Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Управління системою “Розумний будинок” з використанням хмарних обчислень

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Гайдук В.І.
(підпис) (прізвище та ініціали)

Керівник _____ Мацюк О.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Микитишин А.Г.
(підпис) (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Гайдуку Владиславу Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Управління системою "Розумний будинок" з використанням хмарних обчислень

Керівник роботи Мацюк Олександр Васильович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 16 » березня 2022 року № 4/7-161

2. Термін подання студентом завершеної роботи 20.06.22р.

3. Вихідні дані до роботи Інтернет джерела, технічна документація

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. 1.1 Поняття концепції "Розумного будинку". 1.2 Аналіз можливих способів побудови "Розумного будинку". 1.3 Поняття хмарної інфраструктури. Види хмар. 1.4 Різновиди моделей хмарних послуг: Saas, Paas, Iaas. 1.5 Існуючи способи підключення інтерфейсу "Розумного будинку" до хмарної інфраструктури. 1.6 Висновки до першого розділу. 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ. 2.1 Опис архітектури системи управління "Розумний будинок". 2.2 Створення програмного забезпечення для веб-сервера. 2.3 Створення програмного забезпечення для головного контролера. Опис взаємодії приладів. 2.4 Створення панелі керування. 2.5 Висновки до другого розділу. 3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ. 3.1 Психологічні чинники небезпеки. 3.2 Оцінка технологічного процесу щодо умов пожежонебезпеки при роботі з електронно-обчислювальною технікою.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Акутальність. Мета і завдання. Концепція "Розумного будинку". Можливі способи реалізації системи. Хмарна інфраструктура та хмарні послуги. Інтеграція "Розумного будинку" з хмарною інфраструктурою. Опис обраної архітектури "Розумного будинку". Реалізація ПЗ хмарного вебсервера. Розробка ПЗ головного контролера. Розробка панелі керування. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., доцент		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про розробку систем	04.01.2022-30.01.2022	*Виконано*
	управління “Розумний будинок” з використанням		
	хмарних обчислень		
3.	Переклад та опрацювання джерел про розробку	31.01.2022-06.02.2022	*Виконано*
	системи управління “Розумний будинок” з		
	використанням хмарних обчислень		
4.	Розробка системи управління “Розумний будинок”	07.02.2022-13.02.2022	*Виконано*
	з використанням хмарних обчислень		
5.	Оформлення розділу «Аналіз предметної області»	14.02.2022-06.03.2022	*Виконано*
6.	Оформлення розділу «Практична реалізація»	07.03.2022-03.04.2022	*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	06.06.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	07.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	20.06.2022	

Студент _____ Гайдук В.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мацюк О.В,
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Управління системою "Розумний будинок" з використанням хмарних обчислень // Кваліфікаційна робота освітнього рівня «Бакалавр» // Гайдук Владислав Іванович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2022 // С.56, рис. – 21 , табл. – 0, кресл. – 0, додат. – 7, бібліогр. – 33.

Ключові слова: Розумний будинок, хмарні обчислення, IoT, SaaS, PaaS, IaaS, API, NodeJS.

Кваліфікаційна робота присвячена дослідженню існуючих рішень та способів розгортання системи "Розумний будинок" та розробці власної системи управління будинком з використанням хмарних технологій.

Мета роботи: надати користувачеві доступ до системи управління житловою інфраструктурою його будинку чи квартири, що повинно забезпечити комфортніше проживання, підвищити рівень безпеки його житла та оптимізувати використання ресурсів.

В першому розділі кваліфікаційної роботи проаналізовано можливі способи побудови системи "Розумний будинок" з використанням хмарних технологій. Також було розглянуто концепцію хмарної інфраструктури, види хмар та різновиди моделей існуючих хмарних послуг.

В другому розділі кваліфікаційної роботи було описано архітектуру системи "Розмного дому", обрано технології для розробки програмного забезпечення веб-сервера, головного контролера та панелей керування і адміністрування. Також було описано взаємодію між приладами системи.

ANNOTATION

Smart home system management using cloud computing // Qualification work of the educational level «Bachelor» // Haiduk Vladyslav Ivanovych // Ternopil National Technical University named after Ivan Pulyuy, Faculty of Computer Information Systems and Software Engineering // Ternopil, 2022 // / Explanatory note size – 56 pages, contains 21 illustrations, 0 tables, 7 appendix, 33 bibliography items.

Key words: Smart house, cloud computing, IoT, SaaS, PaaS, IaaS, API, NodeJS, Nest.js.

Qualification work is devoted to investigating existing solutions and methods of the deploying a "Smart house" system and developing of own smart home system using cloud technologies.

Purpose: provide users the opportunity to manage the residential infrastructure of their house or apartment to ensure a comfortable life, improve home safety and optimize resource usage.

The first section of the qualification work contains an analysis of possible methods of developing a Smart house system using cloud technologies. Concepts of the cloud infrastructure, cloud types, and types of cloud computing services were also considered.

The second section of the qualification work contains a description of the Smart house system architecture, and a description of selected technologies to develop programs for web-server, main controller, control, and admin panels. Interaction between devices was also considered.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT (Internet of Things) – Інтернет Речей.

SaaS (Software as a service) – програмне забезпечення як послуга.

PaaS (Platform as a Service) – платформа як послуга.

IaaS (Infrastructure as Service) – інфраструктура як послуга.

GPIO (General-purpose input/output) – інтерфейс вводу/виводу загального призначення

API (Application Programming Interface) – інтерфейс прикладного програмування.

AWS (Amazon Web Services) – найпопулярніша платформа хмарних сервісів, що підтримує більше чим 200 різноманітних сервісів.

HTTP (HyperText Transfer Protocol) – протокол передачі даних, що використовується в роботі комп’ютерних мереж.

TCP (Transmission Control Protocol) – один із основних протоколів управління передачі даних в Інтернеті.

JSON (JavaScript Object Notation) – текстовий формат передачі даних, що заснований на базі JavaScript.

TS (TypeScript) – мова програмування від Microsoft, яка призначена для розширення можливостей JavaScript.

ЗМІСТ

ВСТУП

Актуальність теми. На сьогоднішій день актуальність технологій є дуже часто обговорюваною темою серед людей. Ми живемо, вчимося та працюємо, використовуючи технологічні прилади, які надають нам можливість спілкуватись один з одним, фотографувати природу або красиву інфраструктуру мегаполісів і публікувати це для інших, постити цікаві статті, слідуквати за друзями в соціальних мережах тощо. Майбутні покоління людей вже не будуть настільки актуалізовувати розвиток технологій, оскільки життя в високотехнологічному просторі уже буде здаватись звичним явищем. Люди навчились використовувати технології не тільки в науці, а й в повсякденному житті. Купуючи квиток в громадському транспорті або бронюючи номер в готелі, ви берете участь в обміні інформацією між електронними приладами та людьми. Це допомагає отримувати важливі дані чи повідомлення набагто швидше, чим передавати їх особисто, наприклад, через написаний лист. Бажання людей почувати себе безпечніше та комфортніше – є вагомим важілем впливу на рух технологічного розвитку.

Для покращення свого комфорту люди завжди будуть розвивати технології, які ставатимуть актуальними в той чи інший проміжок часу. Наприклад, не так давно користувачі використовували електронні листування для спілкування з друзями чи колегами, згодом настала ера месенджерів, які заполонили простори всесвітньої мережі. Люди полюбили месенджері через їх доступність та простоту в користуванні. Також люди прагнуть зручності і при користуванні житловою інфраструктурою своїх домівок. Ми всі знаємо про існування електричних духовок з таймерами і нагадуваннями про готовність страви, про електронні жалюзі, які можна активувати через кнопку на пульті керування, про клондиціонери з доступною функцією підтримки сталої теператури в приміщенні. Це далеко не повний список найпопулярніших “розумних” побутових приладів. Також важливим аспектом цієї теми є економія ресурсів. Дуже часто ви забуваєте вимкнути світло, коли покидаєте

кімнату, це призводить до надмірного використання електроенергії або інших ресурсів, це може бути актуальним для країн або регіонів з високими цінами на електроенергію. Також варто задати питання щодо безпеки. Більшість квартирних пожеж виникають через несправність електричних кабелів або витік газу. Такі небезпеки можна передбачити, якщо використовувати прилади індентифікації витоку газу чи детектори диму. Тому багато домовласників прагне мати в користуванні унікальну систему управління житловою інфраструктурою, що буде поєднувати всі вище описані аспекти та вирішувати загальні проблеми більшості домогосподарств.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є:

– Проаналізувати та дослідити існуючі рішення та методи розгортання системи управління “Розумний будинок”.

– Дослідити актуальність хмарних технологій в поєднанні з концепцією “Розумного будинку”.

– Визначити способи підключення хмарної інфраструктури до інтерфейсу системи “Розумний будинок”.

– Запропонувати власне рішення на базі хмарої технології, що буде включати в себе програмне забезпечення для системи “Розумний будинок”.

– Побудувати архітектуру системи управління “Розумний будинок”.

– Розробити програмне забезпечення для основних частин системи.

– Протестувати програмне забезпечення на функціональному рівні.

– Висвітлити питання щодо безпеки та охорони праці.

Практичне значення одержаних результатів. Для людей, що проживають в своїх домівках чи квартирах буде наданий доступ до системи управління житловою інфраструктурою. Це дозволить отримувати необхідну інформацію про температуру, освітленість тощо, керувати функціоналом певних приладів віддалено. Користувач зможе отримати певний рівень комфорту, керувати витратами ресурсів його домівки та отримувати певний рівень безпеки для своєї домівки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття концепції “Розумний будинок”

Системи, що дозволяють здійснювати контроль та керування житловою інфраструктурою дому, квартири, чи навіть, цілого міста або його районів останім часом часом отримали дуже велику популярність. Системи схожого типу функціонують в багатьох містах західних країн Європи та надають можливість зручно організовувати роботу підземного та наземного транспорту і користуватись адміністративними послугами. Поняття “Розумний дім” зєявилось ще в 70-х роках минулого століття, проте дана система не мала чітких реалізацій в той час [1]. Автоматизована система управління будинком означає наявність налаштованих автоматизованих інженерних систем управління та збору даних, що дають користувачеві певний рівень комфорту, слідкують за безпековою ситуацією в житлі. Такі системи зазвичай керуються з єдиного центру або інтерфейсу. Першу реалізацію дана система отримала для оптимізації використання ресурсів. Робота такої системи включала в себе контроль за невикористовуваними пристроями, які користувач забув вимкнути. Також збирались та опрацьовувались дані щодо спожиавння електроенергії в будинку чи квартирі.

За допомогою “Розмного дому” можна полегшити планування деяких домашніх справ, наприклад вам не потрібно більше пилососити кожної суботи, оскільки ви можете просто запланувати цю роботу в системі і робот пилосос зробить це за вас. Також вам не потрібно більше думати про пожежну небезпеку чи небезбеку витоку газу, оскільки система “Розумний будинок” самомтійно за допомогою датчиків газу та диму здатна аварійно перекрити подачу газу в квартиру чи повідомити вас про наявність диму в приміщені. Також вам не потрібно більше турбуватись про ризики щодо хатніх крадіжок, оскільки система контролює всі замки та вікна в вашому домі чи квартирі, а

датчики руху передадуть інформацію про підозрілу ситуацію на пульт охорони та ввімкнуть сигнальну триву.

За допомогою такої розумної системи можна програмувати будь-який сценарій поведінки всіх підключених приладів. Ви можете запланувати ваше пробудженння, коли система автоматично відкриватиме жалюзі на вікнах та включатиме спокійну мелодію в кімнаті. Система “Розумний будинок” здатна створити будь-яку атмосферу у вашій домівці. Отже, система “Розумний будинок” здатна вирішувати три важливі проблеми домогосподарства – це: комфорт, економія ресурсів та безпека. В основу концепції такої системи полягає автоматизоване керування всіма приладами житлової інфраструктури та відпрацювання різноманітних сценаріїв поведінки.

Звісно, на даний момент не існує єдиної уніфікованої системи “Розумний будинок”, яка б повністю тримала б управління житловою інфраструктурою. Перш за все, варто зазначити, що завжди будуть існувати різноманітні чинники природного і техногенного характеру, які будуть впливати на систему і на домівку в цілому. На даний момент питання безпеки є найбільш спірним в контексті розмов про “Розумний будинок”. Інколи сама система стає причиною пожеж або помилкового блокування всіх дверей будинку. Неможливо передбачити всі сценарії поведінки. Також важливою проблемою такої системи є неповна сумісність та інтеграція з іншими приладами. Для того, щоб “Розумний будинок” зміг тримати під контролем всю житлову інфраструктуру, або хоча б більшу її частину, нам потрібні прилади з підходящим інтерфейсом для можливого підключення до них розумної системи. Більшість приладів такого інтерфейсу не мають. Також ми могли б виготовляти самостійно побутові прилади з таким інтерфейсом, або хоча б модернізовувати вже існуючі. В такому випадку собівартість системи значно виросте. Тому на даний момент практична реалізація системи “Розумний будинок” орієнтована повністю під користувача або замовника [2]. Також варто зазначити, що в замоників можуть бути різні побажання, що також унеможливлює процес уніфікації інтерфейсу “Розумного будинку”.

Питання вартості та обслуговування є також важливим. При наявності великої кількості різних датчиків, електронних приладів і механізмів замовник повинен розуміти про необхідність своєчасного технічного огляду системи. Якщо ми хочемо покрити всю площу дому чи квартири системою “Розумний будинок”, ми також повинні розуміти про високу вартість такої роботи. Також існуватиме проблема інтеграції такої системи з різними екстреними і міськими службами.

Як вже було зазначено вище, основна концепція “Розумного будинку” повинна покривати комфорт, контроль за ресурсами та безпеку домівки. Практична реалізація буде орінтуватись тільки на побажання користувача або замовника, оскільки не існує єдиного стандарту чи правил для проектування та розробки таких систем.

1.2 Аналіз можливих способів побудови “Розумного будинку”

Спершу небхідно визначати, які критерії вибору ставлять перед собою замовники, коли розглядають варіант обладнання їх домівок розумними системами. Перш за все це – *вартість* робіт та обладнення. На другому місці щодо критерій вибору буде стояти *безпека*. Справді, белика кількість замовників прагне отримувати високий рівень безпеки їх домівок, що на даний момент забезпечити доволі складно. Тому, враховуючи ці два критерії, ми можем виділити два способи розгортання та побудови: *професійний* та *бюджетний* [3].

Професіний варіант побудови “Розумного будинку” є прерогативою спецальних компаній, що спеціалізуються на наданні послуг по розробці розумних систем. В світі не існує великої кількості таких компаній. Інакше кажучи, існує багато підприємств, що надають схожі послуги, але вони не є висококваліфікованими в цій сфері і їхні послуги можна віднести більше до бюджетних варіантів розрогртання, чим до професійних. Послуги професійних компаній зазвичай потребують великої вартості. Такі компанії купують тільки

найкраще і найдорожче обладнення від офіційних брендів чи представників. Також такі послуги від професійних підприємств передбачають довгострокову гарантію та технічне обслуговування.

Висококваліфіковані компанії в більшості випадків працюють на контрактній основі з своїми замовниками. Замовник може бути впевнений в гарантованій безпеці. Домовласнику надається та налаштовується додаткове обладнення, яке буде працювати в екстрених випадках, наприклад, в разі відключення Інтернету чи електроенергії. Професійні способи реалізації “Розумного будинку” передбачають проводове підключення, оскільки система, що з’єднана кабелями матиме більшу швидкість передачі даних і буде більш захищеною від зовнішніх впливів, чим наприклад, система з зв’язком через радіохвилі. Всі інженери повинні мати відповідну сертифікацю для виконання такого роду робіт. При професійному способі реалізації системи управління “Розумний будинок” користувач отримає доступ до спеціального інтерфейсу керування, що має забезпечити зручність в керованості та швидку доступність інформації про стан будинку. Система матиме змогу в будь-який момент в режимі реального часу надсилати сповіщення про стан домівки. В випадку екстерної ситуації сповіщення повинно приходити на телефони родичів домогосподарства, чи друзів, а також система має повідомити екстрені служби.

Бюджетний варіант розгортання системи “Розумний бдинок” може виконуватись будь-ким. В більшості випадків інформація береться з підрчуних матеріалів або статтей з Інтернету. Також замовник матиме можливість значно заощадити кошти при виборі датчиків та інших електонних приладів. В більшості випадків вибір буде падати на постачальників мінікомп’ютерів або одноплатних пристроїв та контролерів, таких як Arduino (Див. рисунок 1.1) або Raspberry Pi (Див. рисунок 1.2). На ринку мінікомп’ютерів та контролерів існує безліч інших не дорогих аналогів, які можуть надавати схожий функціонал до Arduino чи Raspberry Pi. Важливим критерієм при виборі є можливість підключати і підтримувати інтерфейси інших приладів через різні типи зв’язку.

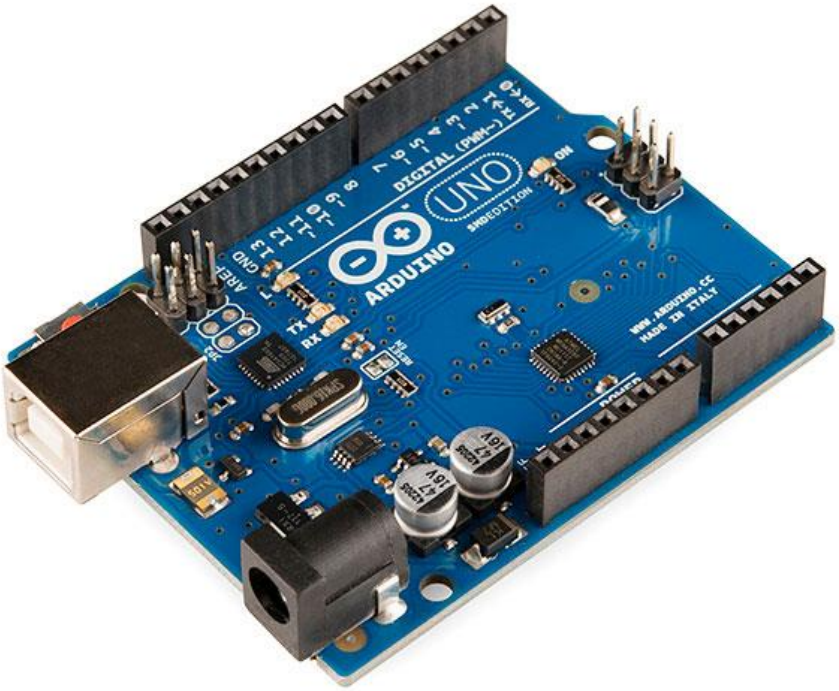

Рисунок 1.1 – Одноплатний комп’ютер Arduino

Рисунок 1.2 – Одноплатний комп’ютер Raspberry Pi

Дешевизна датчиків та інших приладів може значно посилити безпекову проблему системи. Такі прилади здатні швидко виходити з ладу, та будь-яка система, що побудована на базі таких приладів, не може гаратнтувати довгострокову роботу в подальшому. У випадках бюджетної реалізації “Розумного будинку” зазвичай використовують датчики та прилади з

бездротовим інтерфейсом, які не є дорогими, проте значно збільшують ризики щодо втручанння в роботу системи ззовні та не гарантують чіткий бездротовий зв’язок. Бюджетна система не зможе безперебійно функціонувати в випадку зникнення подачі електроенергії чи підключення до мережі. Також варто зазначити що бюджетність системи не буде впливати на зручність використання ПЗ, швидкий доступ та керованість програми можна також реалізувати і в бюджетних умовах.

На просторах Інтернету можна знайти безліч безкоштовних додатків, які надають можливість отримувати сповіщення на мобільні телефони чи планшети через Інтернет або мережу GSM про стан будинку або квартири, чи навіть про надзвичайну ситуацію [4]. Такі додатки не гарантуватимуть належний рівень безпеки при підключені їх до системи “Розумний дім”. Останнє, що варто зазначити – це проблема планування розумної системи при використанні бюджетного способу. Замовник часто повинен сам планувати розміщення різноманітних приладів в себе в квартирі чи будинку і не завжди отриманий результат буде забезпечувати належний рівень комфорту.

1.3 Поняття хмарної інфраструктури. Види хмар

Поняття хмарних технологій з’явилось не так давно, проте зараз воно набуває великої популярності. Особливість хмарних технологій полягає в тому, щоб надати користувачеві комп’ютерні ресурси і потужності як послугу через мережу Інтернет. Користувачеві не потрібно займатись налаштуванням серверних операційних систем, програм та серверної інфраструктури вручну. Все ПЗ знаходить на віддаленому сервері і подається користувачеві як послуга, за яку він платить. Хмарна інфраструктура надає доступ до ресурсів з будь-якого місця планети через Інтернет та має певні рівні захисту. Всього визначено три типи хмарних інфраструктур: *public, private, hybrid* [5].

- *Public* (Публічна хмара) – послуги такої хмари споживаються великою аудиторією людей в режимі спільного використання через мережу

Інтернет. Адмініструванням і управлінням інфраструктурою хмари займається провайдер послуги. Користувач хмари не має доступу до управління чи налаштування інфраструктури. Публічну хмару варто використовувати в тих випадках, коли важлива швидкість розгортання або впроваження, від користувачів публічної хмари не вимагається високо рівня знань і кваліфікації.

- *Private* (Приватна хмара) – послугами такої хмари користується тільки один користувач або абонент і він має право використувати таку послугу у власних цілях без додаткових обмежень (в рамках законодавста). Інфраструктура такої хмари може налаштовуватись і бути розміщеною як і на віддаленому сервері (на сервері провайдера), так і на сервері користувача, або частково в користувача і провайдера. Саме приватна хмара є найбільш підходящим типом хмарної інфраструктури для розробки системи управління “Розумним будинком”, оскільки в такому випадку можна обмежеити доступ до системи для певного кола осіб і це повинно мінімізувати безпекові ризики.
- *Hybrid* (Гібридна хмара) – такий тип інфраструктури поєднує в собі якості приватних і публічних хмар. На практиці це пояднання реалізовується шляхом розміщення одної частини системи в публічній хмарі (наприклад, в дата-центрах), а іншої в приватній хмарі. Також для гібридних інфраструктур є притаманним – надавати можливість користувачеві самому обирати сценарій розгортання його системи, комбінувати елементи публічної та приватної хмари в таких співвідношеннях, як це може бути вигідно для замовника. Гібридна хмара не встановлює ліміти для показників навантаженості системи, це дозволяє значно підвищити адаптивність ПЗ. Користувач самостійно вирішує яку частину його системи варто залишати в публічні, а яку – в приватній.

1.4 Різновиди моделей хмарних послуг: SaaS, PaaS, IaaS

Серед великого спектру різноманіття хмарних послуг базовими і найпопулярнішими залишаються *SaaS*, *PaaS* та *IaaS* (Див. рисунок 1.1) [6].

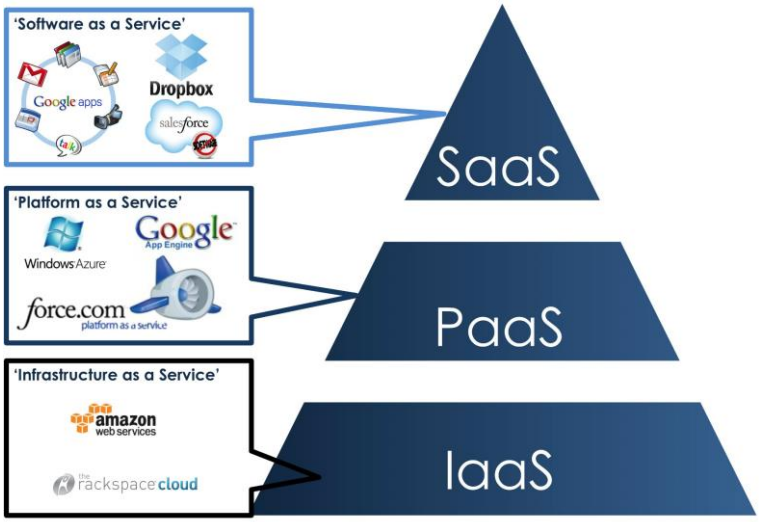

Рисунок 1.3 – Види хмарних моделей послуг і приклади їх застосування

SaaS (Інфраструктура як послуга) модель надає користувачеві можливість використовувати прикладне ПЗ, що надається провайдером. Контроль, управління і адміністрування є прерогативами провайдера хмари, користувач доступу до цих функцій не має. Кінцевий користувач має доступ до програмного забезпечення, зазвичай, через веб браузер або веб додатки. Даний різновид хмарних послуг має суттєву перевагу в тому, що користувачам не потрібно займатись налаштуванням, встановленням, оновленням та підтримкою ПЗ. Прикладами моделі SaaS можуть бути: SalesForce, Dropbox, Cisco, Zendesk, Hubspot, Google Apps та інші.

Щодо використання SaaS моделі для розгортання ПЗ “Розумного будинку” можна зазначити, що в такому випадку система управління будинком буде розгорнута на базовому рівні з обмеженими функціями. SaaS модель не дозволяє реалізувати масштабування системи, тому ПЗ “Розумного будинку”

буде обмежуватись певним визначеним функціоналом, ммможливо з декількома веб-сторінками або елементами веб інтерфейсу.

PaaS (Платформа як полуга) модель надає хмарний функціонал для розгортання власного ПЗ, що було розроблено визначеними провайдером мовами програмування, фреймворками та технологіями. На сьогоднішній день це найпопулярніша модель хмарних послуг на ринку, якою користуються у 35% всіх робочих процесів різних компаній. PaaS модель передбачає надання користувачеві фреймворку або інших інструментів для розробки власного кастомного ПЗ. Хмарний провайдер повністю контролює сервери, системи зберігання даних, бази даних, мережі, операційні системи та іншу хмарну інфраструктуру, натомість користувач може контролювати, розробляти і оновлювати своє ПЗ на базі PaaS хмарної моделі. Також користувач має доступ до налаштування параметрів конфігурацій, які потрібні для розгортання його ПЗ. Як можна побачити, для використання PaaS хмарної моделі користувачам необхідно володіти певними навичками програмування, зазвичай знати і володіти мовами програмування для серверної веб-розробки та налаштування серверної конфігурацій веб застосунків. Така хмарна модель є більш підходящою для реалізації та розгортання системи управління “Розумний будинок”, оскільки розробнику надається певна незалежність та самостійність при написанні ПЗ. Це дозволить створити потрібну логіку веб-сервера, який зможе опрацьовувати запити від контролерів, що будуть працювати через мережу Інтернет. Найпопулярні приклади PaaS хмарної інфраструктури: SAP, Heroku, деякі сервіси AWS, Google App Engine, Force.com, Windows Azure.

IaaS (Інфраструктура як послуга) хмарна модель надає користувачам повний доступ до налаштованих віртуальних серверів з обраною користувачем обчислювальною потужністю. Звісно, це дозволяє заощадити на вкладеннях в фізичну серверну інфраструктуру, проте користувач буде змушений в будь-якому випадку оплачувати обчислювальні потужності, в залежності від вибраних ним параметрів та характеристик системи. Користувач має повну самостійність при виборі технологій розробки чи інструментів розгортання. Із

наданою самостійністю з’являється інша проблема – це проблема контролю за серверною інфраструктурою та програмами. Користувач змушений контролювати, скільки обчислювальних ресурсів він витрачає, також він змушений налаштовувати і підтримувати середовище для розгортання ПЗ самостійно. Даний вид хмарних послуг підходить тим, кому не вистачає функціональності і додаткових можливостей SaaS і PaaS хмарних сервісів. Користувач повинен володіти дещо більшими навичками, чим просто розробка ПЗ. Він повинен добре знати операційні системи, зазвичай це Linux, вміти налаштовувати конфігурації веб-серверів під різні потреби, мати практичні навички в адмініструванні мереж, знати і вміти працювати з ПЗ для автоматизації розгортання і управління застосунками, наприклад, Doker. Люди обирають IaaS, коли прагнуть розробляти комерційні масштабовані високо навантажені веб-застосунки. Провайдер в свою чергу використовує найновітніше апаратне забезпечення на своїх серверах, що дозволяє практично не відчувати брак обсислювальних потужностей чи брак пам’яті. Даний вид хмарних послуг найкраще підходить для розробки і розгортанння ПЗ для системи управління “Розумним будинком”, оскільки практично не обмежує розробника в виборі технологій чи інструментів розробки. Найпопулярніші приклади IaaS хмарних послуг: DigitalOcean, Linode, AWS, Microsoft Azure, RackSpace.

1.5 Існуючі способи підключення інтерфейсу “Розумний будинок” до хмарної інфраструктури

Перш ніж визначати способи підключення розумної системи до хмарної інфраструктури, потрібно проаналізувати, які типи архітектури системи управління “Розуумний будинок” ми можемо розглянути, оскільки ми повинні опиратись на можливості і потреби розумної системи, чим навпаки – на хмарну інфраструктуру. Розгляд можливої архітектури повинен проходити з урахуванням критерію щодо працездатності системи в автономному режимі без

підключення до мережі Інтернет або до електроенергії. Варто зазначити, що такі критерії ставлять серйозні перешкоди для реалізації системи управління “Розумний бдинок” і спонукають розробника використовувати бюджетний спосіб при проектуванні архітектури.

Отже можна виділити два основних способи реалізації архітектури системи управління “Розумний будинок”:

1. *Реалізація системи з використанням головного контролера та другорядних пристроїв.* Другорядні пристрої надсилатимуть сигнали та повідомлення головному контролеру через будь-який тип зв’язку, контролер матиме зв’язок до Інтернету та надсилатиме запити до веб-сервера в хмарі [7]. На даний момент – це найбільш підходящий спосіб для реалізації розумних систем, оскільки завжди буде можливість додавати нові прилади та підключати їх до головного контролера. Контролер повинен мати мережевий шлюз для виходу в Інтернет та базову операційну систему, зазвичай Linux, що дозволить розробнику підключати нові інтерфейси різних приладів та приймати від них сигнали. Також варто додати, що такий спосіб має низьку собівартість, оскільки одноплатні комп’ютери не є дорогими, а вони якраз ідеально підходять для проектування в ролі головного контролера.
2. *Реалізація системи, де кожен пристрій має вихід в хмару.* Реалізація архітектури, де кожен прилад, датчик матимуть мережевий шлюз є доволі складною, оскільки на даний момент існує не велике різноманіття таких приладів у вільному доступі. Купівля кожного такого пристрою з мережевим шлюзом і базовою операційною системою значно збільшить собівартість розробки. Такий спосіб реалізації має також значні переваги, оскільки дозволяє пренести функціонал головного контролера в хмарний веб-сервер, це надасть нам можливості для подальшого масштабування та розширення “Розумного будинку”.

Також варто додати, що при плануванні щодо розгортання таких систем, ми можем використовувати будь-які пристрої, від будь-якого виробника. В нас немає архітектурних обмежень, щодо брендів чи виробників, лише потрібно пам’ятати, що пристрій повинен мати інтерфейс для підключення його до головного контролера, або базову операційну систему для виходу в хмару. Для обох вище описаних архітектурних способів ми можем використовувати як і саморобні пристрої в поєднанні з головним контролером, так і новітні IoT прилади, наприклад від Google або Amazon [8]. Все залежить від визначеного бюджету на розробку системи. У випадку використання нових приладів, ми можем гарантувати замовнику стабільнішу роботу, практично безперебійний зв’язок і кращу безпеку.

1.6 Висновки до першого розділу

Актуальність теми “розумних” технологій буде збільшуватись із кожним роком. Такі технології знаходять популярність в багатьох сферах нашого життя, включаючи побут. Бажання людини отримати додатковий рівень комфорту, підвищити рівень безпеки та встановити контроль використання ресурсів є основними причнами для продовження розвитку такого роду технологій. Система “Розумний будинок” є доволі актуальною на сьогоднішний день, оскільки вона здатна вирішити проблеми безпеки, зробити життя комфортнішим, допомогти з економією ресурсів. На даний час не існує єдиної уніфікованої системи “Розумний будинок”, яка здатна б контролювати всі прилади і елементи житлової інфраструктури. Процес розробки і розгортання таких систем відбувається виключно під потреби замовників.

Актуальність поєднання системи управління “Розумним будинком” з хмарними технологіями є також важливою. Хмарні технології дозволяють використовувати обчислювальні потужності та ресурси як послугу віддалено, тому мною було визначено основні типи можливих способів реалізації системи “Розумний будинок” з використанням хмарних веб-серверів.

РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ

2.1 Опис архітектури системи управління “Розумний будинок”

Перш ніж приступати до реалізації системи, необхідно описати її загальну архітектуру. Варто визначити, які технології, фреймворки та інтерфейси будуть використовуватись. Загальний опис допоможе нам сформувати певний план, згідно з яким ми зможемо поетапно здійснювати розробку різних функціональностей системи.

Як вже зазначалось раніше, наша система управління будинком повинна мати мережевий шлюз для виходу в інтернет. Ми можемо обирати пристрої, кожен з яких буде мати мережевий шлюз для виходу в хмару, або ж один головний контролер з мережевим шлюзом для виходу в Інтернет, що матиме зв’язок з іншими пристроями без мережевого шлюзу. Наша система буде функціонувати саме так, як це описано в другому варіанті – через головний контролер з мережевим шлюзом. В якості головного контролера було обрано одноплатний мінікомп’ютер Raspberry Pi, на який встановлена базова операція система Raspberry Pi OS. Дана операційна система була розроблена на базі Debian Linux, отже це дозволить використовувати нам стандартні пакетні менеджери і завантажувати більшість Linux сумісних пакетів для нашого проекту [9]. Також варто додати, що Raspberry Pi плата підтримує GPIO інтерфейс, що дозволить нам підключати до контролера різноманітні переферійні прилади, та обмінюватись з ними сигналами. GPIO – це інтерфейс універсального вводу та виводу, даний інтерфейс представлений контактами, що знаходяться на боковій стороні Raspberry Pi (Див. рисунок 2.1) [10]. Така можливість Raspberry Pi є ключовою для нашого проекту, оскільки це допоможе нам опрацьовувати сигнали із зовнішньої периферії. Кожен контакт GPIO здатний функціонувати як перемикач, який можна вмикати та вимикати. Базові плати Raspberry Pi місять 26 контактів і 9 з них – це контакти живлення

та заземлення. Контакти заземлення знаходяться в кінці кожного кола, через який повинен проходити електричний струм [11].

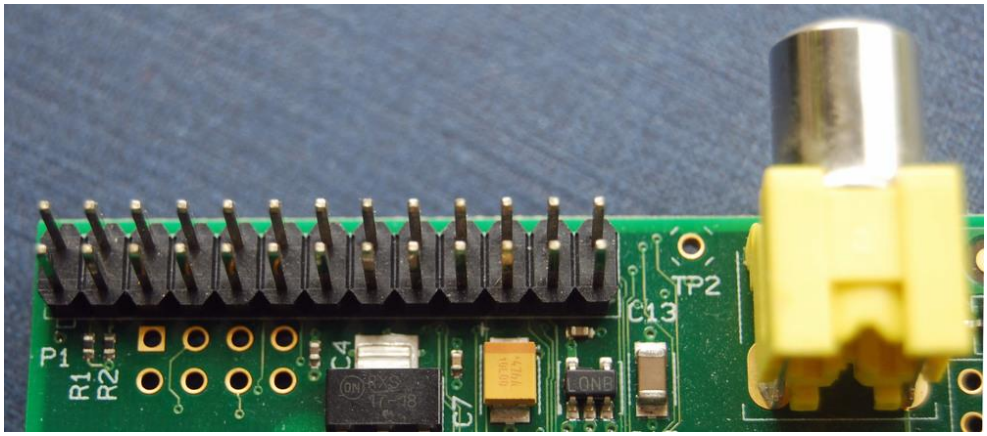

Рисунок 2.1 – Інтерфейс GPIO на боковій стороні плати, поряд з гніздом відеовиходу

Після вибору апаратного забезпечення, та опису його функціональностей, які знадобляться нам для реалізації комунікації пристроїв, необхідно визначаити і програмний стек техннологій. Raspberry Pi (головний контролер) – це умовний передавач інформації на веб-сервер та з веб-серверу на переферію. Отже, нам необхідно не тільки реалізувати реалізувати програмний функціонал прийняття та відправки інформації між зовнішньою локальною перефирією, а й реалізувати функціонал обміну данними між хмарним веб-сервером. Для цього нам може знадобитись налаштування локально сервера на Raspberry Pi, який буде в режимі реального часу приймати повідомлення від переферії та передавати їх на хмарний веб-сервер, а також пересилати сигнали до переферії. Для цього нам знадобиться стек технологій: *Node.js, http* модуль для надсилання запитів на хмарний веб-сервер, базова бібліотека *onoff*, яка надаватиме змогу читати та записувати повідомлення або сигнали для зовнішніх пристроїв, бібліотека Socket.IO, яка ідеально підходить для комунікації різноманітних IoT пристроїв з веб-сервером [12]. Дана бібліотека реалізовує протокол WebSocket, який є надбудовою над простим TCP зв’язком. WebSocket дозволяє створити двонапрямлену комунікацію між клієнтом та

сервером в реальному часі. Натомість HTTP протокол є однонапрямленим, тобто клієнт надсилає запит і очікує на відповідь, потім з'єднання з сервером розривається. Для реалізації проекту на простому HTTP нам було б необхідно кожен раз надсилати запит на хмарний веб-сервер і "запитувати його", чи не має він інформації для сторони клієнта (Див. рисунок 2.2) [13]. Оскільки Raspberry Pi буде слугувати нам як головний контролер, нам необхідно налаштовувати його як клієнт, який буде відкривати WebSocket з'єднання з хмарним веб-сервером.

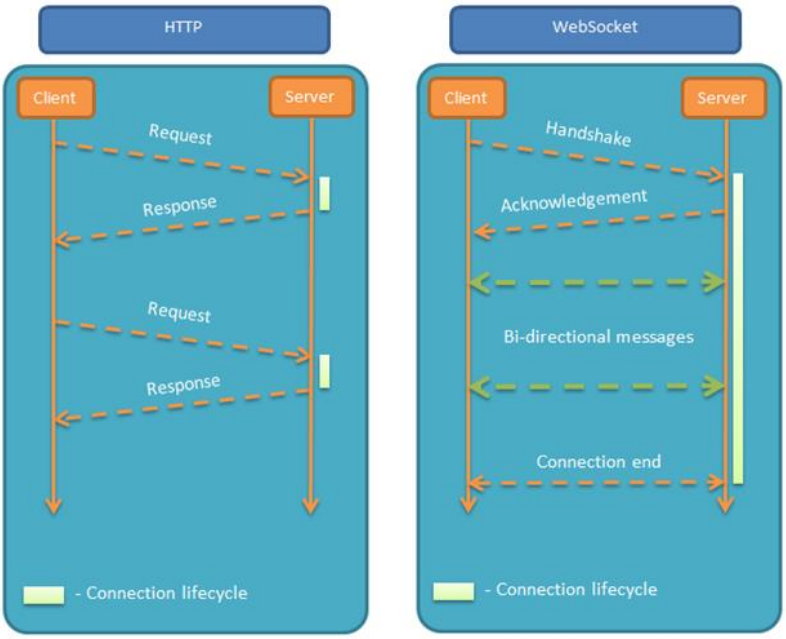

Рисунок 2.2 – Основні відмінності між протоколами HTTP та WebSocket

На стороні хмарного веб-сервера нам також необхідно розробити ПЗ для обробки запитів від головного контролера. Перш за все необхідно обрати хмарну інфраструктуру, яка б ідеально підходила б нам для нашої реалізації. DigitalOcean – це IaaS модель хмарних послуг, дозволяє використовувати виділений приватний віддалений веб-сервер для будь-яких цілей, без будь-яких обмежень чи контролю [14]. Цей варіант ідеально підходить для реалізації ПЗ веб-сервера в хмарі. DigitalOcean дає можливість обрати ОС, параметри та характеристики обчислювальних потужностей серверів (Див. рисунок 2.3).

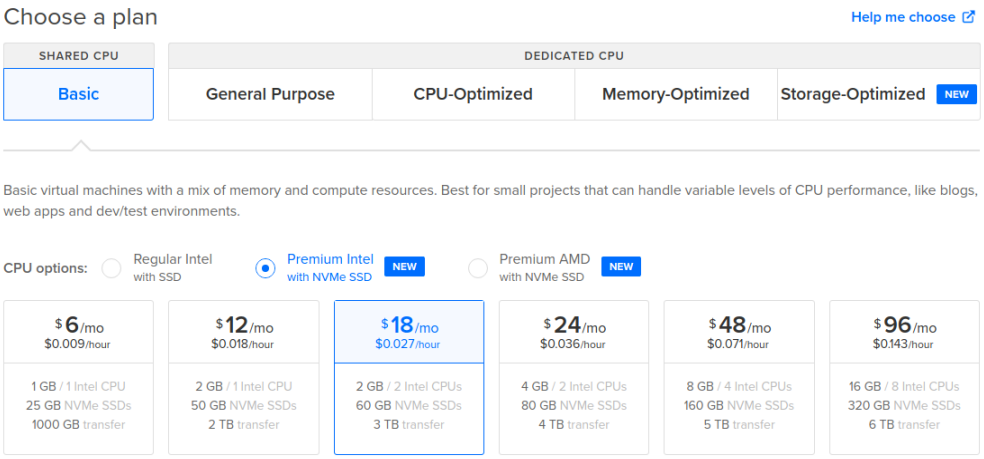

Рисунок 2.3 – Вигляд інтерфейсу вибору типу віддаленоо хмарного веб-серверу від DigitalOcean

На стороні веб-сервера буде використовуватись Node.js і TypeScript, реляційна база даних – PostgreSQL, ORM – TypeORM. Основна функція хмарного веб-серверу – це приймання, відправка та обробка інформації з і на зовнішню переферію через головний контролер (Raspberry Pi). На веб-сервері також необхідно налаштувати WebSocket з’єднання, оскільки сервер буде постійно обмінюватись інформацією з головним контролером. Також не варто забувати про те, що кінцевий користувач має мати змогу переглядати зібрану інформацію, створювати нові сценарії та підтримувати систему. Для цього необхідно реалізувати API на веб-сервері, що дасть нам можливість отримувати доступ до управління системою через протокол HTTP [15].

Також варто пам’ятати і про адміністрування та управління, для цього було прийнято рішення створити панель управління, яка комунікуватиме з веб-сервером через вище описане API. Для реалізації адмін панелі було обрано бібліотеку React.js та MUI (бібліотека створена, для полегшення розробки графічних інтерфейсів) [16].

Отже, вище описана архітектура (Див. рисунок 2.3) надасть нам змогу в повному обсязі реалізувати більшість поставлених задач для програмування та розгортання системи та надасть користувачеві можливість отримувати інформацію про стан системи та здійснювати управління та адміністрування.

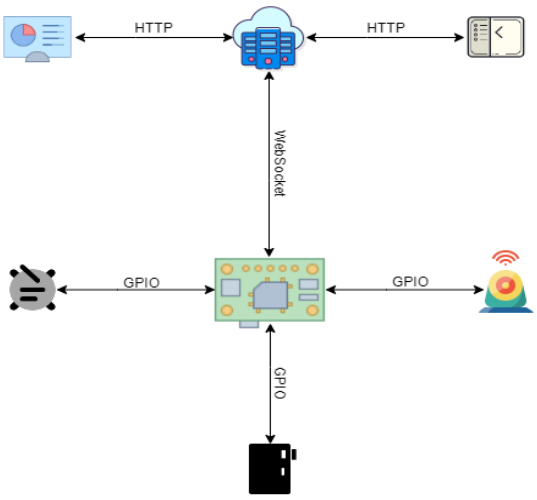

Рисунок 2.3 – Діаграма архітектури системи управління “Розумний будинок”

Архітектура системи також дозволятиме додавати нові функціональності без необіхдності внесення глобальних змін.

2.2 Створення програмного забезпечення для веб-сервера

Перш ніж розпочати розробку серверної частини, нм необхідно налаштувати середовище та хмарний сервіс. Як описувалось раніше, мною був обраний IaaS хмарний сервіс – DigitalOcean, що надає нам повну свободу дій в розробці та налаштуванні. Для можливості зручного розгортання системи мною був обраний Docker, це допоміжне ПЗ для розгортання та адміністрування застосінків. Спочатку мною було проведено налаштування Dockrer файлів, системи контролю версій git, також необхідно було визначити структуру майбутнього проекту.

У файлі *docker-compose.yml* було описано, які сервіси будуть створені, та з яких образів вони будуть завантажуватись (Див. лістинг 2.1) [17].

Лістинг 2.1 – описані сервіси у файлі docker-compose.yml та конфігурації до них

```
version: '3.7'
```

```
services:

  api-server:
    container_name: api-server
    build:
      context: ./api
      dockerfile: Dockerfile
    volumes:
      - ./api:/app
      - /app/node_modules
    environment:
      POSTGRES_HOST: postgres
      POSTGRES_PORT: 5432
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: smart_house
    ports:
      - '3000:3000'
      - '30000:9229'
    restart: always

  postgres:
    container_name: postgres
    image: postgres:latest
    ports:
      - '5432:5432'
    volumes:
      - ./data/db/postgres:/var/lib/postgresql/data
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: smart_house
```

Наш файл містить два описаних сервіси, перший – це безпосередньо веб-сервер, він має назву *api-server*, другий – це база даних PostgreSQL.

Наш проект містить зворотній проксі-сервер, що буде розташований в каталозі *proxy* та має свій власний Docker файл, він знадобиться нам при фінальному налаштуванні проекту, та вихідний код Node.js сервера в каталозі *api*. Вихідний код написаний на мові програмування TypeScript. TS компілятор буде компілювати код та генерувати js код, який буде запускатись платформою Node.js [18]. Як вже було сказано ранше, для роботи з базою даних мною була обрана бібліотека TypeORM. Дана ORM має хороший функціонал порівняно з аналогами, та підтримує реляційні та документні бази даних. Перед початком

написання коду мною було проведено базові налаштування Docker файлу для сервісу *server-api* (Див. лістинг 2.2), налаштовано файл проекту *package.json*, та файл конфігурації мови програмування TypeScript – *tsconfig.json*. Також було здійснено налаштування файлу конфігурацій для бібліотеки TypeORM (Див. лістинг 2.3).

Лістинг 2.2 – інструкції Docker файлу для сервісу api-server

```
FROM node:17.0.1

WORKDIR /app
COPY ./package*.json ./
RUN npm install
RUN npm install -g nodemon

COPY . .
EXPOSE 3000

CMD ["npm", "run", "start:dev"]
```

Лістинг 2.3 – конфігурації файлу tsconfig.json

```
{
  "compilerOptions": {
    "strictPropertyInitialization": false,
    "inlineSources": true,
    "downlevelIteration": true,
    "removeComments": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "forceConsistentCasingInFileNames": true,
    "module": "commonjs",
    "esModuleInterop": true,
    "target": "es6",
    "moduleResolution": "node",
    "sourceMap": true,
    "outDir": "dist",
    "baseUrl": ".",
    "paths": {
      "*": [
        "node_modules/*"
      ]
    },
  },
  "include": [
    "./*.ts"
  ],
  "exclude": [
```

```
    "node_modules",
    "migrations",
  ]
}
```

Docker файли містять важливі інструкції, які будуть виконуватись перед або під час запуску Docker контейнерів. Файл *tsconfig.json* містить параметри налаштування TypeScript компілятора, які прописуються індивідуально для кожного проекту чи модуля програми в залежності від потреб та специфіки розробки [19].

Отже, базова структура після налаштування середовища розробки і опису всіх конфігураційних файлів виглядатиме так (Див. рисунок 2.4):

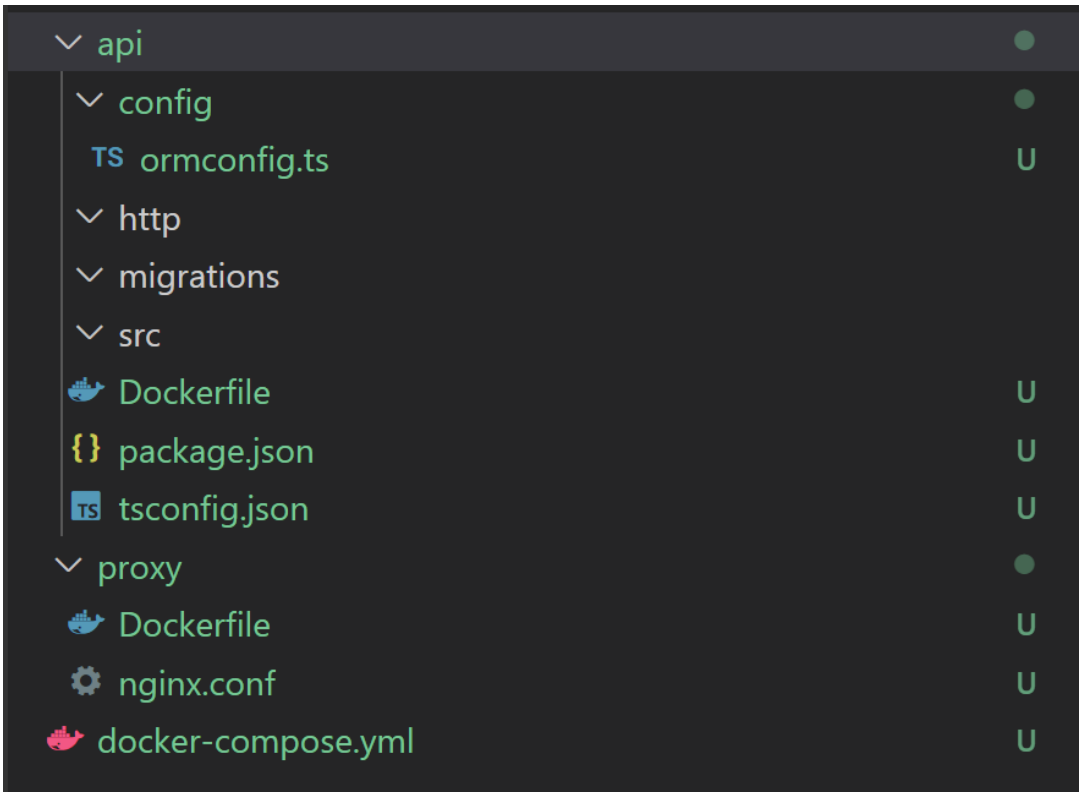

Рисунок 2.4 – Вигляд базової структури проекту після початкового налаштування проекту

Після базового налаштування мною було проведено інсталяцію всіх образів, описаних в файлі *docker-compose.yml*. Описані інструкції в Docker файлі дозволять запустити наш проект, проте поки в режимі розробника, тому

компілятор не буде зберігати, скомпільовані з мови програмування TypeScript, фали в файловій системі, а буде тримати їх в кеші. При фінальному запуску проекту на веб-сервері розробник зазвичай налаштовує режим *production*, в якому скомпільовані файли в кеш не попадають, а зберігаються в файловій системі та виконуються платформою Node.js [20]. Також разом з інсталяцією всіх образів Docker запустить та налаштує базу даних таким чином, як ми це описали в файлах конфігурації. Базове налаштування проекту дає нам можливість розпочати створення міграцій в базі даних та описувати сутності.

Базові сутності програми:

- *User* – стуність, яка нам необхідна для наданя доступу в систему упраління “Розумний будинок”.
- *Device* – сутність, яка необхідна для обробки запитів із головного контролера, оскільки запит міститиме інформацію про пристрій та інформацію, яку отримав пристрій. Це також допомое нам розіпзнавати який пристрій нам потрібно активувати, та яку інформацію на нього необхідно передавати.
- *Message* – сутність, яка є дозволить нам відслідковувати всі повідомлення, що були надіслані до головного контролера та прийняті від нього. Це допоможе нам відслідкувати стан кожного девайсу, якщо він матиме змогу відправити повідомлення, а також бачити історію всіх повідомлень, які були надіслані до головного контролера.
- *Room* – сутність, яка відображатиме кімнату або частину житлового приміщення домівки, дана сутність відображає дані про кімнату в повідомленнях від гооловного контролера, іншими словами, ми можемо знати з якого пристрою та якої кімнати прийшло повідомлення.
- *Command* – сутність, яка відображає дію зовнішнього пристрою, що надіслана від веб-сервера до головного контролера. Головна мета дадавання цієї сутності – є необхідність для попереднього опису команд для різноманітних пристроїв.

- *Access Token* – сутність, що дозволить адмініструвати вхід в систему. У випадку, якщо користувач введе коректні авторизаційні дані, програма повинна згенерувати access token, з яким в подальшому можна відвідувати розділи панелі адміністрування. Access token використовується в заголовках HTTP запитів. Бекенд частина повинна валідувати access token [21].

Основна логіка програми веб-сервера описується в сервісах та контролерах, слідуючи найпопулярнішому шаблону розробки – MVC. Для проекту мною було реалізовано головний роутер, який приймає запити від клієнта та призначає кожному запиту свій обробник та контролер. Для кожного endpoint API можна призначити свій метод контролеру та обробник (Див. лістинг 2.4), який може здійснювати валідацію або перевірку даних, що прийшли в запиті.

Лістинг 2.3 – приклад опису endpoint з призначеним контролером та обробником

```
import { Route } from "./types/Route";
import httpMethods from "http-methods-constants";
import FileController from "../controllers/UserController";
import PaginationRequest from "../requests/PaginationRequest";

/**
 * File routes
 */
export const file: Array<Route> = [
    {
        method: httpMethods.GET,
        path: "/api/users",
        requests: [
            new PaginationRequest
        ],
        middleware: UserController.list
    },
    {
        method: httpMethods.GET,
        path: "/api/user/{{userId}}",
        middleware: UserController.index
    }
];
```

Для того, щоб мати можливість отримувати повідомлення з головного контролера, мною було налаштовано зв’язок з головним контролером через протокол WebSocket. Деталі цього налаштування містяться в лістингу 2.4.

Лістинг 2.4 – створення обробника подій для комунікації з головним контролером через WebSocket протокол

```
io.listen(server).sockets.on('connection', (socket) => {
    socket.on("message", (msg) => {
        await router.readSocketMessage(msg);
    });

    socketEmitter.on("sendMessage", message => {
        socket.json.send(message);
    });
});
```

В даному проекті бізнес логіка є розділеною на моделі, представлення, контролери та сервіси. Представленням в даній програмі слугує JSON формат та формат передачі даних через WebSocket протокол. Спосіб передачі даних через JSON також був розділений мною на два основних – це: відповідь на запит одного елемента або відповідь на запит цілої колекції з використанням пагінації. Платформа Node.js та модуль http надає нам можливість без додаткових налаштувань використовувати формат передачі даних JSON.

Поняття моделі передбачає прямий зв’язок з данними застосунку. Це головний компонент шаблону проектування MVC (Див. рисунок 2.5) [22]. В даному проекті модель буде використовуватись для управління даними та відображення основних вище описаних сутностей.

В контролерах відбувається основна реалізації прив’язки моделі до представлення, в нашому випадку моделі – до формату JSON. Контролер не повинен виконувати функції отримання даних, структурування чи серіалізації. В цьому проекті кожен контролер має функцію прив’язки моделі до представлення даних клієнтській стороні, тому для кожної моделі мною був реалізований контролер, який надаватиме можливість створення, редагування

та видалення даних, а також передачі чи отримання повідомлень від головного контролера через протокол WebSocket.

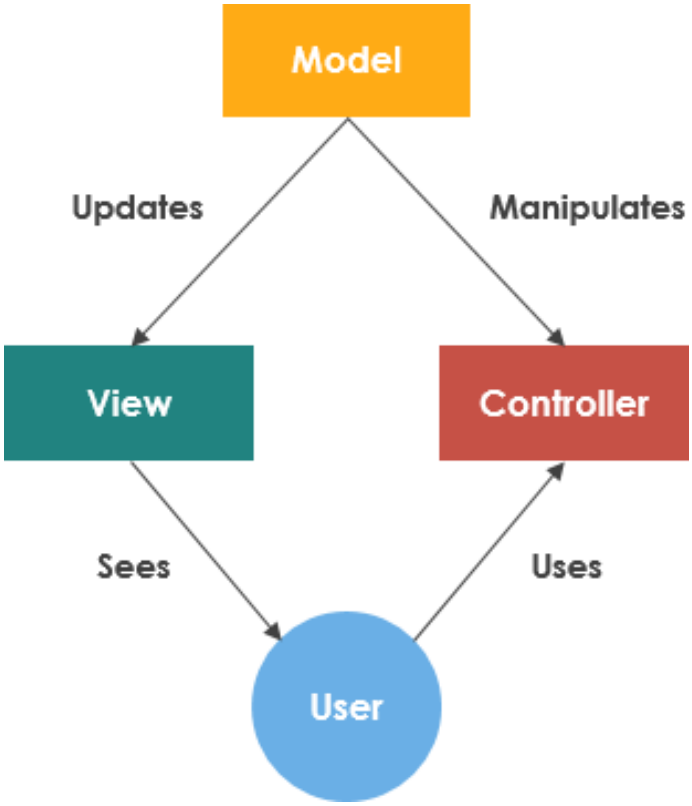

Рисунок 2.5 – Діаграма опису поняття шаблону проектування MVC

Сервіси – це додаткові компоненти, вони можуть слугувати шаром між контролером та моделлю, а також можуть містити елементи контролера [23]. Спосіб використання сервісів визначається особливостями архітектури та іншими аспектами чи потребами проекту. В даному проекті сервіси реалізовані як спосіб структурування та отримання даних для передачі їх контролеру, а контролер в свою черегу відображає зв’язок між собою та представленням. В сервісах описані методи отримання даних та логіка відправки і передачі повідомлень від і до головного контролера. Розділення логіки програми на сервіси та контролери дозволить розробнику підтримувати краще масштабування проекту та надасть можливість реалізовувати додатковий функціонал в програмі, який не може бути опиманий в контролерах, як

наприклад передача або обробка запитів від Raspberry Pi через протокол WebSocket із застосуванням бібліотеки Socket.IO.

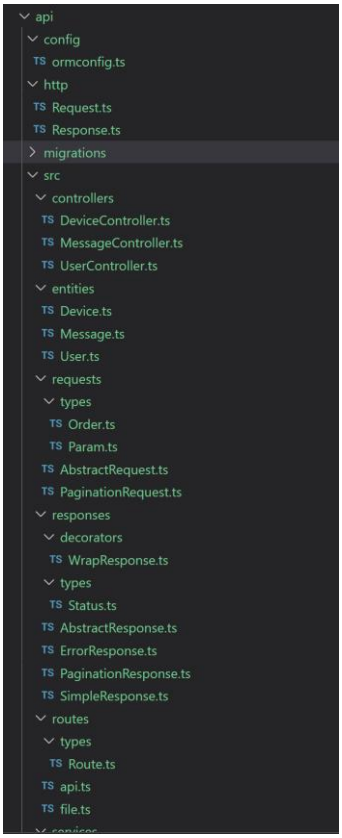

Рисунок 2.6 – Вигляд кінцевої структури проекту веб-сервера

Отже, після створення всіх моделей, функціоналу контролерів та сервісів, проект для веб-сервера матиме стурктуру типового MVC веб-застосунку (Див. рисунок 2.6).

2.3 Створення програмного зебезпечення для головного контролера. Опис взаємодії приладів

Основний принцип роботи головного контролера – це передавання сигналів від зовнішньої переферії до хмарного веб-серверу і навпаки. За основу головного контролера було взято одноплатний мінікомп’ютер Raspberry Pi з ОС Raspberry Pi OS. Дана операційна система розроблена на біза Linux, отже ми зможемо використати той ж стек технологій, що і для веб-сервера.

Для початку мною було налаштовано середовище для розробки. Node.js сервер на Raspberry Pi також запускається в Docker контейнері. Оскільки головний контролер буде виконувати функцію провідника повідомлень, в файлі *docker-compose.yml* ми можемо описати лише один сервіс *client* (Див. лістинг 2.5).

Лістинг 2.5 – вміст файлу docker-compose.yml для розгортання системи під Raspberry Pi

```
version: '3.7'

services:

  api-server:

    container_name: client
    build:
      context: ./
      dockerfile: Dockerfile
    volumes:
      - ./api:/app
      - /app/node_modules
    ports:
      - '3000:3000'
      - '30000:9229'
    restart: always
```

Для початку мною було налаштовано середовище для розробки. Node.js сервер на Raspberry Pi повинен бути налаштований таким чином, щоб він зміг отримувати повідомлення від зовнішніх пристроїв через інтерфейс gpio та передаввати їх на хмарний веб-сервер через налаштоване WebSocket підключення. Також Raspberry Pi контролер має можливість отримувати повідомлення від веб-сервера, та передавати їх на зовнішні пристрої. Для організації можливості такого обміну повідомленнями на Raspberry Pi мною було встановлено спеціальний пакет *onoff* [24]. Даний пакет надає можливість Node.js серверу взаємодіяти з інтерфейсом gpio. Також наш Node.js сервер використовує бібліотеку *onoff*, яка надає доступ до об’єкту gpio (Див. лістинг 2.5).

Лістинг 2.5 – приклад використання бібліотеки onoff, з передачою сигналу на зовнішній пристрій з портом під номером 4.

```
import { Gpio } from "onoff";

const gpio = Gpio(3, 'out');

async function write(value) {
    await gpio.write(value);
}

write(1);
```

Варто зазначити, що дана бібліотека може працювати не з всіма пристроями, тому існує необхідність розробки такої архітектури проекту, яка надавала б змогу працювати з іншими пристроями та підключати інші бібліотеки та функціонал. Тому мною було прийнято рішення розробити абстрактний клас *Device*, з якого будуть наслідуватись всі інші класи, що відображатимуть та реалізовуватимуть зв’язок з конкретним переферійним пристроєм, що підключений через зовнішній інтерфейс до Raspberry Pi.

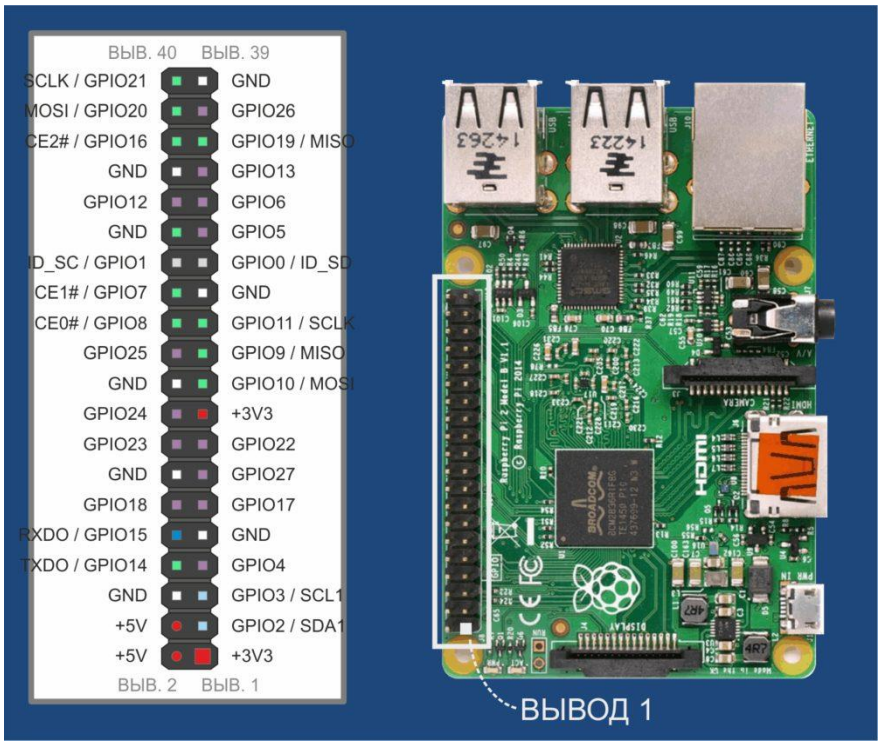

Рисунок 2.7 – Схема портів виводів Raspberry Pi

Основна мета головного контролера – це розпізнати який девайс передає повідомлення, або якому девайсу необхідно передати сигнал. Для цього назріває необхідність реалізувати зв’язок номеру порту Raspberry Pi з класом конкретного девайсу. Для допомоги в пошуку необхідних портів мною було використано схему портів виводів та роз’ємів одноплатного мінікомп’ютера (Див рисунок 2.7). Варто зазначити, що не всі пристрої можливо підключити через інтерфейс gpio. Архітектура Node.js застосунку повинна передбачати й підключення через usb роз’єми. Тому програма надаватиме можливість підключати інші бібліотеки та функціональності для комунікації з зовнішніми девайсами. Можна навести список найпопулярніших бібліотек та платформ для Node.js для роботи з IoT переферією [25]:

- Johnny-Five.
- Cylon.js.
- Node-Red.
- Jerryscript.
- NodeMcu.
- IoTjs.
- IoT-Nodejs.
- Zetta.
- Noduino.
- Node serialport.
- It.

2.4 Створення панелі керування

Після реалізації та налаштування головного контролера та веб-сервера виникає доцільність розпочати створення фронтенд частини проекту. Як вже зазначалось раніше, користувач повинен мати зручний інтерфейс для можливості управління та адміністрування системи “Розумний будинок”. На

практиці замовник через графічний інтерфейс повинен мати доступ до інформації про стан приміщень житла, пристрої, що виконують певні функції (наприклад, відкривають або закривають жалюзі на вікнах або показують температуру), команди, які можна подати на виконання певним пристроям, важливі повідомлення про надзвичайні випадки. Як вже було сказано раніше, будь-яка “розумна” система розробляється виключно під потреби замовника, тому архітектура графічного інтерфейсу повинна також давати можливість підтримувати масштабування та додавати новий функціонал або покращувати існуючий.

Мною був обраний стек React.js і MUI для створення графічного інтерфейсу користувача.

React.js – це JavaScript бібліотека з відкритим кодом для розробки графічних інтерфейсів користувачів в браузері [26].

MUI (Material UI) – JavaScript бібліотека, яка надає готові графічні рішення для забезпечення швидкої та простої веб-розробки [27].

React.js – це бібліотека, яка працює через компонентний підхід, тобто елементарний графічний елемент – це компонент (Див. лістинг 2.6).

Лістинг 2.6 – приклад простого компоненту React.js

```
function App() {
  return (
    <div className="App">
    </div>
  );
}
```

React.js містить готовий інструмент *create-react-app* для початкового створення та налаштування проекту. Перед викликом даної функції, нам необхідно впевнитись, що версія node.js в нашому локальному середовищі і в середовищі веб-серверу є вищою чим 16. Після виклику даної команди ми можемо спостерігати початкову структуру проекту, що наведена на рисунку 2.8.

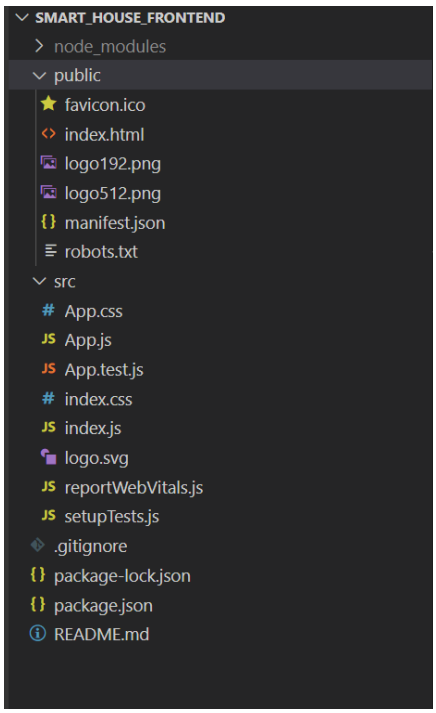

Рисунок 2.8 – Початковий вигляд структури React.js проекту

Після автоналаштування стартової версії проекту, нам необхідно виконати команду *npm start* для запуску застосунку. Відкривши посилання *localhost:3000* в браузері, ми можемо спостерігати, що наш проект запустився успішно (Див. рисунок 2.9) [28]. Далі можна приступати безпосередньо до самого створення графічних компонентів фронтенду. Мною було прийнято рішення про розробку загальної панелі керування з вкладками, кожна з яких відповідатиме певному розділу адміністрування. Було передбачено реалізувати п’ять розділів: Головна сторінка, Кімнати, Пристрої, Команди та Безпека.

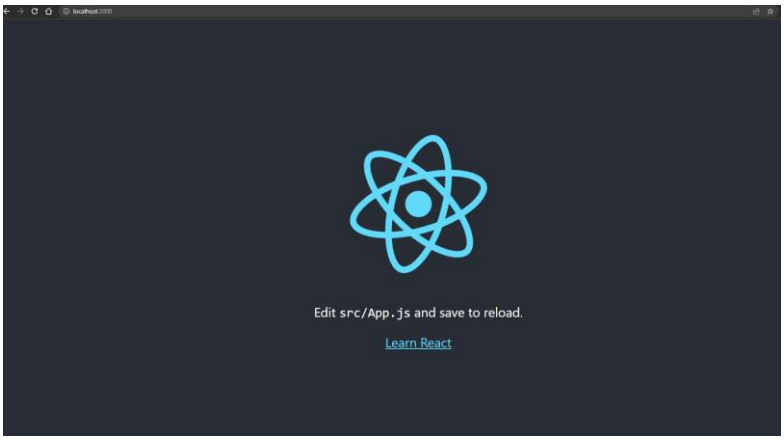

Рисунок 2.9 – Успішний запуск Reaact.js застосунку

Основною функцією фронтенд частини є надсилання запитів до API хмарного веб-серверу з метою передати або отримати дані. Тому перш за все мною була реалізована сторінка авторизації користувача (Див. рисунок 2.10) [29].

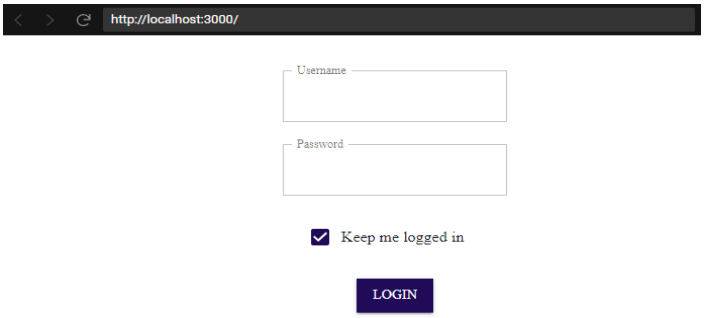

Рисунок 2.10 – Сторінка авторизації користувача

Для виконання запитів до віддаленого веб-сервера мною була обрана бібліотека *axios* [30]. Приклад запиту до сервера з використанням *axios* можна побачити в лістингу 2.7. React.js надає доволі хороший функцонал для реалізації отримання різних ключів доступу та їх подальшого використання.

Лістинг 2.6 – приклад реалізації запиту за допомогою бібліотеки axios

```
const response = await axios.post(
     LOGIN_URL,
     JSON.stringify({ user, pwd }),
        {
          headers: { "Content-Type": "application/json" },
          withCredentials: true,
        }
);
```

Як вже було сказано вище, панель адміністрування матиме п’ять основних розділів. Перш за все мною було реалізовано розділ головної сторінки (Див. рисунок 2.11).

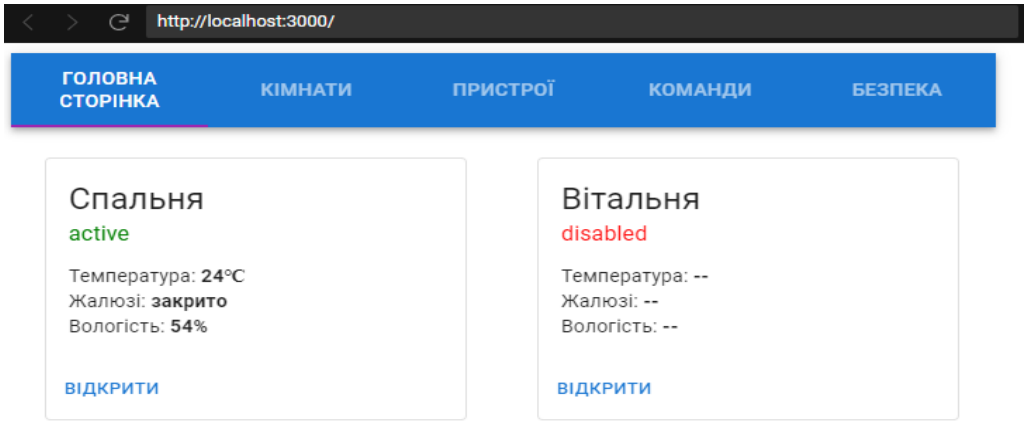

Рисунок 2.11 – Вигляд головної сторінки

Головна сторінка містить всі кімнати, що були додані та налаштовані користувачем, та основну інформацію про них із датчиків та інших приладів. Кожна кімната має статус: *active* або *disabled*. Якщо кімната була відключена від системи, то інформація з приадів в ній не відображатиметься. Наявність назв приладів та інформації, що вони передають залежить від того, чи були вони додані в систему.

Розділ Кімнати містить список кімнат, які раніше додав користувач (Див. рисунок 2.12).

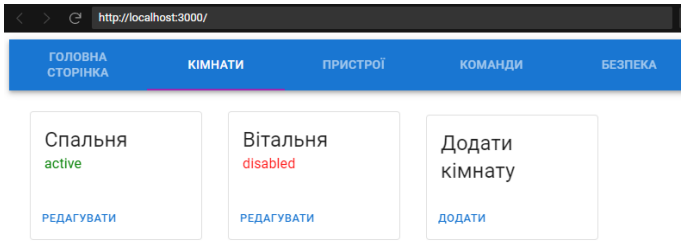

Рисунок 2.12 – Вигляд сторінки кімнат

Також існує можливість додати нову кімнату або редагувати існуючу (Див. рисунок 2.13).

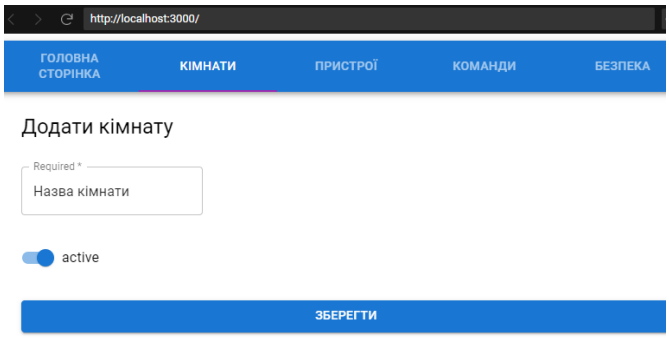

Рисунок 2.13 – Вигляд сторінки створення або редагування кімнат

Кожну створену кімнату можна додати в пристрій в розділі Пристрої (Див рисунок 2.14).

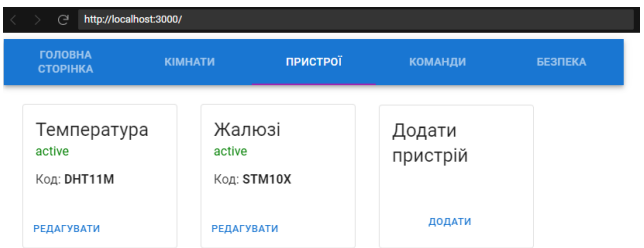

Рисунок 2.14 – Вигляд сторінки Пристрої

Кожен пристрій також має статус *active* або *disabled*, а також свій власний код, по якому ми зможемо розпізнавати зовнішні девайси в повідомленнях.

Розділ Команди (Див. рисунок 2.15) містить створені користувачем команди.

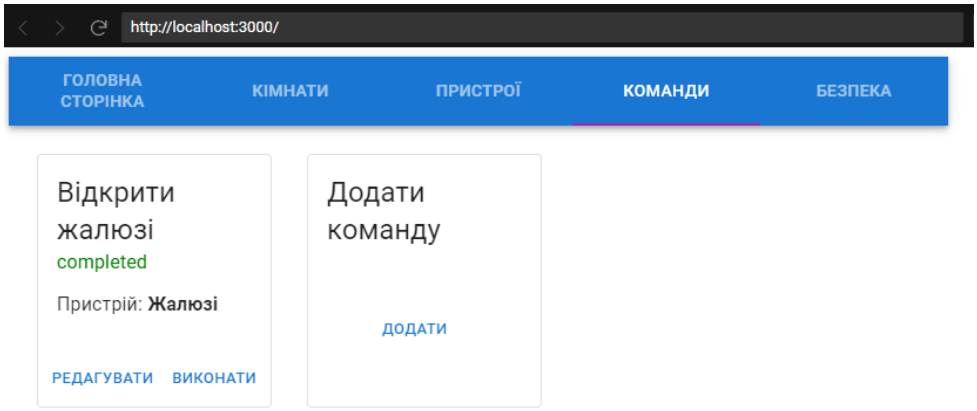

Рисунок 2.15 – Вигляд сторінки команд

Кожну команду можна запустити на виконання або редагувати (Див рисунок 2.16). Також є можливість додати нову команду.

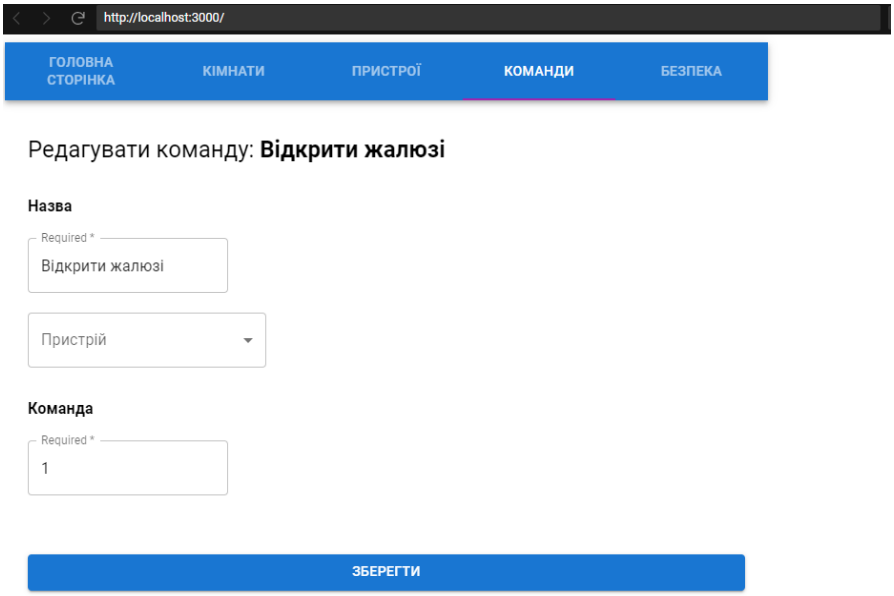

Рисунок 2.16 – Вигляд сторінки редагування команди

Кожна команда має статус: *pending* – у випадку якщо в даний момент відбувається виконання, *completed* – якщо команда була виконана успішно і *failed* – якщо команда не виконалась через помилку.

Останій розділ графічного інтерфейсу – Безпека. В даному розділі реалізовано взаємодія із системою охорони домівки та системою пожежної безпеки (Див. рисунок 2.17).

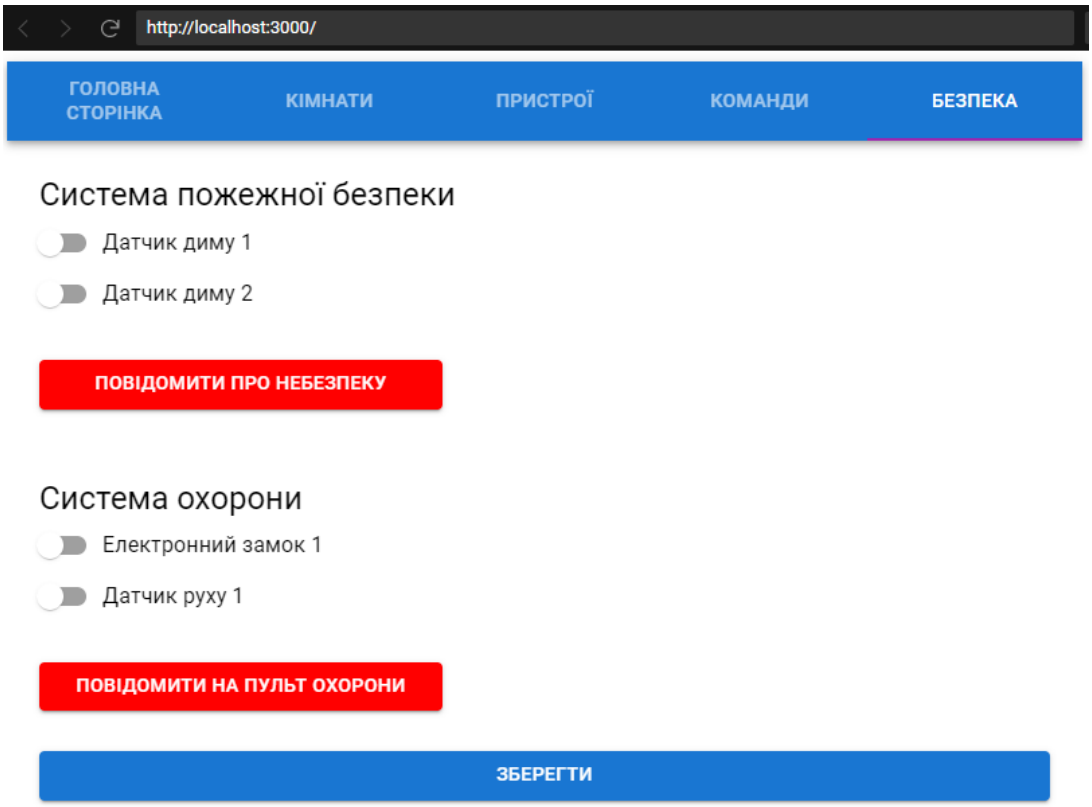

Рисунок 2.17 – Вигляд розділу Безпека

Дана вкладка надає нам можливість керувати системою охорони та пожежної безпеки, проте існують різні системи охорони, деякі не дають доступу для взаємодії через прикладні інтерфейси, а для деяких систем необхідно запитувати спеціальні дозволи, теж саме і можна сказати про систему пожежної безпеки. Тому даний розділ не буде активним та функціональним, якщо розробник не здійснить підключення системи управління “Розумний будинок” до зовнішніх прикладних інтерфейсів спеціальних служб, що є доступним не в кожній країні або регіоні.

2.5 Висновки до другого розділу

В даному розділі мною була здійснена і описана програмна реалізація системи управління “Розумний будинок”. Ситема передбачає три програмні застосунки: фронтенд, ПЗ для хмарного веб-сервісу та ПЗ для Raspberry Pi. Всі застосунки розроблялись та проектувались на базі платформи Node.js та на мові програмування TypeScript. Архітектура системи була спроектована так, щоб хмарний веб-сервер приймав на себе якомога більше функцій. Головний контролер слугує провідником повідомлень і команд між хмарним сервером та зовнішнніми пристроями. Для безперервної передачі повідомлень мною було прийно рішення реалізувати комунікацію головного контролера та веб-сервера через WebSocket з’єднання з використанням JavaScript бібліотеки Socket.IO. Також мною було спроектовано та реалізовано фронтенд частину, яка була написана на базі фреймворку React.js та бібліотеки MUI (Material UI). React.js використовує компонентний підхід до розробки графічних інтерфейсів, тому це спрощує підтримку масштабування графічних додатків та надає можливість повторного використання раніше написаного коду.

Також варто зазначити, що реалізація такої системи управління повністю буде залежати від потреб кінцевого користувача. Під час роботи над програмою в розробника можуть виникнути певні труднощі щодо роботи з зовнішніми інтерфейсами різноманітних пристроїв.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Психологічні чинники небезпеки

Аналіз статистичних даних та висновних різних експертів у галузі безпеки життєдіяльності дають можливість стверджувати, що від 60 до 90% травм у побуті та на виробництві відбувається з вини потерпілих. Основні причини цього такі: низький рівень професійної підготовки з питань безпеки, недостатнє виховання, слабка установка людини на дотримання вимог безпеки, допуск до небезпечних робіт осіб з підвищеним ризиком травматизму, перебування людей у стані втоми чи інших психічних станах, які знижують безпеку діяльності.

Виділяють комплекс чинників, що збільшують індивідуальну схильність людини до небезпеки. Це особливості темпераменту, функціональні зміни в організмі, дефекти органів відчуття, незадоволення даним видом діяльності.

Несприятливий характер діяльності (значні фізичні та розумові зусилля, незручна робоча поза, високий темп праці, нервово-емоційні перевантаження, перенапруга слухових та зорових аналізаторів, несумісність робочого місця, засобів праці, антропометричних даних людини) призводять до підвищеної фізичної та нервової втоми, яка послаблює психіку, знижує швидкість та точність орієнтації, притупляє пильність та увагу, порушує сприйняття того, що коїться. Це також спричинює травматизм. Психологи виділяють спеціальний розділ, психологію безпеки, в якому розглядають психічні властивості та різноманітні форми психічних станів, що спостерігаються у процесі трудової діяльності. Психічні процеси становлять основу психічної діяльності. Без них неможливе формування знань та надбання життєвого досвіду. Розрізняють пізнавальні, емоційні та вольові психічні процеси.

Психічні властивості – це стійкі особливості особи: інтелектуальні, емоційні, вольові, трудові. Психічні стани зумовлюють особливості психічної діяльності у конкретний період часу та можуть позитивно чи негативно

впливати на всі психічні процеси. На думку багатьох психологів, ефективність діяльності (працездатність) людини залежить від рівня психічного напруження. Підвищення рівня психічного напруження істотно збільшує ефективність праці. Але існує критична межа активації, після якої результати праці знижуються аж до повної втрати працездатності. Існують два типи позамежевого психологічного напруження – гальмівний та збудливий.

Гальмівний тип характеризується скутістю та сповільненістю рухів. Людина не здатна з колишньою спритністю виконувати професій ні дії. Знижується швидкість реакцій, сповільнюється процес мислення, погіршується згадування, розпорошується увага та виникають інші негативні прояви, не властиві даній людині у спокійно

Збудливий тип проявляє себе гіперактивністю, багатомовністю, тремтінням рук та голосу. Оператори здійснюють численні, не продиктовані конкретною потребою, дії. Вони перевіряють стан приладів, крутять регулятори, поправляють одяг, розтирають руки. У них з'являється дратівливість, запальність, невластива їм різкість, грубість, уразливість.му стані.

Позамежеві форми психічного напруження часто лежать в основі помилкових дій та неправильної поведінки у складній ситуації. що може спричинити травматизм та аварії. Серед особливих психічних станів, які мають істотне значення для безпеки життєдіяльності, психологи виділяють пароксизмальні розлади свідомості, психогенні зміни настрою та афектні стани, пов'язані з вживанням психічно активних засобів (стимуляторів, транквілізаторів, алкогольних напоїв).

Параксизмальні стани – група розладів, яка характеризується-короткочасною (від кількох секунд до хвилин) втратою свідомості. Такі стани характерні для деяких органічних захворювань головного мозку, епілепсії. Сучасні методики дають змогу своєчасно визначити осіб із прихованою схильністю до пароксизмальних станів. Цим людям протипоказана робота на висоті, водіями автотранспорту та інша робота із підвищеною небезпекою.

Психогенні зміни настрою та афектні стани виникають під впливом психічних дій. Зниження настрою та апатія можуть бути наявні від кількох хвилин до одного – двох місяців, Погіршення настрою спостерігається внаслідок конфліктних ситуацій, після загибелі близьких та в інших випадках. При цьому з'являються байдужість, млявість, загальна скутість, загальмованість, сповільнення темпу мислення . Погіршення настрою супроводжується погіршенням самоконтролю, що може стати причиною травматизму та збільшує ризик виникнення небезпечних ситуацій.

Афектні стани (афект – вибух емоцій) можуть виникнути внаслідок виробничих невдач, під впливом образи. У стані афекту у людини розвивається емоційне звуження обсягу свідомості. Можуть спостерігатися різкі рухи, агресивні та руйнівні дії. Особи, схильні до афектних станів, належать до категорії з підвищеним ризиком травматизму та не повинні призначатися на посади з високою відповідальністю. Використання психічно активних засобів, включаючи алкоголь, збільшує ризик травматизму та знижує рівень безпеки діяльності. Вживання легких стимуляторів (чай, кава) допомагає у боротьбі з сонливістю і може сприяти підвищенню працездатності на короткий період. Вживання ж активних стимуляторів на відповідальних роботах здатне викликати негативний ефект – погіршується самопочуття, зменшується швидкість реакції. Використання транквілізаторів, які діють заспокійливо та запобігають розвитку неврозів, може знижувати психічну активність, уповільнювати реакцію, викликати апатію та сонливість. Особливо потрібно підкреслити вплив на безпеку діяльності алкогольних напоїв. За різними даними, автомобільний травматизм у 60% випадків пов'язаний з вживанням алкоголю. Встановлено [31], що 64% смертельних випадків на виробництві викликано вживанням алкоголю та помилковими діями загиблих. Для безпеки праці особливе значення має після алкогольна астенія (похмілля), яка не лише знижує працездатність, а й пригнічує нервову систему. Тривале вживання алкоголю спричинює алкоголізм, який супроводжується різним ступенем деградації особи. Люди, які страждають на алкоголізм, втрачають властиву їм

точність та охайність у роботі. Вони дедалі частіше допускають помилки та стають нездатними для вирішення складних проблем, до швидкої та правильної орієнтації у нестандартних ситуаціях.зводить до загальмованості та притуплення відчуття обережності. Тривале вживання алкоголю спричинює алкоголізм, який супроводжується різним ступенем деградації особи. Люди, які страждають на алкоголізм, втрачають властиву їм точність та охайність у роботі. Вони дедалі частіше допускають помилки та стають нездатними для вирішення складних проблем, до швидкої та правильної орієнтації у нестандартних ситуаціях.

3.2 Оцінка технологічного процесу щодо умов пожежонебезпеки при роботі з електронно-обчислювальною технікою.

Будь-яка оцінка пожежонебезпеки здійснюється за результатами відповідного аналізу пожежонебезпеки приміщень, будівель та умов роботи в них. Таким чином, методика аналізу пожежонебезпеки зводиться до виявлення і оцінки потенційних та наявних джерел запалювання, умов формування горючого середовища, умов виникнення контакту джерел запалювання та горючого середовища, умов та причин поширення вогню в разі виникнення пожежі або вибуху, наявності та масштабів імовірної пожежі, загрози життю і здоров'ю людей, навколишньому середовищу, матеріальним цінностям [32].

Необхідність матеріальної оцінки пожежонебезпеки потребує чітких критеріїв її визначення. Відомі два підходи до питань нормування в галузі вибухопожежонебезпеки: імовірнісний та детермінований.

Імовірнісний підхід, що ґрунтується на концепції допустимого ризику, передбачає недопущення впливу на людей і матеріальні цінності небезпечних факторів пожежі з імовірністю, яка перевищує нормативну.

Детермінований підхід базується на розподілі об'єктів за ступенем пожежонебезпеки на категорії і класи з позначенням їх конкретних кількісних меж залежно від параметра, що характеризує можливі наслідки пожежі.

Основою для встановлення нормативних вимог щодо конструктивних та планувальних рішень на об'єктах, де розміщуються електронно обчислювальні прилади, а також інших питань забезпечення їхньої вибухопожежобезпеки є визначення категорій приміщень та будівель різного призначення за пожежною небезпекою.

Існують два основних джерела виникнення пожеж в приміщеннях, де працюють електронно обчислювальні прилади:

– можливе виникнення пожежі через коротке замикання.
– виникнення пожежі через наявність легкозаймистих матеріалів.

Коротке замикання – це наслідок контакту двох проводів з різними потенціалами. Коротке замикання виникає внаслідок неправильної експлуатації, перегріву або порушення ізоляції проводів, затоплення квартири або інших чинників [33].

Для того, щоб попередити коротке замикання, слід:

– не використовувати поламані електроприлади;
– ремонтувати або замінювати їх новими своєчасно;
– розетки і вимикачі, що іскрять, ремонтувати в терміновому порядку;
– не маючи навичок і знань, не ремонтувати електропроводку самостійно;
– встановити автомат і пристрій захисного відключення (УЗО);
– 2 рази на рік проводити діагностику електропроводки викликаючи майстра зі спеціальним інструментом.

Також варто зазначити про вплив наявності легкозаймистих матеріалів на пожежонебезпеку. До пожеженебезпечних речовин належать лаки, фарби, горючі й мастильні речовини, розчинники, аерозолі, газ тощо. Пожежонебезпечними матеріалами є меблі, одяг, газети, журнали, книжки, картон, дерев'яні споруди, пластмасові вироби та покриття, дрова, вугілля, деревна стружка. Поблизу цих матеріалів категорично забороняється палити, користуватися відкритим вогнем. Якщо необхідно розігріти пожежонебезпечну рідину, це роблять за допомогою гарячої води. У разі використання

імпортних речовин та матеріалів необхідно суворо дотримуватися вказівок та інструкцій щодо виконання робіт, які додаються. Не дозволяється використовувати речовини та матеріали, які не мають характеристик пожежної безпеки, а також вазівок або інструкцій щодо безпечного виконання робіт. Для виконання робіт з використанням горючих речовин має застосовуватись інструмент, виготовлений з матеріалів, які не дають іскор у разі удару (алюміній, мідь, пластмаса, бронза).

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розглянуто основну концепцію системи “Розумний будинок”, способи розробки та розгортання даної системи, проблеми, які потенційно можуть виникнути при проектуванні архітектури “розумних” систем. Метою реалізації “Розумного будинку” є вирішення трьох важливих проблем: проблема комфорту жителів будинк чи квартири, проблема надмірного споживання ресурсів та проблема безпеки. Розробка таких проектів орієнтується виключно на потреби замовника, оскільки не існує загальних правил та способів реалізацій. Тому інженер має можливість використовувати будь-які технології, які здатні підтримувати мережеві інтерфейси та веб-протоколи. В даній роботі вибрано такий спосіб реалізації системи управління, програмна логіка якого буде працвати в хмарі, це дозволить значно зекономити обчислювальні ресурси, та мінімізувати проблеми масштабування.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Подано інфрормацію про типи арзхітектур та методи, якими можна реалізувати управління системою “Розумний будинок”.

– Розглянуто різновиди хмарних інфраструктур і найпопулярніші типи хмарних послуг.

– Висвітлено проблеми реалізації таких систем.

– Проаналізовано основні критерії, які беруть до уваги замовники, при виборі сценаріїв розробки “Розумного будинку”.

В другому розділі кваліфікаційної роботи:

– Спроектовано можливу архітектуру ПЗ для різних елементів системи управління “Розумний будинок”.

– Запропоновано реалізувати ПЗ для ефективного управління системою “Розумний бдинок” згідно спроектованої архітектури з використанням хмарних технологій.

– Розроблено ПЗ для всіх елементів системи “Розумного будинку” та налаштовано комунікацію переферійних приладів з хмарним веб-сервером.

У розділі «Безпека життєдіяльності, основи охорони праці»

Висвітлено питання про безпеку життєдіяльності та охорону праці, а саме:

– Психологічні чинники небезпеки.
– Проведено оцінку технологічного процесу щодо умов пожежонебезпеки при роботі з електронно-обчислювальною технікою.

ПЕРЕЛІК ДЖЕРЕЛ

1 Розумний будинок – з чого він складається та чи потрібен вам? [Електронний ресурс] – Режим доступу до ресурсу: https://nachasi.com/tech/2018/06/25/smart-house-faq/

2 How to design a Smart Home system [Електронний ресурс] – Режим доступу до ресурсу: https://www.smartsecurity.guide/how-to-design-a-smart-home-automation-system/

3 СЦЕНАРІЇ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ «РОЗУМНИЙ ДІМ» З ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ [Електронний ресурс] – Режим доступу до ресурсу: http://journals.nupp.edu.ua/mist/article/view/555

4 Intelligent Information Technologies [Електронний ресурс] – Режим доступу до ресурсу: http://ekmair.ukma.edu.ua/bitstream/handle/123456789/16991/Hlybovets_Analiz_proprohramn_system_pidtrky_rozumnoho_budynku.pdf?sequence=1&isAllowed=y

5 Private Cloud vs. Public Cloud: What is the difference? system [Електронний ресурс] – Режим доступу до ресурсу: https://www.vmware.com/topics/glossary/content/private-cloud-vs-public-cloud.html#:~:text=In%20its%20simplest%2C%20a%20private,customers%20who%20want%20similar%20services.

6 Difference between IAAS, PAAS and SAAS [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/difference-between-iaas-paas-and-saas/#:~:text=IAAS%20gives%20access%20to%20the,access%20to%20the%20end%20user.&text=It%20is%20a%20service%20model,computing%20resources%20over%20the%20internet.

7 Top Level Architecture of a Smart home [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Top-Level-Architecture-of-a-Smart-home_fig9_221907506

8 Google Cloud vs AWS in 2022 (Comparing the Giants) home [Електронний ресурс] – Режим доступу до ресурсу: https://kinsta.com/blog/google-cloud-vs-aws/

9 Raspberry Pi OS [Електронний ресурс] – Режим доступу до ресурсу: https://www.raspberrypi.com/software/

10 Connect your Raspberry Pi – Projects [Електронний ресурс] – Режим доступу до ресурсу: https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started/3

11 Raspberry Pi GPIO Pinout Projects [Електронний ресурс] – Режим доступу до ресурсу: https://learn.sparkfun.com/tutorials/raspberry-gpio/all

12 Introduction to IoT with Raspberry Pi and Node.js using RGB LED lights [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/sysf/introduction-to-iot-with-raspberry-pi-and-node-js-using-rgb-led-lights-77f4750a5ea9

13 WebSockets vs. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://ably.com/topic/websockets-vs-http#:~:text=Unlike%20HTTP%2C%20where%20you%20have,request%2Fresponse%2Dbased%20methods.

14 How to Create a Droplet from the DigitalOcean Control Panel [Електронний ресурс] – Режим доступу до ресурсу: https://docs.digitalocean.com/products/droplets/how-to/create/

15 HTTP Methods - REST API Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/http-methods/

16 Move faster with intuitive React UI tools [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/

17 Overview of Docker Compose [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/compose/

18 What is TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/learn/nodejs-with-typescript

19 What is a tsconfig.json [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/handbook/tsconfig-json.html

20 Node.js, the difference between development and production [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.dev/learn/nodejs-the-difference-between-development-and-production

21 How to Build an Authentication API with JWT Token in Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.section.io/engineering-education/how-to-build-authentication-api-with-jwt-token-in-nodejs/

22 MVC Framework – Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

23 What is the difference between Controllers and Services in Node REST API's? [Електронний ресурс] – Режим доступу до ресурсу: https://www.coreycleary.me/what-is-the-difference-between-controllers-and-services-in-node-rest-apis/

24 onoff – npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/onoff

25 Top 10 Javascript Frameworks To Use In Your IoT Project [Електронний ресурс] – Режим доступу до ресурсу: https://analyticsindiamag.com/top-10-javascript-frameworks-to-use-in-your-iot-project/

26 React – Getting Started [Електронний ресурс] – Режим доступу до ресурсу: https://en.reactjs.org/docs/getting-started.html

27 MUI – Example projects [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/material-ui/getting-started/example-projects/

28 Create a New React App [Електронний ресурс] – Режим доступу до ресурсу: https://en.reactjs.org/docs/create-a-new-react-app.html

29 Create a Login Page – React App [Електронний ресурс] – Режим доступу до ресурсу: https://serverless-stack.com/chapters/create-a-login-page.html

30 Axios – Getting Started [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/docs/intro

31 Психологічні чинники небезпеки, – ОСНОВИ БЕЗПЕКИ [Електронний ресурс] – Режим доступу до ресурсу: https://subject.com.ua/safety/bezpeka/30.html

32 Пожежна безпека в Україні - Служба охорони праці [Електронний ресурс] – Режим доступу до ресурсу: https://www.sop.com.ua/article/1013-pojejna-bezpeka

33 Пожежна небезпека коротких замикань, перенавантажень і перехідного опору [Електронний ресурс] – Режим доступу до ресурсу: https://ns-plus.com.ua/2017/02/12/analiz-pozhezh-v-ukrayini-za-2016-rik/.

ДОДАТКИ

Додаток А

Програмний код

Лістинг 1.1 – файл ormconfig.ts

```
    import { ConnectionOptions } from 'typeorm';

    const      entityPath:      string      =      'production'      ===
    process.env.environment    ?    '/app/dist/src/entities/*.js'    :
    '/app/src/entities/*.ts';

    /**
     * TypeORM connection options
     */
    const config: ConnectionOptions = {
        type: 'postgres',
        host: process.env.POSTGRES_HOST,
        port: Number(process.env.POSTGRES_PORT),
        username: process.env.POSTGRES_USER,
        password: process.env.POSTGRES_PASSWORD,
        database: process.env.POSTGRES_DB,
        migrations: [
            '/app/migrations/*{.ts,.js}',
        ],
        entities: [
            entityPath
        ],
        cli: {
            migrationsDir: 'migrations',
        }
    };

export = config;
```

Додаток Б

Програмний код

Лістинг 1.2 – файл Request.ts

```
import { IncomingMessage } from "http";
import { URLSearchParams } from "url";
import formidable, { Fields, Files } from "formidable";

/**
 * Class Request
 */
export default class Request
{
    /**
     * Incoming message
     * @private
     * @readonly
     * @type IncomingMessage
     */
    private readonly incomingMessage: IncomingMessage;

    /**
     * Request method
     * @public
     * @type string
     * @default null
     */
    public method: string | null = null;

    /**
     * Request path
     * @public
     * @type string|null
     * @default null
     */
    public path: string | null = null;

    /**
     * Request headers
     * @public
     * @type object
     * @default {}
     */
    public headers: object = {};

    /**
     * Request queries
     * @public
     * @type object
     * @default {}
     */
    public queries: object = {};
```

```
/**
 * Request path params
 * @public
 * @type object
 * @default {}
 */
public params: object = {};

/**
 * @param incomingMessage
 * @public
 */
public constructor(incomingMessage: IncomingMessage)
{
    this.incomingMessage = incomingMessage;

    const { url, headers, method } = this.incomingMessage;

    this.setHeaders(headers);
    this.setQueries(url);
    this.setPath(url);
    this.setMethod(method)

    this.readJsonBody = this.readJsonBody.bind(this);
    this.readFormBody = this.readFormBody.bind(this);
}

/**
 * Set request method
 * @public
 * @param method
 * @returns Request
 */
public setMethod(method: string | undefined): this
{
    if (method) this.method = method;

    return this;
}

/**
 * Set request path
 * @public
 * @param url
 * @returns Request
 */
public setPath(url: string | undefined): this
{
    if (url) this.path = url.split('?')[0];

    return this;
}

/**
```

```
     * Set request queries
     * @public
     * @param url
     * @returns Request
     */
    public setQueries(url: string | undefined)
    {
        if (url) {
            url = url.replace('/', '').split('?')[1] ?? "";
            const params = new URLSearchParams(url);
            for (const param of params.entries()) {
                this.queries        =        {...this.queries,
...{[param[0]]: param[1]}};
            }
        }

        return this;
    }

    /**
     * Set request headers
     * @public
     * @param headers
     * @returns Request
     */
    public setHeaders(headers: object): this
    {
        for (const entry of Object.entries(headers)) {
            this.headers  =  {...this.headers,  ...{[entry[0]]:
entry[1]}};
        }

        return this;
    }

    /**
     * Set path params
     * @public
     * @param params
     * @returns Request
     */
    public setParams(params: object)
    {
        this.params = params;

        return this;
    }

    /**
     * Read json request body
     * @public
     * @returns Promise<object>
     */
```

```
        public async readJsonBody(): Promise<object>
        {
            const dataListener = new Promise((resolve, reject) =>
    {
                this.incomingMessage.on('data', resolve);
            });

            return   await   dataListener.then((chunk:   any)   =>
    JSON.parse(chunk.toString()));
        }

        /**
         * Read multipart-form request body
         * @public
         * @returns Promise<object>
         */
        public async readFormBody()
        {
            const form = formidable({ multiples: true });

            const formListener = new Promise((resolve, reject) =>
    {
                form.parse(this.incomingMessage,     (err:     any,
    fields: Fields, files: Files) => {
                    if (err) {
                        reject(err);
                    } else {
                        resolve({body: fields, files: files});
                    }
                });
            });

            return await formListener.then((data: any) => data);
        }
}
```

Додаток В

Програмний код

Лістинг 1.3 – файл Response.ts

```
import { OutgoingHttpHeaders, ServerResponse } from "http"

/**
 * Class Response
 */
export default class Response
{
    /**
     * Server response
     * @public
     * @readonly
     * @type ServerResponse
     */
    public readonly serverResponse: ServerResponse;

    /**
     * Response code
     * @public
     * @type number
     */
    public statusCode: number;

    /**
     * Response headers
     * @public
     * @type object
     */
    public headers: OutgoingHttpHeaders = {};

    /**
     * Response body
     * @private
     * @type string
     */
    private body: string | null = null;

    /**
     * @public
     * @param serverResponse
     */
    public constructor(serverResponse: ServerResponse)
    {
        this.serverResponse = serverResponse;

        this.send = this.send.bind(this);
    }

    /**
```

```
         * Set response body
         * @public
         * @param body
         * @returns Response
         */
        public setBody(body: object | undefined): this
        {
            if (body) this.body = JSON.stringify(body);

            return this;
        }

        /**
         * Set response headers
         * @public
         * @param headers
         * @returns Response
         */
        public setHeaders(headers: OutgoingHttpHeaders): this
        {
            if (headers) this.headers = headers;

            return this;
        }

        /**
         * Send response
         * @public
         * @returns void
         */
        public send(): void
        {
            this.serverResponse.writeHead(this.statusCode,
    {...this.headers, ...{"Content-Type": "application/json"}});
            this.serverResponse.end(this.body);
        }
}
```

Додаток Г

Програмний код

Лістинг 1.4 – файл Dockerfile

```
FROM node:17.0.1

WORKDIR /app

COPY ./package*.json ./

RUN npm install
RUN npm install -g nodemon

COPY . .

EXPOSE 3000


CMD ["npm", "run", "start:dev"]
```

Додаток Д

Програмний код

Лістинг 1.5 – файл index.ts

```
import { createServer, IncomingMessage, ServerResponse } from
    "http";
import Request from "./http/Request";
import Response from "./http/Response";
import Router from "./src/Router";
import { routes } from "./src/routes/api";
import { createConnection } from "typeorm";
import config from "./config/ormconfig"
import ErrorResponse from "./src/responses/ErrorResponse";
import io from "socket.io"

const router = new Router(routes);

(async () => {await createConnection(config)})();

const server = createServer(async (incomingMessage:
    IncomingMessage, serverResponse: ServerResponse) => {
    const response = new Response(serverResponse);

    try {
            await router.initRoutes(new Request(incomingMessage),
    response);
    } catch (e: any) {
         return new ErrorResponse(response, 500, "Internal Server
    Error", {stack: e.stack})
    }
});

io.listen(server).sockets.on('connection', (socket) => {
    socket.on("message", (msg) => {
        await router.readSocketMessage(msg);
    });

    const socketEmitter = new EventEmitter();

    socketEmitter.on("sendMessage", message => {
        socket.json.send(message);
    });
});

server.on("listening", () => {
    console.log("🌎 Server started at http://localhost:3000");
});

server.on('error', (error: NodeJS.ErrnoException) => {
    if (error.syscall !== 'listen') {
                                            throw error;
```

```
    }

    // handle specific listen errors with friendly messages
    switch (error.code) {
        case 'EACCES':
                    console.error(`Port  3000  requires  elevated
    privileges`);
            process.exit(1);
            break;
        case 'EADDRINUSE':
            console.error(`Port 3000 is already in use`);
            process.exit(1);
            break;
        default:
            throw error;
    }
});

server.listen(3000);
```

Додаток Е

Програмний код

Лістинг 1.6 – файл tsconfig.json

```
    {
        "compilerOptions": {
          "strictPropertyInitialization": false,
          "inlineSources": true,
          "downlevelIteration": true,
          "removeComments": true,
          "experimentalDecorators": true,
          "emitDecoratorMetadata": true,
          "forceConsistentCasingInFileNames": true,
          "module": "commonjs",
          "esModuleInterop": true,
          "target": "es6",
          "moduleResolution": "node",
          "sourceMap": true,
          "outDir": "dist",
          "baseUrl": ".",
          "paths": {
            "*": [
              "node_modules/*"
            ]
          },
        },
        "include": [
          "./*.ts"
        ],
        "exclude": [
          "node_modules",
          "migrations",
        ]
  }
```

Додаток Є

Програмний код

Лістинг 1.7 – файл docker-compose.yml

```
version: '3.7'

services:
  api-server:
    container_name: api-server
    build:
      context: ./api
      dockerfile: Dockerfile
    volumes:
      - ./api:/app
      - /app/node_modules
    environment:
      POSTGRES_HOST: postgres
      POSTGRES_PORT: 5432
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: smart_house
    ports:
      - '3000:3000'
      - '30000:9229'
    restart: always
  postgres:
    container_name: postgres
    image: postgres:latest
    ports:
      - '5432:5432'
    volumes:
      - ./data/db/postgres:/var/lib/postgresql/data
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: smart_house
```