МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Дипломна магістерська робота

на тему на тему: РОЗРОБКА ІНТЕРФЕЙСУ СИСТЕМИ УПРАВЛІННЯ РОЗУМНИМ БУДИНКОМ

Виконав: студент групи МгІТ-1-20
спеціальності 122 Комп'ютерні науки
освітньої програми Комп'ютерні науки

Микита КОЛЬВА

Керівник к.т.н., доц. Тетяна АСТІСТОВА

Рецензент д.т.н., проф. Володимир ЩЕРБАНЬ

Київ 2021

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КНТ
_____проф.Щербань В.Ю.
"_____" _____2021 року

З А В Д А Н Н Я

НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Кольві Микиті Андрійовичу

1. Тема роботи *Розробка інтерфейсу системи управління розумним будинком*,
науковий керівник роботи Астістова Тетяна Іванівна, к.т.н., доцент,
затверджені наказом закладу вищої освіти від "04" 10 2021 року № 286

2. Строк подання студентом роботи 10.12.2021 р.

3. Вихідні дані до роботи Розробка кафедри комп'ютерних наук і технологій. *Система управління розумним будинком «Smart Home Systems»*

4. Зміст дипломної роботи: Вступ, Розділ 1. Стек технологій та їх поняття; Розділ 2. Архітектура додатку; Розділ 3. Програмне забезпечення;

5. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	*Тетяна АСТІСТОВА.к.т.н., доц.*		
Розділ1	*Тетяна АСТІСТОВА.к.т.н., доц*		
Розділ 2	*Тетяна АСТІСТОВА.к.т.н., доц.*		
Розділ 3	*Тетяна АСТІСТОВА.к.т.н., доц.*		
Охорона праці	*Тетяна АСТІСТОВА.к.т.н., доц.*		

6. Дата видачі завдання 10.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	05. 10.2021	
2	Розділ 1 Стек технологій та їх поняття	25.10.2021	
3	Розділ 2 Архітектура додатку	30.10.2021	
4	Розділ 3 Програмне забезпечення	10.11.2021	
5	Висновки	10.11.2021	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	15.11.2021	
7	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	*10.12.21*	
8	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	*14.12.21*	
9	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	*30.11.2021 до 17.12.21*	

Студент _____ *Микита КОЛЬВА*

Науковий керівник роботи _____ Тетяна АСТІСТОВА

Директор НМЦУПФ _____ Олена ГРИГОРЕВСЬКА

АНОТАЦІЯ

Кольва М. А. Розробка інтерфейсу системи управління розумним будинком

Дипломна магістерська робота за спеціальністю 122 - «Комп'ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2021 рік.

Дипломна робота присвячена аналізу вже реалізованих рішень та IoT технологій. Впродовж роботи було розроблено інтерфейс веб-додатоку, за допомогою якого, можна контролювати показники розумних пристроїв в рамках локальної мережі.

Для досягнення результату був розроблений інтерфейс веб-додаток за допомогою якого можна підключит різних види пристроїв та відслідковувати їх показники.

Front-end частина написана на мові програмування ReactJS з використанням state-management бібліотеки Redux для зберігання локальних та серверних даних під час користування додатком; візуалізація додатку створена за допомогою каскадної таблиці стилів CSS; Server-side частина написана на мові програмування NodeJs з використанням веб-сокетів (Websockets) для встановлення без перервної передачі даних між клієнтом та сервером; підходом до розробки була обрана методологія SDLC(Software Development Lifecycle

Дипломний проєкт демонструє взаємодію між розумними пристроями та їх контролем, що дає повне розуміння про технологію інтернетречей у сфері повсякденного життя та сучасних реалій.

Ключові слова: Система розумного будинку, IoT , веб-додаток, ReactJS , NodeJS, інтерфейс, SDLC.

ANNOTATION

MYKYTA KOLVA Creating «Smart Home Systems» Web Application for IoT Technologies - Manuscript.

Master's thesis on the specialty 122 - "Computer science". - Kyiv National University of Technology and Design, Kyiv, 2021.

Thesis is devoted to the analysis of already implemented solutions and IoT technologies. During the work, a web application was developed that allows you to control smart devices within the local network.

To achieve this, a web application has been developed that allows you to connect to a smart device.

The front-end part is written in the ReactJS programming language using the Redux state-management library to store local and server data while using the application; application visualization created using a cascading table of CSS styles; The server-side part is written in the NodeJs programming language using Websockets for installation without continuous data transfer between the client and the server; SDLC (Software Development Lifecycle) methodology was chosen as the approach to development

The thesis project demonstrates the interaction between smart devices and their control, which gives a full understanding of IoT technology in everyday life and modern realities.

Keywords: Smart Home System, IoT, Internet of Things, Website, Web Application, ReactJS, NodeJS, SDLC.

ЗМІСТ

ВСТУП

Актуальність. Актуальність веб-додатку «Smart Home Systems» полягається в тому що, в новітньому суспільстві все частіше і частіше розвиваються технології розумного будинку і постає питання полегшення використання побутових речей користувачами.

Мета і завдання. Щодня з'являється все більше і більше гаджетів, здатних зробити управління будинком простіше і комфортніше, тому будинки з великою кількістю «розумної» техніки стали називати «розумними». Розумний будинок - це система датчиків і техніки, об'єднаних в єдину систему і які підтримують управління та налаштування шз пристрою, яким зручно користуватися людині у своєму побуті, наприклад, це може бути смартфон, лептоп та ін. Метою цієї роботи є розробка інтерфейсу системи розумного будинку під будь-які типи датчиків. Завдання – розробити програмне забезпечення, за допомогою якого можна підключати датчики різних видів та типів, таких як: датчик температури, датчик вологості, датчики пересування, датчик поливу та інше.

Об’єкт та предмет дослідження. Об’єктом даного дослідження є розробка програмного забезпечення для відтворення температурних показників приміщення та їх контролю.

Безліч домашніх пристроїв, здатних виконувати дії і вирішувати певні повсякденні завдання об’єднуються в єдину систему управління, що однозначно раціоналізує життя людини. За допомогою системи розумного будинку, посилюється контроль власного приміщення та більш детальне розуміння кліматичних показників.

Методи та засоби дослідження. Для візуалізації програмного забезпечення був розроблений інтерфейс веб-додатоку «Система Розумного Будинку».

Технології для реалізації: Front-end частина написана на мові програмування ReactJS з використанням state-management бібліотеки Redux

для зберігання локальних та серверних даних під час користування додатком, візуалізація додатку створена за допомогою каскадної таблиці стилів SCSS, Server-side частина написана на мові програмування NodeJs з використанням веб-сокетів(Websockets) для встановлення без перервної передачі даних між клієнтом та сервером, підходом до розробки була обрана методологія SDLC(Software Development Lifecycle) – за допомогою цього життєвого циклу можна розроблювати додатки на різних мовах програмування.

Результат дослідження. Результатом дослідження є розробка інтерфейсу веб-додаток у«Smart Home Systems», який відоброжає температурні показники підключених пристроїв до локального середовища та здійсненються контроль над ними. За допомогою сучасних технологій Front-end розробка перейшла на інший рівень, якщо раніше робились одно-сторінкові сайти та веб-магазини, вже зараз можна розроблювати цілі веб-додатки які за своєю реактивністю та ростом технологій будуть працювати все швидше і швидше, саме тому була обрана мова програмування ReactJS, одна з лідируючих мов у Front-end розробці яка оснащена не тільки своєю реактивністю, а й ще захистом від XSS атак.

Наукова новизна. Наукова новизна, полягається в тому що, в сучасному світі, технології розумного будинку ще не достатньо розвинуті та з кожним роком набирають більш великіх обротів.

Теоретична цінність. Теоретичною цінність даної роботи полягає в тому, що був проведений аналіз аналогів даного продукту на ринку та було прийнято рішення створити власну систему контролю та моніторингу температурних показників. Яка є унікальною розробку та має переваги у простоті використання, зрозумілому інтерфейсу та швидкісній та оптимізованій роботі, завдяки використанням сучасних технологій.

Практична цінність. Практичною цінністю є розробка комплексної програмі для моніторингу та контролю температурних показників за

допомогою інтерфейсу до веб-додатку, який має інтуїтивно простий інтерфейс з використанням сучасних технологій в розробці.

Апробація результатів роботи. Результити

Цей проєкт зайняв друге місце на конкурсі стартапів КНУТД .

Результати роботи були апробовані на міжнародній конференції та в статті.

1. Astistova T.I, Smart house management system, user interface/ T.I. Astistova, M.A. Kolva //Тези V Міжнародної науково-практичної конференції «Мехатронні системи: інновації та інжиніринг – «MSIE-2021» К. КНУТД , 4 листопада 2021р. - С.156-157

2. Астістова Т. І., Кольва М. А., Розробка інтерфейсу системи управління розумним будинком / Т.І. Астістова , М.А.Кольва // Інформаційні технології в науці, виробництві та підприємництві: зб. наук. праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій. – К. : Освіта України, 2021 р. – С. 212 - 214

Розділ 1 СТЕК ТЕХНОЛОГІЙ ТА ЇХ ПОНЯТТЯ

1.1 Software Development Lifecycle

Одним із найважливіших процесів розробки додатку, це правильно обрана методологія розробки та її життєвий цикл. Методологія SDLC поділяється на 6 типів: Agile, Lean, Waterfall, Iterative, Spiral, DevOps. Мною був обраний тип Agile.

Agile методологія дозволяє уникнути ризику витрачати місяці чи роки на процес, який в кінцевому підсумку зазнає невдачі через якусь невелику помилку на ранній стадії. Натомість він покладається на довіру співробітників і команд, які працюють безпосередньо з клієнтами, щоб зрозуміти цілі та надати рішення швидким і поетапним способом.

Швидше, менше. Традиційна розробка програмного забезпечення покладалася на такі етапи, як визначення вимог, планування, проектування, створення, тестування та доставка. Agile методологія, навпаки, передбачає розгортання першого кроку за пару тижнів, а всього програмного забезпечення — за пару місяців.

Спілкування. Гнучкі команди в бізнесі щодня працюють разом на кожному етапі проекту шляхом особистих зустрічей. Така співпраця та спілкування забезпечують стабільність процесу, навіть якщо умови змінюються.

Зворотній зв'язок. Замість того, щоб чекати фази доставки, щоб оцінити успіх, команди, які використовують методологію Agile, регулярно відстежують успіх і швидкість процесу розробки. Швидкість вимірюється після доставки кожного кроку.

Довіра. Agile команди та співробітники самоорганізуються. Замість того, щоб слідувати маніфесту правил від керівництва, призначених для досягнення бажаного результату, вони розуміють цілі та створюють власний шлях для їх досягнення.

Налаштуйте. Учасники постійно налаштовують і коригують процес, дотримуючись принципу KIS або Keep It Simple.

Agile SDLC поділяється на 6 типів розробки

- Аналіз та збір вимог
- Розробка дизайну
- Кодування або впровадження
- Тестування
- Розгортання
- Технічна підтримка

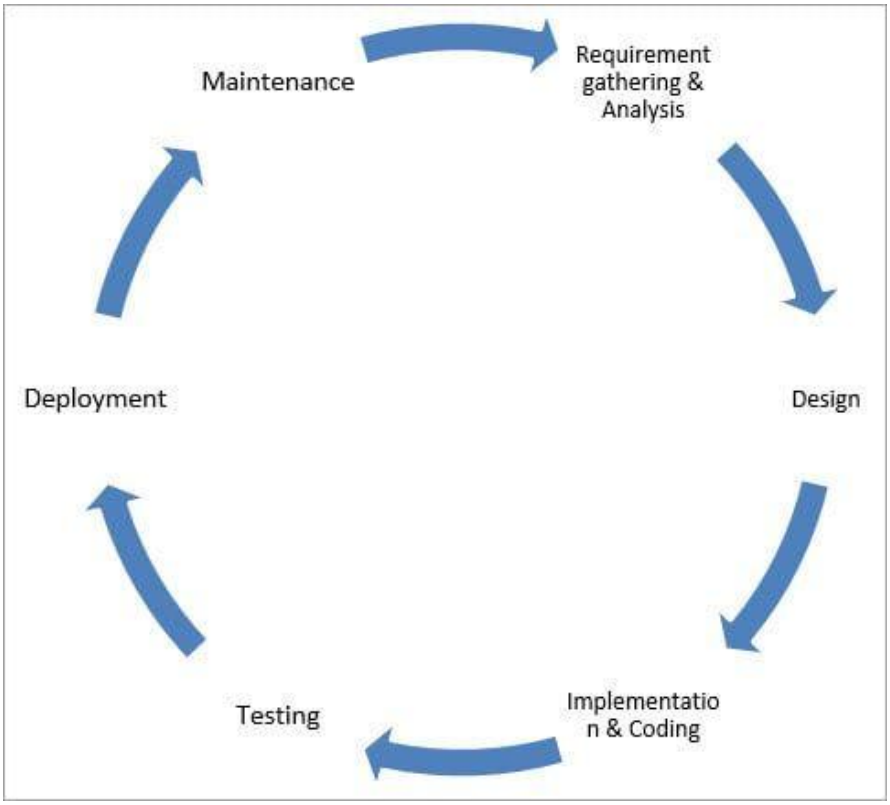

Рис.1.1- Технологія Agile SDLC

Аналіз та збір вимог - на цьому етапі від клієнта збирається вся відповідна інформація для розробки продукту відповідно до його очікувань. Будь-які неясності повинні бути вирішені лише на цьому етапі.

Бізнес-аналітик та менеджер проекту організували зустріч із замовником, щоб зібрати всю інформацію, наприклад, що замовник хоче створити, хто буде кінцевим споживачем, яка мета продукту. Перш ніж створювати продукт, дуже важливо розуміння або знання продукту.

Наприклад**,** Клієнт хоче мати додаток, який передбачає грошові операції. У цьому випадку вимога повинна бути чіткою, наприклад, який вид операцій буде здійснено, як це буде зроблено, в якій валюті це буде зроблено тощо.

Після завершення збору вимог проводиться аналіз для перевірки доцільності розробки продукту. У разі виникнення будь-якої неясності проводиться дзвінок для подальшого обговорення.

Після чіткого розуміння вимоги створюється документ SRS (Software Requirement Specification). Цей документ повинен бути добре зрозумілий розробниками, а також повинен бути переглянутий замовником для подальшого використання.

Розробка дизайну - на цьому етапі вимога, зібрана в документі SRS, використовується як вхідна інформація та виводиться архітектура програмного забезпечення, що використовується для реалізації розробки системи.

Кодування або впровадження - починається після того, як розробник отримає проектний документ. Дизайн програмного забезпечення переведений у вихідний код. На цьому етапі реалізовано всі компоненти програмного забезпечення.

Тестування - починається після завершення кодування та випуску модулів для тестування. На цьому етапі розроблене програмне забезпечення ретельно перевіряється, і всі виявлені дефекти приписуються розробникам для їх усунення. Повторне тестування, регресійне тестування проводиться до

моменту, коли програмне забезпечення відповідає очікуванням замовника. Тестери направляють документ SRS, щоб переконатися, що програмне забезпечення відповідає стандартам замовника.

Розгортання - після того, як продукт тестується, його застосовують у виробничому середовищі або спочатку UAT (Тест прийняття користувача) робиться залежно від очікувань замовника.

У випадку UAT створюється копія виробничого середовища, і замовник разом із розробниками проводить тестування. Якщо замовник знаходить заявку, як очікувалося, тоді клієнт надає підписку для запуску.

Технічна підтримка - після розгортання продукту на виробничому середовищі розробники піклуються про технічне обслуговування продукту, тобто якщо виникає якась проблема, яка потребує виправлення або потрібно зробити будь-яке вдосконалення.

1.2 Визначення Інтернету речей

Інтернет речей (англ. *internet of things, IoT*) — концепція мережі передавання даних між фізичними об’єктами (*«речами»*) — це не тільки виконавчий холодильник, який сам замовляє улюблену їжу господаря, або послужливий чайник, який кип'ятить воду на першу вимогу зі смартфона. Це розумні датчики на полях, дрони з камерами, завдяки яким можна віддалено моніторити стан ґрунтів, це датчики у громадському транспорті та єдині системи для моніторингу життя міста. Інакше кажучи, вже за кілька років інтернетом речей стане світ навколо нас.

Концепція сформульована у 1999 році як осмислення перспектив широкого застосування засобів радіочастотної ідентифікації для взаємодії фізичних предметів між собою та із зовнішнім оточенням. Наповнення концепції різноманітним технологічним змістом та впровадження практичних рішень для її реалізації починаючи з 2010-х років вважається стійкою тенденцією в інформаційних технологіях, насамперед завдяки повсюдному поширенню бездротових мереж, появі хмарних обчислень,

розвитку технологій міжмашинної взаємодії, початку активного переходу на IPv6 та освоєння програмно-визначуваних мереж.

Пристрої інтернету речей є частиною ширшої концепції домашньої автоматизації, яка може включати освітлення, опалення та кондиціювання повітря, медіа-системи та системи безпеки, а також системи відеоспостереження. Довгострокові вигоди можуть включати економію енергії за рахунок автоматичного відключення світла та електроніки або за рахунок інформування мешканців будинку про їх використання.

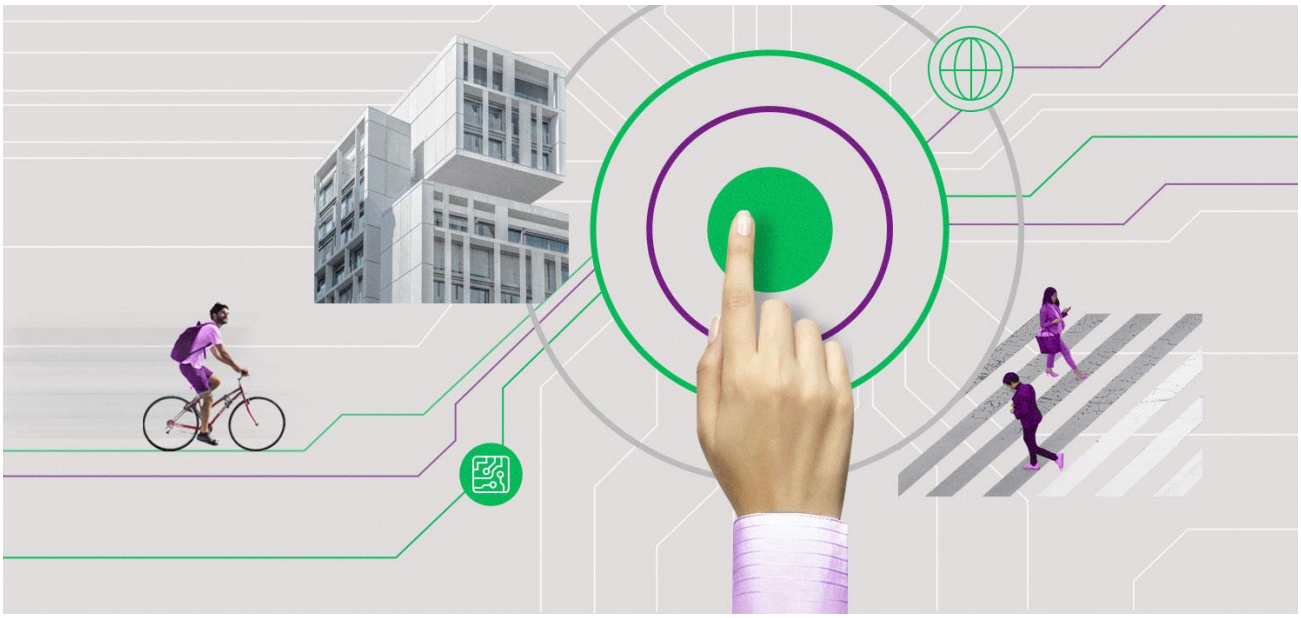

Рис. 1.2. Інновації

У розвинених країнах IoT вже давно не є інновацією, перейшовши до розряду повсякденних завдань, які вирішуються за допомогою пристроїв та алгоритмів. Прориви зосереджені насамперед у галузях, де IoT та алгоритмізація можуть принести ефект миттєвої економії та швидкої монетизації, а сама галузь є великою за розмірами. Наприклад, сільське господарство чи транспорт та логістика «кишать» новаціями, що вражають розум обивателя. При цьому сьогодні вже стоїть питання не створення новації, а її імплементації у великих масштабах. Багато з того, що необхідно для повсюдної зміни усталених операцій та дій, вже є, але потрібно інтегрувати та почати застосовувати ці технології.

Статистична довідка повсякденного життя: середня кількість смартфонів на душу населення останніми роками становила приблизно 2,53 одиниці. Це пристрій, який переріс своє використання тільки для Інтернеті та отримання дзвінків у частину життя людини, як незамінний помічник. Цей пристрій робить повний структурний аналіз щосекунди за рахунок збирання інформації приблизно двадцятьма датчиками одночасно.

Найпоширеніший спосіб «зв'язати» пристрої – за допомогою стільникової мережі. Сучасні рішення хоч і дають змогу використовувати головні функції інтернету речей (наприклад, віддалений доступ до пристроїв), але не розкривають його потенціал. Великий «зазор» між запитом користувача та відповіддю пристрою унеможливлює застосування нових технологій, наприклад, у медицині, де від швидкості прийняття рішень залежать життя людей, а також в енергетиці, на промислових підприємствах (технологія «цифрового двійника» дозволяє створити віртуальну копію об'єкта та повноцінно керувати ним з будь-якої точки світу), у сільському господарстві, сфері ЖКГ, пасажирських та вантажних перевезеннях.

Виходячи з логічних міркувань, поняття Інтернет речей можна розділити на зовнішній та внутрішній. Так до зовнішнього поняття можна віднести глобальні сфери його розповсюдження, наприклад, орієнтування на покращення досвіду клієнтів та інше. До внутрішнього поняття підходить саме внутрішня організація виробничих процесів, інтеграція у систему виробництва продукції або послуг за рахунок раціоналізації їх систем.

Найгарячіша тема 2019-2021 років – мережі нового покоління 5G, про них буквально складають легенди. В інфраструктурі IoT нові мережі допоможуть вирішити проблему тривалого відгуку пристроїв. Але розібратися із проблемою «зазору» між запитом користувача та відгуком пристрою може не тільки розгортання мереж 5G, а й існуюча інфраструктура.

Щороку удосконалюються мікропроцесорні технології, процесори стають дедалі потужнішими але меншими за розміром та дешевшими по вартості. Завдяки дротовому або бездротовому інтернету отримується доступ до інтерфейсу, що дає змогу надсилати чи отримувати дані або команди через мікропроцесорні пристрої. Найчастіше використовуються такі стандарти як WLAN, Bluetooth або мобільних пристроїв, такі як UMTS та LTE. У сукупності основу Інтернету речей складають саме інтернет та мікропроцесорні технології.

Рис. 1.3. Інтернет речей

Наразі у приватному секторі для розумного будинку та автоматизації будівель використовується технологія де типові програми включають управління освітленням, сигналізацією та дистанційним моніторингом, автоматичним опаленням і клімат-контролем або розумним вимірюванням тощо. Все це є численними додатками Інтернету речей.

Пристрої Інтернету речей можна використовувати для забезпечення віддаленого моніторингу стану здоров'я та систем оповіщення про

надзвичайні ситуації. Ці пристрої для моніторингу стану здоров'я можуть змінюватись від моніторів артеріального тиску та серцевого ритму до сучасних пристроїв, здатних контролювати спеціалізовані імплантати, такі як кардіостимулятори, електронні браслети Fitbit або вдосконалені слухові апарати. Деякі лікарні почали впроваджувати "розумні ліжка", які можуть визначати, коли вони зайняті та коли пацієнт намагається встати. Він також може самостійно налаштовуватись.

За даними оцінок деяких компаній, доходи глобального ринку промислового Інтернету речей досягнуть 484 млрд євро в 2025 році, основними сферами застосування технології стануть ЖКГ, охорона здоров'я, промисловість, ЖКГ, технології «Розумного дому». Прогнозується зростання загального обсягу корпоративного та користувальницького ринку інтернету речей до 4,3 трлн дол.

1.3 Основні поняття про ReactJS

ReactJS - бібліотека JavaScript, розроблена для створення інтерфейсів користувача. З її допомогою розробляються високореактивні та стабільно працюючі сторінки/веб-сайти. З нею можна забути про традиційні веб-сторінки, які змушують вас гаяти час через затримку при переході на наступну сторінку. "Блискавичні" веб-сайти, розроблені за допомогою ReactJS, дозволяють уникнути цього циклу та оновлювати контент на тій самій сторінці в момент походження події.

React може використовуватися для розробки односторінкових та мобільних додатків. Його мета — надати високу швидкість, простоту та масштабованість. Як бібліотека для розробки інтерфейсів користувача React часто використовується з іншими бібліотеками, такими як MobX, Redux і GraphQL.

Особливості. Однонаправлена передача даних

Властивості передаються від батьківських компонентів до дочірніх. Компоненти отримують властивості як безліч незмінних (англ. immutable)

значень, тому компонент не може безпосередньо змінювати властивості, але може викликати зміни через callback-функції. Такий механізм називають «властивості вниз, події нагору».

Віртуальний DOM. React використовує віртуальний DOM (англ. virtual DOM). React створює кеш-структуру в пам'яті, що дозволяє обчислювати різницю між попереднім та поточним станом інтерфейсу для оптимального оновлення DOM браузера. Таким чином, програміст може працювати зі сторінкою, вважаючи, що вона оновлюється вся, але бібліотека самостійно вирішує, які компоненти сторінки необхідно оновити.

JSX (JavaScript XML (JSX) - це розширення синтаксису JavaScript, яке дозволяє використовувати HTML-подібний синтаксис для опису структури інтерфейсу. Як правило, компоненти написані з використанням JSX, але є можливість використання звичайного JavaScript. JSX нагадує іншу мову, створену в компанії Фейсбук для розширення PHP, XHP.

Ось як ви могли б зробити це з React:

```
render: function() {
   return <header>
      {this.state.name? <div>this.state.name</div> : null }
   </header>;
}
```

Ми можемо відразу сказати, як компонент буде відресований. Якщо ви знаєте стан, ви знаєте результат відресовування. Не потрібно простежувати хід виконання програми. Коли розробляється складна програма, особливо в команді, це дуже важливо.

2. Зв'язування JavaScript та HTML у JSX робить компоненти простими для розуміння. Дивне поєднання HTML/JavaScript може вас збентежити. Нас вчили не вставляти JavaScript у DOM (наприклад: обробники OnClick), ще в той час, коли ми були «маленькими» розробниками (ор: since we were wee developers).

Зазвичай поділяється уявлення (HTML) та функціональність (JavsScript). Це призводить до монолітного JavaScript файлу, що містить всю функціональність для однієї сторінки, і ви повинні стежити за складним потоком JS->HTML->JS->неприємна ситуація.

Зв'язування функціональності безпосередньо з розміткою та упаковка цього в портативний, автономний «компонент», зробить код загалом кращим. Javasacript «добре знайомий» з HTML, так що змішувати їх має сенс.

Мінуси.

Не забувайте, що React це лише уявлення.

1. Ви не отримаєте систему подій (відмінну від нативних DOM подій), роботу з AJAX, тощо.

2. Невдало подано документацію.

Якщо дивитися на першу частину бічної панелі, можна побачити три окремі, що конкурують туторіали для початківців.

3. React досить великий, включаючи кросбраузерну підтримку (без бібліотеки в reacat-with-addons, яка буде потрібна для розробки реальної програми, без бібліотеки ES5-Shim, необхідної для підтримки IE8).

Попри деякі незручності залишаються такі переваги:

правильна крива навчання.

Крива навчання - це важливий фактор, який потрібно враховувати при виборі UI-фреймворку. У цьому слід зазначити, що у React є менше абстракцій, ніж у Angular. Якщо ви знаєте JavaScript, то, можливо, ви зможете навчитися писати React-додатки буквально за день. Звичайно, для того, щоб навчитися робити це правильно, потрібно деякий час, але приступити до роботи можна дуже і дуже швидко.

Якщо ж проаналізувати Angular, то виявиться, що якщо ви вирішите освоїти цей фреймворк, вам доведеться вивчити нову мову (Angular

використовує TypeScript), а також навчитися використовувати засоби командного рядка Angular та звикнути до роботи з директивами;

особливості механізму прив'язування даних.

Angular має систему двосторонньої прив'язки даних. Це, наприклад, виявляється у тому, що зміни у формі елемента призводять до автоматичного оновлення стану програми. Це ускладнює налагодження та є великим мінусом цього фреймворку. При такому підході, якщо щось йде не так, програміст не може точно знати про те, що саме стало причиною зміни стану програми.

У React, з іншого боку, використовується одностороння прив'язка даних. Це великий плюс цієї бібліотеки, оскільки виявляється це в тому, що програміст завжди точно знає про те, що призвело до зміни стану програми. Подібний підхід до прив'язування даних значно спрощує налагодження програм;

функціональний підхід до розробки.

Однією з найсильніших сторін React є те, що ця бібліотека не примушує розробника до використання класів. У Angular всі компоненти повинні бути реалізовані у вигляді класів. Це призводить до надмірного ускладнення коду, не даючи жодних переваг.

У React всі компоненти інтерфейсу користувача можуть бути виражені у вигляді наборів чистих функцій.

1.4 Основні поняття про Redux.

Redux — відкрита JS бібліотека призначена для управління станом програм JavaScript. Найчастіше використовується разом з React або Angular для побудови інтерфейсів користувача.

Redux — це контейнер станів для застосунків JavaScript. Він допомагає розробникам оптимізувати код програми. Крім того, він забезпечує вдосконалення досвіду розробника, наприклад, редагування живого коду в поєднанні з відладчиком, що працює під час роботи.

Redux можна використовувати разом із React або іншими бібліотеками. Розмір файлу Redux невеликий, 2kB, включаючи залежності.

Redux зберігає стан всього застосунку в дереві об’єктів в одному сховищі. Одне дерево станів полегшує налагодження або перевірку програми; це також дозволяє зберігати стан вашого застосунку в процесі розробки, для прискорення циклу розробки.

Стор (Store) - це об'єкт, який з'єднує екшени, які представляють факт того, що "щось сталося" і редюсери, які оновлюють стан (state) відповідно до цих екшинів разом.

- Стор містить стан додатку (application state);
- надає доступ до стану за допомогою getState()
- може випускати оновлення стану за допомогою dispatch(action)
- обробляє скасування реєстрації слухачів за допомогою функції, що повертається subscribe (listener).

Стан призначений тільки для читання

Єдиний спосіб змінити стан — це виокремити дію, об’єкт, що описує те, що сталося. Це гарантує, що ні перегляди, ні зворотні виклики мережі ніколи не будуть змінювати стан. Натомість вони виражають тільки намір це зробити. Всі зміни — централізовані і відбуваються одна за іншою у чіткій послідовності.

Оскільки дії є простими об’єктами, вони можуть бути зареєстровані, серіалізовані, збережені та пізніше відтворені для налагодження або тестування.

Екшени - це структури, які передають дані з застосування в стан. Вони є єдиними джерелами інформації для стану. Redux відправляє їх їх в стор, використовуючи метод store.dispatch().

Зміни за допомогою чистих функцій

Редьюсери (redusers) — це лише чисті функції, які приймають попередній стан і дію тобто state, і повертають наступний стан. Редюсери

визначають, як стан додатки змінюється у відповідь на екшени (action), які відправлені в стан. Екшени тільки описують, що сталося, але не описують, як змінюється стан додатка.

В процесі розробки редьюсери можуть бути розділені на дрібніші редьюсери, які управляють певними частинами дерева станів. Оскільки редьюсери — це лише функції, ви можете контролювати порядок їх надсилання, передавати додаткові дані або навіть створювати повторювані редьюсери для звичайних завдань, таких як розбиття на сторінки.

Нижче наведено декілька причин, коли слід розглянути.

Стан кешування сторінки - коли користувач переглядає сторінку, а потім, коли переходить на іншу сторінку і повертається назад, очікується, що сторінка буде в тому ж стані. Оскільки редуктори ініціалізуються та живуть протягом усього сеансу, вони можуть зберігати стан сторінки.

Управління станом компонента - Redux використовується, коли нам доводиться керувати станом компонентів.

Глобальні компоненти легко доступні - Вони мають термін служби додатків, що дозволяє закусочних, сповіщень, підказок тощо. Redux має надзвичайно важливе значення для створення дій для відправки команд. Як приклад, якщо код генерує запит, який є асинхронним, він створює дію снек-бар, коли запит не відповідає відносно бекенда. У тих випадках, коли користувач не використовує Redux, йому потрібна інша система подій або ж йому потрібно інстанціювати компонент закусочної, коли б він не використовувався.

Якщо є численні реквізити, пов’язані з висококласним компонентом, з якого використовується лише декілька з них, то їх можна вважати рефактором за допомогою Redux.

Це в основному відбувається в компонентах обгортки, які не потребують великої кількості даних або конфігурації. Таким чином, в таких

випадках дуже важливо вводити бічну ланцюг Redux в компонент нижчого рівня.

Цей самий фрагмент стану програми потрібно відобразити на декілька компонентів контейнера. Redux - це зручний та найкращий спосіб поділитися станом.

Переваги Redux

Центральний магазин - За допомогою скорочення будь-який компонент може отримати доступ до будь-якого стану з магазину. Він також зберігає стан події компонента після відключення компонента.

Коли стан змінюється, він повертає новий стан і запобігає непотрібним повторним рендерингам.

Це виграє при тестуванні, оскільки воно розділяє інтерфейс користувача та управління даними.

Зберігається історія держави, що допомагає в застосуванні таких функцій, як скасування.

Redux полегшує налагодження програми. За допомогою редукса легко зрозуміти мережеві помилки, помилки кодування та інші форми помилок.

Організовані коди дозволяють професіоналам зрозуміти структуру різних додатків Redux. Це, у свою чергу, робить його простою у користуванні бібліотекою JavaScript з відкритим кодом.

1.5 Основні поняття про ScSS

Sass (англ. *Syntactically Awesome Stylesheets*) — скриптова метамова, яка інтерпретується в каскадні таблиці стилів (CSS). Sass призначений для підвищення рівня абстракції коду та спрощення файлів CSS.

Мова Sass має два синтаксиси:

- **sass** (оригінальний) — відрізняється відсутністю фігурних дужок, в ньому вкладені елементи реалізовані за допомогою відступів, а правила відокремлюються переведенням рядка;
- **scss** (новий) — використовує фігурні дужки (подібно до CSS).

Файли **sass**-синтаксису мають розширення .sass, **scss**-синтаксису — .scss.

Sass розширює CSS, надаючи кілька механізмів, доступних в більш традиційних мовах програмування, зокрема об'єктно-орієнтованих мовах, але недоступних для CSS. Інтерпретатор Sass транслює SassScript у блоки правил CSS. По суті, Sass — це синтаксичний цукор для CSS.

Що таке Sass і навіщо він потрібен

Всі, хто стикається з CSS розміром більше 500 рядків, мають справу зі складнощами на тему того, як би його спростити. На жаль, з часів розробки стандартів каскадних стилів їх структура кардинально не змінювалася. Вимоги до верстки — ускладнилися в рази. Колись 50 -70 рядків стилів могли оформити простий сайт, сьогодні цього замало.

2007 року з'явилася перша версія SASS, простий набір правил і примітивний компілятор замислювалися тоді як одні з основних інструментів фронтенд-майстрів і верстальників сучасного інтернету.

Оскільки розширення SASS-файлів можуть бути .sass і .scss, що залежить від обраного синтаксису, але при цьому браузер не розуміє жодного з них, тому для взаєморозуміння потрібно використовувати компілятор. Його завдання – привести SASS в зрозумілий класичний CSS, який буде розпізнано будь-яким браузером.

Роль компілятора може виконувати серверний js або програма, встановлена у вас на робочій машині і моніторять зміни в робочих файлах.

Які бувають синтаксиси в Sass

У мови є два основних «діалекти»: SASS і новіший SCSS. Відмінності між ними невеликі, проте порушення правил синтаксису не дозволить скомпілювати файл. У SASS-синтаксисі немає фігурних дужок, вкладеність елементів в ньому реалізована за допомогою відступів, а стильові правила обов'язково відокремлені новими рядками.

Незалежно від синтаксису, SCSS назад сумісний з CSS. Тобто будь-який CSS обов’язково буде дійсним SCSS-кодом.

Через відсутність дужок і крапок з комою зворотної сумісності у SASS-синтаксису з CSS немає.

Змінні (variables)

Sass дозволяє призначати змінні — і це одна з ключових переваг. Змінна, за аналогією з php, починається зі знака долара ($), значення присвоюються за допомогою двокрапки.

Змінні в Sass можна розділити на 4 типи:

1. Число (int)
2. Рядок (string)
3. Логічний тип (так/ні, boolean)
4. Кольори (ім’я, імена)

Можливе написання доповнень (mixin)

Правило «Do not Repeat Yourself» (або скороченно DRY) реалізовано в Sass за допомогою техніки mixin. Частини коду, які в CSS зазвичай вам доводилося дублювати, тут можна зберегти в окремій змінної і вставляти в потрібних місцях. Компілятор, зустрівши таку змінну, збереже замість неї потрібну частину коду.

Що таке наслідування (extend)

Наслідуваний елемент отримає всі властивості вихідного класу, які ми можемо доповнити будь-якими іншими. Тому створивши одного разу будь-яке правило, ми можемо використовувати його всередині іншого.

Ці прості можливості збільшують швидкість верстки і не дають загубитися в купі коду. У всякому разі, мені. Корисно запам’ятати, що вся документація по SASS є на офсайті мови з прикладами і докладним описом.

Що таке компілятори та їх типи

Програми-компілятори перевіряють ваші .scss і .sass-файли на зміни і автоматично компілюють з них готові стилі.

Є певний перелік типів компеляторів, які радять розробники, а саме:

- CodeKit (платний, Mac)
- Compass.app (платний, Win/Mac)
- Hammer (платний, Mac)
- Koala (безкоштовний, Win/Mac) — читайте обзор на Koala
- LiveReload (платний, Win/Mac)
- Mixture (безкоштовний, Win/Mac)
- Prepros (платний, Win/Mac)
- Scout (безкоштовний, Win/Mac)

Як висновок можна резюмувати, що при необхідності заощадити час і сили, доцільно використовувати Sass. Це зручне і просте рішення для прискорення розробки сайтів.

1.6 Основні поняття про Node.JS

Node.js — це серверна платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків мовою JavaScript.

Платформа *Node.js* перетворила JavaScript на мову загального використання з великою спільнотою розробників. Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то *node.js* створено для створення масштабованих програм, керованих подіями та надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання.

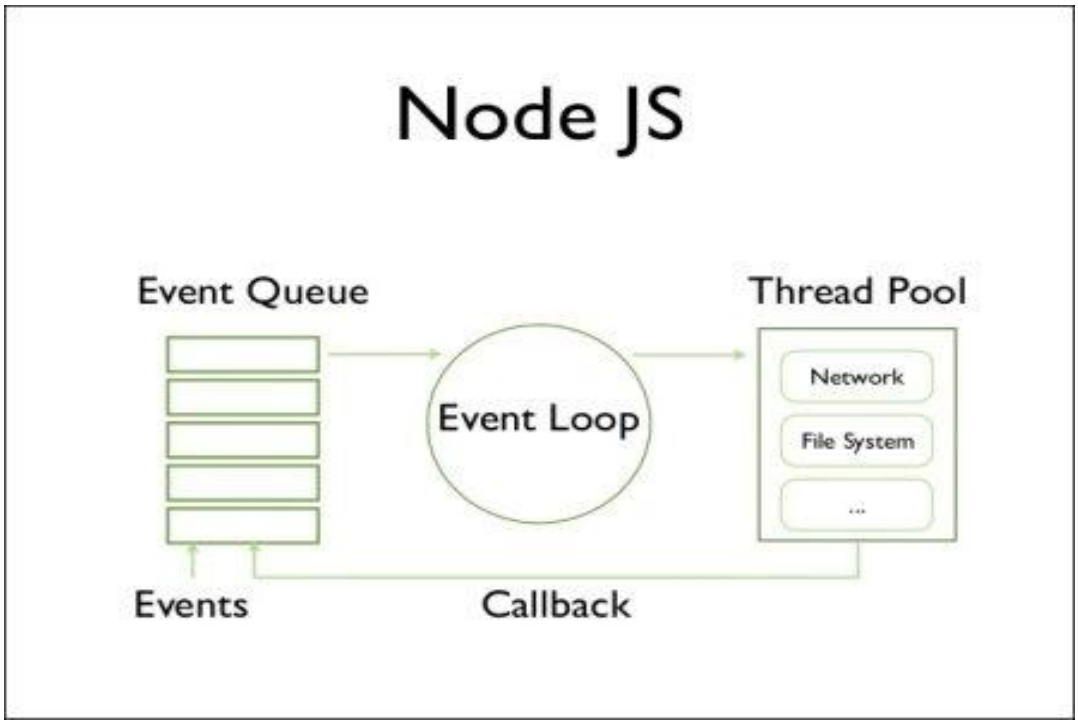

Рис. 1.4 Схема роботи платформи Node.js

Node.js має наступні властивості:

- асинхронна одно-нитева модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8;

Для керування модулями використовується пакетний менеджер npm (node package manager).

Платформа *Node.js* призначена для виконання високопродуктивних мережевих застосунків, написаних мовою програмування JavaScript. Платформа окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських та серверних програм.

Node.js стає особливо привабливим завдяки цим двом ознакам:

перша пов'язана з усіма накладними витратами та упаковкою, які використовуються існуючими технологіями для розмови в Інтернеті;

друга - Node.js пов'язаний з моделлю подій веб-програмування. Більшість існуючих технологій написані для отримання «великих прогалин»

даних для кожного запиту та відповіді. В платформі використовується розроблений компанією Google рушій V8.

Асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback), використовується для забезпечення обробки великої кількості паралельних запитів у Node.js. Epoll, kqueue, /dev/poll і select підтримується як способи мультиплексування з'єднань. Бібліотека libuv, для створення пулу нитей (thread pool) задіяна бібліотека libeio, для виконання DNS-запитів у неблокуючому режимі інтегрований c-ares використовується для мультиплексування з'єднань. Всі системні виклики, що спричиняють блокування, виконуються всередині пулу нитей і потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe).

Перевага використання Node.js полягає в тому, що програмісти можуть використовувати свої поточні знання програмування і починати кодування з Node.js оскільки це набагато легше.

1.7 Основні поняття про react-native

React Native (рис. 1.5) — це фреймворк, який дозволяє створювати нативні мобільні додатки за допомогою JavaScript. Зазвичай вам потрібно запрограмувати свій мобільний додаток за допомогою Java (для Android) і Swift/Obj-C (для iOS).

React Native усуває цю вимогу, що призводить до створення повністю функціональних програм на обох платформах за набагато менший час і з використанням лише однієї мови кодування.

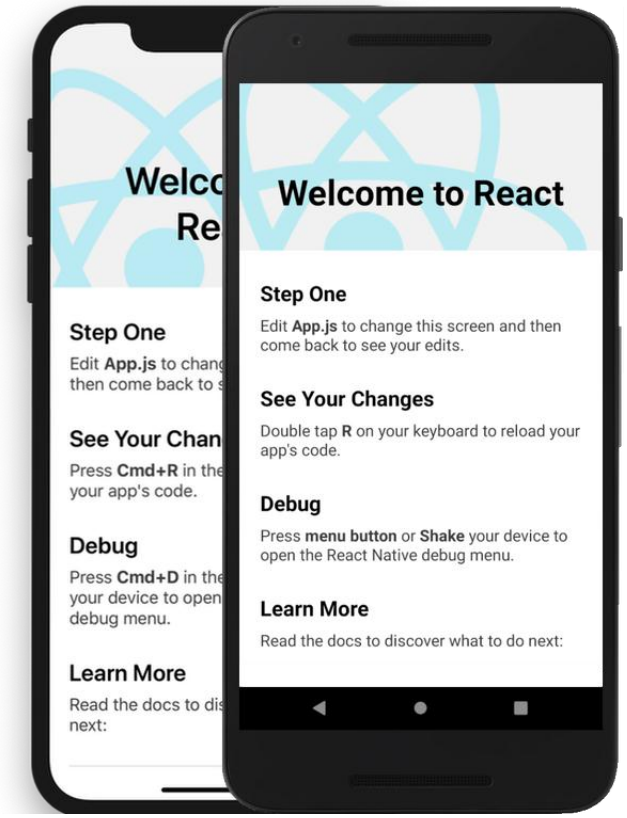

Рис. 1.5 React native

Facebook (рис. 1.6) є компанією, яка стоїть за ReactJS і React Native. Насправді, Facebook вперше створив React. Після подальшої розробки Facebook випустив ReactJS для Інтернету як відкритий вихідний код.

Але Facebook все ще боровся з їхнім мобільним додатком. Їм потрібно було підтримувати дві кодові бази: одну для iOS, одну для Android. Функції, реалізовані в Swift на iOS, довелося реалізувати окремо в Java на Android, що призвело до дублювання роботи та асиметричних програм.

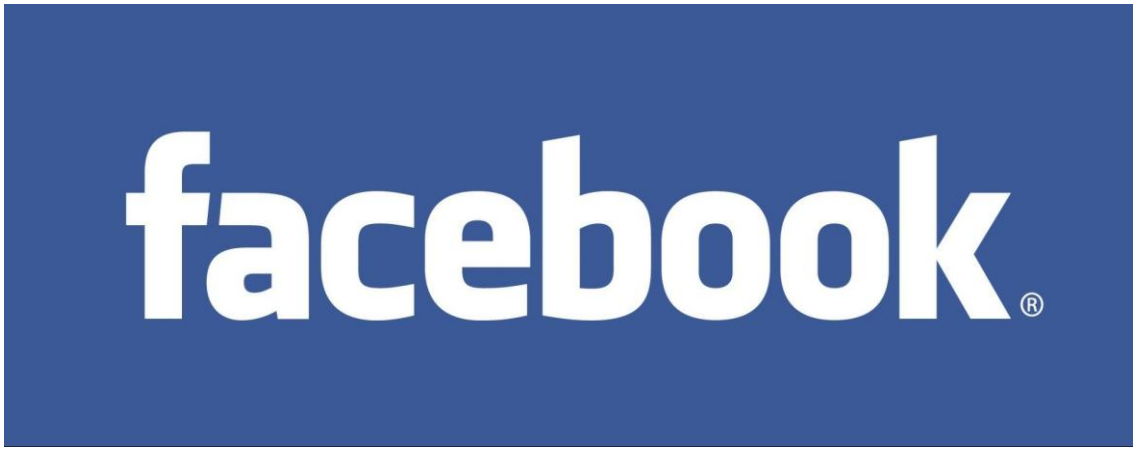

Рис. 1.6 Facebook

React Native акуратно вирішує цю проблему. Наслідуючи ReactJS, мета React Native полягала в тому, щоб полегшити створення мобільних додатків. Все просто: якщо ви можете один раз закодувати програму на JavaScript і розгорнути її як на Android, так і на iOS, ваше життя стане набагато простіше.

Якщо ви коли-небудь використовували офіційний додаток Facebook на Android або iOS, ви бачили React Native в дії. (Те саме стосується мобільного додатка Airbnb.)

Довгий час React Native вважався комерційно нежиттєздатним. Він був недостатньо розроблений або підтримуваний, щоб створювати «нативні» програми. React Native набирає популярності, завойовує підтримку спільноти та завойовує все більшу частку ринку. Писати чудові програми за допомогою React Native стає все легше й легше — і світ помічає.

За допомогою React Native ви створюєте одну кодову базу, яка працює як на Android, так і на iOS. І він не просто «працює» — він компілюється до рідного коду Java (рис 1.7) та Swift (рис 1.8). Зокрема, React Native створює міст між компонентами веб-інтерфейсу та їхніми рідними аналогами Java/Swift.

Рис. 1.7 Swift

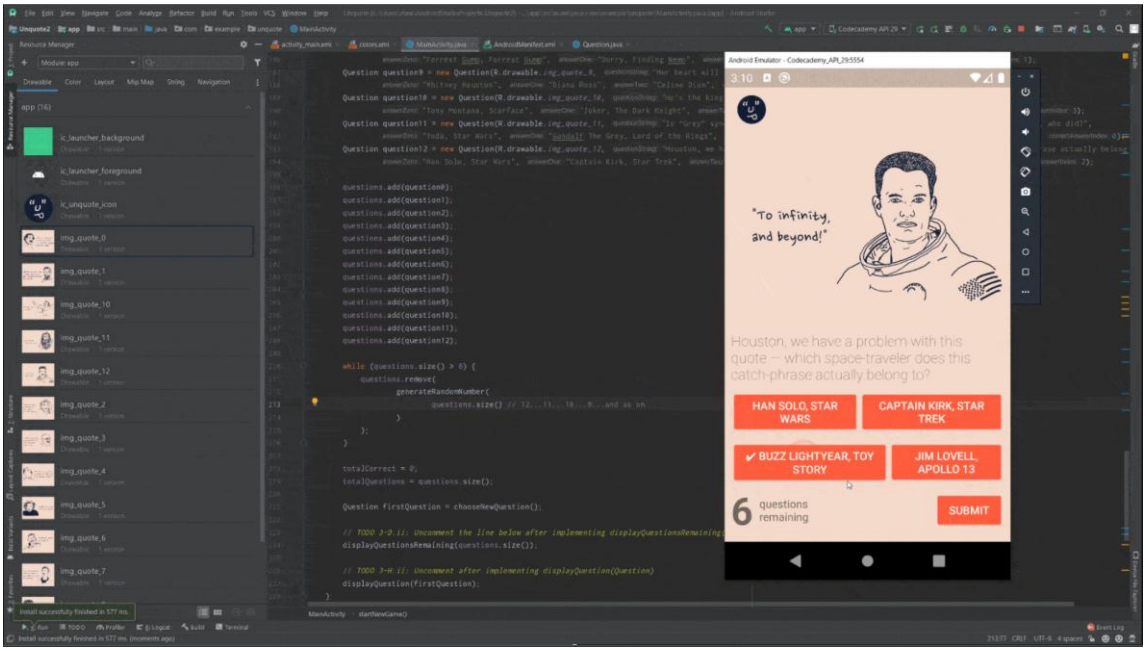

Рис. 1.8 Java

Подумайте про наслідки для вашого програмного проекту. Немає необхідності в двох командах розробників для двох платформ. Немає необхідності синхронізувати функції та макети. Ви просто розвиваєтеся швидше і можете отримати більше від свого бюджету.

React Native чудово підходить для мобільних додатків. Він забезпечує гладкий, плавний і чуйний користувальницький інтерфейс, значно скорочуючи час завантаження. Також набагато швидше і дешевше створювати програми в React Native, ніж створювати нативні, без необхідності йти на компроміс з якістю та функціональністю.

1.7 Висновок до першого розділу

Стек технологій (від англ. stack - "Стопка") – це набір технологій, на основі яких розробляється сайт або додаток.

Вибрані технології мають визначати функціонал продукту і те, чи можна його масштабувати в майбутньому. Також від обраного стека залежать оплата спеціалістів та час на розробку.

Головна ідея технічної сторони проекту – використовування нових кросплатформних технологій, що дозволять безперешкодно підтримувати користувача актуальною інформацією та у будь який момент дати йому змогу зреагувати на подію, що сталася в системі.

Розділ 2. АРХІТЕКТУРА ДОДАТКУ

2.1 Загальна концепція додатку

У повсякденному житті більшість людей не мають інформації згідно температурних показників приміщення, що дає великі витрати на підігрів приміщення. Об’єктом для відтворення температурних показників може слугувати любе приміщення де було розміщено сам пристрій. За допомогою системи управління розумним будинком можна підключити такі пристрої як:

- пристрій моніторингу температурних показників
- пристрій моніторингу вологості повітря.

Загальна схема взаємодії додатку

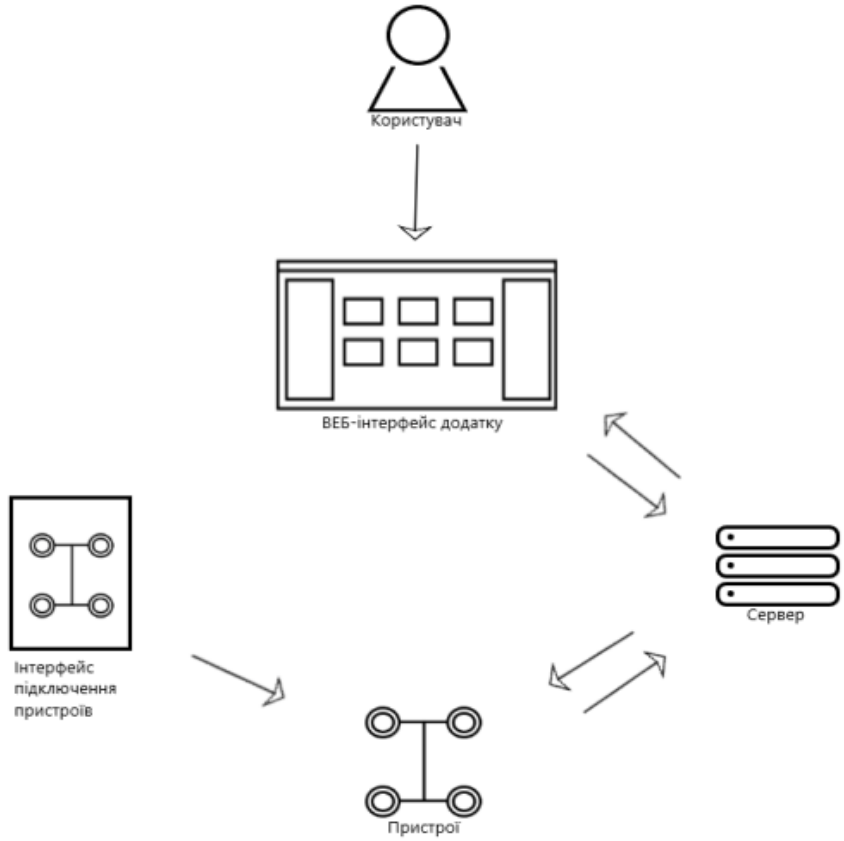

Рис 2.1 Загальна схема додатку

Згідно загальної схеми додатку(рис. 2.1), ми можемо побачити, що є всього 5 ключових одиниць:

- ВЕБ-інтерфейс додатку являє собою оболочку яка відповідає за відтворення змін на сервері та відправлення даних до серверу
- інтерфейс підключення пристроїв, іншими словами хаб, він являється серцем керування підключеними пристроями, саме хаб підключається до локальної мережі та синхронізується з сервером для подальшої обробки даних
- пристрої підключаються за допомогою хабу та ініцілаізуються через додаток для підключення (синхронізації) до серверу
- сервер відповідає за зберігання та опрацьовування отриманих даних, завдяки WebSocket-у сервер має постійне підключення між додатком та пристроями, та всі зміни які відбувалися на пристроях(наприклад зміна температури або показників вологості приміщення) будуть автоматично опрацьовані та надіслані користувачу. Такий же самий сценарій для дій користувача, якщо користувач, наприклад, змінює температурний показник датчику контролю температурою, ці дані відправляються на сервер, з серверу на пристрій, з пристрою назад на сервер та сервер передає оновлені дані на ВЕБ-інтерфейс користувача

2.2 Пристрій зчитування температурних показників

На рисунку 2.1 представлений датчик DHT11 для зчитування вологості та температури повітря, цей датчик має невеликий діапазон зчитування даних, та саме призначений для невеликих приміщень.

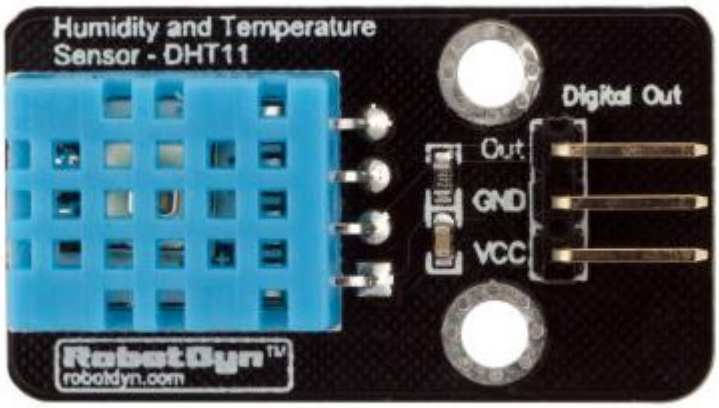

Рис. 2.1 Модуль з датчиком вологості та температури

Датчик DHT-11 має три головних підключення(рис. 2.2):

- Out – для виведення інформації щодо температури та показників вологості повітря, цей пін підключається до другого каналу на платі та подається живлення по 5 вольтовій лінії через резистор 4.7 кОм
- GND – другий пін підключається до 3-го каналу живлення GND
- VCC – третій пін підключається до 4-го каналу живлення 5 вольтової ліній

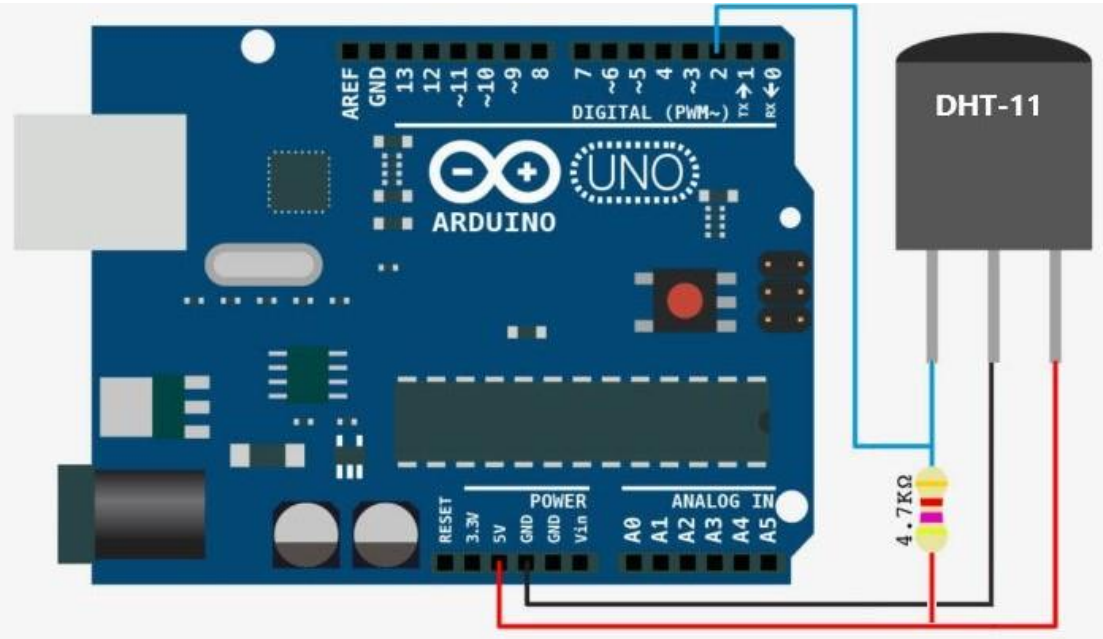

Рис. 2.2 Схема підключення модуля

Для того щоб датчик DHT-11 почав працювати та передавати температурні показники та показники вологості повітря, треба завантажити скетч(прошивка, в якій описано як повинен працювати датчик DHT-11) на пристрій Arduino (рис. 2.3).

Arduino Uno - это плата микроконтроллера на базе ATmega328. Arduino - это платформа для прототипирования с открытым исходным кодом, и ее простота делает ее идеальной для использования любителями, а также профессионалами. Arduino Uno имеет 14 цифровых входов / выходов (из которых 6 могут использоваться как выходы ШИМ), 6 аналоговых входов, кварцевый генератор 16 МГц, USB-соединение, разъем питания, заголовок ICSP и кнопку сброса. Он содержит все необходимое для поддержки микроконтроллера; просто подключите его к компьютеру с помощью кабеля USB или включите адаптер переменного тока в постоянный или аккумулятор, чтобы начать работу.

Arduino Uno отличается от всех предыдущих плат тем, что не использует микросхему драйвера FTDI USB-to-serial. Вместо этого он оснащен микросхемой микроконтроллера Atmega8U2, запрограммированной как преобразователь USB-to-serial.

«Uno» в переводе с итальянского означает «один» и назван так в честь предстоящего выпуска Arduino 1.0. Arduino Uno и версия 1.0 будут эталонными версиями Arduino в будущем. Uno - последняя в серии плат USB Arduino и эталонная модель для платформы Arduino.

```
#include <OneWire.h>

/*
    * Опис взаємодії з цифровим датчиком ds18b20
    * Підключення ds18b20 до ардуїно через пін 8
*/

OneWire ds(8); // Створюємо об'єкт OneWire для шини 1-Wire, за допомогою якого здійснюватиметься робота з датчиком

int temperature = 0; // Глобальна змінна для зберігання значення температури датчика DS18B20

long lastUpdateTime = 0; // Змінна для зберігання часу останнього зчитування з датчика
const int TEMP_UPDATE_TIME = 1000; // Визначаємо періодичність перевірок

void setup() {
    Serial.begin(9600);
}

void loop() {
    detectTemperature(); // Визначаємо температуру від датчика DS18b20
    Serial.println(temperature); // Виводимо отримане значення температури
    // Так як змінна temperature має тип int, дробова частина просто відкидатиметься
}

int detectTemperature() {
    byte data[2]; // Місце значення температури

    ds.reset(); // Починаємо взаємодію зі скидання всіх попередніх команд та параметрів

    ds.write(0xCC); // Даємо датчику DS18b20 команду пропустити пошук на адресу. У нашому випадку тільки один пристрій
    ds.write(0x44); // Даємо датчику DS18b20 команду виміряти температуру.
                    // Саме значення температури ми ще не отримуємо - датчик його покладе у внутрішню пам'ять

    if (millis() - lastUpdateTime > TEMP_UPDATE_TIME) {
        lastUpdateTime = millis();

        ds.reset(); // Тепер готуємось отримати значення виміряної температури

        ds.write(0xCC);
        ds.write(0xBE); // Просимо передати нам значення регістрів зі значенням температури

        // Отримуємо та зчитуємо відповідь
        data[0] = ds.read(); // Читаємо молодший байт значення температури
        data[1] = ds.read(); // А тепер старший

        // Формуємо значення
        temperature = (data[1] << 8) + data[0]; temperature = temperature >> 4;
    }
}
```

Рис. 2.3 Скетч

2.3 Серверна частина

Серверна частина реалізована на мові програмування Node.js, та сучасного фреймворку Express. Для правильного налагодження серверу, будуть використовуватись такі протоколи:

- протокол HTTPS
- протокол MQTT
- WEBSockets

Протокол HTTPS розподіляється на дві частини:

- HTTP(HyperText Transfer Protocol) – відноситься до списку не захищених протоколів
- HTTPS(HyperText Transfer Protocol Secure) – протокол являється розширенням протоколу HTTP з підтримкою шифрування для підвищення рівня безпеки

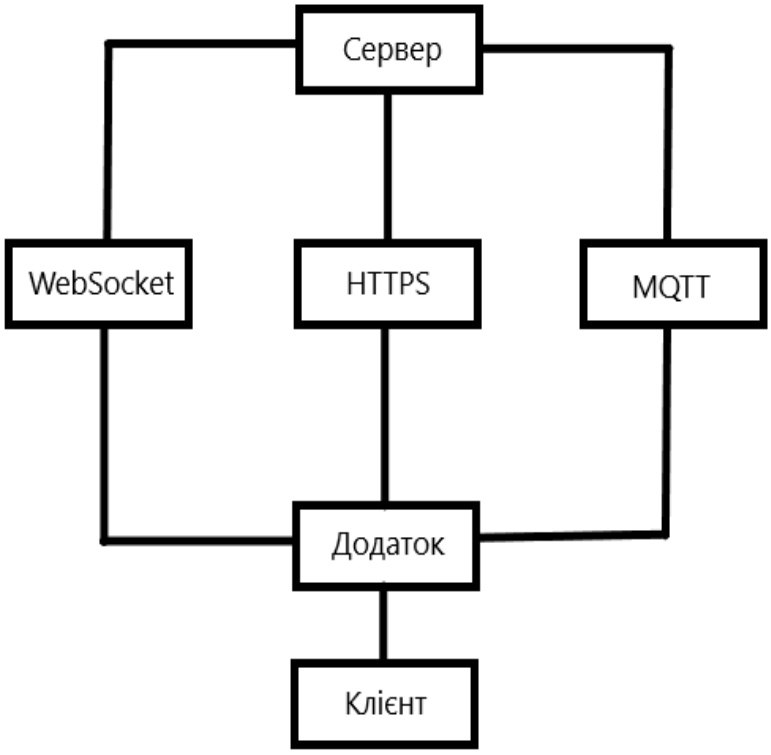

Рис. 2.4 Схема взаємодії серверу з додатком

Згідно схеми взаємодії серверу з додатком(рис. 2.4) усі учасники однієї мережі контактують між собою за допомогою сервера. Щоб забезпечити без перервну передачу даних між серверу та веб-додатком, використовується підключення по WebSocket-у, та за протоколом підключення MQTT відправляються дані з розумних пристроїв на сервер.

Серверна частина розроблена за стандартом CRUD:

- Create – створення даних
- Read – зчитування даних
- Update – оновлення даних
- Delete – видалення даних

Сховищем для збереження серверних даних виступає MongoDB. MongoDB являє собою документно орієнтовану базу даних, та відноситься до типу NoSQL баз даних. MongoDB є не реляційною базою даних та використовує колекції та документи. Основною одиницею даних служать документи, які складаються з пар ключ-значень. В колекціях містяться набори функцій та документів, що є еквівалентом таблиць реляційної бази даних.
MongoDB широко використовується у співпраці з AWS, Azure та багатьма іншими джерелами даних для розробки та функціонування додатків. Дозволяючи зберігати та запитувати великі обсяги даних, він пропонує такі надійні функції:

- Краще виконання запитів із належними функціями індексації та обробки.
- Аналітика в режимі реального часу та оптимізована обробка даних із використанням спеціальних запитів.
- Покращена доступність і гнучкість даних завдяки надійним функціям реплікації.
- Спільне використання даних дозволяє розділяти великі блоки даних для розподіленого та швидшого процесу виконання запитів.

MongoDB зберігає дані у форматі JSON з парами ключів і значень для кожної сутності (рис. 2.5), тоді як бази даних SQL зберігають дані у вигляді запису в рядку таблиці(рис. 2.6).

Рис. 2.5 Формат даних JSON

Name	age	contact-mobile	home-address
Perry	20	9273723723	perry street

Рис. 2.6 Формат даних SQL

MongoDB являється кращою базою даних для горизонтального та простого масштабування. Гнучка база даних яку постійно можна вдосконалювати, додавати більше серверів, розширювати сховища та налаштовувати їх і має такі переваги, як висока продуктивність, простота розгортання, простота використання та зручне зберігання даних. Дані групуються та зберігаються у наборах даних. Кожен набір даних є набором, кожна база даних містить кілька наборів.

2.4 Клієнтська частина

На рисунку 2.6 представлена клієнтська частина, на якій реалізована форма входу та форма реєстрація для нових користувачів.

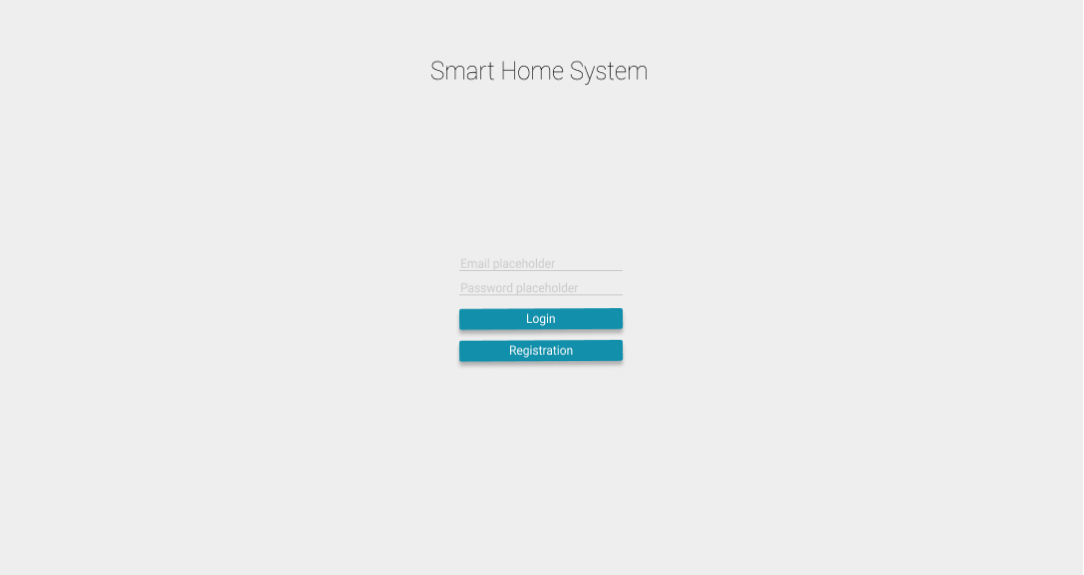

Рис. 2.7 Smart Home Systems форма входу

Веб-додаток розроблений на мові програмування TypeScript з використанням однієї з найпопулярнішою бібліотекою React. TypeScript являється тією ж самою мовою програмування JavaScript, але має більш розширені можливості ніж JavaScript та більш типізовану структуру коду.

Для забезпечення з'єднання серверу з додатком, використовуються FETCH запити та для отримання даних у реальному часі використовується WebSocket.

FETCH запити поділяються на такі методи:

- POST – для відправки даних на сервер
- GET – для отримання даних з серверу
- PATCH – для оновлення даних на сервері
- DELETE – для видалення даних з серверу

Щоб відкрити нове веб-сокет з'єднання, спочатку потрібно зробити запит на підключення указавши посилання з спеціальним протоколом ws, так само як і у HTTP, веб-сокет може бути з зашифрованим протоколом wss(рис 2.6). Слід зауважити, що краще використовувати зашифровані протоколи підключення до серверів, бо саме такі протоколи гарантують безпечний транспортний рівень та шифрує усі дані від відправника та розшифровує їх на стороні отримувача, що захищає від зловмисників.

```
const socket = new WebSocket('wss://javascript.info');
```

Рис. 2.8 Веб-сокет з зашифрованим протоколом

Після ініціалізації нового сокет підключення слід розробити спілкування між сервером та додатком(рис 2.8).

```
socket.onopen = function (event) {
    console.log('[open] Connected', event);
    console.log('Sending data tto the server');
};

socket.onmessage = function (event) {
    console.log(`[message] Getting data from the server: ${event.data}`);
};

socket.onclose = function (event) {
    if (event.wasClean) {
        console.log(`[close] Connection closed, code=${event.code} reason=${event.reason}`);
    } else {
        console.log('[close] Connection dropped');
    }
};

socket.onerror = function (error) {
    console.log(`[error] ${error.message}`);
};
```

Рис 2.8 Спілкування між сервером та додатком

Змінна socket являє собою об’єкт з чотирьома функціями:

- onopen – функція, яка встановлює підключення між сервером та додатком
- onmessage – функція, яка після успішно встановленого з’єднання передає дані з серверу до додатку
- onclose – функція, яка закриває підключення між сервером та додатком у випадку розірваного з’єднання або помилки серверу
- onerror – функція, яка виводить усі помилки серверу

Для зберігання серверних даних використовується state-management бібліотека Redux з використанням redux-thunk бібліотеки для обробки асинхронних запитів. Бібліотека Redux має в собі основні модулі для розробки:

- Slice(Reducer)
- Actions
- Selector’s

Найголовнішим модулем являється Slice, в цьому модулі розробляються редьюсери, та зберігається логіка обробки та запису даних в змінні через actions. В слід за Slice по пріоритетності йдуть Actions, де зберігаються асинхронні функції які відправляють та отримують дані з серверу, самі actions являються дуже гнучкими, в них можна описувати не тільки логіку обробки серверних даних, а ще й управляти станом компоненту на «відстані», передавати локальні дані, те що. Selector’s являють собою функції в яких описані посилання на змінні в Slice, іншими словами, через Selectors витягуються серверні дані.

На рисунку 2.9 представлена модель взаємодії Redux з компонентами де зберігається основна логіка додатку – такі компоненти називаються Features(основні компоненті, в яких зберігається логіка обробки запитів).

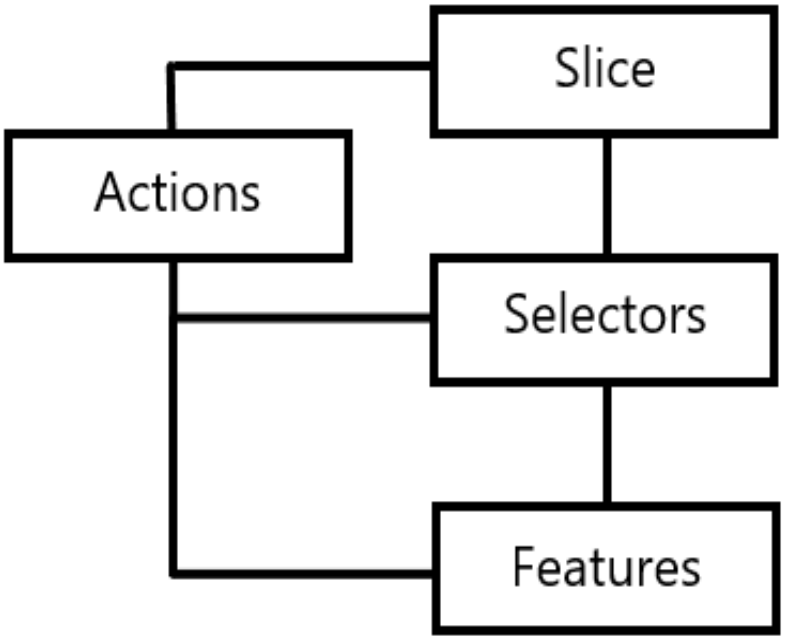

Рис. 2.9 Схема взаємодії Redux з Features

2.5 Висновки до другого розділу

За допомогою сучасної JavaScript бібліотеки React можна створювати інтерактивні користувацькі інтерфейси дуже швидко та просто. Створювати інкапсульовані компоненти з власним станом. Основна логіка компонентів написана на JavaScript завдяки чому, можна з легкістю передавати різні дані по всьому додатку.

Завдяки зашифрованому підключенню по WebSocket користувачі зможуть не перейматися за конфіденційність своїх даних та отримувати оновлені дані своєчасно та без перешкоджань бачити всі зміни на клієнтській частині.

Розділ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка форм ідентифікації

На початковій сторінці вас зустрічає форма логіну(рис 3.1) та по натисканню на кнопку «Registration» ви, як користувач будете перенаправлені на сторінку реєстрації нових користувачів(рис. 3.1)

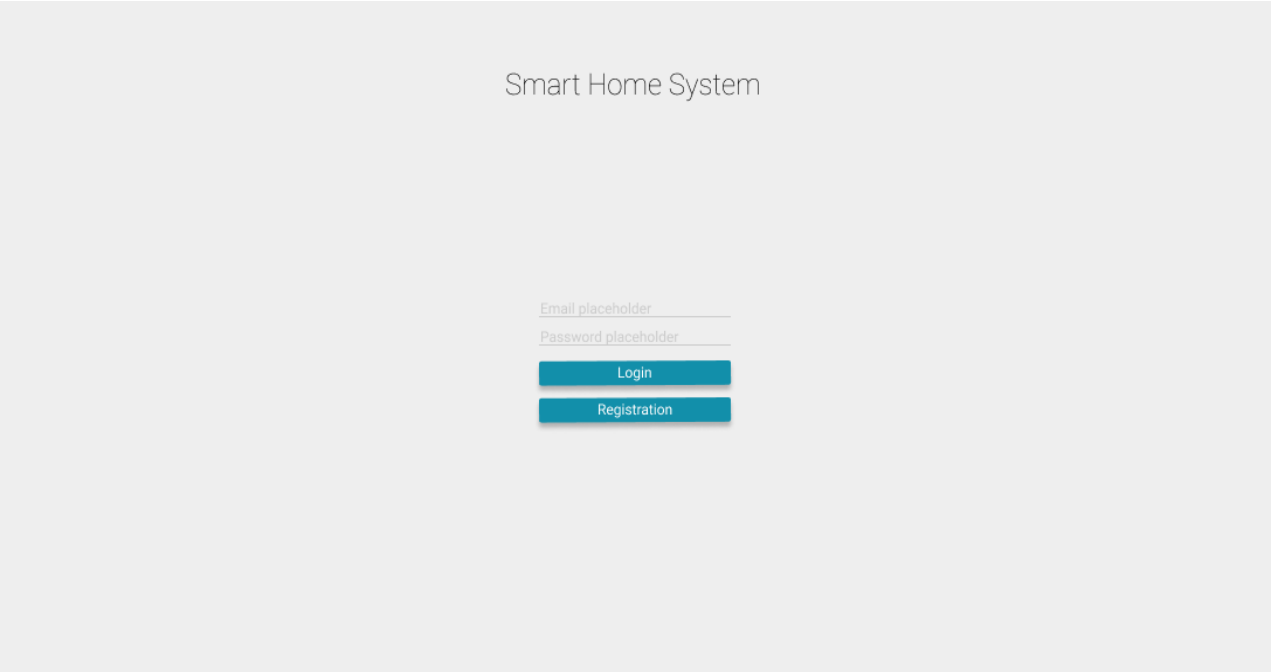

Рис. 3.1 Форма входу

На рисунку 3.2 після заповнення всіх полів та підтвердження реєстрації, в правій частині екрану випливає поп-ап повідомлення про успішно створеного користувача.

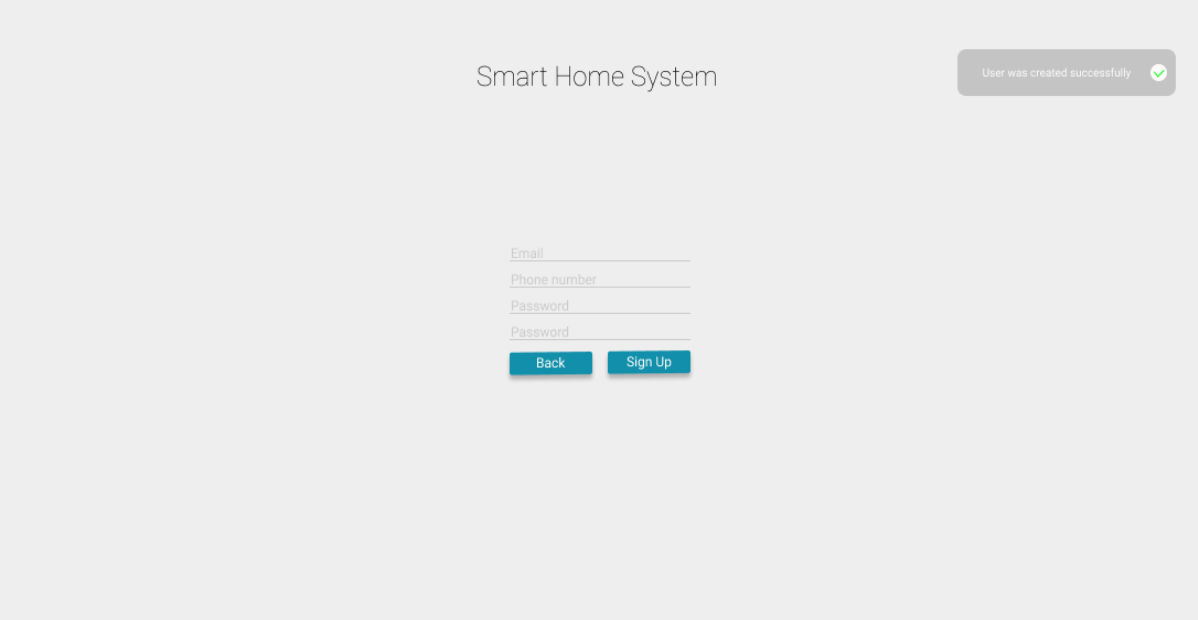

Рис. 3.2 Форма реєстрації користувачів

3.2 Корекція складових інтерфейсу

Після реєстрації або логіну на початковій сторінці, вас перенаправляє на сторінку відображення та підключення нових пристроїв(рис 3.3). У лівій частині сторінки є вікно відображення підключених пристроїв та кнопка підключення пристроїв(рис 3.4). У правій частині екрану знаходиться інформаційна панель, при натисканні на обраний пристрій в інформаційній панелі відображаються дані підключеного пристрою(рис. 3.5).

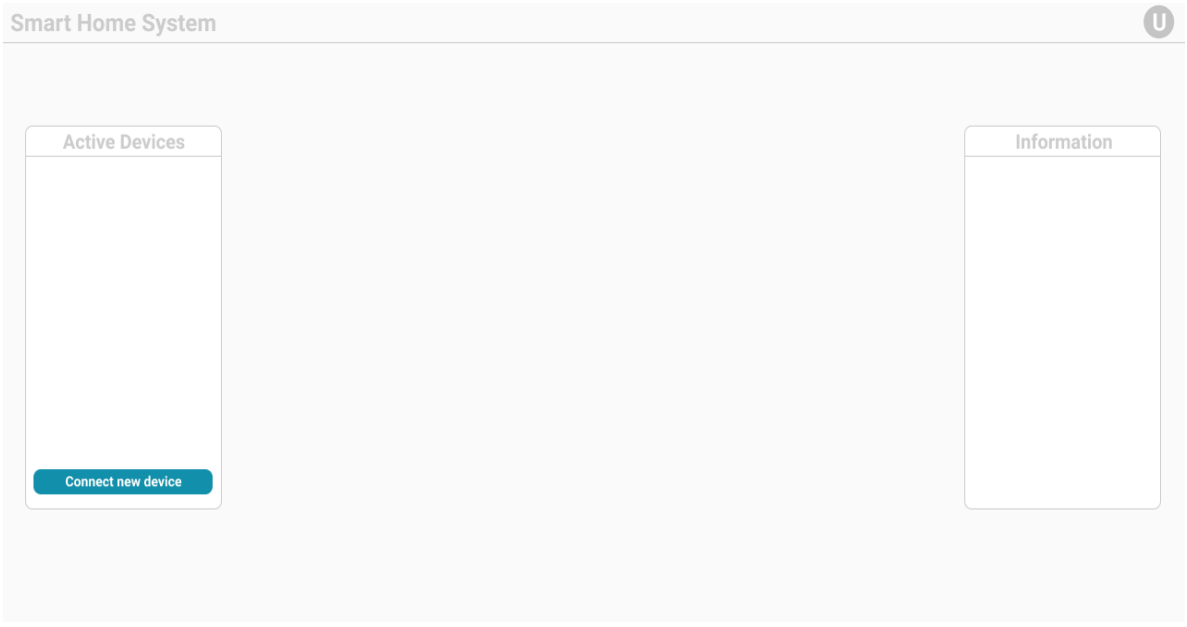

Рис. 3.3 Початкова сторінка після авторизаці

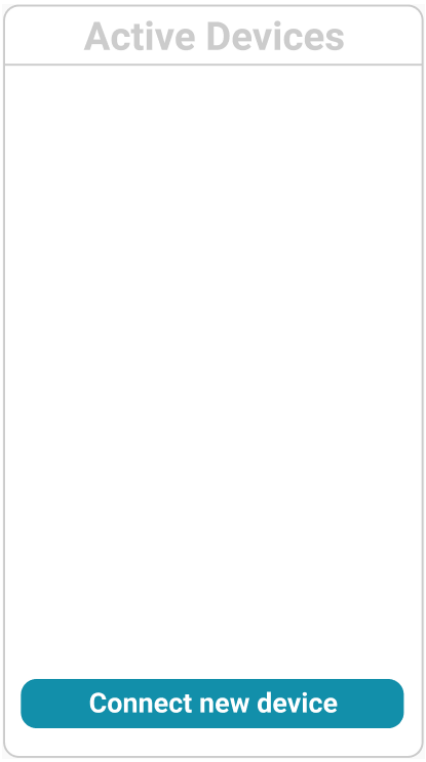

Рис. 3.4 Підключення пристроїв

Рис. 3.5 Інформаційна панель

У правій верхні частині додатку розположена кнопка, яка перенаправляє на сторінку налаштувань персональних даних(рис.3.6).

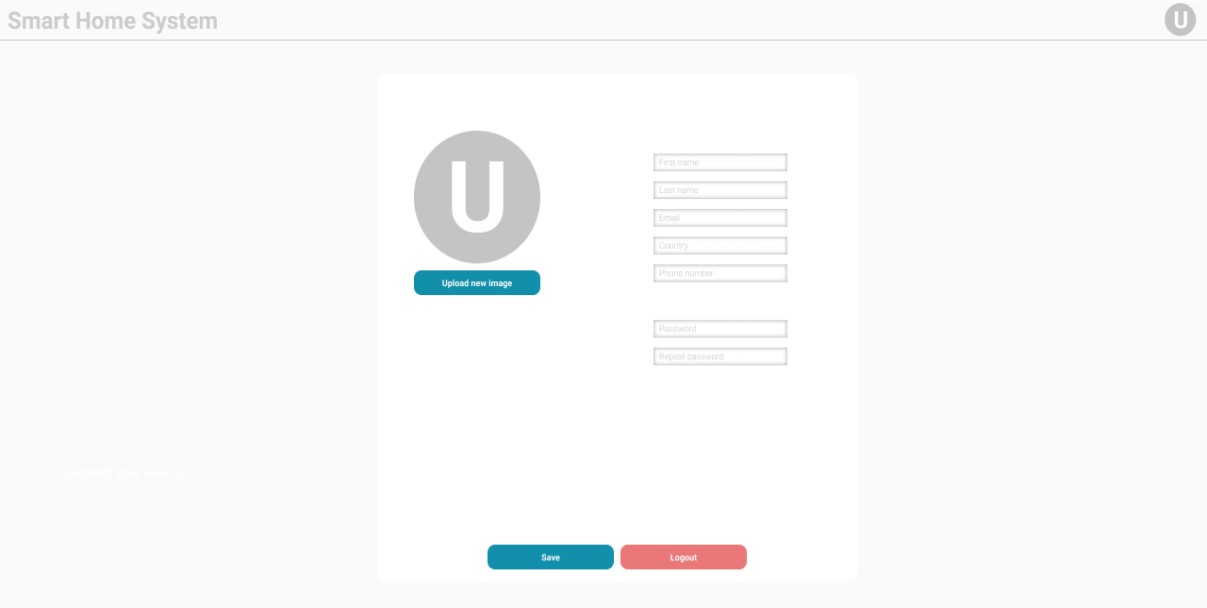

Рис. 3.6 Налаштування аккаунту

Ця сторінка містить такі основні компоненти:

- Компонент загрузки фотографії(рис. 3.7)
- Компонент персональних даних(рис. 3.8)
- Компонент зміни паролю(рис. 3.9)
- Кнопку збереження змінених даних(рис. 3.10)
- Кнопку виходу з додатку(рис. 3.11)

Рис. 3.7 Загрузка фотографії

Рис. 3.8 Форма заповнення персональних даних

Рис. 3.9 Форма зміни паролю

Рис. 3.10 Кнопка збереження змін

Рис. 3.11 Кнопка виходу з додатку

3.3 Розробка модального вікна підключення пристроїв

При натисканні на кнопку «Connect new device» відкривається модальне вікно підключення нового пристрою до вашої системи(рис. 3.12)

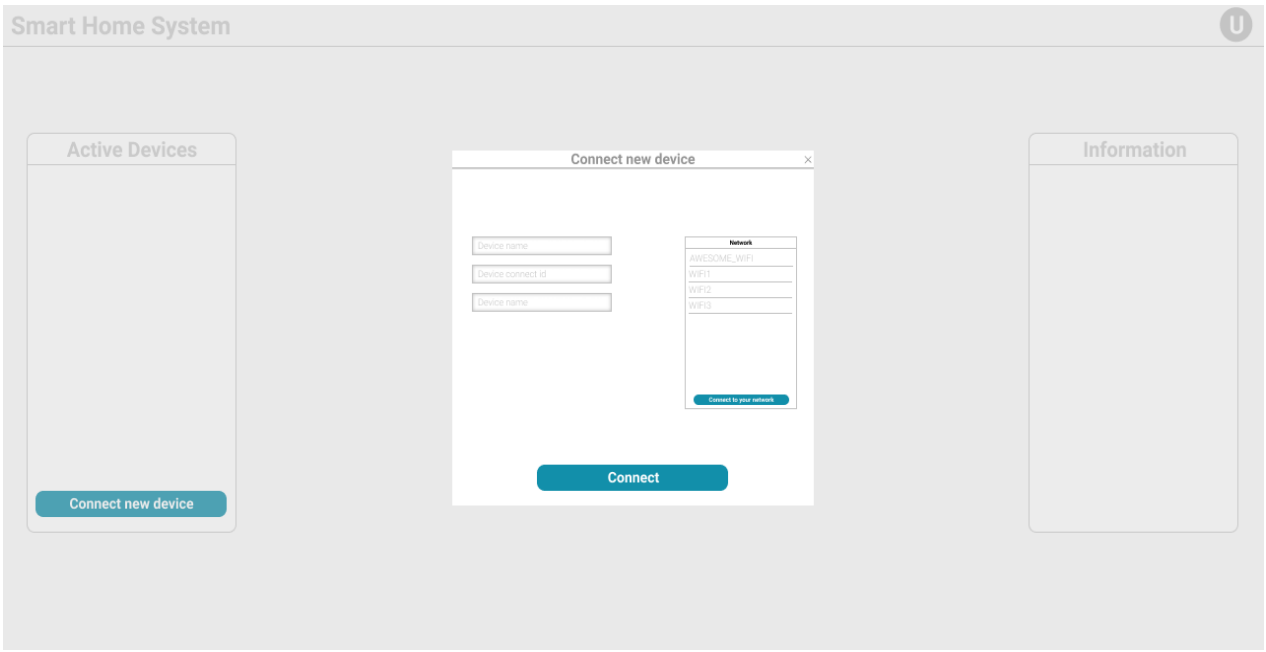

Рис. 3.12 Модальне вікно підключення пристрою

Сторінка підключення пристроїв містить такі поля:

- Поле введення назви пристрою(рис. 3.13)
- Поле введення унікального ідентифікатору пристрою(рис. 3.14)
- Поле введення групи до якої відноситься пристрій(рис. 3.15)
- Форма вибору вашої локальної мережі(рис. 3.16)

Рис. 3.13 Назва пристрою

Рис. 3.14 Унікальний ідентифікатор пристрою

Рис. 3.15 Група до якої належить пристрій

Рис. 3.16 Підключення до локальної мережі

Після натискання на кнопку «Connect» модальне вікно закриється та ви отримаєте поп-ап повідомлення про вдале підключеня пристрою до вашої системи(рис 3.17)

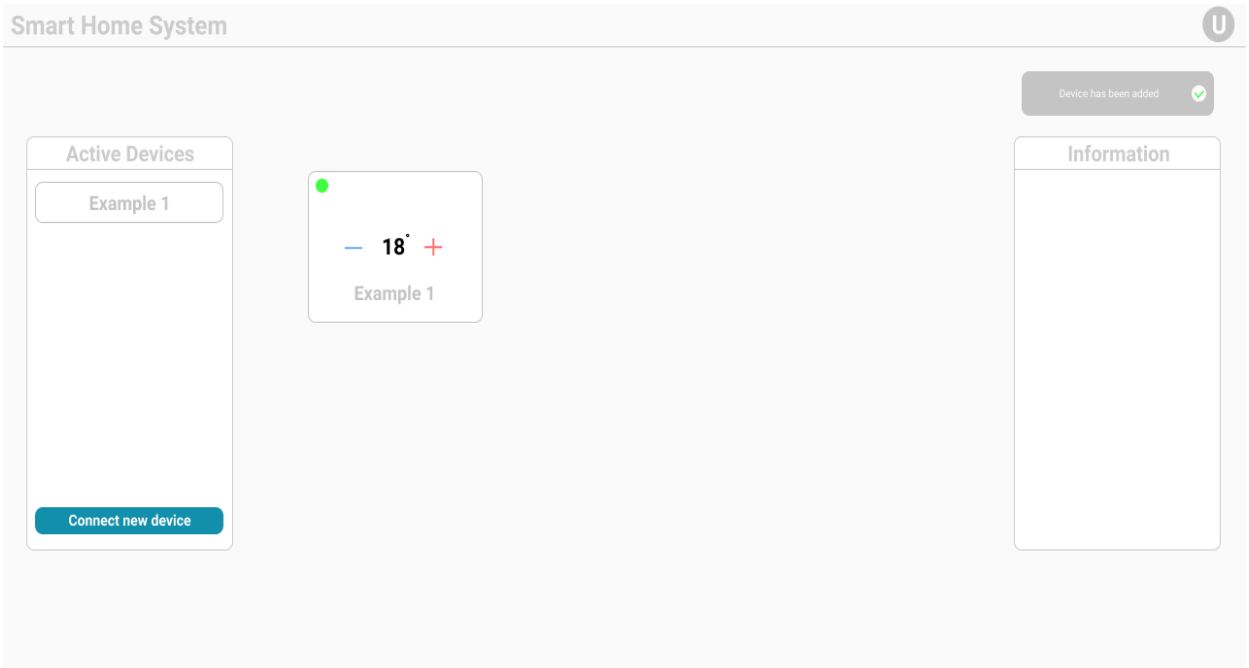

Рис. 3.17 Підключений пристрій

3.4 Аналіз стану показників пристрою

При натисканні на пристрій, він підсвічується сірим кольором для розуміння саме який пристрій був обраний. В інформаційній панелі відображаються такі поля(рис. 3.18):

- Статус пристрою
- Назва пристрою
- Группа пристрою

Та кнопки маніпулювання обраним пристроєм:

- Кнопка відключення пристрою
- Кнопка відкриття налаштувань пристрою

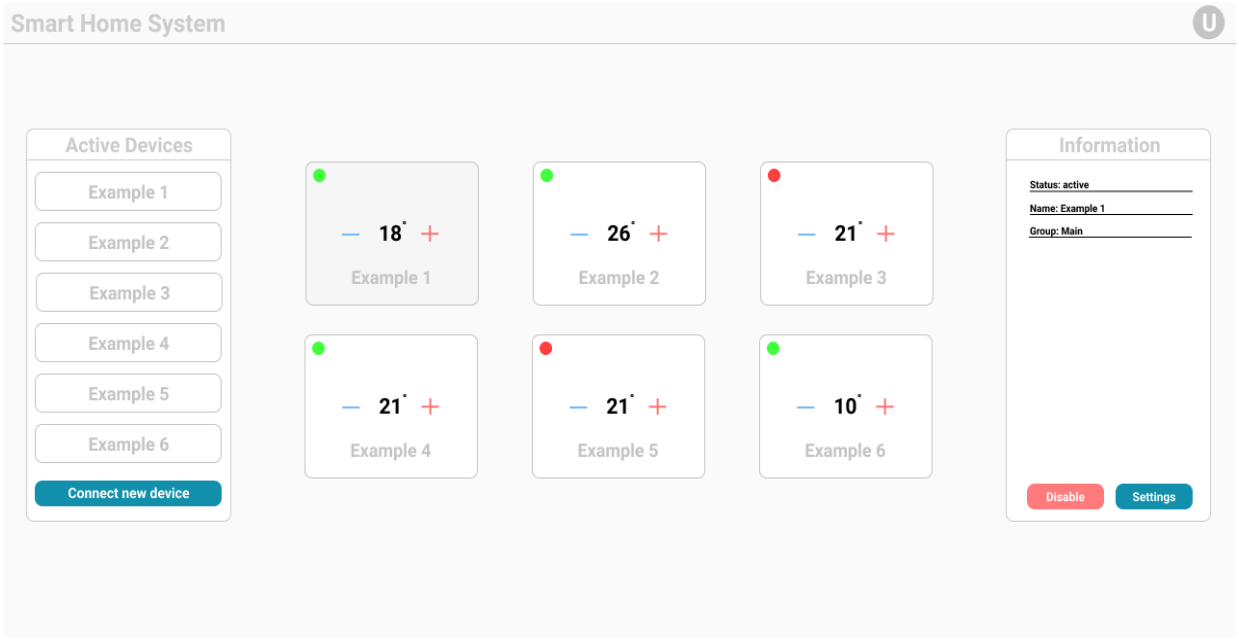

Рис. 3.18 Налаштування пристрою

При натисканні на кнопку відключення пристрою, він переходить до режиму сну, та змінюється стан статусу з Active(активований) на Disabled(відключений). Кнопка Settings(налаштування) відповідає за відкриття модального вікна з налаштуваннями цього пристрою, де саме росписано більш детельна інформація про сам пристрій та знаходиться 3 головних сторінки:

- 1 сторінка – налаштування пристрою;

- 2 сторінка – історія всіх змін;
- 3 сторінка – налаштування автоматичної праці пристрою.

Рис. 3.20 Модальне вікно налаштування пристрою

На рисунку 3.20 розташовані головні компоненти:

- Компонент регулювання температури пристрою
- Компонент відображення статусу та характеристик пристрою
- Графік з точними даними про пристрій

Графік має 4 лінії відображан, помаранчевим кольором позначено сила току, фіолетовим кольором позначено напругу, сірим кольором позначено потужність та синім кольором позначено температуру пристрою.

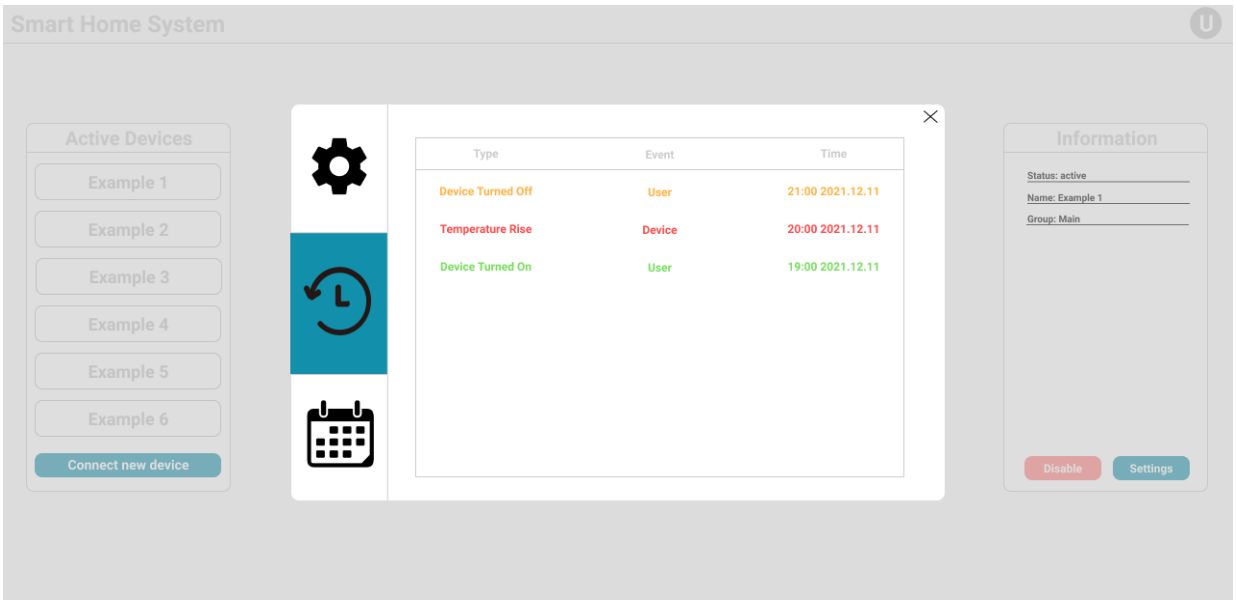

Рис. 3.21 Історія всіх змін

На рисунку 3.21 представлена сторінка з повною історією змін на пристрої, таблиця поділяється на 3 поля:

- Type – тип події
- Event – причина події, причина події може відображатися в залежності хто спричинив цю подію, це може бути User(користувач) чи сам Device(пристрій)
- Time – час коли подія була записана

Зараз відображається 3 статусі які були записані самим пристроєм:

- Помаранчевим кольором відображається статус відключення пристрою
- Червоним кольором відображається статус перегріву датчика
- Зеленим кольором відображається статус включення пристрою

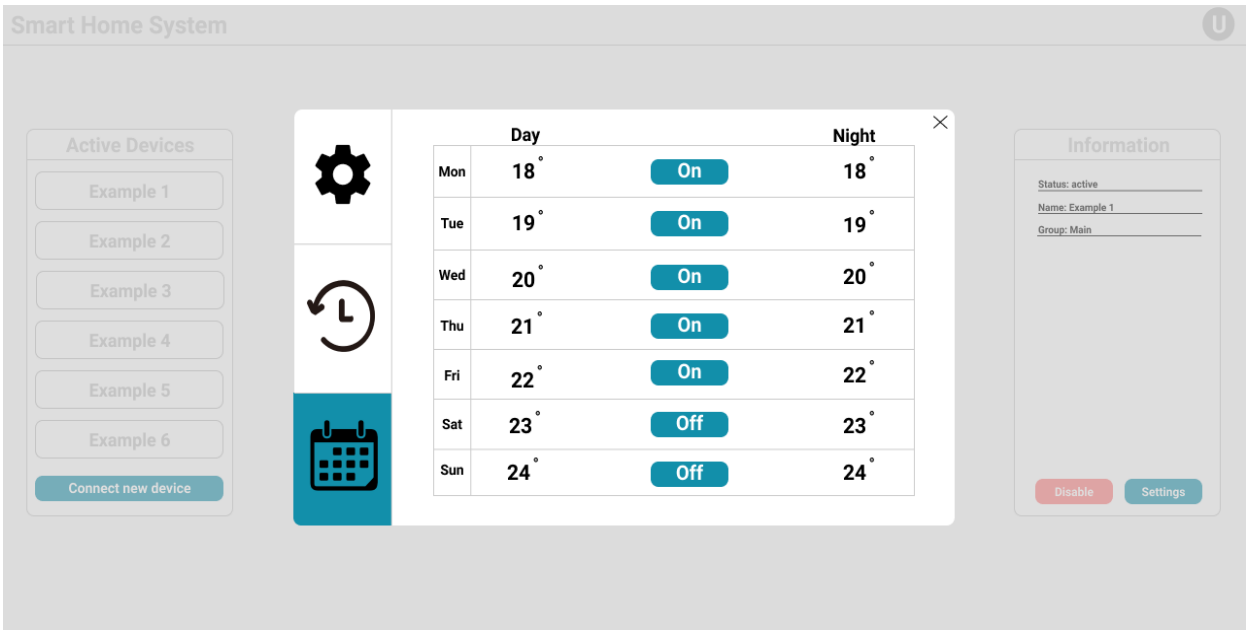

Рис. 3.22 Робота пристрою по графіку

На рисунку 3.22 представлена сторінка налаштування пристрою по розкладу. На кожень день тижня можна вказати температуру датчику, включити розклад або виключити його.

3.5 Мобільний додаток

Для розробки мобільного додатку за основу був взятий існуючий дизайн, так само, як і на веб-версії додатку, є форма входу користувача(рис. 3.23). Після авторизації користувач потрапляє на головну сторінку, де відображаються підключенні пристрої. При натисканні на сам пристрій ви потрапляєте на сторінку керування пристроєм, на цый сторінці, ви побачите основні відомості щодо температурних показників вашого приміщення та споживана потужність пристрою. На другій сторінці відображаються усі зміни в додатку. На третій сторінці ви зможете налаштовувати працю пристрою по графіку.

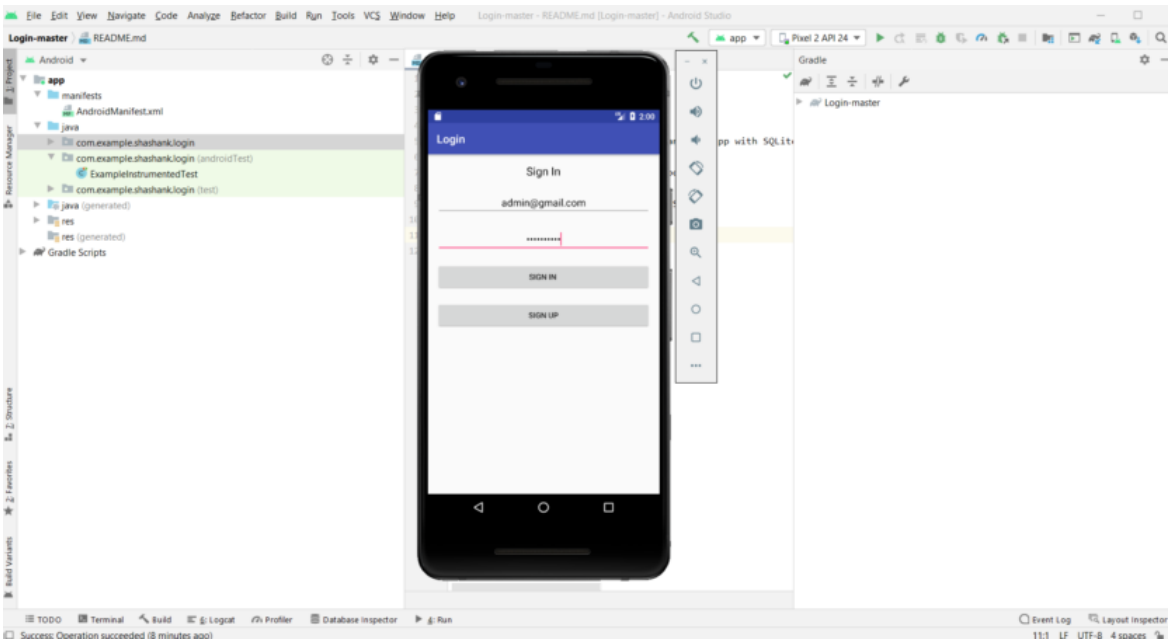

Рис. 3.23 Форма логіну на мобільному пристрої

Також, в мобільному додатку буде реалізована сторінка налаштувань аккаунту.

3.6 Висновок до третього розділу

Програмне забезпечення має досить зручний та сучайсний дизайн, який був розроблений для зручного користування як на компютерах та планшетах. Всі кнопки мають інтуїтивно зрозуміле місце для більш зручного користування на планшетних пристроях. Велика база бібліотеки React, дозволяє розробити унікальний та зручний в користуванні мобільний додаток . Завдяки бібліотеці react-native розробка під основні платформи iOS та Android становиться на багато легше, так як написаний JavaScript код компілюється одночасно під Java та Swift код.

ВИСНОВКИ

На протязі виконання проекту, було розроблено інтерфейс до веб-додатку по контролю температурних показників у вашому приміщенні – Smart Home Systems. Веб-додаток наділений сучасним та інтуїтивним дизайном, який оптимізован пад дві платформи, планшети, та персональні компютери. Головним плюсом додатку є те, що він знаходиться в інтернеті, тобто додаток розташований на властному домені та немає необхідності встановлювати його на персональний компютер або планшет. Цей додаток також може бути конвертований під додатки для платформ OS Windows, MacOS та Linux, за допомогою розширення Electron веб-додаток можна конвертувати під перечислені операційні системи, додаток не буде важити забагато, так як за допомогою React більшість встановлених розширень будуть конвертовані в build.js файл який займає всього до 10 мб. Розроблений інтерфейс до веб-додатку дає змогу користувачу працювати з додатком комфортно та без зайвих проблем.

Результатом дослідження є розробка інтерфейсу веб-додаток у«Smart Home Systems», який відоброжає температурні показники підключених пристроїв до локального середовища та здійсненються контроль над ними.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. IoT – Інтернет речей
2. React – JavaScript бібліотека для розробки додатків
3. Redux – сховище серверних даних
4. MQTT - Message Queue Telemetry Transport
5. HTTPS – протокол являється розширенням протоколу HTTPS
6. WebSocket – безперервний обмін даних між сервером та додатком

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SDLC phases [Електронний ресурс] – Режим доступу до ресурсу: https://uk.myservername.com/sdlc-phases.
2. ReactJS: понятное руководство для начинающих [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/ruvds/blog/428077/.
3. React [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/React.
4. Redux [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Redux.
5. Що таке Redux? [Електронний ресурс] – Режим доступу до ресурсу: https://uk.education-wiki.com/3707369-what-is-redux.
6. Как работает Redux? [Електронний ресурс] – Режим доступу до ресурсу: https://ivaneroshkin.medium.com/%D0%BA%D0%B0%D0%BA-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82-redux-a967d8616398.
7. ScSS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Sass.
8. Керівництво по Sass [Електронний ресурс] – Режим доступу до ресурсу: https://tokar.ua/read/6672.
9. NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js.
10. Про NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/about/.
11. Websockets [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/WebSocket
12. Інтернет речей [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%

80%D0%BD%D0%B5%D1%82_%D1%80%D0%B5%D1%87%D0%B5%D0%B9.

13. Що таке інтернет речей? [Електронний ресурс] – Режим доступу до ресурсу: http://iot.lviv.ua/%D1%89%D0%BE-%D1%82%D0%B0%D0%BA%D0%B5-%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D1%80%D0%B5%D1%87%D0%B5%D0%B9/.
14. NodeJS Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/nodejs/.
15. Введение в NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/web/nodejs/1.1.php
16. Основы CSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics
17. Самоучитель CSS [Електронний ресурс] – Режим доступу до ресурсу: http://htmlbook.ru/samcss
18. CSS по БЭМ [Електронний ресурс] – Режим доступу до ресурсу: https://ru.bem.info/methodology/css/
19. Все спецификации [Електронний ресурс] – Режим доступу до ресурсу: CSS https://www.w3.org/Style/CSS/specs.ru.html
20. How to setup Redux with ReduxJS toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://www.softkraft.co/how-to-setup-redux-with-redux-toolkit/
21. Redux Toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://redux-toolkit.js.org/
22. WebSocket [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/WebSocket

23. Learn Websocket [Електронний ресурс] – Режим доступу до ресурсу: https://learn.javascript.ru/websockets
24. Использование WebSocket API в веб-приложении [Електронний ресурс] – Режим доступу до ресурсу: https://netbeans.apache.org/kb/docs/javaee/maven-websocketapi_ru.html
25. Figma [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Figma
26. Гид по Figma [Електронний ресурс] – Режим доступу до ресурсу: https://tilda.education/articles-figma
27. Жизненный цикл разработки ПО [Електронний ресурс] – Режим доступу до ресурсу: https://beqa.pro/blog/sdlc/
28. SDLC - жизненный цикл разработки [Електронний ресурс] – Режим доступу до ресурсу: https://myalm.ru/news/SDLC-%D0%B6%D0%B8%D0%B7%D0%BD%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9-%D1%86%D0%B8%D0%BA%D0%BB-%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B
29. В чем различия между HTTP и HTTPS [Електронний ресурс] – Режим доступу до ресурсу: https://sweb.ru/journal/article/v-chem-razlichiya-mezhdu-http-i-https/
30. HTTPS & HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://hostiq.ua/wiki/http-https/
31. HTTP or HTTPS [Електронний ресурс] – Режим доступу до ресурсу: https://ssl.com.ua/blog/http-vs-https/
32. Обзор протокола HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview

33. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/HTTP
34. Защищенный протокол HTTPS [Електронний ресурс] – Режим доступу до ресурсу: https://serpstat.com/ru/blog/zashishennij-protokol-https-kak-na-nego-perevesti-sajt/?&utm_source=google&utm_medium=cpc&utm_campaignid=14927491902&utm_campaign=s-ru-instrumenty-general&utm_adgroupid=131229090911&utm_adgroupname=&utm_content=&utm_device=c&utm_keyword=&utm_matchtype=&utm_placement=&utm_targetid=dsa-19959388920&utm_loc_interest_ms=&utm_loc_physical_ms=21580&utm_creative=562112631304&utm_adposition=&utm_feeditemid=&gclid=Cj0KCQiA2NaNBhDvARIsAEw55hic3AvLg2huUb9WDDt9a2gkTIzEXHkIILThnxTl6lBEPC-k81pP6XUaAocKEALw_wcB
35. Что такое HTTPS [Електронний ресурс] – Режим доступу до ресурсу: https://cityhost.ua/blog/chto-takoe-https.html

Код форми Логіну:

```
import React from "react";
// import { useSelector } from "react-redux";

import { Header } from "../features";

import { Public } from "./Public";
import { Private } from "./Private";

// import { getUserToken } from "../features/store/app/app.selectors";

import "./App.css";

const App: React.FC = () => {
  // const userToken = useSelector(getUserToken);
  const isLoggedIn = localStorage.getItem("loginToken");

  const route = isLoggedIn ? <Private /> : <Public />;

  return (
    <div className="App">
      <Header />
      {route}
    </div>
  );
};

export default App;
```

```
import React, { useCallback, useState } from "react";
import { useDispatch } from "react-redux";
```

```
import { Input, Button } from "../../components";
import { setToken } from "../store/app";

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

export const Login: React.FC = () => {
  const dispatch = useDispatch();

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const [isEmailError, setIsEmailError] = useState(false);
  const [isPasswordError, setIsPasswordError] = useState(false);

  const onChangeEmail = useCallback((event) => {
    setEmail(event.target.value);
  }, []);

  const onChangePassword = useCallback((event) => {
    setPassword(event.target.value);
  }, []);

  const validateLogin = useCallback((email: string, password: string) => {
    if (!email) {
      setIsEmailError(true);
    } else {
      setIsEmailError(false);
    }
```

```
    if (!password) {
      setIsPasswordError(true);
    } else {
      setIsPasswordError(false);
    }
  }, []);

  const onSubmitLogIn = useCallback(() => {
    const userData = {
      email,
      password,
    };

    validateLogin(email, password);

    if (!isEmailError && email !== "" && !isPasswordError && password
!== "") {
      console.log(userData);
      dispatch(setToken("tokenX23S$asd1233"));
      localStorage.setItem("loginToken", "tokenX23S$asd1233");
    }
  }, [dispatch, email, password, isEmailError, isPasswordError,
validateLogin]);

  return (
    <div className={classes.Login}>
      <div className={classes.LoginWrapper}>
        <Input
```

```
            value={email}
            placeholder="Enter your e-mail"
            onChange={onChangeEmail}
            isError={isEmailError}
            type="text"
          />

          <Input
            value={password}
            placeholder="Enter your password"
            onChange={onChangePassword}
            isError={isPasswordError}
            type="password"
          />

          <Button title="Log In" type="default" onClick={onSubmitLogIn} />
        </div>
      </div>
    );
};
```

Додаток Б

Код підключення Redux:

```
import { createStore, compose, combineReducers, applyMiddleware } from
"redux";

import thunk from "redux-thunk";
import appReducer from "../features/store/app";
import mainReducer from "../features/store/main";

declare global {
  interface Window {
    __REDUX_DEVTOOLS_EXTENSION_COMPOSE__?: typeof
compose;
  }
}

const composeEnhancers =
  (process.env.NODE_ENV === "development"
    ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
    : null) || compose;

const rootReducer = combineReducers({
  app: appReducer,
  main: mainReducer,
});
export const rootStore = createStore(
  rootReducer,
  composeEnhancers(applyMiddleware(thunk))
);
```