МАГІСТЕРСЬКА РОБОТА

15.04 – МР. 1859 “С” 2021.11.1.003 ПЗ

Паламарчука Богдана Олеговича

2022 р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	Завідувач кафедри
Інформаційних технологій	Комп'ютерних систем, мереж та кібербезпеки
_____ Глазунова О.Г., д.пед.н, проф.	_____ Касаткін Д.Ю., к.п.н., доц.
підпис ПІБ, вчене звання і ступінь	підпис ПІБ, вчене звання і ступінь
«___»_____ 2022 р.	«___»_____ 2022 р.

МАГІСТЕРСЬКА РОБОТА

На тему: «Дослідження параметрів системи управління мікрокліматом у розумному будинку з використанням веб-додатку»

Спеціальність 123 «Комп’ютерна інженерія»

Освітня програма Комп’ютерні системи та мережі

Орієнтація освітньої програми _____

Керівник дипломного проекту: _____ / Місюра М. Д. /
підпис ПІБ

Виконав: _____ / Паламарчук Б. О. /
підпис ПІБ

КИЇВ-2022

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
Комп'ютерних систем, мерсж та кібербезпеки
_____ / Касаткін Д.Ю., к.п.н., доц. /
підпис ПІБ, вчене звання і ступінь
«__» _____ 20__ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Паламарчук Богдан Олегович
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): комп'ютерна інженерія
Освітня програма: Комп'ютерні системи та мережі
Орієнтація освітньої програми _____

Тема магістерської роботи: Дослідження параметрів системи управління мікрокліматом у розумному будинку з використанням веб-додатку

затверджена наказом ректора НУБіП України від "1" 11 2021р. № 1859 «С»
Термін подання завершеної роботи на кафедру: _____
Вихідні дані до магістерської роботи: серверна частина реалізована за допомогою Google Firebase, клієнтська частина: в основі мова програмування JS, бібліотека React, фреймворк Next.js, графічна сторона Material UI

Перелік питань, що підлягають дослідженню:
1. Аналітичний огляд
2. Вимоги до системи
3. Проектування та розробка системи

Перелік графічного матеріалу (за потреби) _____

Дата видачі завдання "_____"_____2022 р.

Керівник магістерської роботи _____ Місюра М.Д., к.т.н., доц.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Паламарчук Б.О
(підпис) (прізвище та ініціали студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	18.10.2021	Виконано
2	Проектування системи	10.12.2021	Виконано
3	Реалізація системи	20.03.2022	Виконано
4	Тестування системи	15.05.2022	Виконано
5	Оформлення пояснювальної записки	19.08.2022	Виконано
6	Оформлення графічного матеріалу	10.10.2022	Виконано

Студент _____ _____
(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____ _____
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 58 сторінок, 26 рисунків, 30 джерел.

КОМП'ЮТЕРНА СИСТЕМА, УПРАВЛІННЯ МІКРОКЛІМАТОМ, КЛІМАТ, КОРИСТУВАЧ, ВІЗУАЛЬНА ЧАСТИНА, ІНТЕРФЕЙС, ПАНЕЛЬ КЕРУВАННЯ, REACT, FIRESTORE, GOOGLE FIREBASE, NEXT, MATERIAL UI

Мета – розробка максимально простого та інтуїтивного зрозумілого для користувача інтерфейсу для автоматизованої системи для управління мікрокліматом та задання його параметрів, що покращить керування підсистемою та підвищить комфорт.

Об'єкт – веб-додаток для управління мікрокліматом у розумному будинку.

Предмет розробки – методи та програмні додатки для веб-додатку.

Проект складається з трьох розділів.

Перший розділ присвячено аналізу предметної області. Проводиться детальний огляд об'єкта, переглянуто переваги та недоліки таких систем. Проведено огляд існуючих рішень.

У другому розділі розкриті питання щодо алгоритмів управління та задання параметрів. Проаналізовано етап роботи алгоритму.

Третій розділ присвячено проектуванню та реалізації компонентів системи. Проведено моделювання поведінки та структури системи.

В результаті виконання дипломної роботи проведено аналіз, моделювання та дослідження розроблюваної автоматизованої системи та розроблені рекомендації щодо її проектування.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПК – персональний комп’ютер

HTML – мова розмітки гіпертексту, HyperText Markup Language

CSS – каскадні таблиці стилів, Cascading Style Sheets

front-end – інтерфейс для взаємодії між користувачем і back end

JS – JavaScript

DOM – об'єктна модель документа, Document Object Model

Мб – мегабайт

UI – інтерфейс користувача, User interface

API – програмний інтерфейс застосунку, Application Programing Interface

ВСТУП

Прогрес не стоїть на місці і кожну хвилину все стрімко розвивається. З кожним днем людина хоче полекшити собі життя тим самим покращитий його в кращу сторону. Технології розвиваються і разом з цим пішли тенденції розумних будинків. Свій хід зробили технології віддаленного та безконтактного доступу. Прикладом може слугувати те, що ви хочете мати впевненість, що ваше майно в безпеці коли ви на роботі чи на відпочинку, або ви не хочете вставати з м'якого крісла щоб вимкнути світло чи відрегулювати температуру в будинку. З цією проблемою вам допоможе автоматизована система. Стрімкий ріст популярності систем управління функціями житлових приміщень, таких як Розумний Дім зумовлений людиною прагненням до зручності і комфорту.

Загалом комфорт в домівці залежить від дрібниць, таких як: режим температури, свіжість та вологість повітря, якість освітлення тощо. Але і дуже важливу роль відіграє і зручне задання і керування параметрами для цих умов. Наприклад якщо розглядати сучасні будинки, то в 90% з них встановлені кондиціонери, системи теплої підлоги, системи обігріву тощо. Система управління мікрокліматом чудово підхидить для всіх цих параметрів, щоб вони працювали як одне ціле. Взагалі системи Розумних Будинків мають дуже широкі можливості такі як:

- автоматизація опалення
- автоматизація освітлення
- управління кондиціонерами
- управління шторами
- сигнали про протікання води
- сигнали про відкриття вікон та дверей
- відеоспостереження
- голосове керування
- управління будинком через додаток

- налаштування сценаріїв
- можливість розширення в подальшому такої системи

В результаті виконання роботи було проведено аналіз існуючих систем, їх параметри та загальі затрати. Також були обрані оптимальні налаштування параметрів системи управління з використанням наявних засобів керування. Було створено простий веб-додаток для задання цих параметрів.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Дослідження предметної області

Для початку давайте розглянемо шо таке Розумний дім.

“Розумний будинок” — це система, яка управляє всіма процесами в будинку: опалення, вентиляція, кондиціювання, відеоспостереження та інше. Система може управлятись як в ручну так і автоматично. Тобто власник такої системи може з телефону або планшету самостійно встановити параметри, або включити різну техніку з будь-якої точки світу.

Система «Smart Home» або в простонароді Розумний будинок набирає чималої популярності. Це дуже легко пояснюється зручним централізованим контролем, який вона надає. Перевага її в тому, що система програмується від потреб до власних вподобань мешканців, тобто навіть дитина може власноруч змінити параметри у своїй кімнаті на ті, що їй потрібні.

Наступне цікаве питання про те, чи можу я на довгий період поїхати і залишити це все? Відповідь так, ви завжди зможете переглядати камери і бачити що відбувається у вас в оселі. Також можна за допомогою електронного замку відчиняти двері і пустити когось в середину, а після закрити за ним двері. Також можна запрограмувати, щоб коли ви приходите після роботи додому, то вмикалась ваша улюблена музика, або налаштовувалась температура до заданої норми. Все залежить від того що ви хочете, обладнання і бюджету.

Якщо брати інженерні комунікації з бездротовим керуванням, то на їх основі виконується автоматизація інженерних комунікацій багатоквартирних будинків за основу яких взята програмно-апаратна платформа, яка дає можливість керувати лічильниками, датчиками, системою опалення, освітлення і безпеки через бездротові пристрої, які розташовані по всьому будинку. За допомогою таких платформ дається можливість створювати типові сценарії для кожної квартири, під’їзду чи будинку. Їхня мета — допомогти власникам економити електроенергію, тепло, воду та запобігати різним непередбачуваним

ситуаціям. При цьому кожен власник квартири, за допомогою телефону або планшету бачить всю інформацію про свій дім. Різниця таких систем від провідних в тому, що вони з легкістю дозволяють добавляти нові технології в уже заселені будинки не доставляючи мешканцям незручності.

Якщо розглядати системи з голосовим управлінням, то продажі розумних колонок з голосовим асистентом постійно зростають. На сьогоднішній день ними користується кожен шостий житель США. Більше 30% власників колонок використовують їх для управління розумним будинком. Паралельно цьому збільшилось число продукції розумних будинків з підтримкою голосових помічників (рис. 1.1).

Рисунок 1.1 – Розумна колонка для керування будинком

1.2 Підбір системи

І ось ви вирішили піти на такий серйозний крокнарахунок свого житла, щоб перетворити його в розумний будинок. Є чотири основні кроки, щоб прокласти правильний маршрут до нових технологій.

1.2.1 Вибір цифрового помічника

Перед закупівлею пристроїв для розумного будинку, потрібно зробити вірний вибір на користь віртуального асистента та центру розумного будинку. Які б технології не були б обрані більшість продуктів будуть підключатись через контролер, а віртуальний асистент буде реалізовувати команди користувача системи. На сьогоднішній день є достатня кількість асистентів таких як: Alexa, Siri, Cortana та Google Asistant. Кожен з них має переваги, але потрібно звертати увагу на те, що очікує користувач. Підбирати потрібно на свій смак наприклад Alexa більш повноцінний, але для прихильників Apple більше підійде Siri тому що вони вже використовують продукти цієї корпорації. А ось центр розумного будинку (рис. 1.2) визначає які пристрої можна підключати разом, тому потрібно спочатку прийняти рішення що ви плануєте купити, а потім вже підбирати помічника.

Рисунок 1.2 -Центр розумного будинку

1.2.2 Вибір пристроїв

Серед великої кількості продуктів розумного на ринку досить не просто підібрати саме той, яки підійде саме тобі. В цьому випадку потрібно відштовхуватись від того що саме має виконувати система. Якщо основним завданням буде безпека, то потрібно звернути увагу на розумні камери (рис. 1.3), відеодзвінки тощо. А рішенням для повсякденних задач підійдуть розумні лампочки, розетки, термостати. Ціна звичайно з кожним днем росте але можна на ринку пошукати і доступні.

Рисунок 1.3 — Розумні камери відеоспостереження

1.2.3 Перевірка інтернету

Чим більша кількість пристроїв в будинку, тим кращий повинен бути інтернет та його швидкість, щоб не виникало затримок. Нікому не сподобається система в якій потрібно буде зробити кілька запитів, щоб виконату одну й ту саму дію через повільний інтернет. Якщо не хочете встановлювати дорогий пакет послуг, то проведіть тестування швидкості, щоб дізнатись чи вистачає її для обладнання проте, якщо у вас швидкий інтернет то проблем з підключенням пристроїв не буде. Система розумного будинку керується цифровим методом телефоном або планшетом або автономно(рис. 1.4).

Рисунок 1.4 — Управління розумним будинком

1.3 Огляд існуючих рішень

Прогрес на сьогоднішній день не стоїть на місці і тому свій хід зробили системи автоматизованого управління. Сучасний ринок дає великий вибір технічних приладів для автоматизації будинку, тому розглянемо більш відомі з цих систем. Перед тим як розпочати розбиратись в специфічних та функціональних можливостях різних систем автоматизації будинку, потрібно виділити основні критерії їх оцінювання. Наприклад:

- стандартні пристрої, та можливість розширення системи;
- порядок підключення (провідний, безпровідний);
- спосіб керування (ПК, смартфон, планшет);
- канали зв'язку з користувачем;
- дальність роботи сигналу;
- ціна;
- виробник та ін.;

1.3.1 Система Ajax

Виробник Україна, тому система зразу підтримує український інтерфейс. Дана система автоматизації будинку відразу справляється з двома важливими завданнями:

- забезпечує комфорт і зручність в керуванні життєзабезпечення приміщення;
- гарантує безпеку житла в повному обсязі, контролює можливі загрози для будівлі;

Обладнання Ajax працює на надійно захищеному радіозв'язку Jeweller свого ж виробництва, повністю автономна від мережі завдяки резервному джерелу енергії (рис. 1.5).

Переваги:

- просте встановлення;
- безпровідний канал зв'язку між елементами;
- велика зона дії сигналу (до 2 км);
- захист від зняття будь якого датчику;
- можливість підключення інших користувачів;
- автономна робота від батареї (до 16 годин);
- WI-FI і GSM — зв'язок;
- різновид інформування користувача;
- встановлення по QR-коду і керування за допомогою смартфону (iOS, Andrioid)
- підключення до 100 пристроїв;
- наявність тривожної кнопки;
- не велика ціна комплекту (від 200$);

Недоліки:

- відсутність автономності датчиків;
- відсутність власного відеоспостереження;
- керування тільки через телефон;

Рисунок 1.5 — Система Ajax

1.3.2 Система BroadLink

Виробник Китай. Система не має українського інтерфейсу, але при необхідності його можна встановити. Обладнання BroadLink являє собою комплект сучасних цифрових пристроїв, створених для управління побутовою технікою, та іншими системами в будинку. Кожний елемент цієї системи може працювати як самостійно. Так і взаємодіяти з іншими.

Переваги:

- швидке встановлення та налаштування;
- великий асортимент датчиків;
- можливість підключати та відключати різні пристрої;
- автономна робота датчиків;
- безпровідна взаємодія пристроїв;
- має камеру відеоспостереження;
- керується через мережу з будь-якої точки;
- ціна (від 200$);

Недоліки:

- не велика дальність сигналу;
- відеутність резервного електропостачання;
- пульт працює тільки на прийом сигналу;

Система має великий функціонал та якісне програмне забезпечення, також вона легка в встановленні та користуванні і відносно не дорого коштує. Цей комплекс не потребує центрального контролеру, оскільки його пристрої хоч взаємнопов’язані, але можуть працювати автономно (рис. 1.6).

Рисунок 1.6 — Система BroadLink

1.3.3 Система Fibaro

Виробник Польща. Грішки проблематично найти інтерфейс на українській мові. Не дивлячись на це, система відноситься до професійного обладнання по забезпеченню автоматизації будинку з широким функціоналом (рис. 1.7).

Переваги:

- можливість широкого наповнення датчиками та пристроями;
- наявність камери відеоспостереження;
- великий набір сценаріїв для користувача;
- відправка повідомлень відразу на декілька пристроїв;
- працює на базі протоколу Z-Wave;
- датчик протікання з сиреною;
- розумна розетка з великим функціоналом;

- кожний елемент може бути ретранслятором сигналу;
- голосове керування через Google;

Недоліки:

- висока ціна (від 600$);
- тільки професійне встановлення і налаштування;
- обов'язкове підключення центрального контролеру до інтернету через LAN-кабель;
- не функціонує без центрального хабу;
- не має резервного електропостачання;
- затримка Push-повідомлень;
- урізаний мобільний додаток;

Рисунок 1.7 — Система Fibaro

1.3.4 Система Orvibo

Виробник Китай відповідно вітсутній український інтерфейс, але присутній англійський. Недорогий комплект простого в експлуатації приладдя, головною задачею якого є безпека будинку. В другу чергу може слугувати базою для повноцінної системи(рис. 1.8).

Переваги:

- легка в установці та підключенні;
- автоматичний пошук і підключення сенсорів;
- велика кількість приладів і можливість розширення;
- має свою відеокамеру;
- безпровідний протокол взаємодії між контролером і датчиками;
- автономність деяких пристроїв;
- вибір сценаріїв роботи;
- Wi-Fi — зв'язок з телефоном;
- недороге обладнання (від 150$);

Недоліки:

- мала зона дії;
- невелика базова комплектація;
- відсутність резервної електромережі;
- провідне підключення до інтернету;

З цього можна сказати, що це проміжне обладнання між системою охорони та розумним будинком. Воно доволі просте і має чудові можливості в маштабуванні, але через доступність не має захисту від взлому.

Рисунок 1.8 — Система Orvibo

1.3.5 Система Xiaomi

Виробник Китай відносно відсутній український інтерфейс, що ускладнює встановлення та налаштування. Ця система відноситься до бюджетному класу пристроїв, які дозволяють зробити управління різними пристроями максимально легким та зручним (рис. 1.9).

Переваги:

- автономність пристроїв;
- можливість маштабування;
- наявність своєї камери відеоспостереження;
- безпровідний протокол;
- зручне керування за допомогою смартфону;
- наявність сценаріїв;

- компактність та дизайн;
- низька ціна (90$);

Недоліки:

- мала зона дії сигналу;
- малий набір сенсорів в базовому наборі;
- немає резервного електропостачання.

Рисунок 1.9 – Система Xiaomi

2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ УПРАВЛІННЯ

2.1 Дослідження алгоритмів керування

Є основні вимоги до управління мікрокліматом і освітленням в розумному будинку. Оснивними є: забезпечення мікроклімату комфортного для проживання, освітлення та зменшення затрат на енергоресурси. Дійшли до висновку, що такі вимоги можна досягти шляхом використання телекомунікаційних технологій, та систем управління опаленням, температурою, вентиляцією, кондиціюванням тощо. Проаналізовано засоби програмно-апаратного забезпечення, які є на ринку для систем управління розумним будинком, і показано що недоліком є необхідність адаптації до вимог конкретного застосування, та їх велика ціна, що зменшує їх використання. Найкращим рішенням для систем управління мікрокліматом і освітленням використовувати готові компоненти, які запропоновані у вигляді готових модулів. Системи управління мікрокліматом та освітленням зв'язують в єдиний комплекс різне обладнання та інженерні системи будинку. Розроблено це за допомогою платформи Arduino структури яких адаптуються під певного користувача, забезпеують кращі умови проживання, зменшують енергоспоживання, та мають доволі невелику ціну. Основним компонентом таких систем є плата Arduino, в якій є мікроконтролер atmel AVR, елементів для програмування з іншими пристроями, датчиків температури, руху, освітлеиості, засобиів які забазпечують мікроклімат такі як котли, підігрів підлоги, обігрівачі, витяжки тощо. Розроблено блок-схеми алгоритмів таких систем. Запропоновано використовувати як дротові, так і без дротові засоби зв'язку. Показано, що контроль і встаиовлення параметрів у системах мікроклімату можна здійснювати за допомогою сенсорних пультів, додатків або веб-додатків.

Сучасний вид будинку — це набір складних систем, які керуються за допомогою технології Smart Home, вони створюють комфортне проживання власника будинку і зменшують енерговитрати. Основою інтелектуального управління є принцип неприривного зв'язку всіх діючих функціональних систем

у приміщені таких як: управління мікрокліматом, водними процесами, газопостачанням, освітленням та електрикою. Інформаційні технології будинку повині створити єдиний комплекс, в який будуть входити різні системи житла, і керувати ними так, щоббула забезпечена повна енергоефективність, і відносно створювали за собою комфортний стан для проживання.Нинішній розвиток систем телекомунікації та мікропроцесорної техніки створює вдосконалену роботу управління такими будинками, які відповідають за створення комфортних умов для проживання. Перспективним є створення розумних будинків які вміють працювати за завданими сценаріями, та вміють розпізнавати конкретні ситуації, які відбуваються в приміщенні, та вдало реагувати на них. Тому на мою думку є актуальним розроблення програмно-апаратних засобів керування такими системами, які будуть об'єднуватись в єдиний комплекс, і будуть забезпечувати камфорт проживання і економію ресурсів.

2.2 Аналіз останіх досліджень

Аналіз наявних засобів, які промонує ринок для синтезу систем управління показує, що вони ідуть як готові модулі (Kliuiko & Zlotenko, 2015; Tesliuk, et al., 2012; Medykovskyi, et al., 2015). Їхнім недоліком є відносно велика ціна яка зменшує їх використання. У роботах (Umnyi Dom, n.d.; Inzhenernye sistemy, n.d.; Obzornaia statia o sisteme "Umnyi dom", n.d.) розглядалось основні інженерні системи, та системи управління ними. Визначено їхні можливості та сформовано вимоги до систем клімат контролю, безпеки та освітлення. Показано, що основні недоліки таких систем управління і потрібність до адаптації їх програмно-апаратних засобів до конкретних вимог користування.

В роботах (Vozmozhnosti Umnogo Doma, n.d.; Home Sapiens, n.d.; Hrytsiuk, et al., 2019) було проаналізовано засоби зв'язку які використовувались для

управління системою і виявлено, що можна використовувати як дротові так і без дротові засоби.

З аналізу літературних джерел виявлено, що недоліком цих систем і апаратно-програмних засобів є необхідність до адаптації певного користувача, і їх висока ціна. Метою дослідження є розробка структур та алгоритмів роботи параметрів управління мікрокліматом.

2.3 Основні результати дослідження

Основним завданням системи являється комфорт та зменшене енергоспоживання. Для виконання такого завдання створено базову структуру зображену на (рис. 2.1).

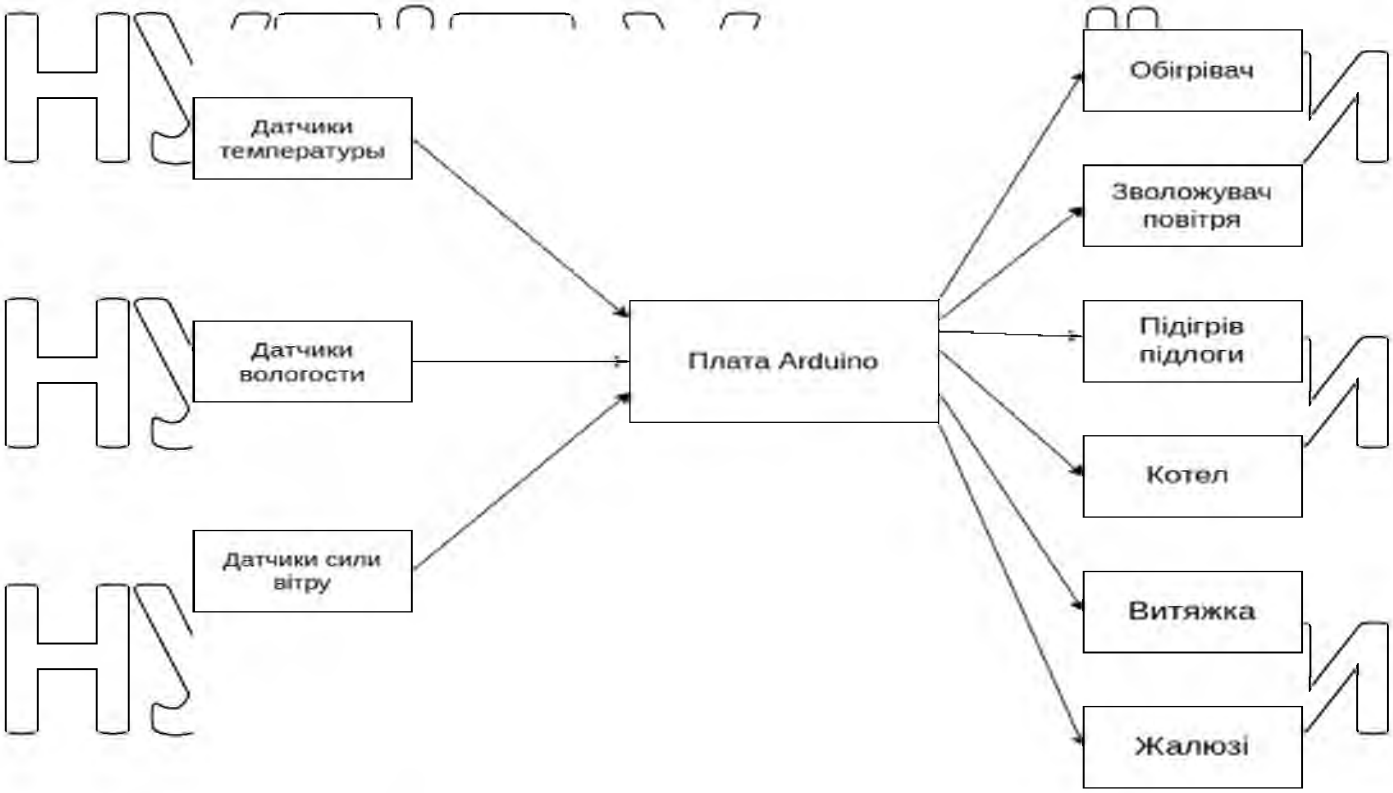


Рисунок 2.1 — Структура управління мікрокліматом будинку

Основні компоненти розробленої системи управління мікрокліматом є плата Arduino, і елементи для програмування та інтеграції з іншими пристроями такими як: датчики температури, датчики вологості повітря, датчики сили вітру, а також засоби забезпечення такі як: підігрів підлоги, обігрівачі, котел, зволожувач повітря, жалюзі та витяжка.

Основним завданням системи освітлення, є зменшення споживання електроенергії, та забезпечення комфортного освітлення приміщення. Для виконання такого завдання було створено структуру зображену на (рис. 2.2)

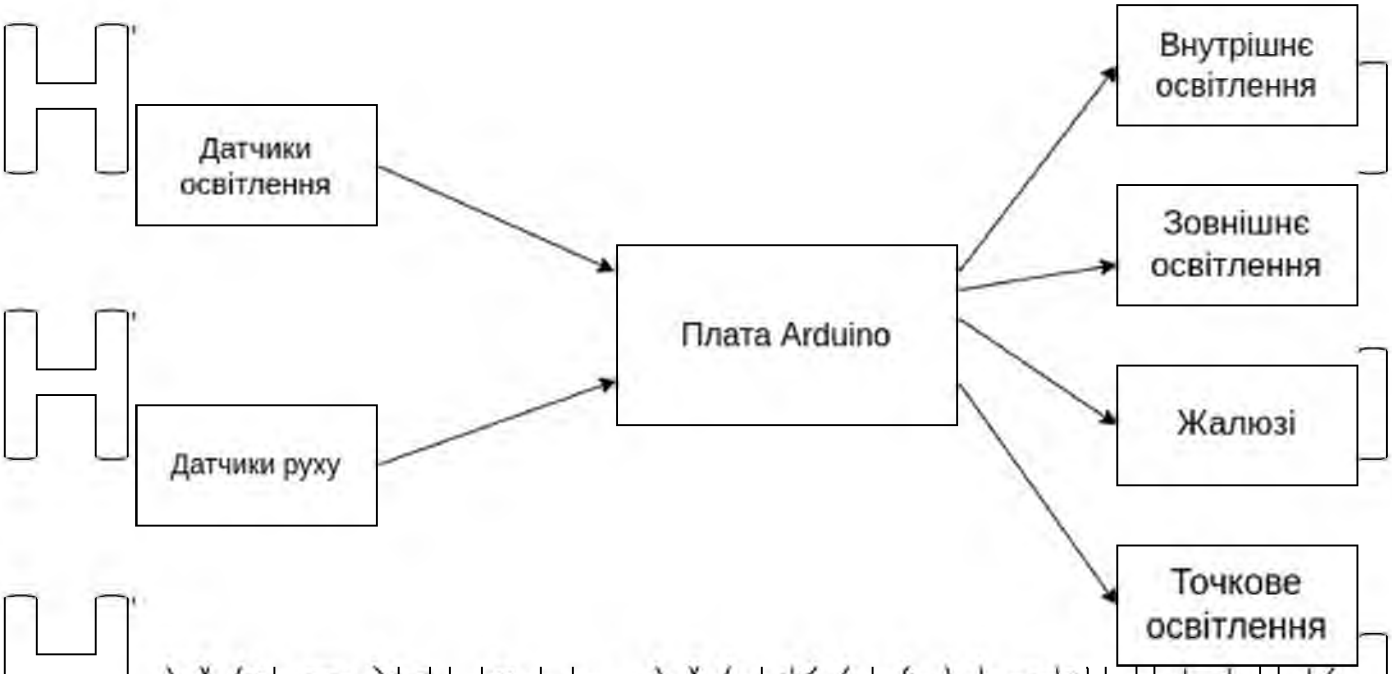


Рисунок 2.2 — Структура управління освітленням будинку

Основні компоненти розробленої системи є плата Arduino, датчики руху, освітлення, жалюзі, засоби завнішнього, внутрішнього та точкового освітлення.

Одним з найскладніших у системі життєзабезпечення будинку є система управління мігрокліматом. Вона забезпечує управління такими процесами як: опалення, вентиляція, кондиціювання, температура підлоги.

Під час традиційної побудови власнику потрібно керувати кожним із цих процесів окремо. Інколи є ситуації, коли в різні сезони система кондиціювання

починає боротися з вентиляцією. Такі дії призводять до швидкого зносу обладнання, та великого енергоспоживання. Щоб уникнути цього потрібно створити єдину систему для задання параметрів всім засобам в будинку. З її допомогою параметри клімату можна задати з телефону, планшету або пульта керування.

Прикладом може слугувати, коли власника немає вдома система всерівно повідомить його про температуру на вулиці і в приміщенні, силу вітру і т.д., отримає і виконає накази — наприклад відкрити вікна або зменшити температуру тощо. Автоматика їх зачинить якщо розпочнеться дощ. У спекотні літні дні увімкне систему кондиціювання чи відчинить вікна. У такій системі мікроклімату можливе виконання багатьох сценаріїв, всі вони залежать від різних ситуацій.

Наприклад якщо вам потрібно поїхати на дачу, то все що вам потрібно це зайти в додаток і дати команду системі, і вона підготує його до вашого приїзду. Блок-схему алгоритму роботи системи мікроклімату зображено на (рис. 2.3)

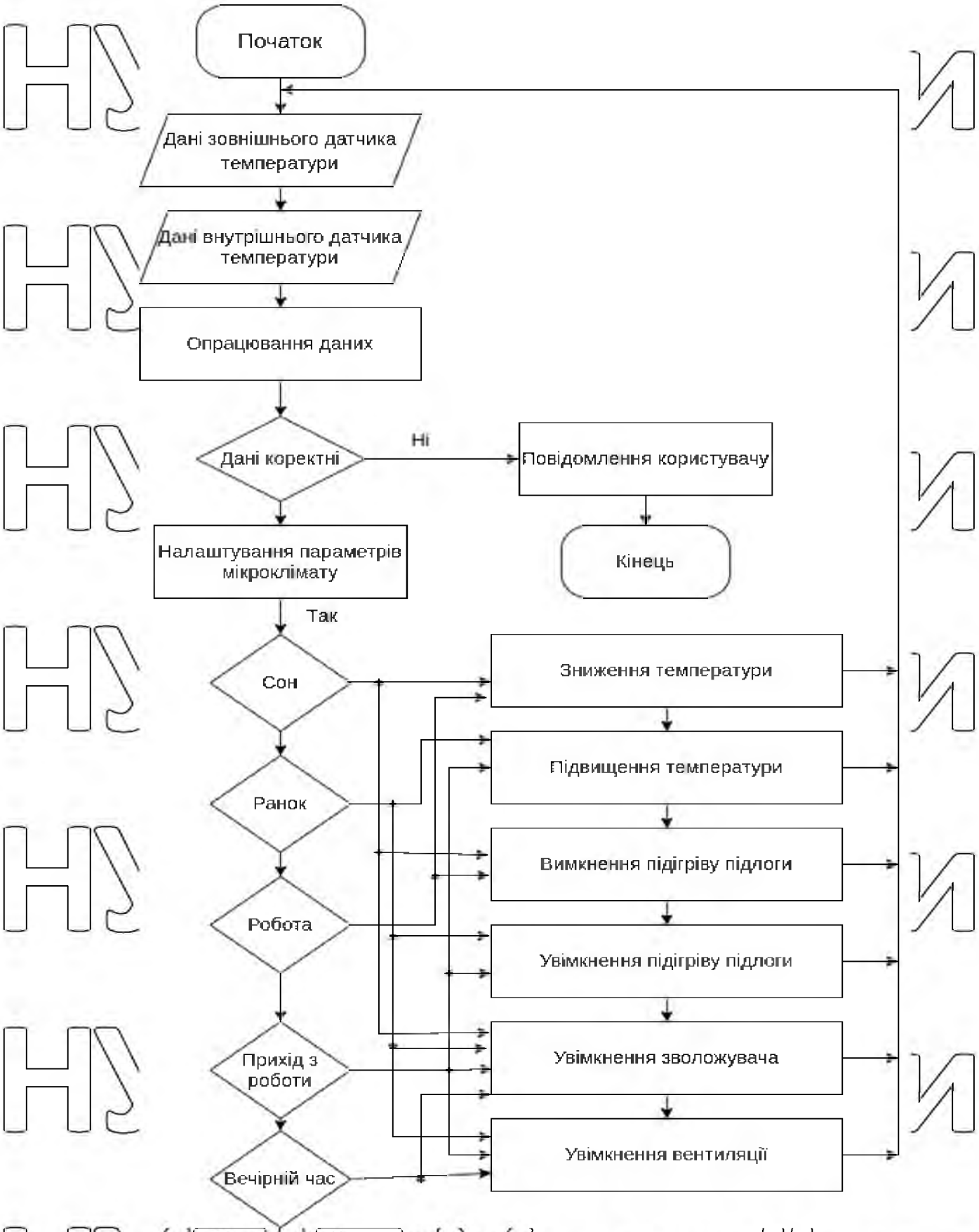


Рисунок 2.3 — Блок-схема роботи системи мікроклімату

У кожному приміщенні система підтримує індивідуальні параметри — вологість, температуру, свіже повітря. Причому залежно від вуличної температури (або за поданою командою власника) вибирає і вмикає необхідну потужність на один або кілька теплових приладів — тепловентилятори, теплі підлоги, радіатори опалення тощо.

Для комфортних умов сну температура до ночі знизиться, а ранком підніметься. Якщо ви ідете з дому на довго, то встановиться економний режим. За кілька годин до повернення можна дати команду телефоном або веб додатком, і до приїзду автоматика задасть для приміщення задані кліматичні параметри.

Управління освітленням, теж відноситься до важливих функцій системи, завдяки якій відбувається не тільки комфорт освітлення а й економія енергоресурсів. У системі управління світлом за допомогою плати Arduino та інших засобів формується сигнал керування джерелами світла в різних приміщеннях і створюється світлова картина покімнатах. Для цього потрібно у пам'ять системи завантажити готовий світловий сценарій, вибір якого дає змогу керувати освітленням по кімнаті, чи у всьому будинку. Основним завданням такої системи є формування сигналу управління внутрішнім, зовнішнім та точковим освітленням. Така система працює за алгоритмом, блок-схема якого наведена на (рис. 2.4).

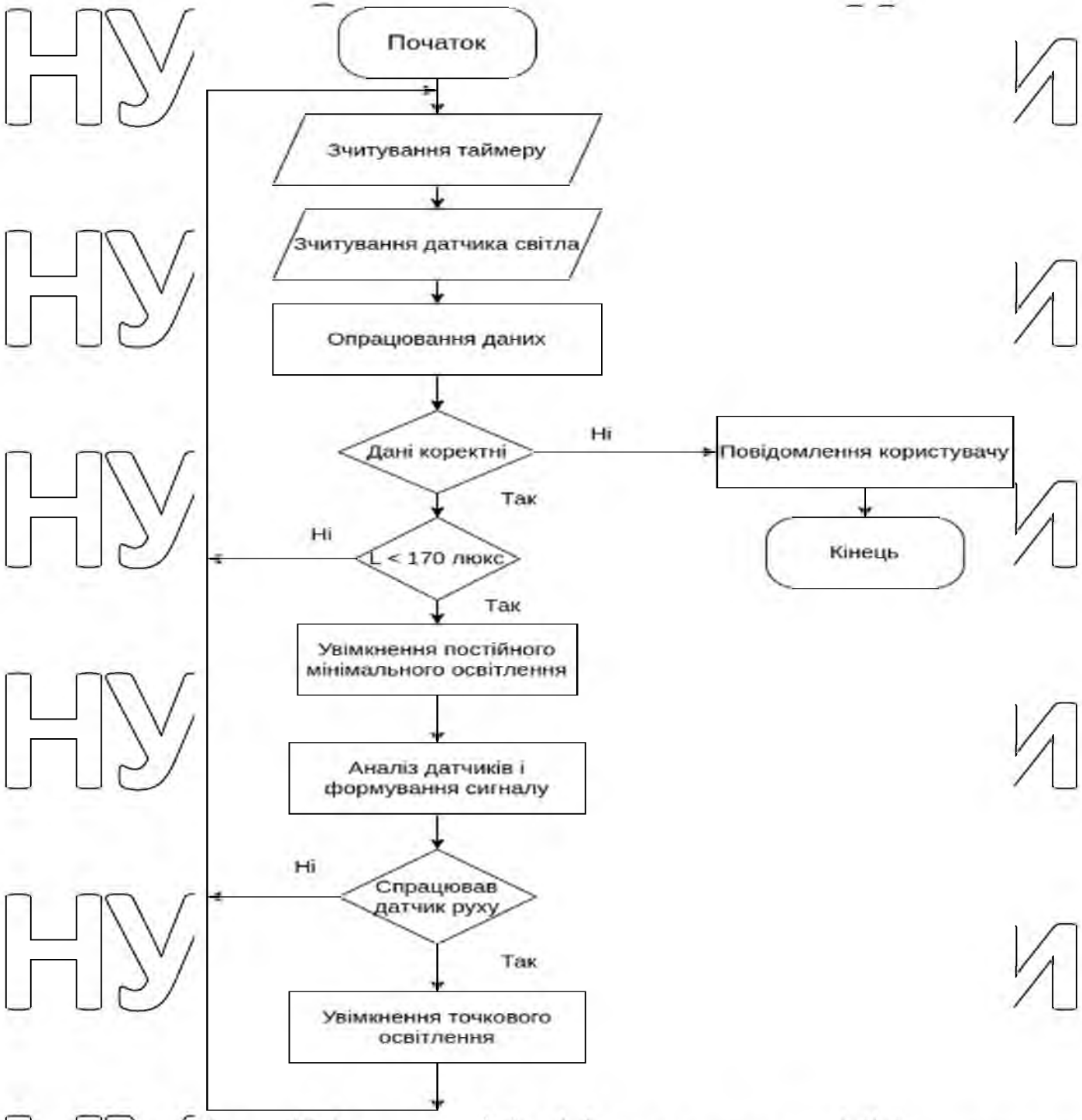


Рисунок 2.4 — Блок-схема роботи системи освітлення

Вентиляція також має свою прив'язку до системи розумного будинку. Тільки згадаємо, що завдавати параметри можна тільки за централізованою системою вентиляції. Така система складається з:

- Витяжно-припливної установки з рекуперацією тепла;
- Мережі повітропроводів до кожної зони;
- Вентиляційних решіток, через які відбувається розподіл повітря до всіх кімнат;
- Витяжних вентиляторів;

Організованість центральної вентиляції розумного будинку ділиться на зони — чисті та брудні. Чисті зони це місця де мешканці проводять більше часу тобто туди потрібно активно подавати свіже повітря. Брудні зони це кімнати в яких потрібно вивести опрацьоване повітря, вологу та неприємні запахи (рис 2.5).

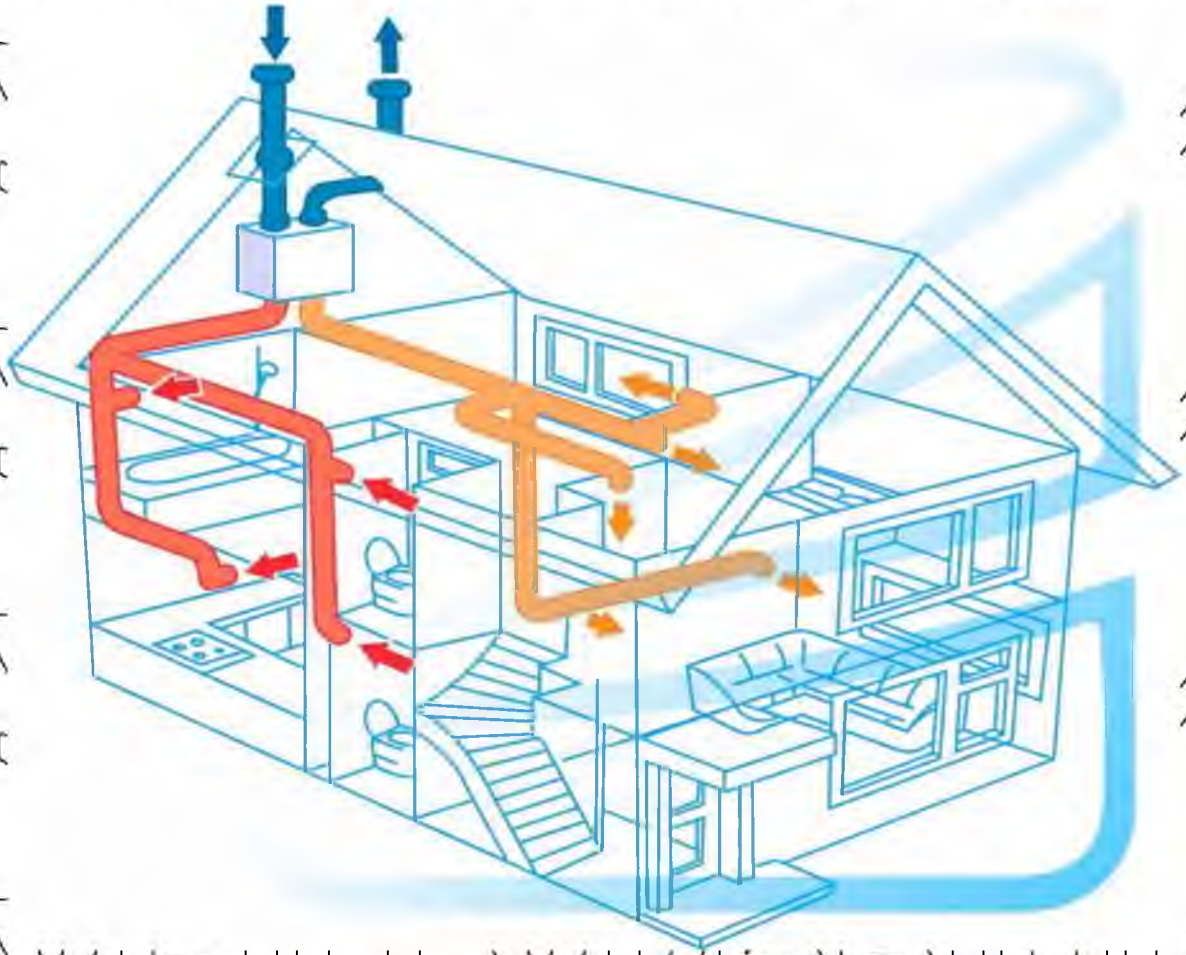


Рисунок 2.5 — Вентиляційна система

На рисунку 2.5 жовтим зображені повітропроводи, які подають повітря в так звані чисті зони. Червоним зображені повітропроводи, які викачують повітря з брудних зон.

Основні модулі, які використовуються для контролю — Modbus і KNX. Деякі виробники наприклад як Maico пропонують Modbus або KNX встановити додатково. Завдяки цим модулям вентиляційна установка може бути підключена до основної системи керування будинком. Така система в будинку дає переваги:

- Контроль увімкнення або вимкнення ПВУ;
- Управління подачі обсягу повітря;
- Контроль фільтрів та їх стану;
- Отримання даних з датчиків;

Потрібно розуміти, що система вентиляції не потребує частих змін в роботі і систематичного контролю. Пояснення полягає в тому, що під час її проетування ведеться детальний розрахунок обсягів повітря. Тому координально змінити її роботу не являється можливим. Але якщо в плани входить зробити повноцінний контроль будинку, то потрібно підключити витяжну установку, щоб мати повний контроль і слідкувати за станом повітря.Вентиляція розумного будинку хоч і не надає такий функціонал як кондиціювання, але вона є зручною для аналізу якості повітря тощо.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1 Призначення специфікації

Вимоги SRS для програмного забезпечення описує функціональні та не функціональні вимоги до програмного продукту. Цей документ встановлює базу того, як повино працювати програмне забезпечення.

Інформаційна система (веб-додаток) призначений для перегляду та встановлення параметрів мікроклімату у Розумному будинку. Описаний функціонал спрощує розробку будь якого програмного продукту. Специфікація SRS являється одною з найголовніших частин складання вимог до ПЗ. Частіше виуористовується на етапі проектування та реалізації продукту. Вона відноситься до програмного продукту, але не до етапу його створення. Для цього потрібна специфікація, яка ділиться на два типа: технічні вимоги та специфічні вимоги.

3.1.1 Контекст

На момент складання документації виріб являється новим продуктом, але має аналоги на ринку. Створюємо мінімальний функціонал для конкретної підсистемя від повноцінної системи. Оскільки продукт відноситься до вуб-додатків і потрібно забезпечити його безпеку та надійність, то він був розміщений на платформі Google Firebase. Таке рішення надає певні переваги:

- швидкий доступ до ресурсу;
- швидкий доступ до БД;
- безкоштовний сервіс з перевагами;
- достатня кількість інструментів для корректної роботи;

Основні операції являються базовими діями по роботі з БД для яких має бути графічний інтерфейс. Якщо виникла помилка, то система повина вжити конкретні заходи, при неможливості таких дій вона повинна вміти їх корректно

обробити. Дані які введе користувач повині бути перевірені на корректність, якщо це потрібно.

Інтерфейс для користуванів повинен бути максимально легкий та зрозумілий для зручного використання. При наявності помилок для користувача має вивести на екран повідомлення про їх наявність. Інтерфейс не повинен заважати роботі системи. Користувач повинен регламентовано виконувати дії за для запобігання виникнення помилок.

Не залежно від прав доступу, додаток повинен обслуговувати користувачів що вказані в таблиці 3.1.

Тип користувача	Характеристика
Гість	Користувач, що не пройшов реєстрацію у додатку, буде мати обмежений доступ. Тобто доступ до реєстрації.
Зареєстрований користувач	Користувач, що вже зареєстрований в системі матиме особистий кабінет який дає функції, які не доступні гостям такі як: • перегляд статистики; • встановлення параметрів температури; • вст-ня параметрів кондиціювання; • вст-ня параметрів вологості повітря;

Таблиця 3.1. Характеристика користуачів за типом

3.1.2 Функціональні вимоги

Вимоги, що описують функції, які виконує програмне забезпечення. Для своєї реалізації система надає функції за типами користувачів такими як гість та зареєстрований користувач.

Фуикції типу «Гість»

Вимога 1 — Дати можливість реєстрації новим користувачам.

Вимога 2 — Можливість моніторингу дій гостей та подача повідомлення про потребу реєстрації.

Функції типу «Зареєстрований користувач» але з закритим доступом для користувачів типу «Гість»

Вимога 1 — Надання доступу зареєстрованим користувачам до всіх можливостей додатку.

Вимога 2 — Надання доступу для перегляду усіх функцій мікроклімату.

Вимога 3 — Надання доступу до можливостей маштабування системи.

Вимога 4 — Можливість повного пересування по додатку для перегляду або змін налаштувань.

Вимога 5 — Можливість контролю дій таких як:

- зміна параметрів контролю температурою;
- зміна параметрів контролю кондиціювання;
- зміна параметрів контролю вологістю повітря;
- зміна сценаріїв;

Вимога 6 — Підтримка нових технологій та оновлень компонентів.

Можливість за бажанням самоочищення статистики.

3.1.3 Моделювання системи

Модель це найголовніший етап у створенні систем. Вона забезпечує інформацією про процеси які відбуваються в додатку і ще побудову моделей. Під час моделювання додаються зміни які спрощують чи навпаки ускладнюють шляком додання нових параметрів або процесів. Надання діаграм описують функціонал та поведінку системи що дозволяє її обговоренню між розробником та замовником.

При моделюванні через діаграму прецедентів переслідуються такі цілі:

- відокремлення оточення від системи;
- взаємодія акторів з системою;
- функціонал акторів;

Вона відображує такі дії:

- актор — фігура людини яка обозначає ролі. Це може бути як реальна людина, так і користувач системи. Актори не можуть бути пов'язані між собою.

- прецедент — фігура еліпсу з підписом. В ній вказуються дії, які виконуються системою і відстежує результати акторів. Його підписом може бути і'мя дії чи опис з точки зору акторів яку дію виконує система. Сценарій пов'язаний з і'мям прецеденту а саме послідовністю дій поведінки системи. Безліч сценаріїв можна поєднати з одним прецедентом.

Діаграма сценаріїв відображає відносини між прецедентами та акторами в системі. За їх допомогою користувач отримує потрібний результат. Для описання сценарію роботи були використані актори які описані в специфікації:

- гість — це актор не зареєстрований в системі. Він має доступ лише до однієї функції такої як реєстрація в системі. Після такого кроку у нього появиться доступ як у зареєстрованого користувача (рис.3.1).

- зареєстрований користувач — це актор який зареєстрований в системі і має більш обширний доступ до функцій додатку та їх зміни та задання нових параметрів, які не має гість(рис. 3.2).

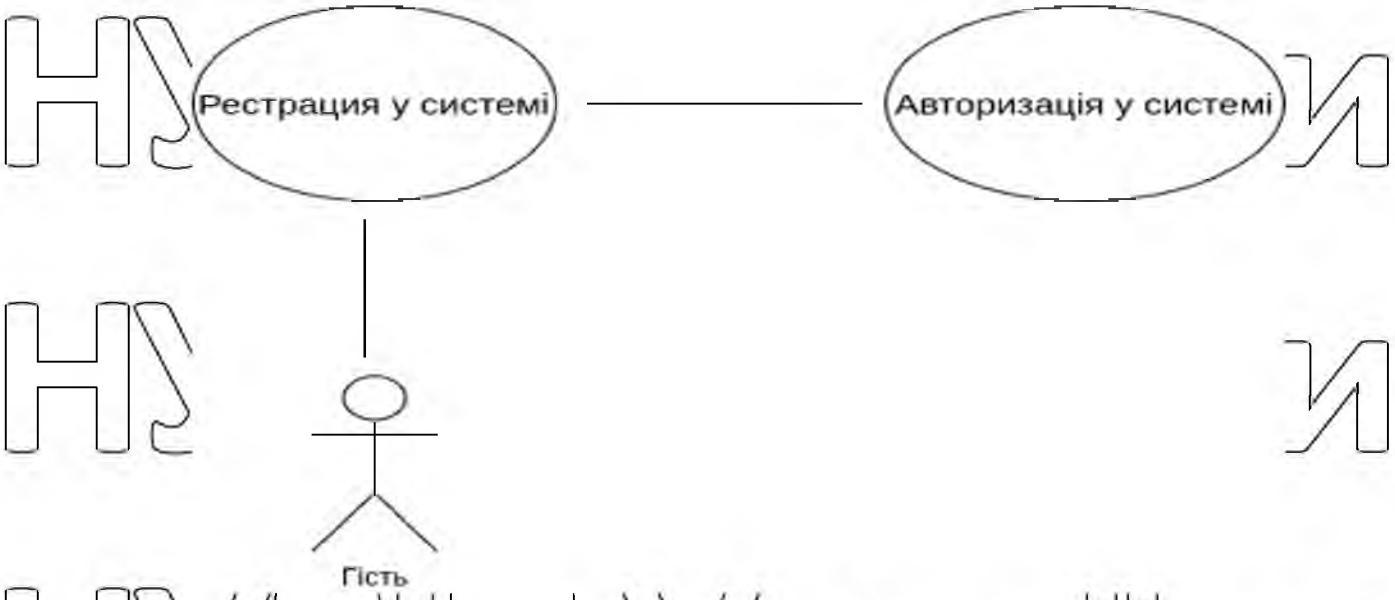


Рис 3.1 Діаграма актора гість

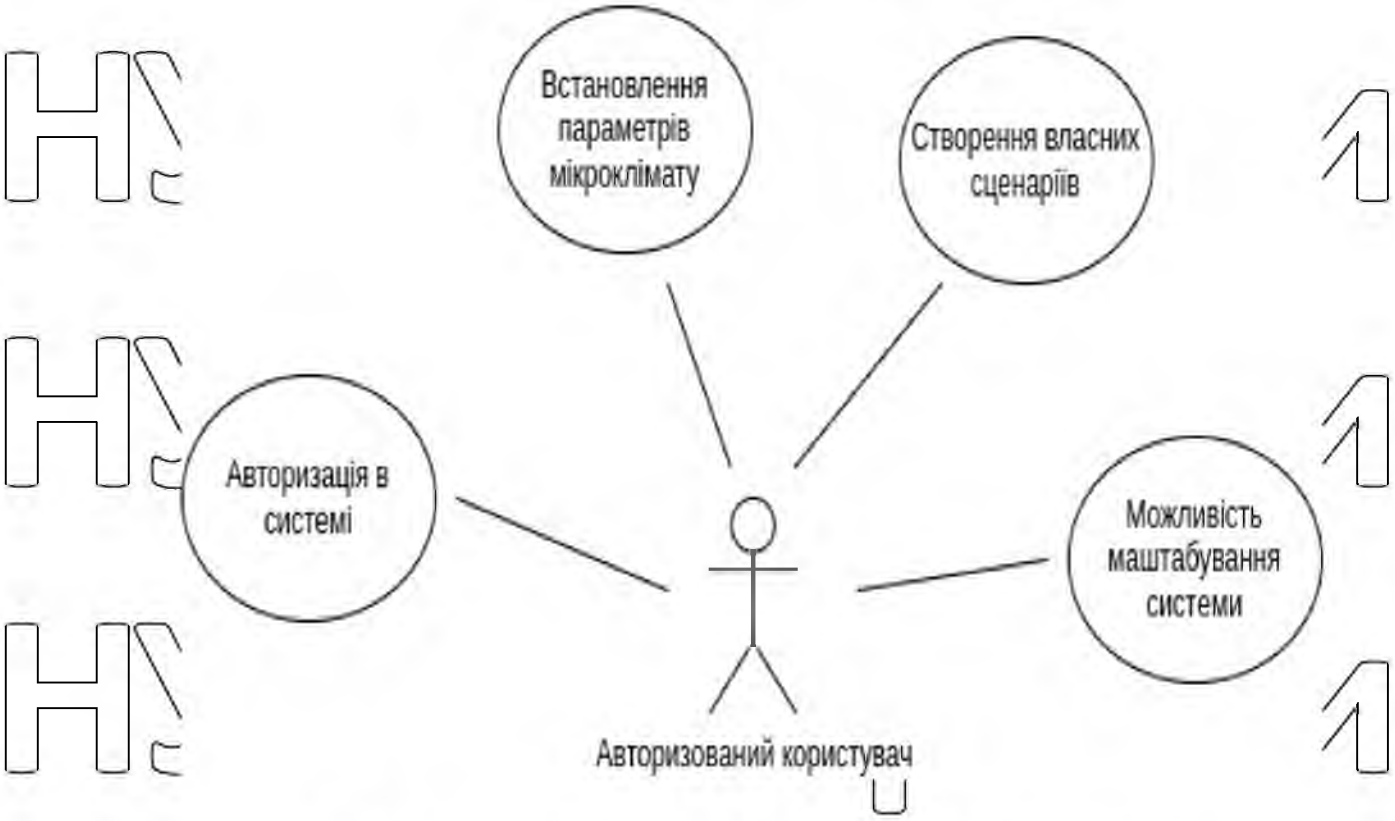


Рис. 3.2 Діаграма актора Авторизований користувач

Також потрібно розглянути діаграму для всіх акторів. Як бачимо на (рис. 3.3) гість не пересікається з авторизованим користувачем бо має обмежені права.

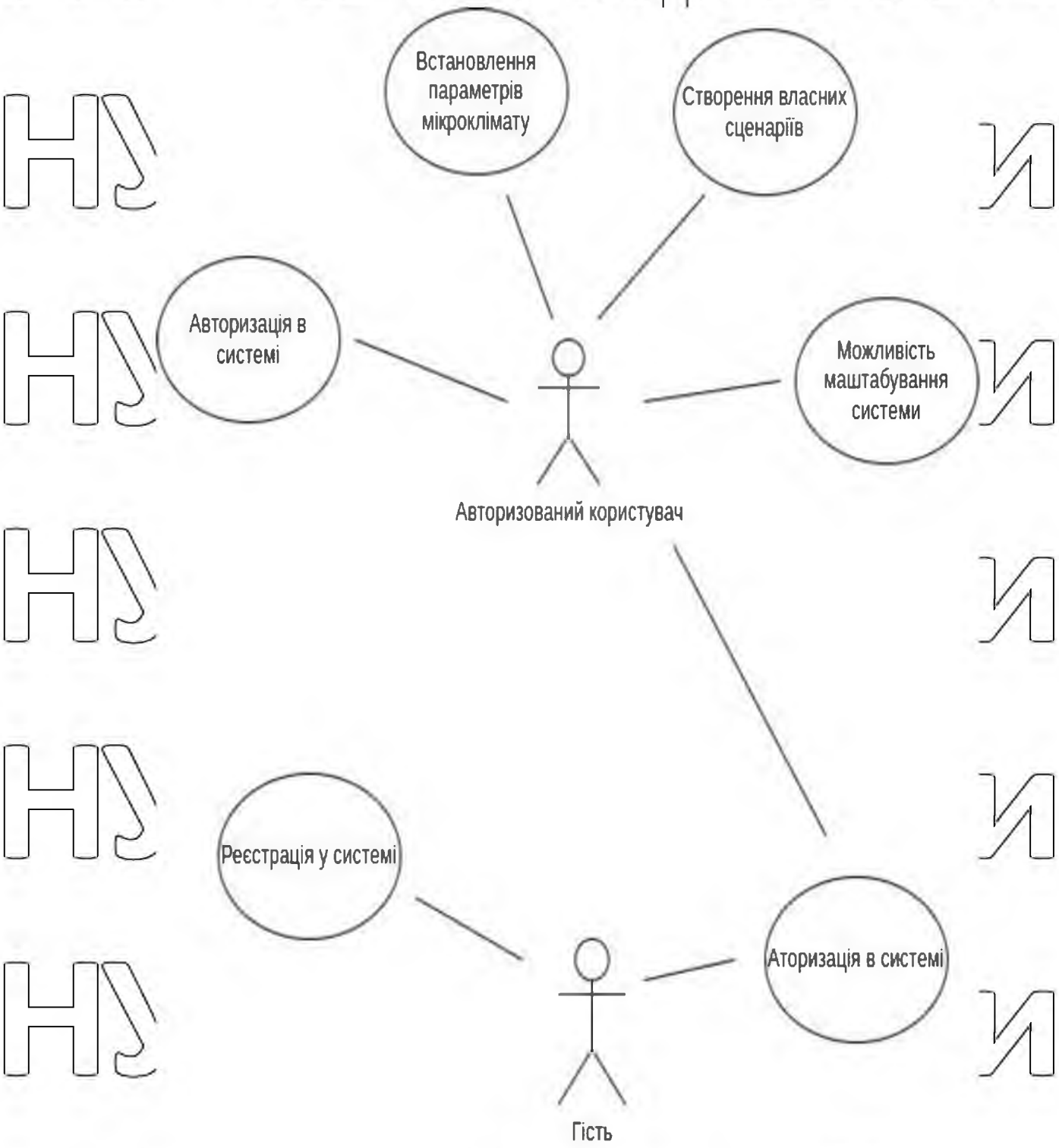


Рис. 3.3 — Діаграма для всіх акторів

3.2 Проектування інтерфейсу користувача

Інтерфейс користувача — це різновид інтерфейсу в якому всі елементи графічно представлені. Інтерфейс який використовується повинен бути налаштований згідно з поставленою задачею. Найбільше потрібно звернути увагу на такі пункти як: простота, функціонал, естетика. Потрібно звернути увагу на зручність та просту логіку під час розробки. Дизайн повинен бути на одному рівні з графічним інтерфейсом і повинен мати цілісну та організовану структуру. Також на оптимізацію системи робить хороший вплив добре розроблений інтерфейс. Якщо користувач зможе без проблем користуватись інтерфейсом веб-додатку то він буде схоче ним користуватись. Якщо все таки користувачеві не сподобається перше відвідування то він буде шукати альтернативу.

Форма реєстрації має бути мінімалістичною та простою без зайвих кнопок та посилань (рис. 3.4).

Рис. 3.4 Макет фарми входу/реєстрації

Відповідно для зареєстрованого користувача головна сторінка буде подавати всю необхідну інформацію (рис. 3.5)

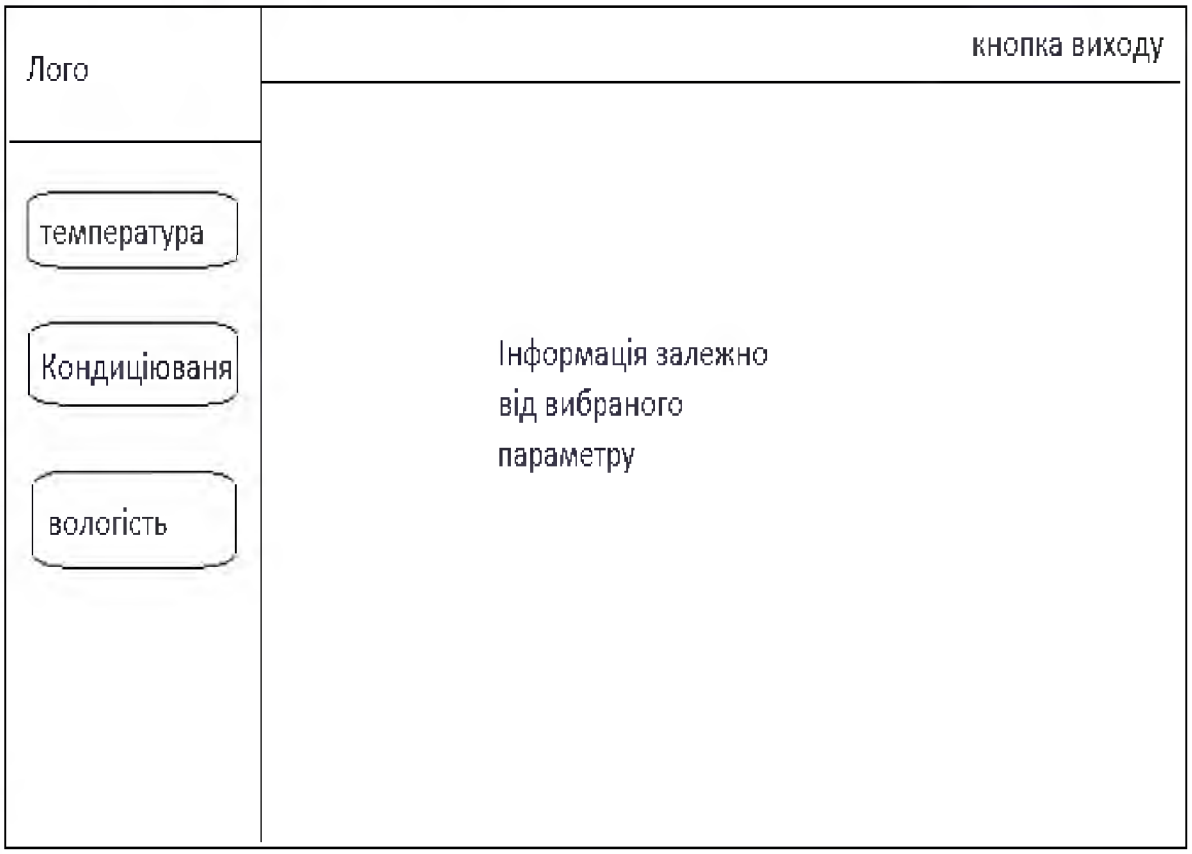


Рис. 3.5 Макет головної сторінки

Контрольна панель для цієї системи це окремий компонент який знаходиться на сторінці веб-додатку. Щоб отримати доступ потрібно перейти до додатку та авторизуватись. Вона відображає інформацію про дані мікроклімату і кількість кімнат підключених до системи та можливість їх маштабування(рис. 3.6). Також на окремій сторінці представлена інформація щодо параметрів кімнат та можливість їх зміни.

Головним недоліком такої панелі керування є те, що не приходять повідомлення про стан системи тому потрібно тримати додаток відкритим. Але цю проблему можливо вирішити підключивши функцію відправки повідомлень на телефон.

Рис. 3.6 Сторінка маштабування

Після маштабування будинку всі додані кімнати додаються у пункт керування, в якому можна проводити маніпуляції з системою (рис. 3.7).

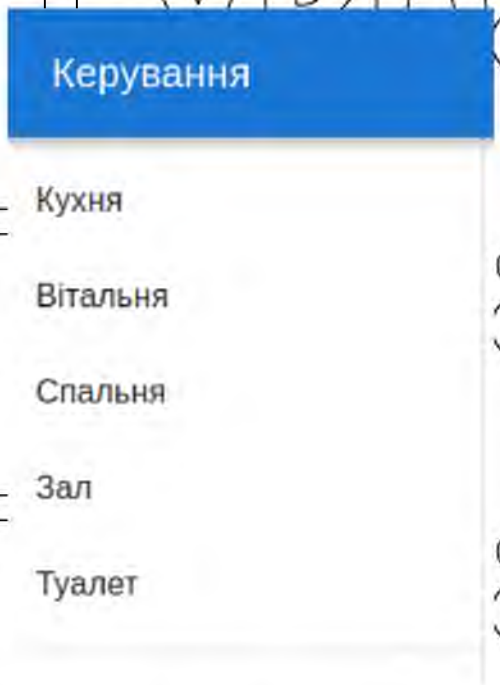


Рис. 3.7 Список кімнат для керування

3.3 Проектування архітектури ПЗ

Розділ специфікації до ПЗ описує зовнішні інтерфейси та компоненти. Також він вказує на клієнт-серверну архітектуру системи. Компоненти клієнт та сервер підлягають до процесу проектування та реалізації, оскільки інші є для того, щоб показати зв'язок який має показати система. Тому під проектуванням ПЗ розуміється саме проектування цих компонентів та їх зв'язки.

В розділі специфікації вимогою є саме розробка інтерфейсу у вигляді SPA. Звичайно існує багато технологій для реалізації цієї вимоги але розглянемо популярні методи. Веб-додатки які використовують технологію SPA імітують роботу настільних додатків. Їх архітектура влаштована так, що при переході на нову сторінку оновлюється тільки контент тому не потрібно буде завантажувати одні й ті самі елементи інтерфейсу.

Основні переваги SPA:

- продуктивність;
- Достатній вибір фреймворків та бібліотек, які дозволяють зробити великою швидкість розробки;
- В SPA є можливість повторного використання коду, тобто можливість використовувати повторно одні й ті ж самі компоненти.

SPA технологія використовується для вирішення не всіх задач інтерфейсу через ряд недоліків:

- недостатня оптимізація;
- при вимкненому JS інтерфейс не доступний, але мало людей вимикають його в браузері тому причин для хвилювання немає;
- SPA залежний від JS це мова з низькою типізацією та замовчуванням,

Проблема поганої SEO який використовує SPA вирішується застосуванням SSR для отримання HTML документів при заниті на сторінку.

Фреймворк — це програмний продукт, що спрощує створення складних та навантажених проектів. Фреймворк містить тільки базові модулі а всі інші

компоненти розробляються на їх основі. З цього ми маємо високу швидкість розробки і продуктивність.

Для створення клієнтської частини потрібно обрати фреймворк який підтримує SPA. Популярніші з них це Vue та React. Та було складено їх порівняння (рис. 3.8).

Рис. 3.8 Порівняння популярності

Через те що React являється одним з найпопулярніших фреймворків, то він буде використовуватись для розробки клієнтської частини. Надалі знайти тих хто може доробити систему буде набагато легше. На даний момент він являється стабільним та оновлюється тому що має підтримку багатьох компаній та має чудову документацію та невеликий поріг входу.

З React найчастіше використовують бібліотеку Redux для реалізації Flux підходу. Це набір шаблонів для побудови користувацького інтерфейсу всіх додатків які побудовані на односпрямованих потоках.

React діє з принципу, що логіка рендерингу пов'язана з логікою UI частини. Замість того щоб розділити логіку React пов'язує все в компоненти, які поміщають в себе все. React можна використовувати і без JS, але більшість людей цінує його за наочність при роботі з UI, що живе в JavaScript-коді. Крім цього

JSX допомагає React робити повідомлення про помилки і попередження зрозумілішe. Компоненти складються з двох основ це state та props. Така структура називається деревом і вона може перекидати відповідальність на один компонент. Наступний відповідає за масив та передає інформацію через props.

Дерево компонентів дозволяє створювати складні інтерфейси. Наприклад додаток не буде плутати свій стан з іншим. Відповідно до цього додаток буде слідувати правильному шляху, щоб відображати правильну інформацію і правильно її оновлювати (рис. 3.9).

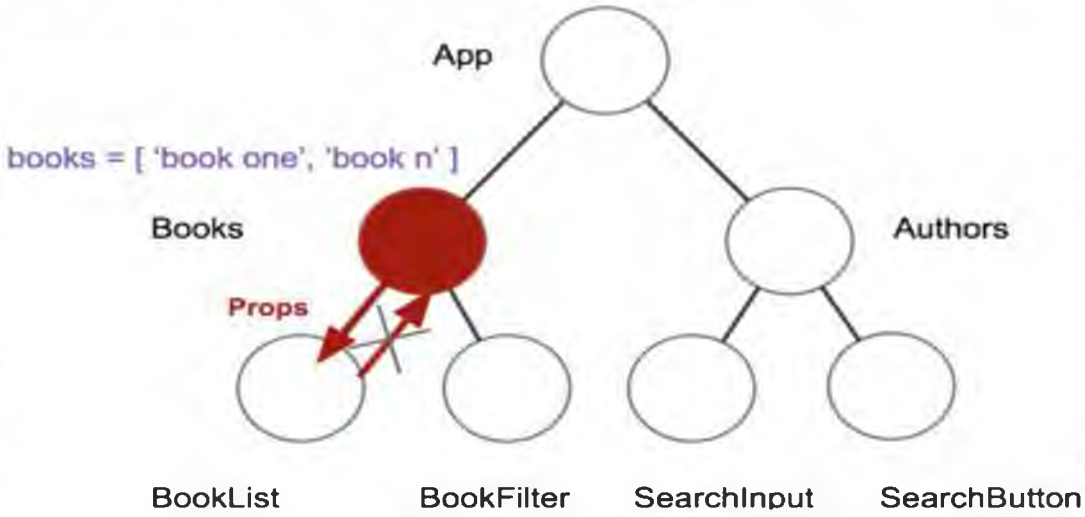


Рис. 3.9 Ієрархія класів

Створення проекту на реакт доволі не складна вона виконується однією командою.

Лістинг 3.1 — Створення проекту

```
npx create-react-app palamarhome
```

Як описано у вимогах рішенням створення клієнта обрана технологія Flux архітектури. Це рішення які використовуються для реалізації та керування контейнером веб-додатку. Саме спільнота розробляє бібліотеку React.js, тому було розроблено та запропоновано розробникам реалізацію цього підходу для React у вигляді бібліотеки Redux. Усі додатки з використанням JS-програмування, використовують компонентну систему. Кожен компонент може складатися з компонентів і має свій життєвий цикл і декоди може містити свій

стан. Компонент може керувати своїм станом і повідомляти свої дочірні компоненти. На рисунку 3.10 показано життєвий цикл React компоненту.

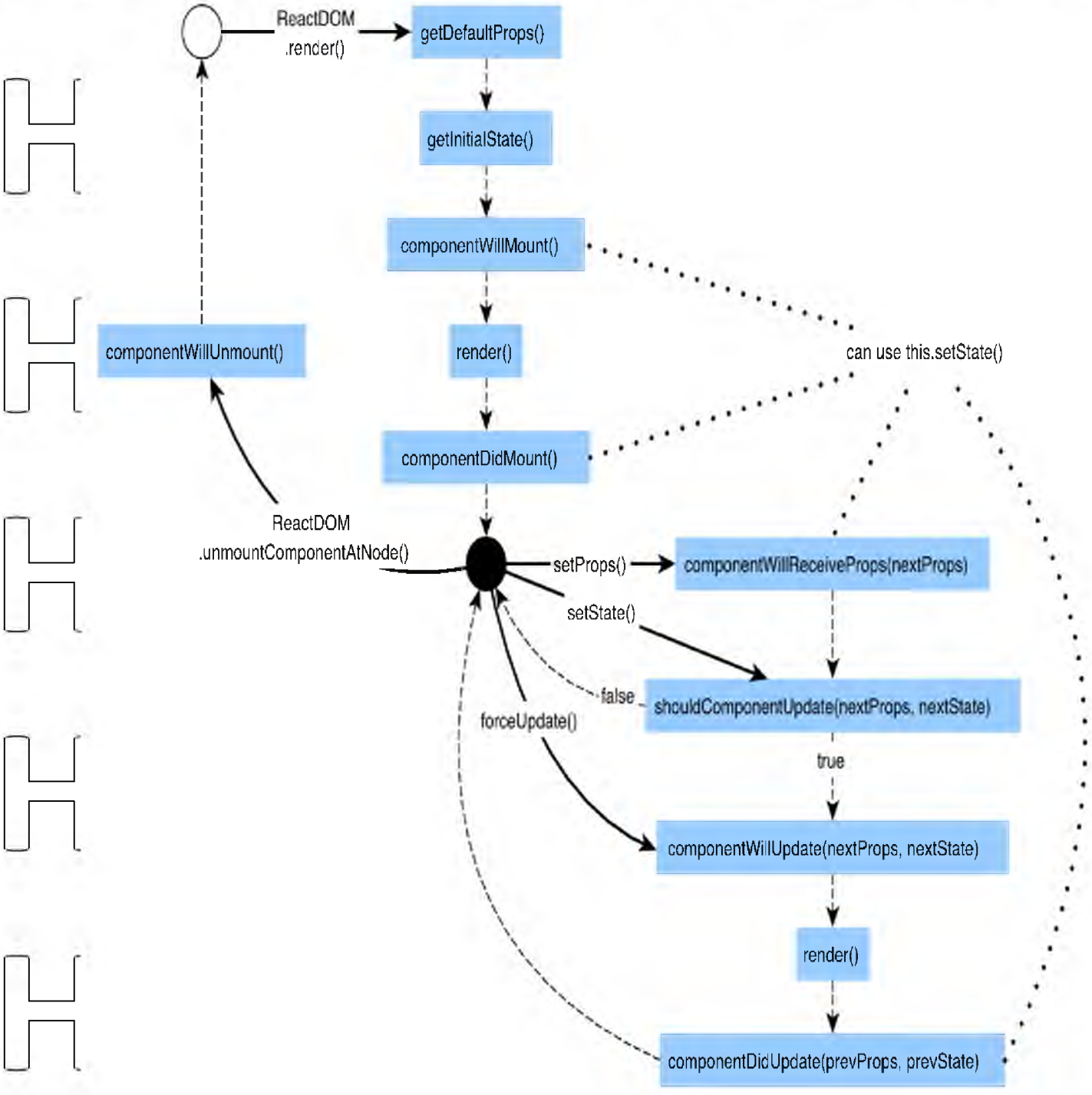


Рис. 3.10 Життєвий цикл React компоненту

Через це існує перевага повторного використання коду, компонент

можна використовувати всередині іншого, які також мають свій стан. Через збільшення проекту, зростає складність керування тому, що збільшується кількість компонентів та їх взаємодія як один з одних та стани один з одним З'являються проблеми з керуванням, так як взаємодії не завжди явні.

Через це компонентами тяжко керувати, потрібно скласти правильні підходи в системі ще на етапі проектування з передбаченням розвивання проекту.

У Redux є стан який представляє стан всієї системи. Після будь-якої події яка виникає в додатку зі сторони користувача дія з даними надсилається до функції що виконує певні операції і оновлює стан. Після таких дій створюється вигляд який бачить користувач з відображенням змін.

Завдяки Redux компоненти відокремлюються внаслідок чого інтерфейс стає простішим. Тоді у функціях закладена бізнес логіка.

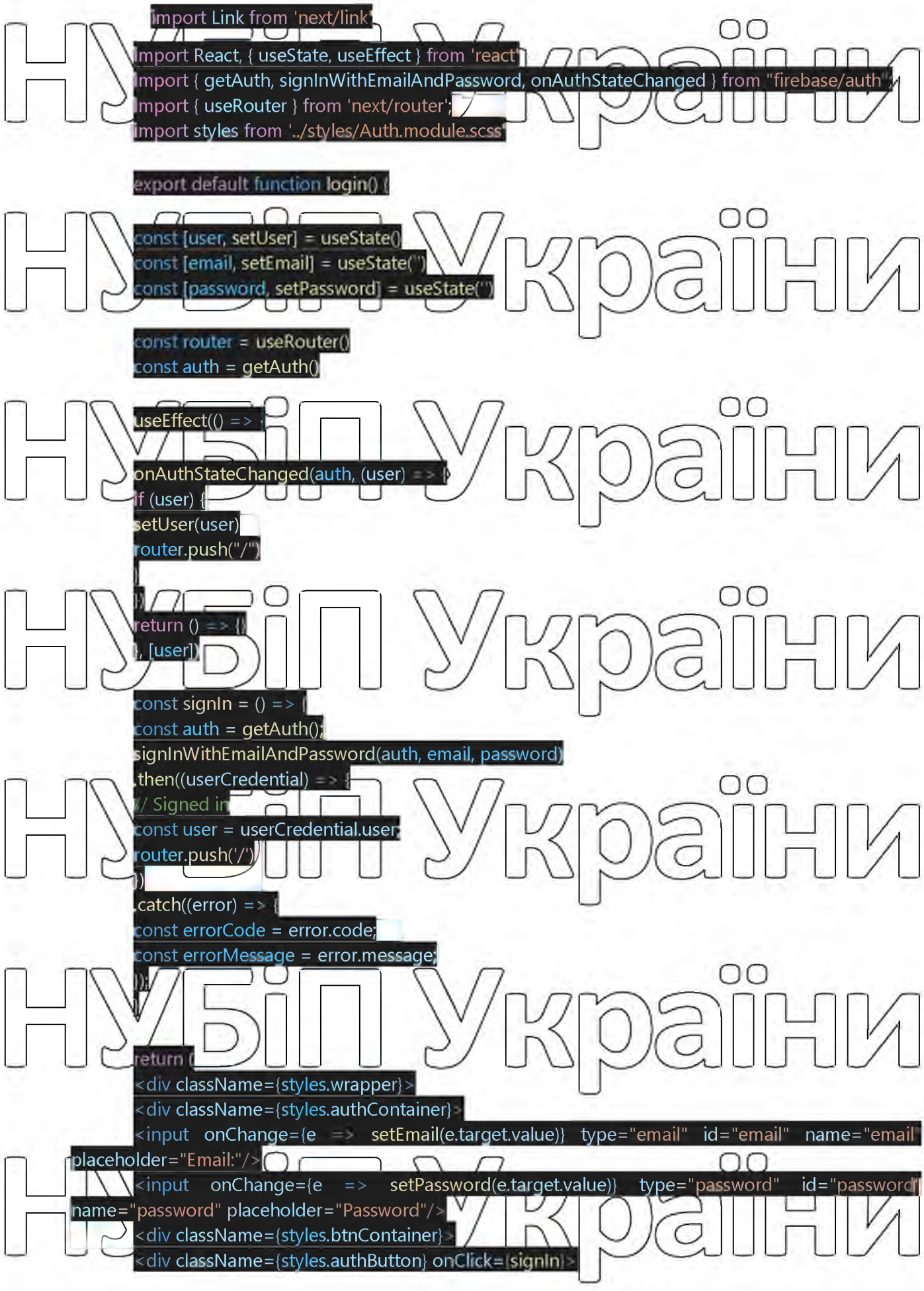
Так як у якості хостингу використовується Google Firebase то потрібно пам'ятати про переваги даного сервісу. На його базі буде розроблена реєстрація та вхід у додаток.

Отже розглянемо код реєстрації у додаток (Лістинг 3.2).

```
import Link from 'next/link'

import React, { useState } from 'react'
import { getAuth, createUserWithEmailAndPassword } from "firebase/auth";
import { doc, setDoc, collection } from "firebase/firestore";
import { db } from '../firebase';
import { useRouter } from 'next/router'
import styles from '../styles/Auth.module.scss'

export default function registr() {

const [email, setEmail] = useState('')
const [password, setPassword] = useState('')
const [fullName, setFullName] = useState('')

const router = useRouter()

const signUp = () => {
const auth = getAuth();
createUserWithEmailAndPassword(auth, email, password)
.then((userCredential) => {
// Signed in
```

```
const user = userCredential.user;
console.log(user.uid)
setDoc(doc(db, "users", `${user.uid}`), {
name: fullName,
email: email,
rooms: []
}).then(() => {
router.push("/")
})
.catch((error) => {
const errorCode = error.code;
const errorMessage = error.message;
// ..
});
}
return (
<div className={styles.wrapper}>
<div className={styles.authContainer}>
<input onChange={e => setFullName(e.target.value)} type="text" id="fullname" name="email"
placeholder="Full name:"/>
<input onChange={e => setEmail(e.target.value)} type="email" id="email" name="email"
placeholder="Email:"/>
<input onChange={e => setPassword(e.target.value)} type="password" id="password"
name="password" placeholder="Password"/>
<div className={styles.btnContainer}>
<div className={styles.authButton} onClick={signUp}>
<p>Sign up</p>
</div>
<div className={styles.goBack}>
<Link href="/login">
<a>Sign In</a>
</Link>
</div>
</div>
</div>
</div>
```

З вище написаного коду можем побачити, що контекст отримує дані з бібліотеки Firebase. У випадку коли користувач авторизований то йому вже потрібна форма авторизації. Розглянемо її код нижче (Лістинг 3.3).



```
import Link from 'next/link'
import React, { useState, useEffect } from 'react'
import { getAuth, signInWithEmailAndPassword, onAuthStateChanged } from "firebase/auth";
import { useRouter } from 'next/router';
import styles from '../styles/Auth.module.scss'

export default function login() {

const [user, setUser] = useState()
const [email, setEmail] = useState('')
const [password, setPassword] = useState('')

const router = useRouter()
const auth = getAuth()

useEffect(() => {
onAuthStateChanged(auth, (user) => {
if (user) {
setUser(user)
router.push("/")
}
});
return () => {}
}, [user])

const signIn = () => {
const auth = getAuth();
signInWithEmailAndPassword(auth, email, password)
.then((userCredential) => {
// Signed in
const user = userCredential.user;
router.push('/')
})
.catch((error) => {
const errorCode = error.code;
const errorMessage = error.message;
});
}

return (
<div className={styles.wrapper}>
<div className={styles.authContainer}>
<input onChange={e => setEmail(e.target.value)} type="email" id="email" name="email" placeholder="Email:"/>
<input onChange={e => setPassword(e.target.value)} type="password" id="password" name="password" placeholder="Password"/>
<div className={styles.btnContainer}>
<div className={styles.authButton} onClick={signIn}>
```

```
<p>Sign In</p>
</div>
<div className={styles.goBack}>
<Link href="/registr">
<a>Sign Up</a>
</Link>
</div>
</div>
</div>
</div>
```

Слід зауважити що у вападку успішної авторизації користувача перекину на годовну сторінку додатку, де він зможе керувати всією підсистемою. Розглянемо код головної сторінки (Лістинг 3.4).

```
import {React, useState, useEffect} from 'react';
import Box from '@mui/material/Box';
import Card from '@mui/material/Card';
import CardActions from '@mui/material/CardActions';
import CardContent from '@mui/material/CardContent';
import Button from '@mui/material/Button';
import Typography from '@mui/material/Typography';
import Image from 'next/image';
import trashIcon from '../../public/delete.png'

import { doc, updateDoc, arrayRemove } from "firebase/firestore";
import {getAuth, onAuthStateChanged} from 'firebase/auth'
import {db} from '../firebase'

export default function RoomCard({roomName}) {

const [user, setUser] = useState()

const auth = getAuth()

useEffect(() => {

onAuthStateChanged(auth, (user) => {
if (user) {
user = auth.currentUser.uid

setUser(user)
}
```

```
return () => {
}, [user])

const removeRoom = () => {
updateDoc(doc(db, "users", `${user}`), {
rooms: arrayRemove(roomName)
});
}

return (
<Card
sx={{
minWidth: 410,
minHeight: 250,
backgroundColor: '#145ea8',
'.css-10xx7c7-MuiPaper-root-MuiCard-root': {
backgroundColor: '#fff'
}
}}>
<CardContent>
<Box sx={{
display: 'flex',
alignItems: 'center',
justifyContent: 'space-between',
width: '100%'
}}>
<Typography variant="h5" component="div">
{roomName}
</Typography>
<Box onClick={removeRoom} sx={{
cursor: 'pointer'
}}>
<Image src={trashIcon} width="30px" height="30px" alt="Trash"/>
</Box>
</Box>
</CardContent>
<Box sx={{
display: 'flex'
}}>
<CardActions>
<Button size="large">Info</Button>
</CardActions>
</Box>
</Card>
);
```

На головній сторінці для авторизованого користувача відкриті всі можливості додатку. Під час маштабування коли користувач додає кімнати

всі інформація а також дані з датчиків записуються в базу яку надає Google Firebase а саме Firestore. Використовуючи пакетний менеджер npm який надає нам Node.js потрібно підключити її до проекту (Лістинг 3.5).

Лістинг 3.5 — підключення Firebase

```
npm install firebase
```

Після встановлення потрібно вказати налаштування бази даних такі як домен, апі ключ і т.д.. Налаштування зображено на (Лістинг 3.6).

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
apiKey: "AIzaSyDex1EVr-FKp7-HfFdjfK1DcxiUA0h63nc",
authDomain: "diploma-ae7a7.firebaseapp.com",
projectId: "diploma-ae7a7",
storageBucket: "diploma-ae7a7.appspot.com",
messagingSenderId: "402441664245",
appId: "1:402441664245:web:4cc126ae56b2e34de1b7bc"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
export const db = getFirestore(app)
```

Після цих маніпуляцій ми отримуємо БД від Firestore (рис. 3.11).

Рис. 3.11 База даних з кімнатами

Також в цю базу даних ми отримуємо і авторизованих користувачів отримуючи їх пошту та пароль тому як під умову реєстрації підлягають кільки ці параметри (рим. 3.12).

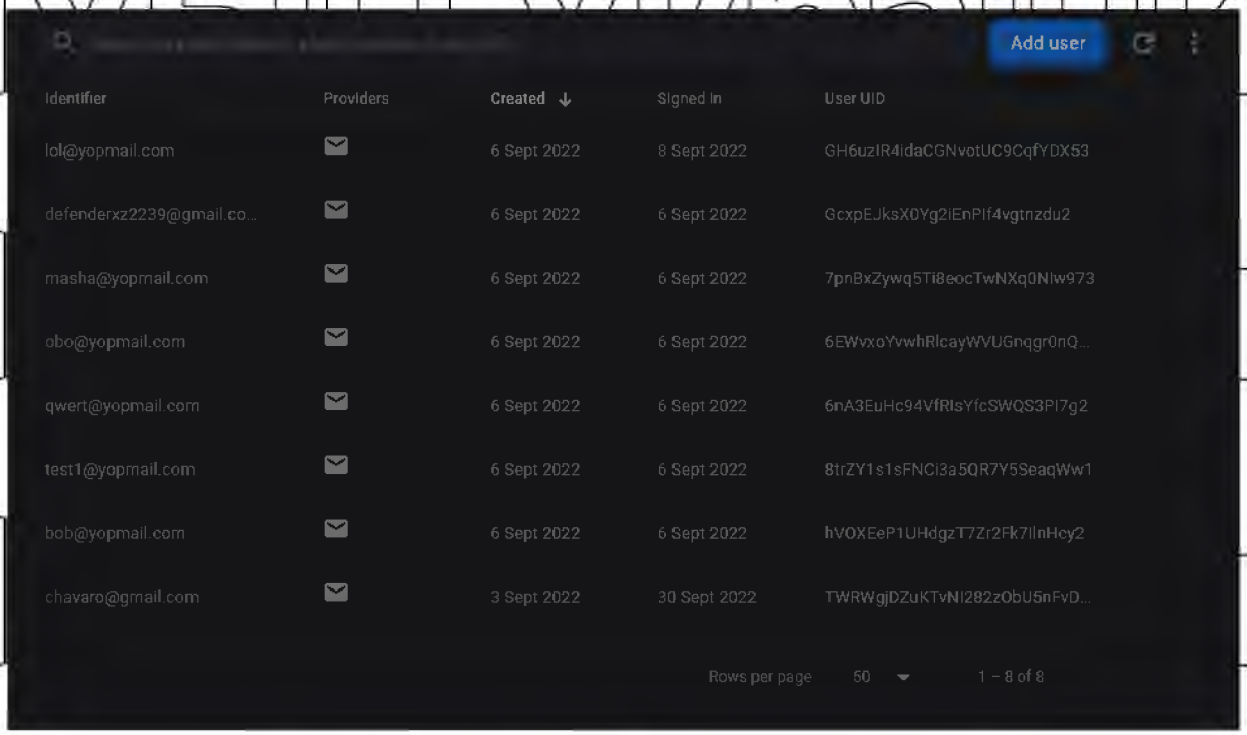


Рис. 3.12 База даних користувачів

Для розробки візуальних компонентів в роботі було використано бібліотеку Material UI.

Material UI — це набір готових стилізованих компонентів бібліотеки React який реалізує Material Design. Їх перевага що вони працюють ізольовано, тобто вони являются самопідтримуючими і вводять тільки ті стилі які вони повині в дображати. Щоб почати роботу з ними потрібно в терміналі проекту запустити команду вказану в (Лістинг 3.6).

Лістинг 3.6 — Встановлення стилів

```
npm install @material-ui/core
```

Крім цього для коррект ної роботи додатку потрібно встановити ще деякі залежності. Всі вони встановлюються за допомогою пакетного менеджера npm. Також існує ще один і користувач сам може вибирати зручніший для себе. Перелік всіх встановлених залежностей розглянемо в (Лістинг 3.7).

```
{
  "name": "diploma",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "@devexpress/dx-core": "^3.0.5",
    "@devexpress/dx-react-chart": "^3.0.5",
    "@devexpress/dx-react-chart-material-ui": "^3.0.5",
    "@devexpress/dx-react-core": "^3.0.5",
    "@emotion/react": "^11.10.4",
    "@emotion/styled": "^11.10.4",
    "@mui/icons-material": "^5.10.3",
    "@mui/material": "^5.10.3",
    "apexcharts": "^3.35.5",
    "firebase": "^9.9.4",
    "hoist-non-react-statics": "^3.3.2",
    "next": "12.2.5",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-dropdown": "^1.10.0",
```

```
"react-charts": "^2.1.14",
"sass": "^1.54.8",
"victory": "^36.6.7"
},
"devDependencies": {
"eslint": "8.23.0",
"eslint-config-next": "12.2.5"
}
```

Відносно модального вікна для створення карток кімнат там теж використовується стилізація Material UI. В ньому користувач вводить данні про кімнату і після того як натисне кнопку збереження, то всі данні перейдуть до Firestore. Код модального вікна розглянемо в (Лістинг 3.8).

```
import {React, useState, useEffect} from 'react';
import Box from '@mui/material/Box';
import Modal from '@mui/material/Modal';
import Button from '@mui/material/Button';
import { doc, setDoc, updateDoc, onSnapshot, arrayUnion } from "firebase/firestore";
import {getAuth, onAuthStateChanged} from 'firebase/auth'
import {db} from '../firebase'

const style = {
position: 'absolute',
top: '50%',
left: '50%',
transform: 'translate(-50%, -50%)',
width: 400,
bgcolor: 'background.paper',
border: '2px solid #000',
boxShadow: 24,
pt: 2,
px: 4,
pb: 3,
};

export default function NestedModal(props) {

const {isOpen, bumer} = props

const [open, setOpen] = useState(false);
const [user, setUser] = useState()
const [nameRoom, setNameRoom] = useState("");
```

```
const [roomsJson, setRoomsJson] = useState({
email: '',
name: '',
rooms: []
})
const [rooms, setRooms] = useState([])

const auth = getAuth()

useEffect(() => {
if(!user) return

onSnapshot(doc(db, "users", `${user}`), (doc) => {
setRoomsJson(doc.data())
});

},[user])

useEffect(() => {

if(!roomsJson) return

setRooms(Object.values(roomsJson["rooms"]))

}, [roomsJson])

useEffect(() => {

onAuthStateChanged(auth, (user) => {
if (user) {
user = auth.currentUser.uid

setUser(user)

}
});

return () => {}
}, [user])

const addRoom = () => {
bumer(false);

if(rooms.length === 0){
setDoc(doc(db, "users", `${user}`), {
// rooms: [{nameRoom: nameRoom, slug: nameRoom}]
```

```
rooms: [nameRoom]
}, { merge: true });
};

updateDoc(doc(db, "users", `${user}`), {
rooms: arrayUnion(nameRoom)
});
};

const handleClose = () => {
bumer(false);
};

return (
<div>
<Modal
open={isOpen}
onClose={handleClose}
aria-labelledby="parent-modal-title"
>
<Box sx={{ ...style, width: 400 }}>
<h2 id="parent-modal-title">Введіть назву кімнати</h2>
<input type="text" onChange={e => setNameRoom(e.target.value)}/>
<Box
sx={{
display: 'flex',
}}
>
<Button onClick={addRoom}>Створити</Button>
<Button onClick={handleClose}>Скасувати</Button>
</Box>
</Box>
</Modal>
</div>
);
}
```

Тестування клієнтської частини відбувалось ручним тестуванням. Ручне тестування – це взаємодія людини та додатку на пряму. В цьому процесі можна отримати зворотній зв’язок про продукт, але це не можливо при автоматизованому тестуванні.

Зазвичай після тестування складається детальний відгук та рекомендації щодо того як можна удосконалити продукт. Також можна порівняти результати очікуваний та отриманий.

Ще особливістю ручного тестування є зворотній зв'язок щодо дизайну інтерфейсу. Так як тільки людина може помітити нюанси кольорів чи читабельність тексту.

На сторінці гостя знаходиться форма реєстрації і форма авторизації. При розробці потрібно було зробити простий інтерфейс, щоб користувач міг швидко розібратись у використанні додатку та авторизуватись у системі. На головній сторінці знаходиться панель керування мікрокліматом. Основною метою було створення простого інтерфейсу керування, щоб користувач міг швидко встановити потрібні характеристики температури, кондиціонування а також вологості повітря. Також, щоб користувач мав можливість переглянути статистику по всім цим параметрам.

ВИСНОВКИ

У дипломній роботі на тему дослідження параметрів управління мікрокліматом у розумному будинку були розглянуті основні завдання а також алгоритми управління та дослідження їх параметрів.

У результаті виконання роботи було спроектовано та розроблено веб-додаток для управління мікрокліматом.

У ході роботи вирішено наступні задачі:

1. Аналіз та опис предметної області програмного продукту. Виділено переваги розробки продукту, його цілі, сегмент споживачів та затрати на підтримку, що значно допомогло у розробці специфікації вимог до системи.

2. Досліджено алгоритми задання параметрів мікроклімату. Було побудовано блок-схеми процесу роботи підсистеми, та подано можливе рішення щодо покращення роботи та досягнення оптимальної ціни.

3. Спроектовано систему, розроблено функціональні вимоги до системи, проведено опис діаграм використання системи за типами користувачів за допомогою UML. Обгрунтовано актуальність технологій розробки, використаних технологій, бібліотек, фреймворків, архітектурних рішень та методологій , що забезпечують швидкість, якість та продуктивність роботи системи.

4. Описано кроки реалізації, наведено код найважливіших компонентів та описано основні архітектурні та методологічні рішення як серверної, так і клієнтської частини проекту.

5. Проведено ручне тестування системи. За результатами тестування створеного додатку показано, що він відповідає вимогам які були створені до неї.

Отримана система являється повністю автоматизована і додає комфорту людині, так як керувати мікрокліматом свого будинку через телефон або комп'ютер це дуже зручно. В подальшому систему можна

вдосконалити, і зробити її повноцінною для всіх розділів Розумного будинку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Дубровин Д.А. РАЗРАБОТКА СИСТЕМЫ "УМНЫЙ ДОМ": научная статья / Д.А. Дубровин – Москва: Финансовый университет при Правительстве Российской Федерации, 2015.
2. Недостатки системы "Умный дом" [Электронный ресурс] // Сонэс. – Режим доступа: http://sones.ru/stati/nedostatki-sistemy-umnyi-dom.html
3. Системы безопасности «умного дома» [Электронный ресурс] // ВИРА-АРТСТРОЙ. – Режим доступа: http://www.eremont.ru/enc/engineer/clever/security-system-smart-home.html
4. Умный дом своими руками [Электронный ресурс] // Dom-electro. – Режим доступа: http://www.dom-electro.ru/умный-дом-своими-руками/
5. Диаграмма прецедентов. URL: https://en.wikipedia.org/wiki/Use_case_diagram
6. Почему интуитивно понятный интерфейс важен для веб-дизайна и разработки программного обеспечения. URL: https://it-wo.ru/it-tehnologii/pochemu-intuitivno-ponyatnyj-interfejs-nastolko-vazhen-dlya-veb-dizajna-i-razrabotki-programmnogo-obespecheniya
7. Single Page Application (SPA) та Multi Page Application (MPA): Переваги та Недоліки. URL: https://merehead.com/ru/blog/single-page-application-vs-multi-page-application/
8. Deploying to Firebase Hosting. URL: https://www.gatsbyjs.org/docs/deploying-to-firebase/
9. Add Firebase to your JavaScript project. URL: https://firebase.google.com/docs/web/setup
10. Памятка UX / UI дизайнеру. 19 принципов построения

интерфейсов. URL: https://habr.com/ru/company/SECL_GROUP/blog/182208/

11. Контекст - React. URL: https://ru.reactjs.org/docs/context.html

12. Анотація до аналізу ринку. URL: https://pro-consulting.ua/ua/issledovanie-rynka/analiz-rynka-sistem-umnyj-dom-v-ukraine-2019-god

13. Системи автоматизації для дому та квартири. URL: https://alterair.ua/avtomatizaciya/dlya-doma-i-kvartiry-pod-klyuch/

14. Розумний будинок LifeSmart — можливості та огляд складових системи. URL: https://bezpeka.club/rozumnyj-budynok-lifesmart-mozhlyvosti-ta-oglyad-skladovyh-systemy/

15. Дизайн сайтів. Як самостійно зробити шаблон сайту URL: https://webstudio2u.net/ua/design-site/257-template-how-to.html.

16. A JavaScript library for building user interfaces URL: https://reactjs.org/

17. Розумний дім. URL: https://uk.wikipedia.org/wiki/Розумний_дім/

18. Знайомство з “розумним будинком”: з чого почати встановлення? URL: https://worldvision.com.ua/4-prostykh-shaga-dlya-nastroyki-vashego-pervogo-umnogo-doma/

19. Material-UI doc. URL: https://mui.com/getting-started/installation/

20. ТОП 5 систем «Умный дом» по производителям. URL: https://vencon.ua/articles/reyting-sistem-umnyy-dom-po-proizvoditelyam

21. СТРУКТУРИ ТА АЛГОРИТМИ РОБОТИ ПІДСИСТЕМ УПРАВЛІННЯ МІКРОКЛІМАТОМ ТА ОСВІТЛЕННЯМ РОЗУМНОГО БУДИНКУ. URL: https://nv.nltu.edu.ua/Archive/2018/28_1/24.pdf

22. Home Sapiens. (n.d.). Home Sapiens – programmnoe obespechenie umnogo doma. Retrieved from: http://techvesti.ru/node/4627. [In Russian].

23. Hrytsiuk, Yu. I., Nazar, M. B., & Polishchuk, M. O. (2010). Alhorytmy ta metody styskannia danykh. Visnyk Lvivskoho derzhavnoho universytetu bezpeky zhyttiedialnosti: zb. nauk. Prats, 4(1), 7–13. Lviv: Vyd-vo LDU BZhD. [In Ukrainian].

24. Inzhenernye sistemy. (n.d.). Inzhenernye sistemy vashei kvartiry i doma. Retrieved from: http://ingsvd.ru/main/smarthome [In Russian].

25. Kliuiko, Yu. I., & Zlotenko, B. M. (2015). Rozrobka intelektualnoi systemy keruvannia osvitlenniam "rozumnoho budynku". Tekhnolohii ta dyzain, 2(15). Retrieved from: http://nbuv.gov.ua/ UJRN/td_2015_2_8. [In Ukrainian].

26. Medykovskyi, M. O., Tkachenko, R. O., Tsmots, I. H., Tsymbal, Yu. V., Doroshenko, A. V., & Skorokhoda, O. V. (2015). Intelektualni komponenty intehrovanykh avtomatyzovanykh system upravlinnia. Lviv: Vydavnytstvo Lvivskoi politekhniky. 315 p. [In Ukrainian].

27. Obzornaia statia o sisteme "Umnyi dom". (n.d.). Retrieved from: http://portfolio.textsale.ru/10/12343 [In Russian].

28. Tesliuk, V. M., Berezkyi, O. M., Berehovskyi, V. V., & Tesliuk, T. V. (2012). Rozroblennia neirokontrolera dlia upravlinnia pidsystemoiu osvitlennia intelektualnoho budynku. Zbirnyk naukovykh prats IPME im. H. Ye. Pukhova NAN Ukrainy, 64, 137–143. [In Ukrainian].

29. Umnyj Dom. (n.d.). Retrieved from: http://electronic-home.com.ua/. [In Russian].

30. Vozmozhnosti Umnogo Doma. (n.d.). URL: http://www.bau.ua/library/art-vozmozhnosti_umnogo_doma [In Russian].

Додаток А

Layout.js

```
import React, { Fragment, Component } from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import Header from "./../../containers/Header/Header";
import SideDrawer from "../../containers/SideDrawer/SideDrawer";
import classes from "./Layout.module.scss";
import { closeErrorModal } from "./../../store/ui/ui.actions";
import Modal from "./../../components/UI/Modal/Modal";
import Login from "../../containers/Login/login";
export class Layout extends Component {
static propTypes = {
isSideDrawerOpen: PropTypes.bool
};
render() {
if(window.location.pathname === '/'){
return(
<>
<Login />
</>
)
}
else{
return (
<Fragment>
<Header />
<SideDrawer isOpen={this.props.isSideDrawerOpen} />
<Modal
data-test="errors-modal"
```

```
show={!!this.props.error}
onCloseModal={this.props.closeErrorModal}
>
{this.props.error && this.props.error.message}
</Modal>
<main className={classes.Main}>{this.props.children}</main>
</Fragment>
);
}
}
const mapStateToProps = state => ({
error: state.ui.error,
isSideDrawerOpen: state.ui.openSideDrawer
});
const mapDispatchToProps = {
closeErrorModal
};
export default connect(
mapStateToProps,
mapDispatchToProps
)(Layout);
```

SideDrawer.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { NavLink } from 'react-router-dom';
import { connect } from "react-redux";
import Navigation from "../../components/Layout/Navigation/Navigation";
```

```
importNavigationItemfrom
"../../components/Layout/Navigation/NavigationItem/NavigationItem";
import { toggleSideDrawer } from "../../../store/ui/ui.actions";
import Button from "../../components/UI/Button/Button";
import classes from "./SideDrawer.module.scss";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faWindowClose } from "@fortawesome/free-solid-svg-icons";
import Backdrop from "../../components/UI/Backdrop/Backdrop";
import { Link } from "@mui/material";
export class SideDrawer extends Component {
static propTypes = {
isOpen: PropTypes.bool
};
render() {
const sideDrawerContainerClasses = [classes.SideDrawerContainer];
if (this.props.isOpen) {
sideDrawerContainerClasses.push(classes.Open);
}
return (
<div className={sideDrawerContainerClasses.join(" ")}>
<Backdrop show={this.props.isOpen} onClick={this.props.toggleSideDrawer}
/>
<Button                                    className={classes.CloseDrawerBtn}
onClick={this.props.toggleSideDrawer}>
<FontAwesomeIcon icon={faWindowClose} />
</Button>
<div className={classes.SideDrawer}>
<div className={classes.Title}>Меню</div>
<Navigation>
<NavigationItem>
```

```
<NavLink to="/dashboard">Кімнати</NavLink>
</NavigationItem>
</Navigation>
<Link href="/">Вихід</Link>
</div>
</div>
);
}
}
const mapStateToProps = state => ({
isOpen: state.ui.openSideDrawer
});
const mapDispatchToProps = {
toggleSideDrawer
};
export default connect(
mapStateToProps,
mapDispatchToProps
)(SideDrawer);
```

Header.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { NavLink } from "react-router-dom";
import { connect } from "react-redux";
import { toggleSideDrawer } from "./../../store/ui/ui.actions";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faBars } from "@fortawesome/free-solid-svg-icons";
import Button from "./../../components/UI/Button/Button";
import Navigation from "./../../components/Layout/Navigation/Navigation";
```

```
import NavigationItem from "./../../components/Layout/Navigation/NavigationItem/NavigationItem";
import classes from "./Header.module.scss";
import { Link } from "@mui/material";

export class Header extends Component {
static propTypes = {
toggleSideDrawer: PropTypes.func
};

render() {
return (
<header className={classes.Header}>
<div className={classes.HeaderContainer}>
<div className={classes.AppName}>Palamar Home</div>
<div className={classes.Navigation}>
<Navigation>
<NavigationItem>
<NavLink to="/dashboard">Кімнати</NavLink>
</NavigationItem>
</Navigation>
<Link href="/">Вихід</Link>
</div>
<div className={classes.MenuBtn}>
<Button onClick={this.props.toggleSideDrawer}>
<FontAwesomeIcon icon={faBars} />
</Button>
</div>
</div>
```

```
</header>
);
}
}

const mapDispatchToProps = {
toggleSideDrawer
};

export default connect(
null,
mapDispatchToProps
)(Header);
```

Login.js

```
import * as React from 'react';
import Avatar from '@mui/material/Avatar';
import Button from '@mui/material/Button';
import CssBaseline from '@mui/material/CssBaseline';
import TextField from '@mui/material/TextField';
import FormControlLabel from '@mui/material/FormControlLabel';
import Checkbox from '@mui/material/Checkbox';
import Link from '@mui/material/Link';
import Paper from '@mui/material/Paper';
import Box from '@mui/material/Box';
import Grid from '@mui/material/Grid';
import LockOutlinedIcon from '@mui/icons-material/LockOutlined';
import Typography from '@mui/material/Typography';
import { createTheme, ThemeProvider } from '@mui/material/styles';
```

```
function Copyright(props) {
return (
<Typography variant="body2" color="text.secondary" align="center"
{...props}>
{'Copyright © '}
<Link color="inherit" href="https://mui.com/">
Your Website
</Link>{' '}
{new Date().getFullYear()}
{'.'}
</Typography>
);
}

const theme = createTheme();

export default function Login() {

return (
<ThemeProvider theme={theme}>
<Grid container component="main" sx={{ height: '100vh' }}>
<CssBaseline />
<Grid
item
xs={false}
sm={4}
md={7}
sx={{
backgroundImage: 'url(https://source.unsplash.com/random)',
backgroundRepeat: 'no-repeat',
```

```
backgroundColor: (t) =>
t.palette.mode === 'light' ? t.palette.grey[50] : t.palette.grey[900],
backgroundSize: 'cover',
backgroundPosition: 'center',
}}
/>
<Grid item xs={12} sm={8} md={5} component={Paper} elevation={6} square>
<Box
sx={{
my: 8,
mx: 4,
display: 'flex',
flexDirection: 'column',
alignItems: 'center',
}}
>
<Avatar sx={{ m: 1, bgcolor: 'secondary.main' }}>
<LockOutlinedIcon />
</Avatar>
<Typography component="h1" variant="h5">
Вхід
</Typography>
<Box sx={{ mt: 1 }}>
<TextField
margin="normal"
required
fullWidth
id="email"
label="Email Address"
```

```
name="email"
autoComplete="email"
autoFocus
/>
<TextField
margin="normal"
required
fullWidth
name="password"
label="Password"
type="password"
id="password"
autoComplete="current-password"
/>
<FormControlLabel
control={<Checkbox value="remember" color="primary" />}
label="Запам'ятати мене"
/>
<Link href="/dashboard">
<Button
fullWidth
variant="contained"
sx={{ mt: 3, mb: 2 }}
>
Вхід
</Button>
</Link>
<Grid container>
<Grid item>
<Link href="#" variant="body2">
```

```
{"Реєстрація"}
</Link>
</Grid>
</Grid>
</Box>
</Box>
</Grid>
</Grid>
</ThemeProvider>
);
}
```

RoomsDashboard.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import { fetchRooms } from "./../../store/rooms/rooms.actions";
import Room from "../../components/Rooms/Room";

import classes from "./RoomsDashboard.module.scss";

export class RoomsDashboard extends Component {
static propTypes = {
fetchRooms: PropTypes.func,
rooms: PropTypes.object
};

componentDidMount() {
if (this.props.fetchRooms) {
this.props.fetchRooms();
```

```
}
}

onClickRoomHandler = roomId => {
// Go to room page
this.props.history.push(`/room/${roomId}`);
};

render() {
if (!this.props.rooms) return null;

return (
<div className={classes.Row}>
{Object.entries(this.props.rooms).map(roomData => {
const roomId = roomData[0];
const room = roomData[1];
return (
<div
data-test={`room-card-${roomId}`}
key={roomId}
className={classes.Column}
onClick={() => this.onClickRoomHandler(roomId)}
>
<Room
id={roomId}
name={room.name}
icon={room.icon}
devicesCount={room.devicesCount}
/>
</div>
```

```
);
})}
</div>
);
}
}

const mapStateToProps = state => ({
rooms: state.rooms.rooms
});

const mapDispatchToProps = {
fetchRooms
};

export default connect(
mapStateToProps,
mapDispatchToProps
)(RoomsDashboard);
```

App.js

```
import React from "react";
import { Route, Switch } from "react-router-dom";
import "./App.module.sess";
import Layout from "./hoc/Layout/Layout";
import RoomsDashboard from "./containers/RoomsDashboard/RoomsDashboard";
import asyncComponent from "./hoc/asyncComponent/asyncComponent";
```

```
import Login from "./containers/Login/login";

function App() {
const AsyncRoomsDevices = asyncComponent(() =>
import("./containers/RoomsDashboard/RoomDevices/RoomDevices")
);
return (
<Layout>
<Switch>
<Route path="/room/:id" exact component={AsyncRoomsDevices} />
<Route path="/dashboard" exact component={RoomsDashboard} />
<Route path="/" exact component={Login} />
</Switch>
</Layout>
);
}

export default App;
```

RoomDevices.js

```
import React, { Component, Fragment } from "react";
import PropTypes from "prop-types";
import { connect } from "react-redux";
import {
fetchRoomDevices,
toggleDeviceSwitch,
updateDeviceControlValue
} from "../../../store/devices/devices.actions";
import Device from "../../../components/Device/Device";
```

```
import { NavLink } from "react-router-dom";
import classes from "./RoomDevices.module.scss";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faChevronLeft } from "@fortawesome/free-solid-svg-icons";

export class RoomDevices extends Component {
static propTypes = {
fetchRoomDevices: PropTypes.func
};

state = {
mountFailded: false
};

componentDidMount() {
if(
!!this.props.match &&
!!this.props.match.params &&
!!this.props.match.params.id &&
!!this.props.fetchRoomDevices
){
const roomId = this.props.match.params.id;
this.props.fetchRoomDevices(roomId);
}
}

/**
* This for toggling the main switch of the device
*/
toggleDeviceSwitch = deviceId => {
```

```
this.props.toggleDeviceSwitch(deviceId);
};
/**
* Evenet handler when a device control value changed
*/
onControlValueChangedHandler = (deviceId, controlId, newValue) => {
this.props.updateDeviceControlValue({ deviceId, controlId, newValue });
};

render() {
if (!this.props.devices) return null;

return (
<Fragment>
<NavLink to="/dashboard" className={classes.BackLink}>
<FontAwesomeIcon icon={faChevronLeft} />
<span>До кімнат</span>
</NavLink>
<div className={classes.RoomDevices}>
{Object.entries(this.props.devices).map(device => {
const deviceId = device[0];
const deviceData = device[1];
return (
<div key={deviceId} className={classes.Column}>
<Device
deviceId={deviceId}
device={deviceData}
onToggleDeviceSwitch={() => this.toggleDeviceSwitch(deviceId)}
onControlValueChanged={this.onControlValueChangedHandler}
```

```
/>
</div>
);
})}
</div>
</Fragment>
);
}
}

const mapStateToProps = state => ({
devices: state.devices.devices
});

const mapDispatchToProps = {
fetchRoomDevices,
toggleDeviceSwitch,
updateDeviceControlValue
};

export default connect(
mapStateToProps,
mapDispatchToProps
)(RoomDevices);
```

devices.action.js

```
import {
FETCH_DEVICES_START,
FETCH_DEVICES_SUCCESS,
```

```
  FETCH_DEVICES_FAILED,
  TOGGLE_DEVICE_SWITCH_START,
  TOGGLE_DEVICE_SWITCH_SUCCESS,
  TOGGLE_DEVICE_SWITCH_FAILED,
  UPDATE_DEVICE_CONTROL_VALUE_START,
  UPDATE_DEVICE_CONTROL_VALUE_SUCCESS,
  UPDATE_DEVICE_CONTROL_VALUE_FAILED
} from "./devices.actiontypes";
import {
  getRoomDevicesApi,
  toggleDeviceSwitchApi,
  updateDeviceControlValueApi
} from "../../utils/api/devices.api";
import { showErrorModal } from "../ui/ui.actions";

/** Fetching Room Devices Actions */
export const fetchRoomDevices = roomId => dispatch => {
  dispatch(fetchRoomDevicesStart());

  getRoomDevicesApi(roomId)
    .then(response => dispatch(fetchRoomDevicesSuccess(response.data.devices)))
    .catch(error => {
      // This to mock an error response
      const errorResponse = {
        message: "Error while getting the devices data"
      };
      dispatch(fetchRoomDevicesFailed(errorResponse));
      dispatch(showErrorModal(errorResponse))
    });
};
```

```
export const fetchRoomDevicesStart = () => ({
type: FETCH_DEVICES_START
});

export const fetchRoomDevicesSuccess = devices => ({
type: FETCH_DEVICES_SUCCESS,
payload: {
devices: devices
}
});

export const fetchRoomDevicesFailed = error => ({
type: FETCH_DEVICES_FAILED,
payload: {
error
}
});

/** Device Switch Toggle Actions */
export const toggleDeviceSwitch = deviceId => dispatch => {
dispatch(toggleDeviceSwitchStart());

toggleDeviceSwitchApi(deviceId)
.then(response => dispatch(toggleDeviceSwitchSuccess(response.data.deviceId)))
.catch(error => {
// This to mock an error response
const errorResponse = {
```

```
        message: "Error while toggle the device switch"
      };
      dispatch(toggleDeviceSwitchFailed(errorResponse));
      dispatch(showErrorModal(errorResponse))
    });
  };

export const toggleDeviceSwitchStart = () => ({
  type: TOGGLE_DEVICE_SWITCH_START
});

export const toggleDeviceSwitchSuccess = deviceId => ({
  type: TOGGLE_DEVICE_SWITCH_SUCCESS,
  payload: {
    deviceId
  }
});

export const toggleDeviceSwitchFailed = error => ({
  type: TOGGLE_DEVICE_SWITCH_FAILED,
  payload: {
    error
  }
});

/** Updating Control Value Handler */
export const updateDeviceControlValue = controlData => dispatch => {
  const payload = {
    deviceId: controlData.deviceId,
```

```
        controlId: controlData.controlId,
        newValue: controlData.newValue
      };

      dispatch(updateDeviceControlValueStart());

      updateDeviceControlValueApi(payload)
        .then(response => dispatch(updateDeviceControlSuccess(response.data.control)))
        .catch(error => {
          // This to mock an error response
          const errorResponse = {
            message: "Error while updating the device value"
          };
          dispatch(updateDeviceControlValueFailed(errorResponse));
          dispatch(showErrorModal(errorResponse))
        });
    };

export const updateDeviceControlValueStart = () => ({
  type: UPDATE_DEVICE_CONTROL_VALUE_START
});

export const updateDeviceControlSuccess = control => ({
  type: UPDATE_DEVICE_CONTROL_VALUE_SUCCESS,
  payload: {
    ...control
  }
});
```

```
export const updateDeviceControlValueFailed = error => ({
type: UPDATE_DEVICE_CONTROL_VALUE_FAILED,
payload: {
error
}
});
```

rooms.actions.js

```
import {
FETCH_ROOMS_START,
FETCH_ROOMS_SUCCESS,
FETCH_ROOMS_FAILED
} from "../../store/rooms/rooms.actiontypes";
import { getRoomsApi } from "../../utils/api/rooms.api";
import { showErrorModal } from "../ui/ui.actions";

export const fetchRooms = () => dispatch => {
dispatch(fetchRoomsStart());
getRoomsApi()
.then(response => dispatch(fetchRoomsSuccess(response.data.rooms)))
.catch(error => {
// This to mock an error response
const errorResponse = {
message: "Error while getting the rooms data"
};
dispatch(fetchRoomsFailed(errorResponse));
```

```
    dispatch(showErrorModal(errorResponse));
  });
};

export const fetchRoomsStart = payload => ({
  type: FETCH_ROOMS_START
});

export const fetchRoomsSuccess = rooms => ({
  type: FETCH_ROOMS_SUCCESS,
  payload: {
    rooms
  }
});

export const fetchRoomsFailed = error => ({
  type: FETCH_ROOMS_FAILED,
  payload: {
    error
  }
});
```

index.js

```
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import { createStore, applyMiddleware, compose, combineReducers } from
"redux";
```

```
import thunk from "redux-thunk";
import { library } from "@fortawesome/fontawesome-svg-core";
import roomsReducers from "./store/rooms/rooms.reducers";
import devicesReducers from "./store/devices/devices.reducers";
import uiReducers from "./store/ui/ui.reducers";
import * as serviceWorker from "./serviceWorker";
import fontawesomeIcons from "./utils/fontawesome.icons";
import App from "./App";

import "./styles/style.scss";
/**
* Supported Fontawesome Icons for Offline usage
*/
library.add(fontawesomeIcons);
/**
* Redux Setup
*/
// Add DevTools Redux Inspector
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
//Combine App Reducers
const rootReduce = combineReducers({
rooms: roomsReducers,
devices: devicesReducers,
ui: uiReducers
});
```

```
// Create the Redux store
const store = createStore(rootReduce,
composeEnhancers(applyMiddleware(thunk)));

/**
* The application JSX code and creation
* Combine Redux and React-Router with the Application
*/

const app = (
<Provider store={store}>
<BrowserRouter>
<App />
</BrowserRouter>
</Provider>
);

ReactDOM.render(app, document.getElementById("root"));

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

serviseWorker.js

```
const isLocalhost = Boolean(
window.location.hostname === 'localhost' ||
// [::1] is the IPv6 localhost address.
window.location.hostname === '[::1]' ||
```

```
// 127.0.0.1/8 is considered localhost for IPv4.
window.location.hostname.match(
/^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/
)
);

export function register(config) {
if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator)
{
// The URL constructor is available in all browsers that support SW.
const publicUrl = new URL(process.env.PUBLIC_URL, window.location.href);
if (publicUrl.origin !== window.location.origin) {
// Our service worker won't work if PUBLIC_URL is on a different origin
// from what our page is served on. This might happen if a CDN is used to
// serve assets; see https://github.com/facebook/create-react-app/issues/2374
return;
}

window.addEventListener('load', () => {
const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;

if (isLocalhost) {
// This is running on localhost. Let's check if a service worker still exists or not.
checkValidServiceWorker(swUrl, config);

// Add some additional logging to localhost, pointing developers to the
// service worker/PWA documentation.
navigator.serviceWorker.ready.then(() => {
console.log(
'This web app is being served cache-first by a service ' +
```

```
'worker. To learn more, visit https://bit.ly/CRA-PWA'
);
});
} else {
// Is not localhost. Just register service worker
registerValidSW(swUrl, config);
}
});
}
}

function registerValidSW(swUrl, config) {
navigator.serviceWorker
.register(swUrl)
.then(registration => {
registration.onupdatefound = () => {
const installingWorker = registration.installing;
if (installingWorker == null) {
return;
}
installingWorker.onstatechange = () => {
if (installingWorker.state === 'installed') {
if (navigator.serviceWorker.controller) {
// At this point, the updated precached content has been fetched,
// but the previous service worker will still serve the older
// content until all client tabs are closed.
console.log(
'New content is available and will be used when all ' +
'tabs for this page are closed. See https://bit.ly/CRA-PWA.'
);
```

```
// Execute callback
if (config && config.onUpdate) {
config.onUpdate(registration);
}
} else {
// At this point, everything has been precached.
// It's the perfect time to display a
// "Content is cached for offline use." message.
console.log('Content is cached for offline use.');

// Execute callback
if (config && config.onSuccess) {
config.onSuccess(registration);
}
}
};
};
})
.catch(error => {
console.error('Error during service worker registration:', error);
});
}

function checkValidServiceWorker(swUrl, config) {
// Check if the service worker can be found. If it can't reload the page.
fetch(swUrl)
.then(response => {
// Ensure service worker exists, and that we really are getting a JS file.
const contentType = response.headers.get('content-type');
```

```
if (
response.status === 404 ||
(contentType != null && contentType.indexOf('javascript') === -1)
) {
// No service worker found. Probably a different app. Reload the page.
navigator.serviceWorker.ready.then(registration => {
registration.unregister().then(() => {
window.location.reload();
});
});
} else {
// Service worker found. Proceed as normal.
registerValidSW(swUrl, config);
}
})
.catch(() => {
console.log(
'No internet connection found. App is running in offline mode.'
);
});
}

export function unregister() {
if ('serviceWorker' in navigator) {
navigator.serviceWorker.ready.then(registration => {
registration.unregister();
});
}
}
```

ModeControl.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

import classes from "./ModeControl.module.scss";

export default class ModeControl extends Component {
static propTypes = {
controlId: PropTypes.string,
name: PropTypes.string,
value: PropTypes.string,
onUpdateValue: PropTypes.func,
options: PropTypes.object
};

onChangeModeHandler = event => {
const updatedValue = event.target.value;
if (updatedValue !== this.props.value) {
this.props.onUpdateValue(this.props.controlId, updatedValue);
}
};

render() {
if (!this.props.options) return null;

const options = Object.entries(this.props.options).map(optionData => {
const optionKey = optionData[0];
const option = optionData[1];
```

```
const isChecked = this.props.value === openKey;
const radioId = `${this.props.controlId}-${openKey}`;
return (
<div className={classes.ModeControl} key={openKey}>
<input
type="radio"
id={radioId}
value={openKey}
className={classes.Radio}
checked={isChecked}
onChange={this.onChangeModeHandler}
/>
<label htmlFor={radioId} className={classes.Label}>
{option.icon ? (
<FontAwesomeIcon icon={{ prefix: "fa", iconName: option.icon }}
className={classes.Icon} />
) : (
<span className={classes.OptionText}>{option.name}</span>
)}
</label>
</div>
);
});
return <div className={classes.ModeControlContainer}>{options}</div>;
}
}
```

ScaleControl.js

```
import React, { Component } from "react";
```

```
import PropTypes from "prop-types";

import classes from "./ScaleControl.module.scss";

export default class ScaleControl extends Component {
static propTypes = {
controlId: PropTypes.string,
name: PropTypes.string,
type: PropTypes.string,
value: PropTypes.number,
min: PropTypes.number,
max: PropTypes.number,
step: PropTypes.number
};
onChangeScaleHandler = event => {
const updatedValue = event.target.value;
if (updatedValue !== this.props.value) {
this.props.onUpdateValue(this.props.controlId, parseInt(updatedValue));
}
};

render() {
return (
<div className={classes.RangeContainer}>
<div className={classes.CurrentValue} data-test="current-value">
{this.props.value}
</div>
<input
className={classes.Range}
```

```
type="range"
min={this.props.min}
max={this.props.max}
step={this.props.step}
value={this.props.value}
onChange={this.onChangeScaleHandler}
/>
</div>
);
}
}
```

TemperatureControl.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faPlus, faMinus } from "@fortawesome/free-solid-svg-icons";
import Button from "../../../UI/Button/Button";

import classes from "./TemperatureControl.module.scss";

export default class TemperatureControl extends Component {
static propTypes = {
controlId: PropTypes.string,
name: PropTypes.string,
unit: PropTypes.string,
value: PropTypes.number,
min: PropTypes.number,
max: PropTypes.number,
```

```
onUpdateValue: PropTypes.func
};

onIncreaseTemperatureHandler = () => {
if (this.props.max === this.props.value) return null;
this.props.onUpdateValue(this.props.controlId, this.props.value + 1);
};

onDecreaseTemperatureHandler = () => {
if (this.props.min === this.props.value) return null;
this.props.onUpdateValue(this.props.controlId, this.props.value - 1);
};

render() {
if (!this.props.value || !this.props.unit) return null;
return (
<div className={classes.TemperatureControl}>
<div className={classes.MinusBtn}>
<Button
onClick={this.onDecreaseTemperatureHandler}
data-test="decrease-temperature-btn"
>
<FontAwesomeIcon icon={faMinus}/>
</Button>
</div>
<div className={classes.Temperature} data-test="temperature">
<div>
{this.props.value} °{this.props.unit.toUpperCase()}
</div>
```

```
</div>
<div className={classes.PlusBtn}>
<Button
onClick={this.onIncreaseTemperatureHandler}
data-test="increase-temperature-btn"
>
<FontAwesomeIcon icon={faPlus} />
</Button>
</div>
</div>
);
}
}
```

ControlsSwitcher.js

```
import React from "react";
import PropTypes from "prop-types";
import { TEMPERATURE, SCALE } from "../../../utils/deviceControls.types";
import TemperatureControl from "./Controls/Temperature/TemperatureControl";
import { MODE } from "./../../../utils/deviceControls.types";
import ModeControl from "../Controls/Mode/ModeControl";
import ScaleControl from "../Controls/Scale/ScaleControl";
import classes from "./ControlsSwitcher.module.scss";

export const controlsSwitcher = device => {};

function ControlsSwitcher(props) {
```

```
if (!props.deviceData) return null;

const controlProps = {
controlId: props.controlId,
onUpdateValue: props.onUpdateValue,
...props.deviceData
};

let control;

switch (props.deviceData.type) {
case TEMPERATURE:
control = <TemperatureControl {...controlProps} />;
break;
case MODE:
control = <ModeControl {...controlProps} />;
break;
case SCALE:
control = <ScaleControl {...controlProps} />;
break;
default:
control = null;
}

if (!control) return null;

return (
<div className={classes.TemperatureControlContainer}>
<div className={classes.Title} data-test="device-title">
{props.deviceData.name}
```

```
</div>
{control}
</div>
);
}

ControlsSwitcher.propTypes = {
controlId: PropTypes.string,
deviceData: PropTypes.object
};

export default ControlsSwitcher;
```

Device.js

```
import React, { Component } from "react";
import PropTypes from "prop-types";
import ControlsSwitcher from "./ControlsSwitcher/ControlsSwitcher";

import classes from "./Device.module.scss";
import Switch from "./../UI/Switch/Switch";

export default class Device extends Component {
static propTypes = {
deviceId: PropTypes.string,
device: PropTypes.object,
onToggleDeviceSwitch: PropTypes.func,
onControlValueChanged: PropTypes.func
};
```

```
/**
* Event fired when the value of a control is changed
*/
onControlValueChangedHandler = (controlId, newValue) => {
this.props.onControlValueChanged(this.props.deviceId, controlId, newValue);
};
render() {
if (!this.props.device) return;
// Checks it has controls before rendering them
let deviceControls;
if (
this.props.device.controls &&
!!Object.values(this.props.device.controls).length
) {
deviceControls = Object.entries(this.props.device.controls).map(
device => {
const controlId = device[0];
const deviceData = device[1];
return (
<div key={controlId} className={classes.DeviceContainer}>
<ControlsSwitcher
controlId={controlId}
deviceData={deviceData}
onUpdateValue={this.onControlValueChangedHandler}
/>
</div>
);
}
```

```
);
}
return (
<div className={classes.Device}>
<div className={classes.Header}>
<div>{this.props.device.icon}</div>
<div className={classes.Title}>{this.props.device.name}</div>
<div className={classes.Switch}>
<Switch
onChange={this.props.onToggleDeviceSwitch}
checked={this.props.device.switch}
/>
</div>
</div>
<div>{deviceControls}</div>
</div>
);
}
}
```