className={style.deviceContainer}>
                      <span className={style.deviceName}>
                        {device.title.toUpperCase()}
                      </span>
                      <span>{buttonName} через</span>
                      <input
                        className={style.timeOutInput}
```

```
                                type='text'
                                onChange = {handleTimeOutChange}>
                              </input>
                                хвилин
                              <StatusButton className={buttonClassName} onClick = {switchDevice} name={buttonName}></StatusButton>
                            </div>

                          )
                        })
                      }
                    </Collapsible>
                  )
                })
              }
            </div>
          )
        }
}

const mapDispatchToProps = {
   loadDevices, editDevice, loadRooms
};
const mapStateToProps = (state) => {
 const {device, loading} = state.device;
 const {rooms} = state.rooms;
 return {
   device,
   rooms,
   loading,
 };
};
export default connect(mapStateToProps, mapDispatchToProps)(DeviceList);
```

1.2.16. Modal Window

```
import React, {Component} from 'react';
import { connect } from 'react-redux';
import CloseButton from '../CloseButton';
import { closeModal } from '../../redux/modules/modals.reducer';
import style from './style.css';
import PropTypes from 'prop-types';
import Modal from 'react-modal';

Modal.setAppElement('#root');
class ModalWindow extends Component{
 static propTypes = {
   modal: PropTypes.object,
 };
 close = () => {
   this.props.closeModal();
```

```
 };
 render(){
   const { isVisible } = this.props;
   return (
     <Modal
       isOpen={isVisible}
       onRequestClose={this.close}
       className = {style.popup}>
       <div className = {style.popupContent}>
         <div className = {style.close}>
           <CloseButton
             onClick = {this.close}
           />
         </div>
         <div className = {style.popupForm}>
           {this.props.content}
         </div>
       </div>
     </Modal>
   )
 }
}
const mapDispatchToProps = {
 closeModal,
};
const mapStateToProps = (state) => {
 const {isVisible, content} = state.modal;
 return {
   isVisible,
   content,
 };
};
export default connect(mapStateToProps, mapDispatchToProps)(ModalWindow);
```

1.2.17. Button

```
import React from 'react';
const StatefullButton=(props) => (
  <button
    type = "button"
    disabled = {false || props.disabled}
    className = {props.className}
    onClick = {props.onClick}>
    {props.name}
  </button>
)
export default StatefullButton;
```

1.2.18. Scene Adding Form

```
import React, { Component }  from 'react';
import { connect } from 'react-redux';
import { loadDevices } from '../../redux/modules/device.reducer';
import { closeModal } from '../../redux/modules/modals.reducer';
import { loadRooms } from '../../redux/modules/rooms.reducer';
import Select from 'react-select';
import style from './style.css';

class SceneForm extends Component {
 constructor(props) {
  super(props);
  this.state = {
   device: null,
   room: '',
   timeOut: 0
  }
 }
 componentWillMount (){
  this.props.loadDevices();
  this.props.loadRooms();
 };
 shouldComponentUpdate() {
  return true;
 };
 componentWillUpdate(nextProps) {
  if(nextProps.device) return nextProps.device;
 };
 handleSubmit = (event) => {
  event.preventDefault();
  this.props.onActionCreate({device: this.state.device, timeOut: this.state.timeOut});
  this.setState({device: null, timeOut: 0});
  this.props.closeModal();
 };
 setRoom = (option) => this.setState({room: option.value});
 setDevice = (option) => this.setState({device: option});
 setTimeOut = (event) => this.setState({timeOut: event.target.value});
 isButtonActive = () => this.state.device;
 render() {
  const device = this.props.device ?  this.props.device : [];
  const rooms = this.props.rooms ? this.props.rooms : [];
  const addDevice = (option) => {
   this.setDevice(device.find(d => d._id === option.value))
  };
  return (
   <form className={style.sceneForm}>
    <div>
     У кімнаті:
      <Select
```

```
          className = {style.select}
          options={rooms.map(room => {
            return {value: room.title, label: room.title}
          })}
          onChange={this.setRoom}
        />
      </div>
      <div>
        увімкнути прилад:
        <Select
          className = {style.select}
          options={device.filter(d => d.room === this.state.room && d.type !== 'temp').map(d => {
            return {value: d._id, label: d.title.toLowerCase()}
          })}
          onChange={addDevice}
        />
      </div>
      <div>
        через &amp;nbsp;&amp;nbsp;
        <input
          className={style.timeOutInput}
          type='text'
          value = {this.state.timeOut}
          onChange = {this.setTimeOut}>
        </input>
        &amp;nbsp;&amp;nbsp;хвилин
      </div>
      <button onClick = {this.handleSubmit} disabled = {!this.isButtonActive}>Додати</button>
    </form>
  )
 }
}
const mapDispatchToProps = {
 loadDevices, closeModal, loadRooms
};
const mapStateToProps = (state) => {
 const {device, loading} = state.device;
 const {rooms} = state.rooms;
 return {
   device,
   rooms
 };
};
export default connect(mapStateToProps, mapDispatchToProps)(SceneForm);
```

2. Сервер

2.1. app.js

```
const express = require('express');
const mongoose = require('mongoose').set('debug', true);
const deviceRouter = require('./routes/device');
const roomRouter = require('./routes/room');
const sceneRouter = require('./routes/scene');
const userRouter = require('./routes/user');
const cors = require('cors');
const corsMiddleware = require('./middlewares/cors');
const app = express();
const uri = "mongodb+srv://mhorova:s1984tramri@cluster0.udb16.mongodb.net/smart_home?retryWrites=true&w=majority";
mongoose
  .connect(uri, {useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex: true})
  .then(() => console.log( 'Database Connected' ))
  .catch(err => console.log( err ));
app.use(cors(corsMiddleware));
app.use(express.json());
app.use('/devices', deviceRouter);
app.use('/devices/:id', deviceRouter);
app.use('/rooms', roomRouter);
app.use('/scenes', sceneRouter);
app.use('/scenes/:id', sceneRouter);
app.use('/users', userRouter);
app.use('/users/:data', userRouter);
app.listen(3003);
```

2.2. CORS config

```
const corsMiddleware = {
    origin: 'http://localhost:5100',
    optionsSuccessStatus: 200,
    methods: "GET, PUT, POST, DELETE"
}
module.exports = corsMiddleware;
```

2.3. Authentication config

```
const getTokenFromHeader = (req) => {
  return req.headers.authorization ? req.headers.authorization : null;
}
const isAuth = jwt({
    secret: 'MySuP3R_z3kr3t',
    userProperty: 'token',
    getToken: getTokenFromHeader,
    algorithms: ['HS256']
});
module.exports = isAuth;
```

2.4. Моделі

2.4.1. deviceModel

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const deviceSchema = new Schema({
    _id: mongoose.ObjectId,
    title: String,
    type: String,
    status: Boolean,
    payload: Number,
    room: String
},  { versionKey: false });
const deviceModel = mongoose.model('device', deviceSchema);
module.exports = deviceModel;
```

2.4.2. roomModel

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const roomSchema = new Schema({
    _id: mongoose.ObjectId,
    title: String
});
const roomModel = mongoose.model('room', roomSchema);
module.exports = roomModel;
```

2.4.3. sceneModel

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const sceneSchema = new Schema({
    _id: String,
    name: String,
    startDate: String,
    isActive: Boolean,
    actions: [],
    repeatDaily: Boolean,
    repeatWeekly: Boolean
});
const sceneModel = mongoose.model('scene', sceneSchema);
module.exports = sceneModel;
```

2.4.4. userModel

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const userSchema = new Schema({
    _id: String,
    userEmail: String,
```

```
    password: String
});
const userModel = mongoose.model('user', userSchema);
module.exports = userModel;
```

2.5. Маршрути

2.5.1. DeviceRoutes

```
const router = require('express').Router();
const Device = require('../models/device');
const isAuth = require('../middlewares/isAuth');
const DeviceController = require('../../espruino/controller');

router.get('/', isAuth, async (req, res) => {
    const devices = await Device.find().exec();
    res.json(devices);
});
router.put('/:id', isAuth, async (req, res) => {
    const id = req.params.id;
    const data = req.body;
    try {
        const device = await Device.findById(id);
        const deviceController = new DeviceController(req.body);
        deviceController.switchDevice();
        await device.updateOne(data);
        res.sendStatus(200);
    } catch(e) {
        res.sendStatus(404)
    }
})
module.exports = router;
```

2.5.2. RoomRoutes

```
const router = require('express').Router();
const Room = require('../models/rooms');
const isAuth = require('../middlewares/isAuth');
router.get('/', isAuth, async (req, res) => {
    const rooms = await Room.find().exec();
    res.json(rooms);
});
module.exports = router;
```

2.5.3. SceneRoutes

```
const router = require('express').Router();
const Scene = require('../models/scene');
const isAuth = require('../middlewares/isAuth');
router.get('/', isAuth, async (req, res) => {
```

```
    const scenes = await Scene.find().exec();
    res.json(scenes);
});
router.get('/:id', isAuth, async (req, res) => {
    const sceneId = req.params.id;
    const scene = await Scene.findById(sceneId).exec();
    if(scene) {
        res.json(scene);
    } else {
        res.sendStatus(404);
    }
});
router.put('/:id', isAuth, async (req, res) => {
    const sceneId = req.params.id;
    const sceneData = req.body;
    try {
        const scene = await Scene.findById(sceneId).exec();
        await scene.updateOne(sceneData);
        res.sendStatus(200);
    } catch(e) {
        res.sendStatus(404)
    }
})
router.post('/', isAuth, async (req, res) => {
    const sceneData = req.body;
    const scene = new Scene(sceneData);
    scene.save();
    res.json(scene);
})
router.delete('/:id', isAuth, async (req, res) => {
    const sceneId = req.params.id;
    await Scene.findByIdAndDelete(sceneId);
    res.sendStatus(200);
})
module.exports = router;
```

2.5.4. UserRoutes

```
const router = require('express').Router();
const argon2 = require('argon2');
const jwt = require('jsonwebtoken');
const User = require('../models/user');

function generateToken(id) {
    const data =  {
      _id: id
    };
    const signature = 'MySuP3R_z3kr3t';
    const expiration = '24h';
    return jwt.sign({ data, }, signature, { expiresIn: expiration });
```

```
}
router.get('/', async (req, res) => {
    const users = await User.find().exec();
    res.json(users);
})
router.get('/:data', async (req, res) => {
    const userData =  req.params.data.split('&');
    const users = await User.find().exec();
    const user = users.find(user => user.userEmail == userData[0]);
    if(user) {
        const correctPassword = await argon2.verify(user.password, userData[1]);
        if(correctPassword) {
            const sessionToken = generateToken(user._id);
            res.json({
                userEmail: user.userEmail,
                token: sessionToken
            });
        } else {
            res.json(null).status(404);
        }
    } else {
        res.json(null).status(404);
    }
});
router.post('/', async (req, res) => {
    const password = req.body.password;
    const passwordHashed = await argon2.hash(password);
    const userRecord = await User.create({
        _id: req.body._id,
        userEmail: req.body.userEmail,
        password: passwordHashed
      });
    userRecord.save();
    res.json({
        userEmail: userRecord.userEmail,
        token: sessionToken
    });
})
module.exports = router;
```

3. Управління фізичними приладами

3.1. BluetoothDevice

```
module.exports = class BluetoothDevice {
  constructor(macAddress) {
    this.address = macAddress;
  }
  switchOn = (delay=0) => {
    NRF.connect(this.address)
    .then(connection => {
      setTimeout(() => connection.write('set();\n', () => {
        setTimeout(() => connection.write('NRF.on("disconnect", () => return);\n'), 1500)
      }), delay)
    })
  }
  switchOff = (delay=0) => {
    NRF.connect(this.address)
    .then(connection => {
      setTimeout(() => connection.write('reset();\n', () => {
        setTimeout(() => connection.write('NRF.on("disconnect", () => return);\n'), 1500)
      }), delay)
    })
  }
}
```

3.2. Led

```
const led = require('../espruino-modcat/modules/@amperka/led');
module.exports = class Led {
  constructor(_id) {
    this._id = _id;
  }
  pinOptions = [
    {_id: '5fae77a371ec2f854e31e029', pin: P1},
    {_id: '5fae78a371ec2f854e31e02f', pin: P2},
    {_id: '5fae78fc71ec2f854e31e031', pin: P3},
    {_id: '5fae7a1671ec2f854e31e038', pin: P4},
  ]
  createController = () => {
    const pin = this.pinOptions.find(opt => opt._id === this._id).pin;
    const controller = led.connect(pin);
    return controller;
  }
  switchOn = (delay=0) => {
    const controller = this.createController();
    setTimeout(() => controller.turnOn(), delay);
  }
  switchOff = (delay=0) => {
    const controller = this.createController();
```

```
        setTimeout(() => controller.turnOff(), delay);
    }
}
```

3.3. MeteoControl

```
const meteoControl = require('../espruino-modcat/modules/@amperka/pid');
module.exports = class MeteoControl {
    constructor(data, type) {
        this.data = data;
        this.control = type === 't' ? meteoControl.create({
            target: 25,
            outputMin: 23,
            outputMax: 27
        }) : meteoControl.create({
            target: 55,
            outputMin: 40,
            outputMax: 70
        });
    }
    act = () => {
        this.control.run(() => {
            const input = analogRead(this.data.read);
            const output = control.update(input);
            analogWrite(this.data.write, output);
        }, 60) //interval - 60 sec
    };
}
```

3.4. MeteoSensors

```
const meteoSensor = require('../espruino-modcat/modules/@amperka/meteo-sensor');
const axios = require('axios');
const api = axios.create({
    baseURL: 'http://localhost:3003/',
});
module.exports = class MeteoSensor {
    constructor(_id, pin) {
        this._id = _id;
        this.controller = meteoSensor.connect({
            pin: pin
        })
    }
    getDevice = async () => {
        const device = await api.get(`devices/${this._id}`);
        return device;
    }
    act = () => {
        setInterval(async () => {
            this.controller.read((data) => {
```

```
            const dbDevice = await this.getDevice();
            dbDevice.payload_T = data.tempC;
            dbDevice.payload_W = data.humidity;
            api.put(`devices/${this._id}`, dbDevice);
          })
        }, 30000)
    };
}
```

3.5. Security

```
const securityControl = require('../espruino-modcat/modules/@amperka/power-control').connect(P0);
module.exports = class Security {
    switchOn() {
        securityControl.turnOn();
    }
    switchOff() {
        securityControl.turnOff()
    }
}
```

3.6. DeviceController

```
const Led = require('./devices/lights');
const Security = require('./espruino/devices/security');
const BluetoothDevice = require('./espruino/devices/bluetoothDevices');

module.exports = class DeviceController {
    constructor(device) {
        this._id = device._id;
        this.mac = device.mac ? device.mac : null;
        this.action = device.status ? 'on' : 'off';
        this.deviceType = device.type;
    }
    switchDevice(delay=0) {
        let deviceControl;
        switch (deviceType) {
            case 'light':
                deviceControl = new Led(this._id);
                break;
            case 'security':
                deviceControl = new Security();
                break;
            case 'bluetooth device':
                deviceControl = new BluetoothDevice(this.mac);
            default:
                break;
        }
        if (deviceControl && this.action === 'off') {
            deviceControl.switchOff(delay);
```

```
        }
        if(deviceControl && this.action === 'on') {
            deviceControl.switchOn(delay);
        }
    }
}
```

3.7. SceneControl

```
const DeviceController = require('./controller');

Module.exports = class SceneControl {
    constructor(scene) {
        this.startDate =  new Date(Date.parse(scene.startDate));
        this.interval = scene.repeatDaily ? 24*60*60*1000 : scene.repeatWeekly && !scene.repeatDaily ? 7*24*60*60*1000 : 0;
        this.actions = scene.actions;
        this.isActive = scene.isActive;
    }
    getStartDate = () => {
        let startDate = this.startDate;
        while(startDate < new Date()) {
            startDate += this.interval;
        }
    }
    play = () => {
        if(!this.isActive) {
            return;
        }
        const today = new Date();
        const startDate = this.startDate > today ? this.startDate : this.getStartDate();
        const delay = startDate - today;
        setTimeout(() => {
            scene.actions.forEach(action => {
                const deviceController = new DeviceController(action.device);
                deviceController.switchDevice(action.timeOut);
            });
        }, delay)
    }
}
```

3.8. index.js

```
const MeteoSensor = require('./devices/meteoSensors');
const MeteoControl = require('./devices/meteoControls');
const DeviceController = require('./controller');
const axios = require('axios');
const SceneControl = require('./sceneController');
const api = axios.create({
    baseURL: 'http://localhost:3003/',
```

```
});
const sensors = [
    {_id: "5fae77d971ec2f854e31e02a", pin: P6},
    {_id: "5fae785571ec2f854e31e02d", pin: P7},
    {_id: "5fae792071ec2f854e31e032", pin: P8},
    {_id: "5fae7a6071ec2f854e31e03a", pin: P9},
];
const tControls = [
    {read: A0, write: P10},
    {read: A1, write: P11},
    {read: A2, write: P12},
    {read: A3, write: P13}
];
const hControls = [
    {read: A4, write: P14},
    {read: A5, write: P15},
    {read: A6, write: P16},
    {read: A7, write: P17}
];
const scenesUpdate = async () => {
    const currentScenes = await api.get(`scenes/`);
    currentScenes.forEach(scene => {
        const sceneControl = new SceneControl(scene);
        sceneControl.play();
    })
}
tempSensors.forEach(sensor => {
    const newSensor = new MeteoSensor(sensor._id, sensor.pin);
    newSensor.act();
});
tControls.forEach(control => {
    const newControl = new MeteoControl(control,'t');
    newControl.act();
});
hControls.forEach(control => {
    const newControl = new MeteoControl(control,'h');
    newControl.act();
})
setInterval(() => {
    scenesUpdate();
}, 60000);
```