Міністерство освіти і науки України
ДВНЗ «Прикарпатський національний університет імені Василя Стефаника»
Кафедра комп'ютерної інженерії та електроніки

Гунда Юрій Олександрович
Hunda Yurii

УДК 004

Спеціальність 123 «Комп'ютерна інженерія»
(шифр та назва спеціальності)

Кваліфікаційна робота
на здобуття освітньо-кваліфікаційного рівня магістр
(бакалавр, спеціаліст, магістр)

База даних з інтерфейсом на основі Wi-Fi технології

The database with an interface based on Wi-Fi technology

Науковий керівник:
кандидат технічних наук,
доцент Голота В. І.

Рецензенти:
д. ф.-м. н.,
проф. кафедри матеріалознавства
і новітніх технологій
Яремій І. П.

Івано-Франківськ
2020

Формат	Зона	Поз.	Позначення	Найменування	К-ть	Прим.
			123. УДК 004	Пояснювальна записка	1	

Змн.	Арк.	№ докум.	Підпис	Дата	123. УДК 004		
Розробив		Гунда Ю.О..			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Голота В.І.					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи на здобуття рівня магістр «База даних з інтерфейсом на основі Wi-Fi технології»: кількість сторінок – 75, джерел – 11.

Об’єкт дослідження – взаємодія між електронними пристроями та IoT-системи.

Предметом дослідження є система автоматичної генерації метео-даних, зі зберіганням, передаванням та представленням цих даних.

Мета роботи - створення системи, яка складається із трьох частин: апаратної частини, бази даних, інтерфейсу користувача; дослідження параметрів та характеристик; визначення фінансових затрат на створення проєкту.

Робота складається з вступу, 5 розділів, висновку, списку використаних джерел інформації.

Ключові слова: мікроконтролер, інтернет речей, веб-застосунок, база даних, інтерфейс користувача, програмне забезпечення.

					123. УДК 004		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Гунда Ю.О.			Анотація	Арк.	Аркуш 3 / Аркушів 2
Перевірив		Голота В.І.					
Н. Контр.							
Затверд.							

ABSTRACT

Explanatory note to the qualification work for obtaining a master's degree "The database with an interface based on Wi-Fi technology": number of pages - 75, sources - 11.

The object of research is the interaction between electronic devices and IoT systems.

The subject of the study is the implementation of a system of automatic generation of meteorological data, with storage, transmission and presentation of this data.

The purpose of the work is to review the types of databases. Research of parameters and characteristics of the considered technologies. Creating a system that consists of three parts: hardware, database, user interface. Determining the financial costs of creating a project.

The work consists of an introduction, 5 sections, conclusion, list of sources of information used.

Keywords: microcontroller, Internet of Things, web application, database, user interface, software.

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Фізико-технічний факультет
Кафедра «Комп'ютерної інженерії і електроніки»

Пояснювальна записка
до кваліфікаційної роботи
на тему:

«База даних з інтерфейсом на основі Wi-Fi технології»

					123. УДК 004		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Гунда Ю.О.			Пояснювальна записка	Арк.	Аркуш 5 / Аркушів 75
Перевірив		Голота В.І.					
Н. Контр.							
Затверд.							

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

JS – JavaScript.

ES – ECMAScript.

HTML (Hyper Text Markup Language) – мова гіпертекстової розмітки.

XML (Extensible Markup Language) – розширювана мова розмітки.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

HTTP (Hyper Text Transfer Protocol) – протокол передачі гіпертекстових документів.

JSON (JavaScript Object Notation) – запис об’єктів JavaScript.

REST (Representational State Transfer) - передача репрезентативного стану.

API (Application Programming Interface) – прикладний програмний інтерфейс.

NoSQL (Not only SQL, non-SQL) – не тільки мова SQL, не SQL.

СУБД – система управління базою даних.

БД – база даних.

ЛФ - Людський фактор.

DBaaS (Database as a Service) – база даних як сервіс.

ЗМІСТ

ВСТУП

Розвиток апаратних та програмних технологій призвів до того, що інформація в даний час поширюється неймовірними темпами. В даній роботі буде розглянуто і розроблено систему автоматичного зберігання, передавання та представлення даних. Для збереження даних використовуються бази даних, які впливають на характеристики створюваної системи і є майже невід'ємною частиною програмного та апаратного забезпечення.

Метою роботи є створення системи, яка складається із трьох частин: апаратної частини, бази даних, інтерфейсу користувача; дослідження параметрів та характеристик; визначення фінансових затрат на створення проєкту.

Об'єктом дослідження є взаємодія між електронними пристроями та IoT-системи.

Предметом дослідження система автоматичної генерації метео-даних, зі зберіганням, передаванням та представленням цих даних.

РОЗДІЛ 1. ТЕХНОЛОГІЇ СТВОРЕННЯ ВЕБ-ЗАСТОСУНКІВ ТА ПРОЄКТУВАННЯ БАЗ ДАНИХ

1.1. Технологія JavaScript

JavaScript ("JS" для стислості) - це повноцінна динамічна мова програмування, яка застосовується до HTML документу, і може забезпечити динамічну інтерактивність на веб-сайтах.

JavaScript спочатку був створений, щоб «оживити веб-сторінки».

Програми на цій мові називаються сценаріями. Їх можна записати прямо в HTML веб-сторінки та запускати автоматично під час завантаження сторінки. Сценарії надаються та виконуються як звичайний текст. Для запуску їм не потрібна спеціальна підготовка чи компіляція.У цьому аспекті JavaScript сильно відрізняється від іншої мови, яка називається Java.

Сьогодні JavaScript може виконуватися не лише у браузері, але і на сервері, або фактично на будь-якому пристрої, що має спеціальну програму, яка називається механізмом JavaScript.

У браузері є вбудований механізм, який іноді називають «віртуальною машиною JavaScript». Різні двигуни мають різні “кодові назви”. Наприклад:

- V8 - у Chrome та Opera.
- SpiderMonkey - у Firefox.
- Є й інші кодові назви, такі як «Chakra» для IE, «ChakraCore» для Microsoft Edge, «Nitro» та «SquirrelFish» для Safari тощо.

Проте, JS використовується не тільки в веб-розробці. Однією з унікальних переваг JavaScript є його поширеність.

Двигуни складні, але ази прості. Механізм (вбудований, якщо це браузер) зчитує («аналізує») сценарій. Потім він перетворює («компілює») сценарій на машинну мову. І тоді машинний код працює досить швидко. Двигун застосовує оптимізацію на кожному кроці процесу. Він навіть спостерігає за скомпільованим сценарієм під час його роботи, аналізує дані, що проходять через нього, і додатково оптимізує цей код на основі здобутих

знань.

Що ж стосується інших можливостей - вони залежать від оточення, в якому запущений JavaScript. У браузері JavaScript вміє робити все, що відноситься до маніпуляції зі сторінкою, взаємодії з відвідувачем і, в якійсь мірі, з сервером:

- Створювати нові HTML-теги, видаляти існуючі, змінювати стилі елементів, ховати, показувати елементи.
- Реагувати на дії відвідувача, обробляти події.

У JavaScript є принаймні три характеристики:

1. Повна інтеграція з HTML / CSS.
2. Простота при створенні не складних проєктів.
3. Підтримка всіх основних браузерів ввімкнена за замовчуванням.

JavaScript - це єдина технологія браузера, яка поєднує ці три речі. Саме це робить JavaScript унікальним. Ось чому це найпоширеніший інструмент для створення інтерфейсів браузера. Тим не менш, JavaScript також дозволяє створювати сервери, мобільні додатки тощо.

Сучасний JavaScript – це «безпечна» мова програмування. Він не забезпечує низькорівневий доступ до пам’яті або центрального процесора, оскільки спочатку був створений для браузерів, які цього не потребують. Можливості JavaScript сильно залежать від середовища, в якому він працює. Наприклад, Node.js підтримує функції, які дозволяють JavaScript читаи / писати довільні файли, виконувати мережеві запити, тощо.

JavaScript у браузері може робити все, що стосується маніпулювання веб-сторінками, взаємодії з користувачем та веб-сервером. Наприклад, JavaScript у браузері може: додавати на сторінку новий HTML, змінювати наявний вміст; надсилати запити через мережу на віддалені сервери; завантажувати файли; працювати з файлами cookie; зберігати дані на стороні клієнта («локальне сховище») . Можливості JavaScript у браузері обмежені заради безпеки користувача. Мета полягає в тому, щоб запобігти доступу

злочинної веб-сторінки до приватної інформації або шкоди даним користувача.

Приклади таких обмежень включають:

- JavaScript на веб-сторінці не може читати / писати довільні файли на жорсткому диску, копіювати їх або виконувати програми. Він не має прямого доступу до функцій ОС.

Сучасні браузери дозволяють йому працювати з файлами, але доступ обмежений і надається лише в тому випадку, якщо користувач робить певні дії, наприклад, "скидає" файл у вікно браузера або вибирає його за допомогою тегу <input>.

- Різні вкладки/вікна, як правило, не знають один про одного. Іноді вони це роблять, наприклад, коли одне вікно використовує JavaScript, щоб відкрити інше. Але навіть у цьому випадку JavaScript з однієї сторінки може не мати доступу до іншої, якщо вони надходять з різних сайтів (з іншого домену, протоколу чи порту).

Це називається "Політика того самого походження". Щоб обійти це, обидві сторінки повинні домовитись про обмін даними та містити спеціальний код JavaScript, який це обробляє.

- JavaScript може легко зв’язатись через мережу з сервером, звідки походить поточна сторінка. Але здатність отримувати дані з інших сайтів/доменів дещо зменшена. Хоча це можливо, для цього потрібна явна згода (виражена в заголовках HTTP) з віддаленої сторони. Ще раз, це обмеження безпеки.

1.2. Технологія AJAX

AJAX розшифровується як Asynchronous JavaScript and XML. Це технологія, яка дозволяє будувати інтерактивні веб-застосунки разом із мовою JavaScript.

За допомогою даної технології можна отримувати дані із зовнішніх ресурсів, уникаючи використання серверів. Дана технологія частіше

використовується, коли немає фахівця із розробки серверної частини та проєктувальника бази даних.

Часто користувачі навіть не здогадуються, що отримують чи передають дані між зовнішніми серверами.

AJAX технологія, яка не належить до жодної мови програмування. Для запиту можна використовувати події, що будуть запускати передавання даних. Такі події не обов'язково повинні мати явне представлення, наприклад у вигляді кнопки «Старт» чи тому подібне, натомість можуть бути використані приховані події: рух курсора миші та інші.

AJAX використовує специфічні технології:

- XML або JSON: формат даних, який представляється у вигляді тексту. Використовується для передачі даних від сервера до браузера користувача.
- CSS: стилізує дані, для кращого подання.
- JavaScript: керує представленням даних.

XMLHttpRequest: ключова особливість технології – метод, що дозволяє передавати дані із сервера до браузера. Всі сучасні браузери підтримують XMLHttpRequest.

Використання AJAX запитів має низку переваг, серед яких можна виділити:

- підвищену швидкість;
- гнучкість використання;
- простота в реалізації;
- реалізований для передавання малої кількості даних;
- не сповільнюється з обмеженою пропускною здатністю.

Властивості об'єкта XMLHttpRequest:

readystate - це властивість об'єкта XMLHttpRequest, що містить статус XMLHttpRequest:

- 0: запит не ініціалізований.
- 1: встановлено з’єднання із сервером.
- 2: запит отримано.
- 3: обробка запиту.
- 4: запит закінчено і відповідь готова.

status - це властивість об'єкта XMLHttpRequest, яке повертає номер статусу запиту:

- 200: «ОК»
- 403: «Заборонено»
- 404: «Не знайдено»

1.3. Інтерфейс користувача

Дизайн інтерфейсу користувача / системи стосується широкого кола користувачів і систем. Людський фактор (ЛФ) - це дисципліна, яка намагається оптимізувати взаємозв'язок між технологіями та людьми. ЛФ виявляє та застосовує інформацію про поведінку людей, здібності, обмеження та інші характеристики до конструкції інструментів, машин, систем, завдань, робочих місць та середовища для продуктивного, безпечного, комфортного та ефективного використання людиною.

Область ергономіки виникла в результаті досліджень того, як люди взаємодіють із навколишнім середовищем, особливо на робочому місці. Проблеми ергономіки, як правило, перебувають на сенсорно-моторному рівні, але з додатковим фізіологічним статусом та акцентом на стресі. Ергономічні дослідження комп’ютерів та робочого місця досліджують взаємозв’язок між людьми та робочим середовищем, включаючи наслідки стресу, звичайні робочі завдання, використання клавіатури, використання миші.

Взаємодія людина-комп'ютер (HCI; також взаємодія комп'ютер-людина, CHI) - це дисципліна, що стосується проєктування, оцінки та впровадження інтерактивних обчислювальних систем для використання людиною (як апаратного, так і програмного забезпечення), а також вивчення взаємодії людини з комп'ютерами. HCI - це підгрупа ЛФ-дисципліни, орієнтована на комп'ютерне обладнання та програмне забезпечення як систему.

Термін інтерфейс часто використовується у ЛФ-полі і повинен бути чіткіше визначений. Інтерфейс - це місце, де незалежні системи зустрічаються, діють або взаємодіють між собою. Вузько визначений, користувацький інтерфейс (UI) складається з пристроїв введення та виведення, а також інформації, яку користувачі бачать та взаємодіють з нею на екрані комп'ютера. У більш широкому розумінні, інтерфейс користувача включає все, що визначає взаємодію користувачів із системами, інформацією та людьми, коли вони виконують завдання за допомогою комп’ютерів.

Розробка інтерфейсу користувача може бути окремою від решти процесу розробки продукту, а може і не бути. Однак фокус, безумовно, інший. Все зосередження приходиться на елементи інтерфейсу та об’єктах, які сприймають та використовують користувачі, а не на функціональних можливостях програми.

Процес, спеціально спрямований на розробку та розробку інтерфейсів користувача, показаний на рис. 1.1.

Чотири основні фази процесу:

- Збір / аналіз інформації про користувача
- Розробка моделі інтерфейсу користувача
- Побудова інтерфейсу користувача
- Перевірка інтерфейсу користувача

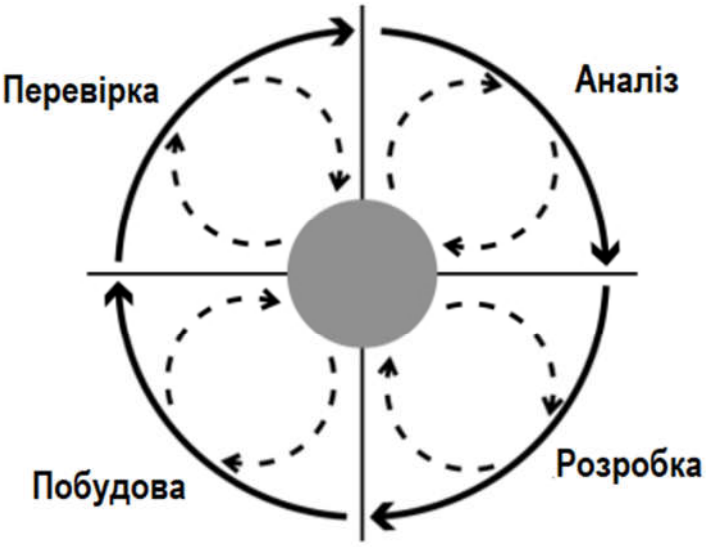

Рисунок 1.1. - Елементи розробки інтерфейсу користувача.

Цей процес не залежить від апаратної та програмної платформи, операційної системи та інструментів, що використовуються для проєктування та розробки продуктів. Всі галузеві керівництва з дизайну інтерфейсу сприяють ітеративному процесу проєктування інтерфейсу. Посібник з проєктування інтерфейсу IBM Common User Access (CUA) вперше описав ітеративний процес проєктування інтерфейсу та використав продукт продажу автосалону як тематичне дослідження.

Дизайн веб-інтерфейсів можна розділити на чотири області: вміст, взаємодія, навігація та графіка. Усі вони є важливими сферами, і їх відносна важливість для веб-сайту варіюється залежно від природи сайту та його використання.

Зміст - це цар, як часто кажуть. Сайт повинен мати вміст, який є цінним для відвідувачів, інакше у людей не буде причин відвідувати сайт. Залежно від типу вмісту, він також повинен бути своєчасним, точним, читабельним та для друку. Взаємодія визначає, як користувачі взаємодіють із сайтом. Чи використовує сайт текстові посилання, кнопки або карти зображень?Чи повинен користувач входити, щоб отримати інформацію? Чи пам’ятатиме сайт попередній сеанс користувача? Як програма може вчитися на поведінці користувача та надавати вміст, який бажає користувач?

Навігація визначає спосіб пересування користувачів на веб-сайті. Багато сайтів забезпечують навігацію лише на одному або двох верхніх рівнях; нижче цих рівнів користувач повинен використовувати кнопки навігації браузера. Чи відповідає структура навігації сайтом очікуванням користувача? Чи повинні користувачі розуміти організацію компанії, щоб користуватися сайтом?

Графіка створює стиль інтерфейсу та метафори для сайту. Вони додають естетики та насолоди від досвіду користувача. Споживчі сайти зазвичай підтримують високий ступінь естетичної привабливості, але це не обов'язково потрібно для успіху сайту. Якщо сайт орієнтований переважно на інформацію, графіка може відволікати користувачів з повільним інтернет-трафіком під час сканування, пошуку та читання інформації.

1.4. Принцип RAIL

RAIL - це модель, в якій призначений для користувача досвід розбивається на кілька ключових дій. Наприклад, клік, протягування (рух з натиснутою клавішою миші або тачпада), скролінг, завантаження.

RAIL встановлює цілі по продуктивності для цих дій. Наприклад, від кліка до відтворення дій з цього кліку має пройти не більше 100 мілісекунд.

RAIL забезпечує структуру для планування робіт з поліпшення продуктивності. Дизайнери і розробники можуть визначити ті моменти, поліпшення яких може дати найбільшу впливу, і працювати над ними.

RAIL - це абревіатура від англійських слів response (відгук), animation (анімація), idle (очікування) і load (завантаження). Ці чотири області слід розглядати для обговорення оптимізації продуктивності сайтів і додатків. Не будемо розглядати даний принцип надто глибоко. Його можна висловити наступним твердженням: якщо застосунок не має критичної потреби в анімаціях, зображеннях та різного роду медіа-файлах, то краще їх туди не добавляти. Це суттєво впливає на продуктивність.

1.5. Технології розробки інтерфейсу користувача

Для відображення роботи програми використовуються інтерфейси користувачів. Відповідно для веб-сайтів використовуються браузери, які в свою чергу обробляють HTML-документи, щоб згенерувати написаний код.

Мова розмітки гіпертексту HTML використовується для створення електронних документів (так званих сторінок), які відображаються у Всесвітній павутині. Кожна сторінка містить ряд з'єднань з іншими сторінками, які називаються гіперпосиланнями. Кожна веб-сторінка, яку ви бачите в Інтернеті, написана з використанням тієї чи іншої версії HTML-коду.

HTML-код забезпечує належне форматування тексту та зображень для Інтернет-браузера. Без HTML браузер не знав би, як відображати текст як елементи або завантажувати зображення чи інші елементи. HTML також забезпечує базову структуру сторінки, на яку накладаються каскадні таблиці стилів, щоб змінити її вигляд.

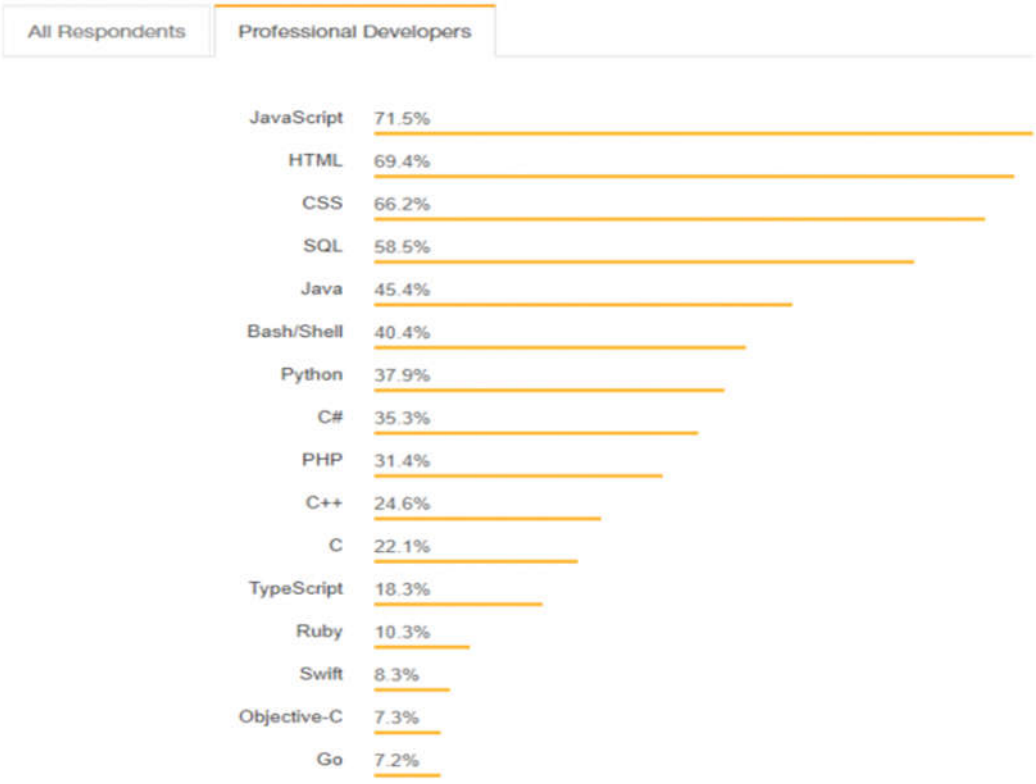

Рисунок 1.2. - Популярність технологій розробки програмного забезпечення, дані на 2018 рік. Джерело: stackoverflow

Мови програмування, мови маніпуляції базами даних, гіпертекстова розмітка, каскадна таблиця стилів – це все називається «технологіями розробки програмного забезпечення».

HTML, JS, CSS є базовими поняттями в розробці веб-сайтів. Ці технології є важливими та забезпечують створення сучасних веб-застосунків, для яких важливими характеристиками є:

1. Швидкодія.
2. Надійність (перевіряється автоматизованими тестами).
3. Верифікація запитів користувача / обробка помилок.
4. Захищеність даних.

Усе це дуже важко забезпечити використовуючи тільки базові технології, або і взагалі неможливо.

Важливість цих параметрів призвела до створення різного виду бібліотек, шаблонів та інфраструктури програмних рішень (надалі – фреймворк), що полегшує розробку складних систем.

Фреймворк відрізняється від бібліотеки тим, що бібліотека може бути використана в програмному продукті просто як набір підсистем функціональності, не впливаючи на архітектуру основного програмного продукту і не накладаючи на неї ніяких обмежень. Фреймворк диктує правила побудови архітектури додатку, задаючи на початковому етапі розробки поведінку за умовчанням, формуючи каркас, який потрібно буде розширювати і змінювати відповідно до зазначених вимог.

Фреймворк - це платформа для розробки програмних додатків. Наприклад, фреймворк може включати заздалегідь визначені класи та функції, які можна використовувати для обробки введення, управління апаратними пристроями та взаємодії з системним програмним забезпеченням. Це спрощує процес розробки, оскільки програмістам не потрібно винаходити колесо кожного разу, коли вони розробляють нову програму.

Можна виділити деякі переваги та недоліки використання фреймворків.

Переваги:

1. Код є більш безпечним.
2. Уникнення зайвого коду.
3. Встановлює правила дизайну коду.
4. Допомагає послідовно розробляти код із меншою кількість помилок.

Недоліки:

1. Довший час завантаження веб-застосунків.
2. Довший час розробки.
3. Встановлює правила дизайну – іноді потрібно порушити структуру програмного коду, що з використанням фреймворку зробити важко.

На 2019 рік у світі найпопулярнішими фреймворками є:

- Angular (використовує мову TypeScript)
- React (легкий фреймворк)
- VueJS

Фреймворк схожий на інтерфейс прикладного програмування (API), хоча технічно фреймворк включає API. Як випливає з назви, фреймворк служить основою для програмування, тоді як API забезпечує доступ до елементів, що підтримуються фреймворком. Структура може також включати бібліотеки коду, компілятор та інші програми, що використовуються в процесі розробки програмного забезпечення. Веб-фреймворк відрізняється від веб-сервера тим, що веб-сервер насправді запускає веб-додаток, тоді як веб-фреймворк більше нагадує віртуальну базу даних або бібліотеку, яка допомагає прискорити процес розробки та написання. Веб-фреймворки допомагають виконувати різноманітні завдання, починаючи від створення шаблонів та доступу до бази даних, до управління сеансами та повторного використання коду.

У Node.js для генерації та видачі HTML-сторінок використовуються шаблонізатори. Шаблонізатор Node.js представляє спеціальний модуль, що використовує більш зручний синтаксис для формування HTML на основі динамічних даних та дозволяє розділити представлення від контролера.

Налаштування Node.js шаблонізатора здійснюється заданням двох параметрів:

views - шлях до каталогів, у якому знаходяться шаблони;

view engine - вказівка самого шаблонізатора.

Шаблони Node js handlebars представляють собою звичайні файли HTML в форматі визначеним шаблонізатором, в яких за допомогою спеціального синтаксису виводяться дані, що передаються. Для відображення значення властивості переданого об'єкта використовується запис *{{{(назва властивості)}}}*.

Використання шаблонизатора покращує читабельність коду і спрощує внесення змін в зовнішній вигляд, коли проєкт повністю виконує одна людина.

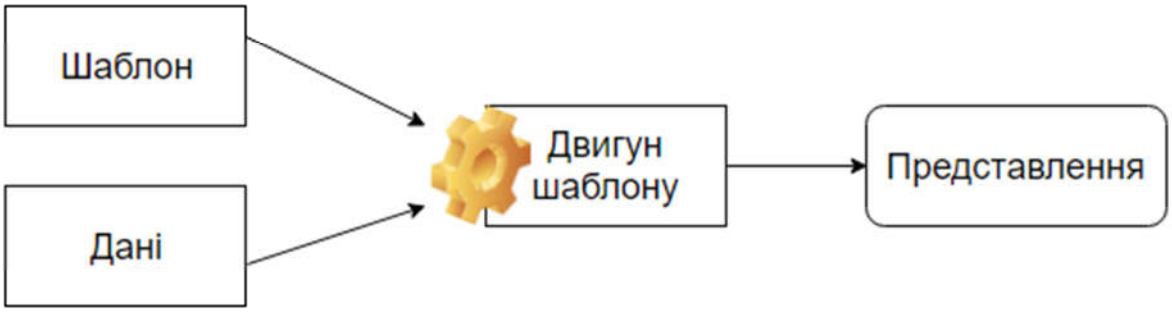

Рисунок 1.3. - Представлення роботи шаблонізатора.

Популярними шаблонізаторами є:

- Hogan.js
- Handlebars.js
- mustache.js
- doT.js
- Bug

1.6. Технологія Node.js

Node.js - це середовище виконання JavaScript, побудоване на механізмі JavaScript V8 Chrome. Node.js використовує керовану подіями неблокувальну модель введення-виведення, що робить його легким і ефективним. NPM - це пакет Node.js з бібліотеки з відкритим кодом, який є найбільшим серед своїх аналогів у світі.

Node створений для побудови масштабованих мережевих додатків. Він може працювати з багатьма одночасними підключеннями асинхронно. Створений у 2009 році.

Платформа Node не підтримує модель запиту/відповіді в багатопотоковому режимі. Натомість використовується однопотокова модель із використанням циклу подій (event loop). Node.js складається з двох основних компоненті: ядра і його модулів. Ядро: він побудований на C і C++. Ядро поєднує в собі JS-двигун Google V8 з бібліотекою Libuv і прив'язками протоколів, включаючи сокети та Http. Цикл подій – це програма, яка очікує на подію і коли подія буде створена – створить відповідну функцію, що буде виконуватись асинхронно разом з іншими функціями такого типу. Неблокуюча модель дозволяє обробляти запити асинхронно, а значить, функція, що обробить запит користувача матиме змогу першою вийти із сховища callback-функцій та видати результат користувачу [1, с. 19].

Варто також відмітити, що сервер може бути написаний на чистому Node.js. Проте існує великий список фреймворків, які спрощують синтаксис, для легшого читання коду. І всі вони є «надбудовою» відносно Node.

Найпопулярнішими є:

- Koa
- Express
- Hapi
- Next

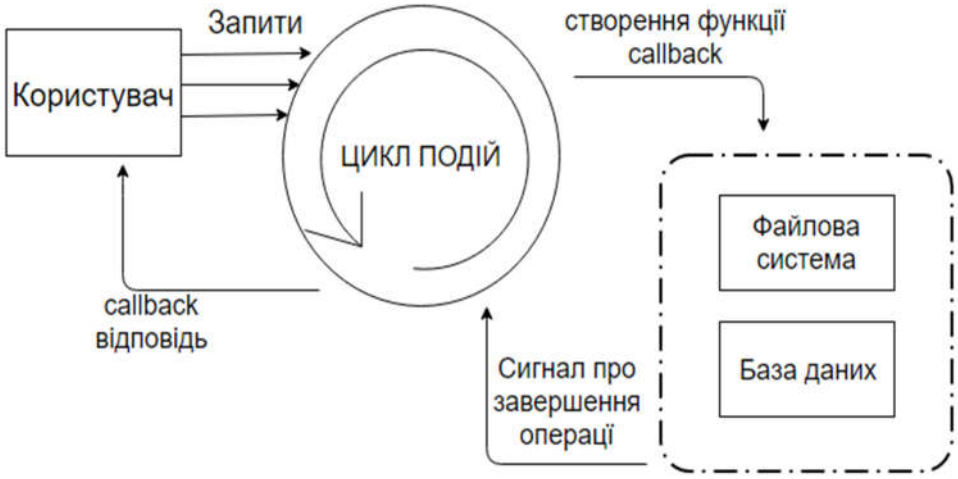

Рисунок 1.4. - Умовне представлення роботи циклу подій, що лежить в основі віртуальної машини V8.

Що це означає з практичної точки зору? Можливість використовувати JavaScript для написання інтерфейсу та серверної частини лежить в основі багатьох переваг Node.js:

- Велика спільнота. Node.js, будучи проєктом з відкритим кодом, заохочує підтримку та внесок, спрямований на вдосконалення та впровадження платформи. Це місія фонду, призначеного для постійного розвитку та вдосконалення Node.js. Таким чином, ви можете бути впевнені, що, з одного боку, Node.js завжди покращується, а з іншого боку, вже існує багато ресурсів для багаторазового використання.
- Надійність. Використання Node.js дозволяє організувати повний стек розробки JavaScript, що забезпечує швидкість і продуктивність програми.
- Масштабованість. Це справжня коштовність середовища розробки Node.js, оскільки дозволяє створювати додатки, які можуть легко розширюватися. Node.js чудово працює в системах, що використовують архітектуру мікросервісів або контейнеризацію, де масштабованість і гнучкість можна досягти швидко і легко.

- Чудова екосистема. NPM (менеджер пакунків Node.js) мітить сотні тисяч бібліотек для вирішення не складних, проте тривіальних проблем. Node використовує мову JavaScript. А це означає, що програмісти, які створюють інтерфейс клієнта з логікою, можуть з легкістю вивчити серверну розробку.
- Двигун V8. В середині Node лежить JavaScript двигун V8, який використовується в браузері Google Chrome та інших браузерах.
- Асинхронність. JavaScript спрощує написання асинхронного і неблокуючого коду, навіть при використанні одного потоку, оскільки використовуються функції зворотного виклику, що базується на подіях. Такий механізм виник в браузерах. При завантаженні даних, користувач повинен очікувати на виконання повного завантаження, при цьому блокуються всі інші операції можливі в системі. Такий підхід давно застарів і тому на наш час все базується на принципі асинхронності.

1.7. Загальний опис баз даних

База даних - це сукупність інформації, організованої таким чином, щоб її можна було легко отримати, керувати та оновлювати. Система баз даних - це інтегрована колекція відповідних файлів разом із деталями інтерпретації даних, що містяться в ній. В основному система баз даних - це не що інше, як комп'ютерна система ведення документації, тобто система, загальною метою якої є запис та підтримка інформації/даних.

Як правило, база даних - це організований збір супутньої інформації. Організована інформація або база даних служить базою, з якої можна отримати бажану інформацію або прийняти рішення шляхом подальшого розпізнавання або обробки даних. Люди користуються декількома базами даних у своєму повсякденному житті. Словник, телефонний довідник, каталог бібліотеки тощо є прикладом для баз даних, де записи розташовані в алфавітному або секретному порядку. Термін "дані" можна визначити як

значення атрибута сутності. Будь-яка колекція пов'язаних елементів даних об'єктів, що мають однакові атрибути, може називатися "базою бази даних". **Поширений збір даних не робить його базою даних; спосіб його організації для ефективного та ефективного використання робить його базою даних.**

На сьогодні виділяють два типи баз даних: SQL і NoSQL. В обох випадках БД можуть розміщуватись на хмарних середовищах.

Система управління базами даних (СУБД) - це програмна система, що забезпечує доступ до даних, що містяться в базі даних. Завдання СУБД - забезпечити зручний та ефективний метод визначення, зберігання та отримання інформації, що міститься в базі даних. СУБД взаємодіє з прикладними програмами, завдяки чому дані, що містяться в базі даних, можуть використовуватися багатьма програмами та користувачами. Крім того, СУБД здійснює централізований контроль над базою даних, запобігає доступу шахрайських або несанкціонованих користувачів до даних та забезпечує конфіденційність даних.

СУБД виконує декілька основних функцій:

- Зберігання даних, пошук та оновлення: база даних може спільно використовуватися багатьма користувачами, отже, СУБД повинна надавати можливість перегляду даних кільком користувачам і дозволяти їм зберігати, отримувати та оновлювати дані легко та ефективно.
- Словник даних та каталог: СУБД повинна пітримувати доступний користувачеві словник даних.
- Цілісність транзакції: транзакція - це послідовність кроків, що становлять деяку чітко визначену ділову діяльність. Для підтримки цілісності транзакцій СУБД повинна надавати засоби для користувача або прикладної програми для визначення меж транзакцій. Потім СУБД повинна здійснити зміни для успішних транзакцій і відхилити зміни для перерваних транзакцій.

- Служби відновлення: СУБД повинна мати можливість відновлювати базу даних у випадках збою системи. Джерела системних збоїв включають помилки оператора, збої головки диска та помилки програми.
- Контроль паралельності: Оскільки база даних є спільною для кількох користувачів, два або більше користувачів можуть спробувати отримати доступ до одних і тих самих даних одночасно. Якщо два користувачі намагаються одночасно оновити один і той самий запис даних, можуть виникнути помилкові результати.
- Механізми безпеки: Дані повинні бути захищені від випадкового або навмисного неправильного використання чи відволікання. СУБД забезпечує механізми контролю доступу до даних та визначення того, які дії може вжити кожен користувач.
- Послуги вбудованості: СУБД повинна надавати засоби, які допомагають користувачам у підтримці цілісності їх даних. Різні перевірки редагування та обмеження цілісності можуть бути запрограмовані на СУБД та її програмні інтерфейси. Зазвичай ці перевірки адмініструються через словник даних.

СУБД - це складна структура, яка використовується для управління, зберігання та управління даними та метаданими, що використовуються для опису даних. Система використовується великою кількістю користувачів для отримання та управління даними. Система складається з набору взаємопов’язаних компонентів:

1. Щонайменш одна особа, яка є власником бази даних і відповідає за неї.

2. Набір правил та взаємозв'язку, що визначає та регулює взаємодію між елементами бази даних.

3. Люди, які вводять дані в базу даних.

4. Люди, які отримують дані з бази даних.

4. Сама база даних.

Архітектура баз даних – це важливе питання для розробника програмного забезпечення.

Архітектура бази даних - це розробка структури бази даних, яка буде використовуватися для зберігання та управління даними, а не конструкція програмного забезпечення СУБД. Після завершення проєктування бази даних СУБД обробляє всі складні дії, необхідні для перетворення думки архітектора на структури в програмному забезпеченні, придатні для використання на комп'ютері.

Погано розроблена база даних, як правило, генерує помилки, які можуть призвести до неправильних рішень. Поганий дизайн бази даних може врешті-решт вилізти боком: організації, які використовують погано спроєктовані бази даних, часто зазнають невдач, оскільки їх менеджери не мають доступу до своєчасної (або навіть правильної) інформації, тим самим домінуючи в поганому дизайні бази даних.

Наявність СУБД дозволяє вирішити набагато складніші способи використання ресурсів даних, якщо база даних призначена для використання цієї доступної потужності. Види структур даних, створені в базі даних, і ступінь взаємозв'язку між ними відіграють важливу роль у визначенні ефективності СУБД. Тому проєктування бази даних стає найважливішим видом діяльності в середовищі бази даних.

Дизайн бази даних стає набагато простішим, коли використовуються моделі. Модель бази даних - це сукупність логічних конструкцій, що використовуються для представлення структури даних та відносин даних, знайдених у базі даних, тобто спрощених абстракцій реальних подій або умов. Якщо моделі не логічно обгрунтовані, конструкції баз даних, отримані з них, не забезпечать системи баз даних ефективною інформацією, отриманою з ефективної бази даних. Хороші моделі дають хороший дизайн бази даних, який є основою для хороших додатків.

Реляційна база даних - це тип бази даних, що зберігає та забезпечує доступ до точок даних, пов’язаних між собою. Реляційні бази даних базуються на реляційній моделі, інтуїтивно зрозумілому, прямому способі представлення даних у таблицях. У реляційній базі даних кожен рядок таблиці є записом з унікальним ідентифікатором, який називається ключем. Стовпці таблиці містять атрибути даних, і кожен запис зазвичай має значення для кожного атрибута, що полегшує встановлення взаємозв’язків між точками даних.

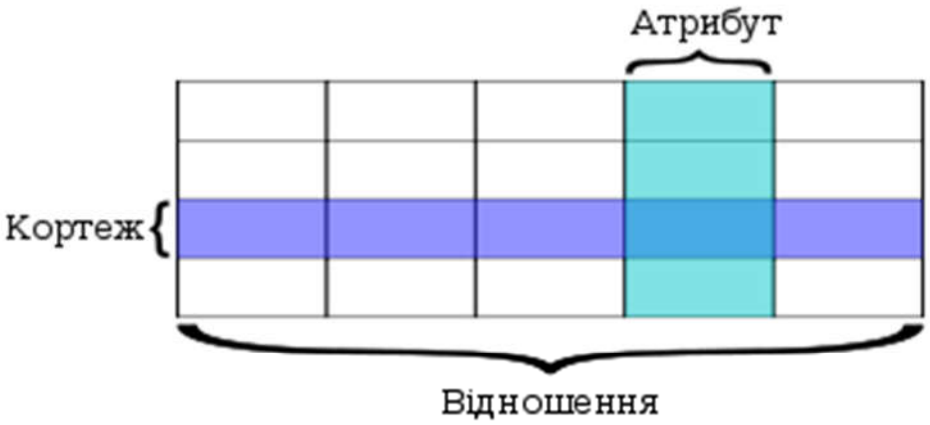

Рисунок 1.5. - Структура реляційної бази даних

Атрибутами реляційних баз даних є:

- Мова запитів SQL

SQL (мова структурованих запитів) - це стандартизована мова програмування, яка використовується для управління реляційними базами даних та виконання різних операцій над даними. Використання SQL включає модифікацію таблиць баз даних та структур індексів; додавання, оновлення та видалення рядків даних; і отримання підмножин інформації з бази даних для обробки транзакцій та додатків аналітики. Запити та інші операції SQL приймають форму команд, записаних як оператори - загальновживані оператори SQL включають вибір, додавання, вставку, оновлення, видалення, створення, та інші операції.

- Транзакції

Транзакції можна визначити, як «все або нічого», тобто існує деяке завдання, яка керується за допомогою операцій SQL, які повинні виконати поставлене завдання. Якщо завдання виконується успішно, то представляється результат виконаної роботи. В іншому випадку, навіть, якщо із 10 операцій не виконалась 1, то це означає помилку при роботі, як результат – не виконаються всі 10 операцій.

- Цілісність даних

Цілісність даних – це відповідність представлення даних. Вони повинні бути точними, відповідати суті представлення та нормалізованими.

- Нормалізація даних

Метою нормалізації є усунення недоліків структури БД, які призводять до шкідливої надмірності в даних, яка в свою чергу потенційно призводить до різних аномалій і порушень цілісності даних. Теоретики реляційних баз даних у процесі розвитку теорії виявили та описали типові приклади надмірності і способи їхнього усунення.

Для правильного представлення даних в базі використовують процес нормалізації. Існує 5 нормальних форм. Для першої нормальної форми:

- Кожна комірка таблиці повинна містити одне значення: цим самим уникати повторень не-ключових значень.
- Кожен запис повинен бути унікальним: іноді, цю вимогу називають атомарністю – кожен атрибут повинен мати одне унікальне значення, а не множину значень.

База даних перебуває у другій нормальній формі, якщо їй властиві наступні характеристики:

- БД уже знаходиться в першій нормальній формі.
- Для кожного кортежа в таблиці існує унікальний ключ (зазвичай ID).

Наступною є третя форма, якій відповідно властиві такі характеристики:

- БД знаходиться в другій нормальній формі.
- Не має перехідних функціональних залежностей – тобто зміна

одного неключової комірки не повинна викликати зміну іншої неключової комірки.

- Якщо жоден екземпляр таблиці бази даних не містить двох або більше, незалежних та багатозначних даних, що описують відповідну сутність, то це в 4-та нормальна форма.
- Таблиця знаходиться в 5-й нормальній формі, лише якщо вона в 4НФ, і її не можна розкласти на будь-яку кількість менших таблиць без втрати даних.

NoSQL - це нереляційна система управління базами даних (СУБД), яка не потребує фіксованої схеми. Бази даних NoSQL уникають об'єднань і їх легко масштабувати. Такі компанії, як Facebook, Google і Twitter, використовують NoSQL для своїх великих даних та веб-додатків у режимі реального часу, збираючи терабайти даних користувачів щодня.

Бази даних NoSQL працюють без наперед строго створеної схеми, дозволяючи додавати нові структурні елементи до існуючої бази без порушення цілісності початкової структури та записаних у базу даних.

Серед **переваг** варто відзначити:

1. Гнучка масштабованість, оскільки ці бази даних призначені для використання з недорогим апаратним забезпеченням.
2. Підтримка додатків для великих даних за допомогою баз даних NoSQL, здатних обробляти великі обсяги даних.
3. Динамічні схеми, оскільки бази даних NoSQL не потребують схем для початку роботи з даними.
4. Сумісність з дешевими товарними апаратними кластерами із збільшенням обсягів транзакцій та даних, що дозволяє обробляти та зберігати більше даних за меншими витратами.
5. Підтримка автоматичного шардінгу, що дозволяє базам даних NoSQL автоматично і автоматично розповсюджувати дані на довільній кількості серверів, не вимагаючи, щоб програма знала про склад задіяних серверів.

Недоліки NoSQL:

1. Відсутність норм стандартизації
2. Обмежені можливості запитів
3. Не працює так добре з реляційними даними
4. Проєкти з відкритим кодом не популярні для підприємств.

Існує декілька типів представлень NoSQL баз даних:

- База даних типу «ключ-значення»
- Документо-орієнтована база даних
- Графова база даних.
- Стовпчикова (колонкова) база даних.

Принцип ACID. Теорема CAP.

Акронім ACID — розшифровується як atomicity (атомність, неподільність), consistency (узгодженість), isolation (ізоляція), and durability (довговічність). Ці принципи дозволяють працювати з даними в безпечному режимі, уникаючи виникнення можливих неточностей та необережностей.

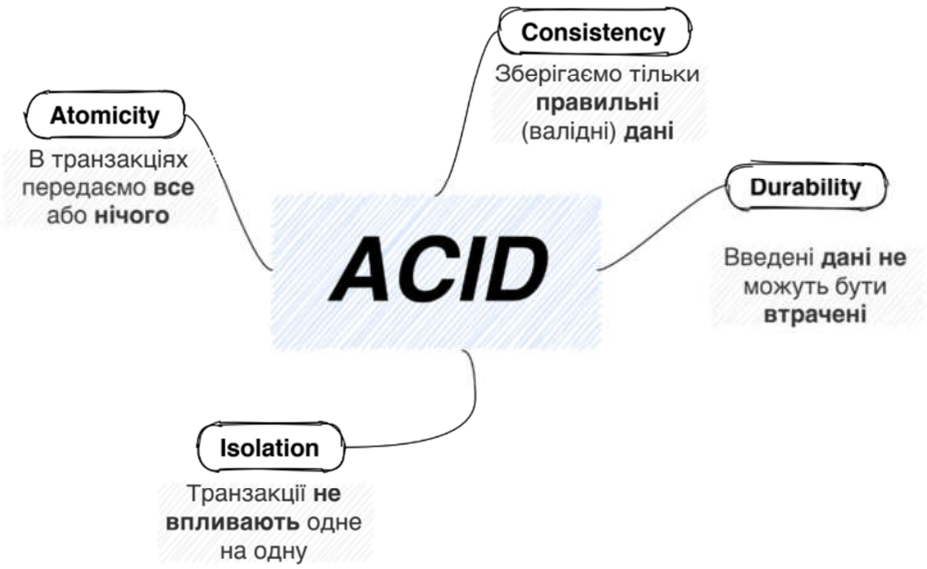

Рисунок 1.6. - Представлення властивостей ACID.

Транзакції реалізуються за допомогою декількох додатків (програмних застосунків) та хостів, які у розподіленій базі даних забезпечують застосування властивостей ACID.

Розглянемо кожну із цих властивостей більш детально:

1. Неподільність — це здатність СУБД гарантувати, що коли користувач створює деяку транзакцію, то варіантів розвитку буде два: транзакція виконається, і виконається успішно; транзакція не виконається. В цьому випадку діє правило “все або нічого”. Тобто якщо відбудеться збій в декількох байтах, з деяких причин вони виявляться недоступними або передадуть неправильну інформацію, то вся транзакція зазнає невдачі і не може бути виконана. Це дуже важливо при роботі з базами даних в банках.

2. Узгодженість гарантує, що база даних залишиться працювати стабільно, незалежно від того чи була виконана транзакція успішно чи ні. Перевіряється до початку транзакції і після її закінчення.

3. Ізоляція вимагає правильної роботи паралельних запитів — транзакції не пов’язані між собою, один користувач не має права бачити кінцевий або проміжний результат запиту іншого користувача.

4. Довговічність стверджує, що після здійснення транзакції її наслідки гарантовано зберігатимуться навіть у разі наступних збоїв.

Сьогодні програмні продукти розвиваються і змінюються з шаленою швидкістю, ринок вимагає від розробників ПЗ однієї важливої характеристики — адаптації, пристосування. Створений програмний продукт може виконувати чітко одну функцію для користувача, а уже через місяць до цієї однієї можуть добавитись пів дюжини інших функцій. В таких випадках реляційні бази даних дещо програють через свою особливість — чітку структуру. NoSQL - це відносно нова технологія СУБД для обробки великих обсягів неструктурованих даних, які не надходять у заздалегідь визначеному форматі. Більшість нереляційних баз даних включені в такі веб-сайти, як Google, Yahoo, Amazon і Facebook, які ініціюють безліч нових додатків з мільйонами користувачів. СУБД не може впоратися з проблемою величезного трафіку, тим самим ведучи до переходу до нового типу СУБД, яка здатна обробляти дані масштабу в мійльйони записів в одиницю часу, беручи їх з інтернету, нереляційно.

Найважливішою особливістю нереляційної бази даних є її масштабованість.

Бази даних NoSQL сумісні з ACID, проте вони змінюють цей підхід, зменшуючи кількість принципів із чотирьох до трьох, і називається такий підхід теоремою CAP, викладеною Еріком Брюерсом. Згідно цієї теореми є три основні функції при роботі з базами даних:

1. Узгодженість: користувачі можуть переглядати одні і ті ж самі дані одночасно.
2. Доступність: при умові доступу до мережі бази даних, з нею можна працювати: оновлювати, додавати дані або видаляти.
3. Розподіл: бази даних можуть розміщуватись на декількох серверах і вони є відмовостійкими.

А тепер повертаючись до теореми CAP: можна вибрати тільки два варіанти із трьох, тобто випадок, коли база даних характеризується трьома пунктами не є можливим.

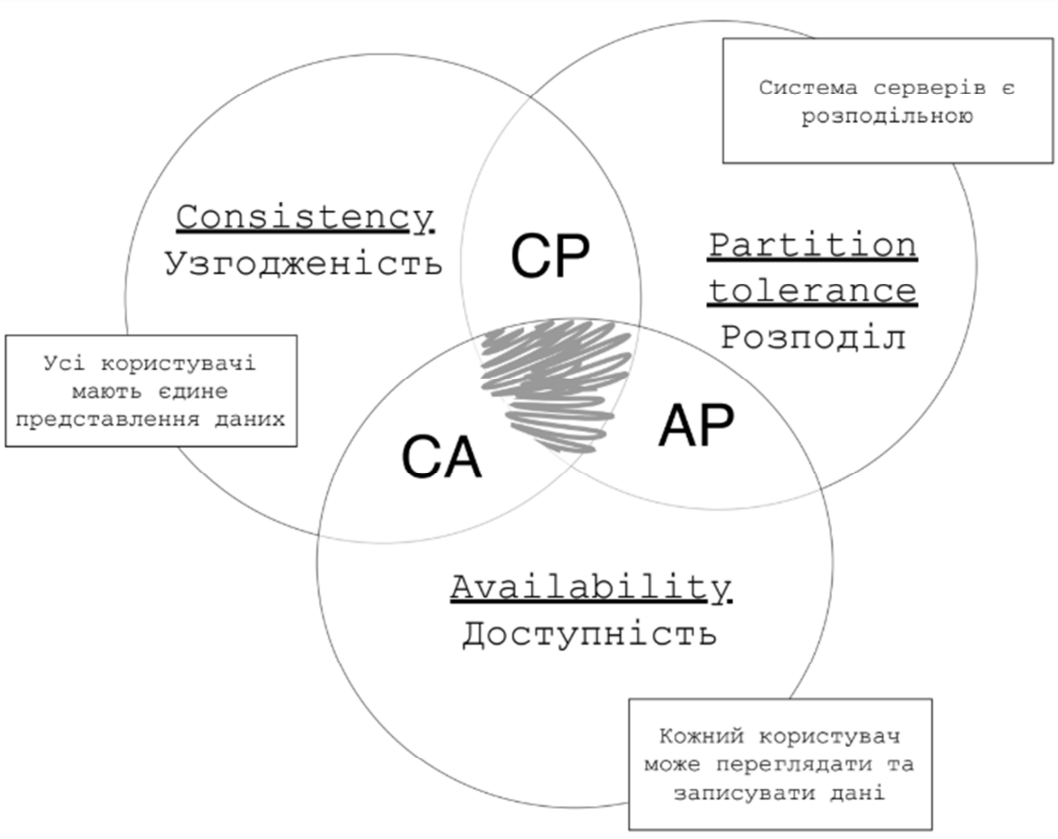

Рисунок 1.7. – Представлення теореми CAP у вигляді діаграми Венна.

PACELC. Ця теорема розширює теорему CAP. У ній говориться, що в разі поділу мережі, як і у випадку CAP, необхідно вибирати між доступністю і узгодженістю. Але навіть якщо поділу немає, потрібно робити вибір між затримками і узгодженістю.

У **висновку** до даного розділу можна відмітити, що мною було:

1. Описано технології сторвення сучасних веб-застосунків.
2. Стисло описано принципи архітектури інтерфейсів, що впливають на швидкодію.
3. Розглянуто типи баз даних.
4. Визначено характеристики, переваги та недоліки описаних баз даних.
5. Описані принципи, на які варто орієнтуватись при розробці структури бази даних.

РОЗДІЛ 2. МОДЕЛЬ СТВОРЕННЯ ІНТЕРФЕЙСУ ТА ПРОЄКТУВАННЯ БАЗИ ДАНИХ

2.1. Бібліотека React

React - це JavaScript-бібліотека для створення користувацьких інтерфейсів на різних платформах. Вона надає потужну ментальну модель для роботи і допомагає створювати декларативні і орієнтовані на компоненти призначені для користувача інтерфейси. Це те, що бібліотека React є в найширшому і загальному сенсі.

UI-компоненти створюються з React на мові JavaScript, а не на спеціальній мові шаблонів. Цей підхід, званий створенням складеного призначеного для користувача інтерфейсу, грає фундаментальну роль в філософії React. UI-компоненти React це автономні блоки функціональності, призначені для конкретної мети. Наприклад, можна уявити собі компоненти для вибору дати, контрольного зображення, введення адреси або поштового індексу. Такі компоненти володіють як візуальним представленням, так і динамічної логікою. Деякі компоненти навіть здатні на самостійні взаємодії з сервером.

Компоненти в React повинні бути такими, щоб їх можна було легко зрозуміти і інтегрувати з іншими компонентами. Вони повинні розвиватися відповідно до передбачуваним життєвим циклом, підтримувати власний внутрішній стан і працювати зі старим добрим JavaScript.

Компоненти - це інкапсульовані блоки функціональності, які є основою в React. Вони використовують дані (властивості і стан) для рендеринга призначених для користувача інтерфейсів. Деякі їх типи також надають набір методів життєвого циклу, які ви можете перехоплювати (hook), інакше - перехоплювачі життєвого циклу.

- Процес рендеринга (висновок і оновлення призначеного для користувача інтерфейсу на основі ваших даних) в React передбачуваний, і компоненти можуть підключитися до нього через API React.
- Бібліотеки React. React містить набір основних бібліотек. Основна бібліотека React працює з інтерактивними бібліотеками react-dom і react-native і орієнтована на специфікацію і визначення компонентів. Вона дозволяє створювати дерево компонентів, які можна використовувати для засобів візуалізації (рендеринга) браузера або іншої платформи. react-dom - один з таких засобів, призначений для рендеринга в браузері і на стороні сервера. Бібліотеки React Native сфокусовані на нативних платформах і дозволяють створювати React-додатки для iOS, Android і інших платформ.
- Сторонні бібліотеки. React не поставляється з інструментами моделювання даних, HTTP-викликами, бібліотеками стилів або іншими поширеними елементами інтерфейсного застосунку. І хоча цими загальними технологіями React не комплектується, широка екосистема складається з неймовірно корисних бібліотек.
- Запуск React-додатків. React-додаток запускається на платформі, для якої було розроблено. У магістерській роботі зосередження буде на веб-платформу, але інші проєкти, такі як React Native і React VR, дають можливість запускати додатки на інших платформах.

Нижче перераховані деякі переваги React перед іншими бібліотеками та фреймворками:

- Прості програми - React використовує компонентну архітектуру з чистим JavaScript, декларативний стиль програмування і потужні, зручні для розробника абстракції DOM (і не тільки DOM, а й iOS, Android і т. д.).
- Швидкий призначений для користувача інтерфейс - React забезпечує швидкодію завдяки своїй віртуальній моделі DOM і алгоритму інтелектуального узгодження (smart-reconciliation algorithm). Одна з побічних переваг - можливість тестування без запуску браузера з відсутнім графічним

інтерфейсом.

- Скорочення обсягу коду: величезне співтовариство React і гігантська екосистема компонентів надають в розпорядження розробника безліч різноманітних бібліотек і компонентів. Це важливий момент при виборі фреймворка, що буде використовуватись для розробки.

Концепція простоти в комп'ютерних технологіях високо цінується як розробниками, так і користувачами. Мова йде не про простоту використання - просте рішення може бути більш складним в реалізації, але в кінцевому підсумку воно виявляється більш елегантним і ефективним, а просте на перший погляд нерідко виявляється складним. Простота тісно пов'язана з принципом KISS (Keep It Simple, Stupid, то тобто «не ускладнюй»). Суть в тому, що прості системи краще працюють.

Підхід React дозволяє будувати простіші рішення із застосуванням кардинально поліпшеного процесу веб-розробки.

У React ця простота досягається завдяки наступним особливостям:

- Декларативний стиль (на відміну від імперативного): React віддає перевагу декларативним стилю замість імперативного, автоматично оновлюючи представлення.
- Компонентна архітектура, яка використовує чистий JavaScript: React не використовує для своїх компонентів предметно-орієнтовані мови (DSL, Domain-Specific Languages), тільки чистий JavaScript. Тут немає поділу, коли працюєш над однією і тією ж функціональністю.
- Потужні абстракції: в React є спрощений механізм реалізації в DOM, який дозволяє нормалізувати обробку подій і інші інтерфейси, що працюють однаково в різних браузерах.

Звичайно, майже у всього є недоліки. Це відноситься і до React, але повний список недоліків залежить від того, кого ви запитаєте. Деякі відмінності, наприклад декларативний стиль проти імперативного, найвищою мірою суб'єктивні.

Таким чином, вони можуть вважатися як достоїнствами, так і недоліками. Нижче наведено мій список недоліків React (як і будь-які списки такого роду, він може бути необ'єктивним):

- React не є повноцінним фреймворком на всі випадки життя (на зразок швейцарського ножа). Розробнику доводиться використовувати React в поєднанні з такою бібліотекою, як Redux або React Router, щоб досягти функціональності, порівнянної з Angular. Це також може стати перевагою, якщо вам потрібна мінімалістична UI-бібліотека для інтеграції з існуючим стеком.
- React використовує дещо нетрадиційний підхід до веб-розробки, а JSX і Flux (часто використовувані з React як бібліотеки даних) можуть здатися страхітливими для новачків.
- React не є реактивним (в сенсі реактивного програмування і архітектури, які в більшій мірі управляються подіями, володіють гнучкістю і швидкістю реакції) в початковому стані.

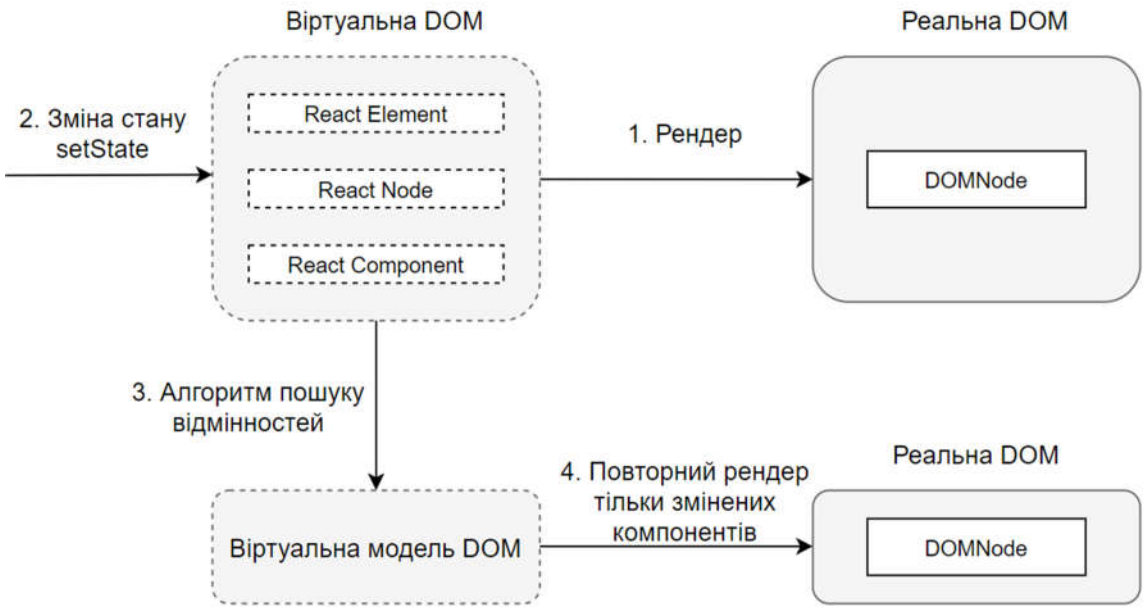

Рисунок 2.1. - Візуалізація компонентів в React.

2.2. Створення інтерфейсу

Інтерфейс створювався з використанням наступних технологій: React, Tailwind CSS. Проте, на етапі тестування, бібліотету Tailwind було вирішено виключити з програмного коду проекту через її маштабність, що впливає на швидкодію.

У даній роботі, при створенні інтерфейсу було використано технологію React, яка детально описана в попередьому підрозділі.

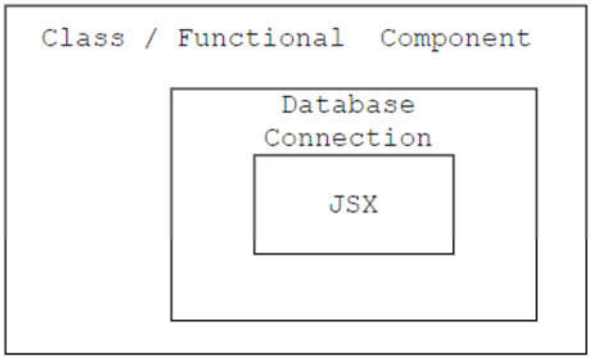

Рисунок 2.2. - Будова React-компоненту

Згідно рисунку 2.2, робота програми представляється в класовому або функціональному компоненті, при цьому можна використовувати два компоненти одночасно. Дана робота використовує класовий компонент, оскільки в ньому присутні дві ключові особливості: state (стан компоненту) та props (властивості). Використовуючи функцію, яка буде слідкувати за станом

компоненту, можна з легкістю керувати відображенням компонентів.

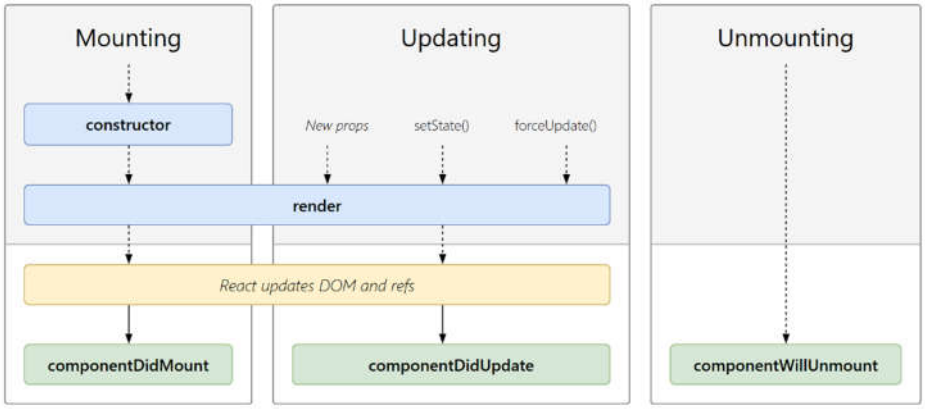

Рисунок 2.3. - Методи життєвого циклу React-компоненту.

При цьому зміна відображення буде відбуватись швидко та без перезавантаження сторінки.

Використовуючи синтаксис JSX можна «прив'язувати» (binding) результат виконання програмного коду до HTML-тегів.

```
<span class="block text-6xl pb-2  transform hover:scale-150">{fixedHum}</span>
    <span class="text-sm text-grey text-2xl">%</span>
    {fixedHum > 90 ? alertHum : humIsGood}
```

Рисунок 2.4. - Представлення «прив'язування» логіки до до HTML-тегу.

На рисунку 2.4 представлено як змінна fixedHum «прив'язується» до тегу span. Окрім цього на рисунку також представлено, що в HTML-тег можна і прописувати логіку: в даному випадку використовується перевірка if-else на значення змінної fixedHum, якщо значення є більшим 90, то тоді повинний виконатись код, що записаний в змінній alertHum, в іншому випадку використовується код humIsGood.

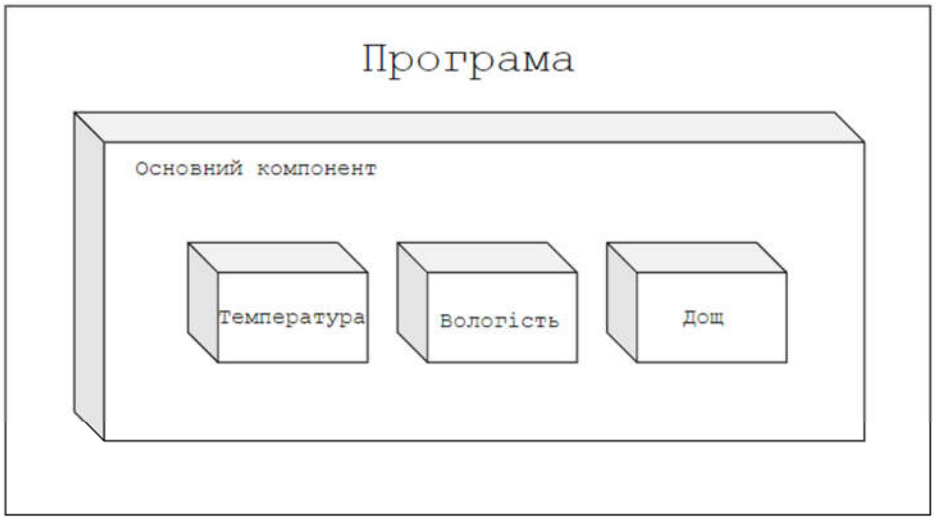

Рисунок 2.5. - Представлення вкладеності компонентів.

На рисунку 2.5 представлено розбиття на компоненти різних представлень для користувача, при чому зміна стану одного із компонентів не буде викликати перезавантаження інших, навіть у випадку коли змінюється дочірній компонент, процес рендерингу не торкнеться батьківськоо компоненту.

Tailwind CSS – бібліотека стилів, що є у відкритому доступі і має уже готові компоненти, які можна вільно добавляти в свої проєкти. Наприклад: навігаційна панель, кнопки, зображення та інші. Використання стилів.

Tailwind CSS - це високо настроюваний, низькорівневий фреймворк CSS, який надає вам усі «будівельні блоки», необхідні для створення індивідуальних проєктів, без будь-яких настирливих стилів, з якими доводиться довго працювати для правильного відображення.

Бібліотека постачається з усіма заздалегідь розробленими компонентами, такими як кнопки, картки та сповіщення. TailwindCSS відрізняється тим, що саме ці розроблені компоненти є відкритими, тобто можна використати уже наявний компонент і внести власні корективи в код CSS під власні потреби. Замість закритих компонентів, бібліотека пропонує низькорівневі утилітні класи, які дозволяєють створювати повністю власні дизайни.

2.3. Створення проєкту та моделі бази даних

Клієнти можуть отримати доступ до бази даних через Інтернет від постачальника послуг хмарних баз даних. Іншими словами, хмарна база даних призначена для віртуалізованого комп'ютерного середовища. Хмарна база даних реалізована за допомогою хмарних обчислень, що означає використання програмних та апаратних ресурсів постачальника послуг хмарних обчислень. Хмарні обчислення розвиваються дуже високими темпами в IT-галузі у всьому світі. Багато компаній почали рухатися до хмарних обчислень і отримувати доступ до своїх даних із хмарних баз даних. Хмарні обчислення можна назвати новим виміром в IT-світі з точки зору економії коштів і швидшої роботи додатків. Хмарна база даних здебільшого використовується як сервіс (або ж послуга). Її також називають База даних як сервіс (DBaaS). База даних повинна бути доступною постійно, щоб користувач міг отримувати дані коли завгодно. Хмарною базою даних повинна легко керуватись адміністратором, а також вона повинна зменшити витрати.

Хмарні обчислення дуже ефективно відновлюють інформацію після збоїв в базі даних.

Працює багато різних постачальників послуг хмарних баз даних, які надають базу даних як послугу, яка далі поділяється на три основні категорії. Існують: реляційна база даних, нереляційна база даних та операційна віртуальна машина, завантажена програмним забезпеченням для локальних баз даних, таких як SQL. Існують різні компанії, що пропонують базу даних як послугу, DBaaS, такі як Amazon RDS, Microsoft SQL Azure, Google AppEngine Datastore та Amazon SimpleDB. Кожен постачальник послуг відрізняється від іншого залежно від якості та виду послуг, що надаються. Вибір DBaaS залежить не тільки від послуг, що надаються компанією, але також залежить від вимог компанії. Існують певні параметри, які можна взяти як орієнтир для вибору найкращих DBaas.

- Розмір сховища.

Кожен постачальник DBaaS має різну ємність зберігання даних в базі даних. Розмір даних дуже важливий, оскільки компанія повинна бути впевнена у розмірі даних, які вони будуть зберігатись у своїй базі даних. Наприклад, Amazon RDS дозволяє користувачеві зберігати до 1 ТБ даних в одній базі даних, з іншого боку, SQL Azure пропонує лише 50 ГБ даних для однієї бази даних.

- Портативність.

База даних повинна бути портативною, оскільки база даних ніколи не повинна бути поза доступом користувача. Постачальник послуг може припинити свою діяльність, тому база даних та збережені дані можуть бути знищені. Якщо такі речі трапляються, повинен бути план надзвичайних ситуацій. Цю проблему можна вирішити, взявши хмарні сервіси в інших компаній, щоб база даних була доступна навіть у надзвичайних ситуаціях.

- Транзакції.

Можливості транзакцій є основною особливістю хмарної бази даних, оскільки завершення транзакції дуже важливо для користувача. Користувач

повинен знати, чи була транзакція успішною чи ні. Є компанії, які здебільшого здійснюють транзакції з грошима, і в цій ситуації необхідно виконати повну операцію читання та запису. Користувачеві потрібна гарантія здійсненої ним транзакції, і така транзакція називається транзакцією ACID.

- Можливість налаштування.

Є багато баз даних, які користувач може легко налаштувати, оскільки більшу частину конфігурації робить постачальник послуг. Таким чином даний пункт можна розглядати з двох сторін: як перевага через зручність та недолік через обмеження до повного доступу налаштування.

Переваги хмарної бази даних.

Хмарні обчислення створили нову гілку в IT-індустрії, і компанії прагнуть застосувати хмарні сервіси, а не інвестувати величезні гроші в отримання інфраструктури для власної системи баз даних. Уже на сьогоднішній день хмарні обчислення є доволі великою частиною ринку програмного забезпечення.

Є цілий ряд особливостей, які вважаються перевагами, в першу чергу це стосується економічної складової. Якщо компанії не використовують хмарні

технології, то їм доводиться вкладати великі кошти на створення власних центрів зберігання та обробки даних, а після наймати працівників, які будуть працювати з цими даними та заново “винаходити велосипед”. Нижче опишу більш детально, про які саме переваги йде мова:

1. Інтернет. Технологія змінила правила ведення бізнесу, тепер люди прагнучи зекономити свій час користуються послугами інтернет-магазинів. Така зміна змусила людей думати, як зробити інтернет-купівлю швидкою та зручною для користувача.

Раніше, доводилось встановлювати програмне забезпечення бази даних, створювати доступ до даних та слідкувати за оновленнями, які випускають компанії, потім реалізувати зміну свого коду під оновлення, яке вийшло. Зараз же, це може бути спрощено за допомогою хмарних технологій.

2. Інша перевага використання хмарної технології полягає в тому, що це економить багато грошей. Компанії не потрібно вкладати великі кошти в

створення власних центрів зберігання та обробки даних, а потім в управління ними, наймаючи додатковий персонал. Більше того, після створення центру обробки даних компанії потрібно буде також купити ПЗ, яке після потрібно обслуговувати.

3. Постачальники послуг хмарних баз даних пропонують масштабованість у пікові години, коли база даних повинна була б “упасти”.

4. Хмарні бази даних дають можливість доступу до інформації з будь-якого місця, використовуючи будь-який електронний пристрій, без будь-який обмежень до доступу до вашого персонального комп’ютера. Це дуже потужна технологія і компанії віддають перевагу саме їй.

5. Також є і інші переваги, які є менш значними. Наприклад — обсяг даних, який може збільшитись одним натискання клавіші миші.

Проте, говорити, що хмарні технології вирішать всі ваші проблеми не будемо. Звісно кожна монета має дві сторони. Нижче опишу **недоліки**, які відносяться до хмарних технологій.

1. Оплата в чітко визначені періоди. Якщо оплата не пройшла в визначений час, база даних закінчує свою роботу і дуже добре, якщо постачальник послуг тільки відключає доступ до самої бази, а не видаляє наявні там дані. Це свого роду як квитанція за комунальні послуги, тільки в цьому випадку за доступ до інформації. Іноді компаніям доводиться переплачувати, якщо виявиться, що трафік був високим і доводилось виходити за межі договору або створювати масштабування фізичного серверу.

2. Інший недолік — відсутність повного контролю над сервером, на якому знаходиться база даних. Ми не можемо контролювати програмне забезпечення, встановлене на цих серверах. Ви, як розробник, не впливаєте на безпеку серверу чи бази даних. Для деяких компаній проблема безпеки може виявитись дуже цінною.

3. Дані, розміщені у хмарі, повністю залежать від постачальника послуг. Дані та інформація про користувачів є найважливішим активом компаній. Ці дані не повинні потрапляти в руки “третіх людей”, а якщо таке сталось, то

компанія зазнає величезних втрат в фінансовому та репутаційному секторах.

4. Оскільки в хмарні базі даних розміщується великий масив даних, то для отримання їх потрібен високошвидкісний інтернет зв'язок. З іншого боку з локальної бази даних вибірка буде набагато швидшою.

5. Якщо клієнт хмарної бази даних захоче змінити постачальника послуг, виникнуть деякі проблеми, а саме різні хмари можутьвикористовувати різні технології та свої власні методи зберігання та обробки даних.

6. У випадку з хмарною базою даних дані слід отримувати через Інтернет, тому, якщо сервер не працює, це може спричинити неможливість доступу до даних із сервера. Це спричиняє величезні втрати, коли інформація недоступна у разі потреби.

Для того, щоб створити базу даних потрібно перейти на сайт https://console.firebase.google.com/ та зареєструватись. Реєстрація відбувається за допомогою Google-аккаунту. Наступним кроком буде створення проєкту:

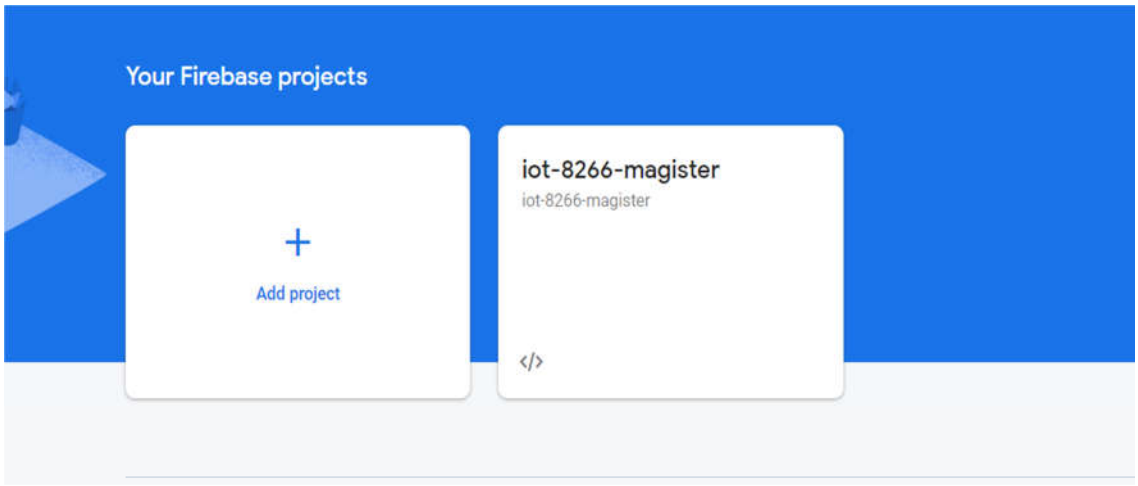

Рисунок 2.6. - Створення проєкту.

Як Ви могли замітити, мною було уже створено проєкт iot-8266-magister.

Наступним кроком створюємо модель бази даних, вибравши графу Realtime Database:

Рисунок 2.7. - Представлення моделі бази даних в реальному часі.

Ключовими полями є humidity, rain, temperature – саме сюди будуть завантажуватись дані із давачів за допомогою WiFi-з'єднання.

Нажавши на шестерню налаштувань та вибравши пункт Project settings > General копіюємо конфігураційні дані доступу до хмарного середовища Firebase.

```
const firebaseConfig = {
  apiKey: "      персональний ключ          ",
  authDomain: "iot-8266-magister.firebaseapp.com",
  databaseURL: "https://iot-8266-magister.firebaseio.com",
  projectId: "iot-8266-magister",
  storageBucket: "iot-8266-magister.appspot.com",
  messagingSenderId: "204368577001",
  appId: "1:204368577001:web:5e8e8cd780c44e8e298b78"
};
```

Рисунок 2.8. - Представлення конфігураційних даних.

За допомогою цих даних встановлюється зв'язок із хмарним середовищем. Замітьте, що встановлюється зв'язок тільки із хмарним середовищем. Наразі без доступу до бази даних. Для доступу до бази даних потрібно використати ключ доступу, який знаходиться в Settings > Service accounts > Database secrets.

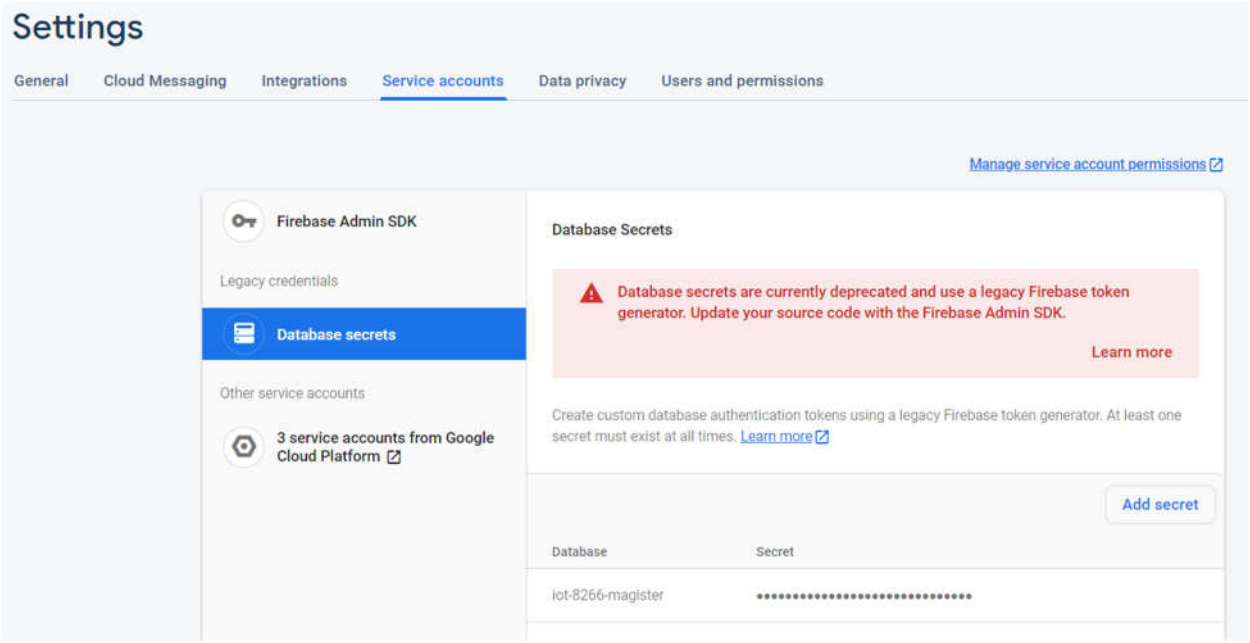

Рисунок 2.9. - Представлення місцезнаходження секретного ключа.

У висновку до даного розділу виділимо наступні пункти:

1. Описано сучасні технології створення веб-застосунків – React.
2. Визначено ключові особливості компонентної архітектури React.
3. Описані переваги та недоліки даної технології.
4. Описано особливості хмарних баз даних.
5. Детально описано створення бази даних Firebase.

РОЗДІЛ 3. СТВОРЕННЯ ТА ПРОГРАМУВАННЯ СИСТЕМИ АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Реалізація робочої системи

Перед вибором компонентів, мною було складено схему роботи такої системи:

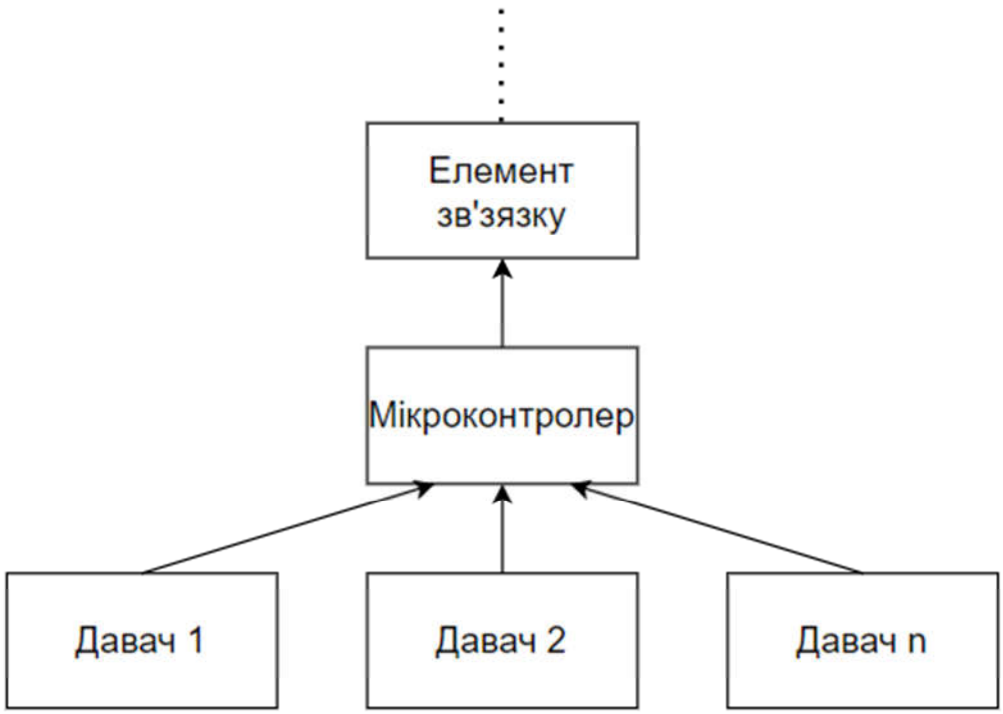

Рисунок 3.1. - Схема взаємодії елементів виконуваного проєкту (апаратна частина).

Аналізуючи ринок мікроконтролерів, мною було вибрано ESP8266, оскільки у ньому є вбудований Wi-Fi модуль, що на рисунку вище відповідає блоку «Елемент з’язку». А це в свою чергу зменшує кількість зовнішніх компонентів та збільшує ергономіку пристрою. Відповідно, керуючись ціновою політикою сучасного ринку, було вибрано саме названий вище мікроконтролер.

Тоді схема, представлена вище набуватиме такого вигляду:

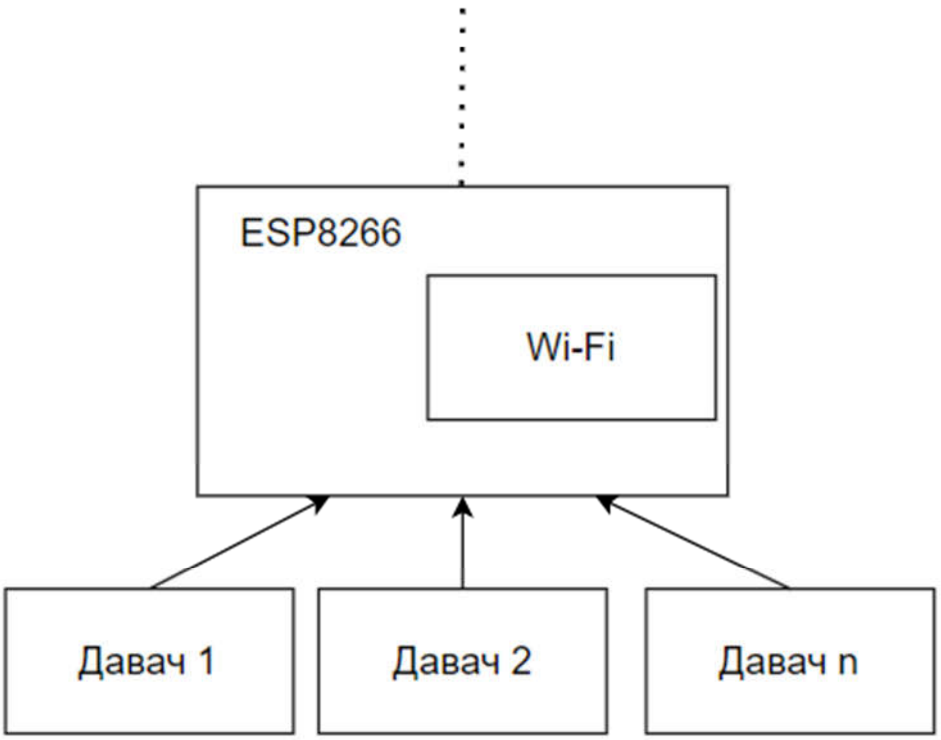

Рисунок 3.2. - Схема взаємодії елементів виконуваного проєкту після вибору мікроконтролера.

Більш детально про характеристики ESP8266 NodeMCU:

Плата оснащена модулем ESP-12E, що містить мікросхему ESP8266 з RISC мікропроцесором Tensilica Xtensa® 32-bit LX106, який працює з регульованою тактовою частотою від 80 до 160 МГц.

Характеристики ESP-12E:

- 32-розрядний LX106 від Tensilica Xtensa®
- Тактова частота від 80 до 160 МГц
- 128 КБ вбудованої оперативної пам'яті
- 4 МБ зовнішньої зовнішньої флеш-пам'яті
- Приймач Wi-Fi 802.11b / g / n.

Також даний модуль має 128 КБ ОЗУ і 4 МБ флеш-пам'яті (для зберігання програм і даних), достатніх, щоб впоратися з великими рядками, які складають веб-сторінки, даними в JSON / XML і всім, що ми сьогодні

додаємо на пристрої IoT. Оскільки діапазон робочої напруги ESP8266 становить від 3 В до 3,6 В, дана плата для підтримки постійної напруги на рівні 3,3 У поставляється з LDO стабілізатором напруги. Він може надійно забезпечувати струм до 600 мА, чого повинно бути більш ніж достатньо, оскільки ESP8266 під час радіочастотних передач споживає до 80 мА. Вихід стабілізатора також виводиться на висновки на сторонах плати і позначений як 3V3. Ці висновки можна використовувати для подачі живлення на зовнішні компоненти.

Вимоги до живлення:

- Робоча напруга: від 2,5 до 3,6 В
- Вбудований стабілізатор: 3,3 В, 600 мА
- Робочий струм: 80 мА
- Споживання в сплячому режимі: 20 мкА

Живлення до ESP8266 NodeMCU подається через вбудований USB-роз'єм MicroB. В якості альтернативи можна використовувати вивід VIN для безпосереднього живлення ESP8266 і його периферії.

Мультиплесковані I/O виводи:

- 1 канал АЦП
- 2 інтерфейсу UART
- 4 виходи ШІМ
- Інтерфейси SPI, I2C

На платі встановлений контролер USB-UART CP2102 від Silicon Labs, який перетворює USB сигнал в сигнал послідовного порту і дозволяє комп'ютеру програмувати і взаємодіяти з мікросхемою ESP8266.

Послідовний зв'язок:

- USB-UART перетворювач CP2102
- Швидкість зв'язку 4,5 Мбіт/с
- Підтримка управління потоком

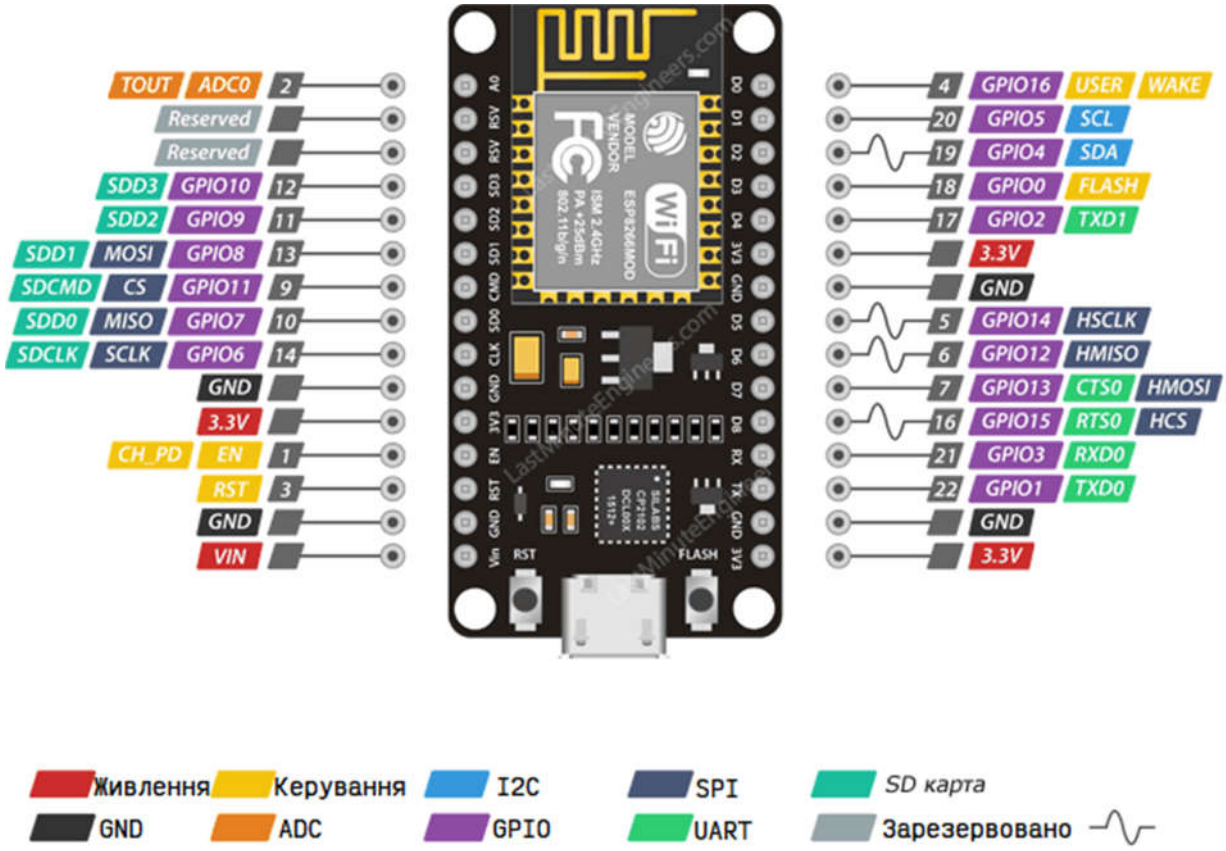

Рисунок 3.3. - Схема розташування виводів ESP8266 NodeMCU.

Є декілька варіантів кодування мікроконтролера, наприклад NodeMCU має власну вбудовану скриптову мову Lua, яка має методи для роботи із створенням, налаштуванням точки доступу. Проте, мною було використано середовище розробки Arduino, і весь програмний код написаний на мові С, оскільки передбачається не тільки робота з точкою доступу мережі, а також робота з базою даних. Тому весь код буде виглядати «читабельним», якщо він буде написаний з використанням однієї технології. Для роботи з ESP8266 NodeMCU в Arduino потрібно встановити додаткові бібліотеки та обов'язково додати нову плату через менеджер плат.

Вибір давача для генерації даних.

При виборі давачів основним критерієм була фінансова вартість по відношенню до характеристик. Вибір впав на давач температури DHT22 та давач даних дощу YL-83.

Більш детально по характеристиках кожного.

Давач дощу YL-83:

- Зібраній на мікросхемі LM393
- Сенсорна поверхня: 5 х 4 см захищено від окислення, провідності, має Висока якість и довговічність.
- Налаштування чутлівості с помощью вбудований потенціометра
- Цифровий и аналоговий виходи
- Напруга живлення давача 3,3 - 5В
- Контакти: живлення, земля, цифровий вивід, аналоговий вивід

Давач температури DHT22:

DHT має 4 виводи:

1. Vcc (3-5V живлення)
2. Data out - виведення даних
3. Не використовується
4. Загальний

- Живлення: від 3.3 до 5.5 В.
- Допустимий діапазон: - 40 С . +80 С.
- Максимальна погрішність: ±1С.
- Градація шкали: 0.1°С.
- Допустимий діапазон: 0.99.9%
- Мінімальна погрішність: ±2% +25 С
- Максимальна погрішність: ±4%
- Градація шкали: 0.1%
- Мінімальний час між зчитуваннями показів: 2 сек.

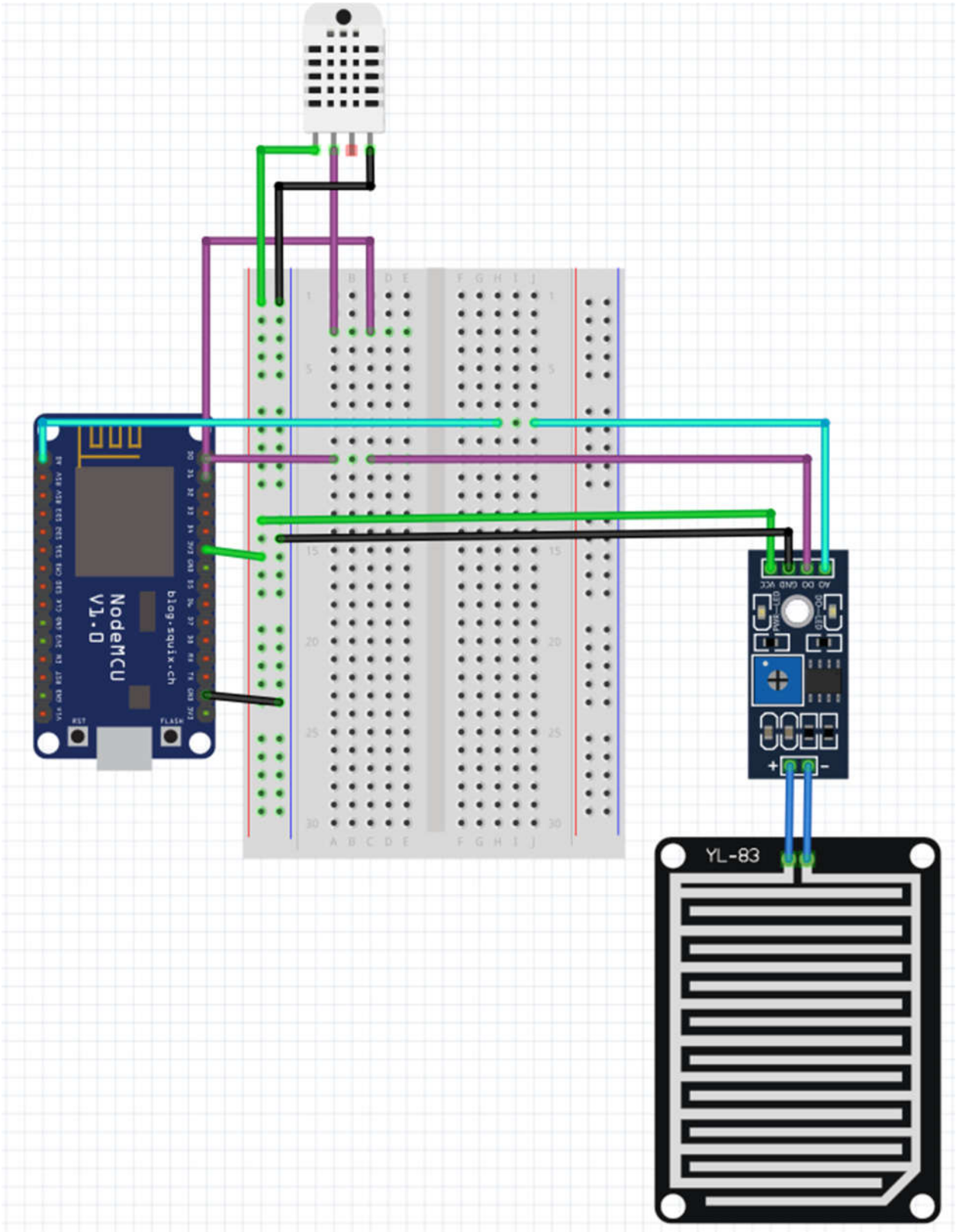

Рисунок. 3.4. - Схема з’єднання фізично наявних елементів.

3.2. Application Programming Interface (API)

Веб-API є невід'ємною частиною нашого взаємопов'язаного світу. Програмне забезпечення використовує такі інтерфейси для обміну даними — від застосунків на смартфонах до серверів баз даних, API-інтерфейси на сьогодні присутні майже всюди. Вони можуть бути простими технічними інтерфейсами або продуктами самі по собі, цілі системи, незалежно від розмірів та призначень, залежать від них. Те ж саме роблять цілі компанії і організації, починаючи з технологічних стартапів та інтернет-гігантів, закінчуючи нетехнічними малими і середніми підприємствами, великими корпораціями та державними структурами.

Мільярди людей володіють смартфонами і використовують їх для обміну фотографіями в соціальних мережах. Без API це було б неможливо. Обмін фотографіями з допомогою мобільного додатку для соціальних мереж передбачає використання різних типів API.

По-перше, щоб зробити фото, мобільний додаток соціальної мережі використовує камеру смартфона за допомогою API. Потім через API він може використовувати якусь бібліотеку зображень, вбудовану в додаток, для інвертування кольорів фотографії. І в підсумку він ділиться зміненої фотографією, відправляючи її на серверний додаток, розміщений на сервері соціальної мережі, використовуючи віддалений API, доступ до якого можна отримати через мережу, зазвичай через інтернет. Таким чином, в цьому сценарії задіяні три різних типи API: апаратний API, бібліотека і віддалений API, відповідно.

API-інтерфейси, незалежно від свого типу, спрощують створення програмного забезпечення, але віддалені API, особливо веб-API, змінили спосіб створення програмного забезпечення. В даний час будь-хто може легко створити що-небудь, зібравши різні частини програмного забезпечення.

Суворо кажучи, API - це тільки інтерфейс, що надається деяким програмним забезпеченням. Це абстракція базової реалізації (базовий код - це

те, що насправді відбувається всередині програмного продукту при використанні API).

Буває і таке, що термін API часто використовується для позначення всього програмного продукту, включаючи API і його реалізацію.

Мобільний додаток, що працює в смартфоні, використовує API, що надається серверним додатком (часто званим бекенд додатком або просто серверної частиною), яке розміщене на віддаленому сервері. Тому мобільний додаток називається споживачем, а серверна частина називається постачальником. Дані терміни також застосовні до компаній і до людей, що створюють додатки, або використовують або надають API. Тут це означає, що розробники мобільного застосування є споживачами, а розробники бекенд-додатків — постачальниками.

Для обміну даними зі своїм бекенд-додатком мобільний додаток зазвичай використовує мережу інтернет. Тут цікавий не самий інтернет - такий обмін даними також може здійснюватися через локальну мережу, - а те, як ці два додатки обмінюються даними по мережі. Коли мобільний додаток відправляє фотографію і повідомлення бекенд-додатком, то він використовує протокол передачі гіпертексту (HTTP).

Тисячі, навіть мільйони мобільних додатків і їх бекенд були створені завдяки веб-API, але це ще не все. Веб-API розвивають творчий потенціал та інновації, перетворюючи програмне забезпечення в багаторазові використовувані блоки, які можна легко зібрати. У світі API існує два типи блоків, що представляють два типи API: відкриті API і закриті API.

Відкриті API-інтерфейси пропонуються в якості сервісу або продукту іншим; ви не створюєте їх, не встановлюєте і не запускаєте - ви тільки використовуєте їх. Відкриті API надаються всім, хто їх потребує і готовий прийняти умови стороннього постачальника. Залежно від бізнес-моделі провайдерів API такі API можуть бути безкоштовними або платними, як і будь-який інший програмний продукт.

Такі відкриті API-інтерфейси розкривають творчий потенціал і інновації, а також можуть значно прискорити створення всього того, про що ви можете мріяти.

Закритий API — це API, який створюєте безпосередньо самі ви, та в результаті ви виступаєте кінцевим споживачем, створеного API.

Питання відкритості / закритості не в тому, як API надається, а кому. Навіть будучи розміщеним в інтернеті, API може бути закритим.

Проєктування API має значення, тому що, коли люди використовують API, вони хочуть використовувати його, не турбуючись про дрібниці, які не мають до них ніякого відношення. А для цього розробка повинна приховувати деталі реалізації (що насправді відбувається).

- RESTful API

Перед визначенням RESTful API, розберемо спочатку, що таке REST.

REST розшифровується як Representational State Transfer, в перекладі на українську — передача репрезентативного стану. Це набір архітектурних принципів для комунікаційної мережі. Мережі, які розроблені згідно даних принципів, характеризуються двома ключовими особливостями: масштабованістю та гнучкістю. Дані принципи можуть відповісти на наступні питання: що є компонентом RESTful системи? Як компоненти вза'ємодіють одне з одним? Як можна змінювати (переміщати) компоненти один відносно інших? Як відбувається масштабування системи?

Термін REST був започаткований в 2000 році Роєм Філдінгом, який є одним із авторів HTTP-протоколу. Системи, що підтримують REST, називаються RESTful-системами.

В загальному визначенні REST — це дуже простий інтерфейс керування даними або інформацією. Кожна одиниця інформації визначаються URL параметром, кожна адреса URL має чітко задану структуру, представлену нижче:

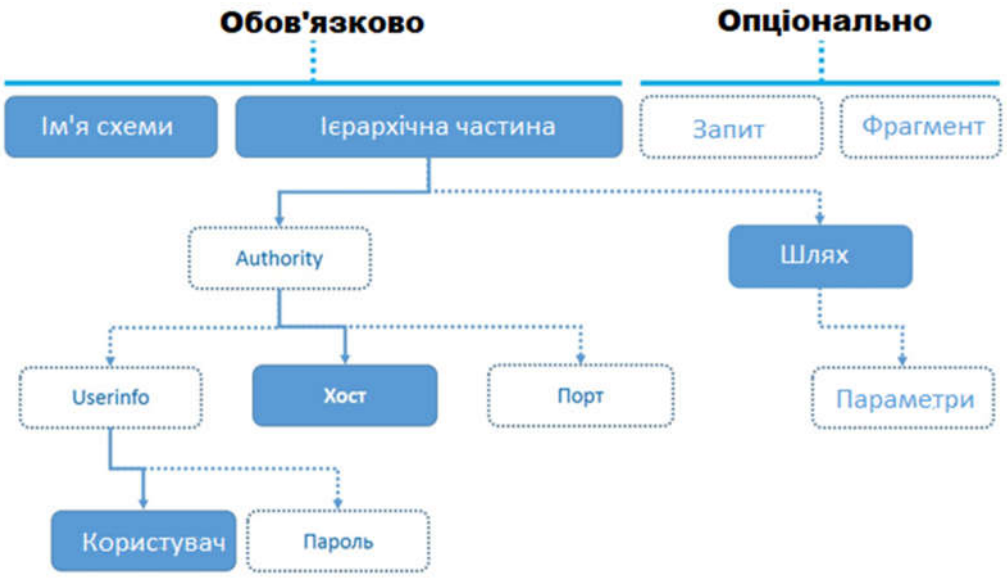

Рисунок 3.5. - Структура URL.

- URI — уніфікований ідентифікатор ресурс. URI — це щось глобальне, яке містить в собі два елементи: URL, що вказує на місцезнаходження ресурсу та URN — ім’я даного ресурсу.
- URL (Uniform Resource Locator), як сказано вище вказує на місцезнаходження ресурсу і також — метод, за допомогою якого здійснюється доступ, наприклад HTTP-протокол із методом GET.

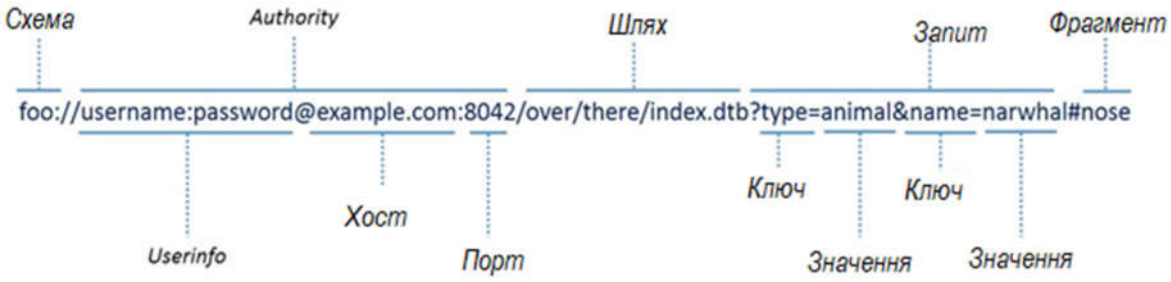

Рисунок 3.6. - Розшифрування пошукової адреси.

Отже, ми дали визначення адреси, за якою можна звернутись в глобальній мережі та отримати (в більшості випадків) або передати дані. В загальному випадку — робота з ресурсами. Саме із цим пов’язана дана магістерська робота.

Ресурс у даному випадку виступає ключовою абстракцією, на якій концентрується протокол HTTP. Ресурс — це дещо, що ви хочете показати світові через ваш інтерфейс користувача.

Наприклад, в випадку даної роботи виводяться метеодані.

REST задає декілька архітектурних правил, іноді кажуть обмежень, які працюють з передачею даних, ось декілька із них:

1. Клієнт-сервер.

Це перше архітектурне обмеження, і воно доволі просте: сервер — це комп'ютер або різного виду сховище, який підключений до мережі та має ресурси, що нас цікавлять. Клієнт — це комп'ютер, що хоче взаємодіяти із цими ресурсами. Система RESTful повинна працювати в моделі клієнт-сервер, навіть якщо компонент іноді діє як клієнт, а іноді - як сервер.

Альтернативою є архітектура на основі подій. Між клієнтом і сервером створюється ще один шар — "наглядач", який слідкує за ресурсами на сервері, і коли ті змінюються, передає зміни клієнтові.

2. Відсутність стану.

Клієнт та сервер повинні працювати незалежно одне від одного, та не мати уявлення про існування одне одного, доки не буде встановлено прямий контакт одного із іншим. Після того, як сервер відповідає клієнтові, він може його забути, тобто сервер не повинен вести облік минулих запитів, якщо це не є ключовою необхідністю. Кожен запит розглядається як окремий.

3. Уніфікований інтерфейс.

Наявність єдиного інтерфейсу гарантує наявність спільної мови для встановлення зв'язку між сервером та клієнтом. Тут мова йде про методи GET (отримання), POST (надсилання), UPDATE (оновлення), DELETE(видалення).

Дані три правила є основними, але також до REST можна добавити кешування відповідей, що є деяким виключенням обмеження про відсутність стану; багатокомпонентність — ввід в систему клієнт-сервера додаткових компонентів, таких як шлюзи або проксі-сервери.

Формат обміну даними.

В даному випадку не має ніяких обмежень щодо формату, проте, використовується JSON-формат, який є популярним та похідним від XML.

Запит, що відправляється користувачем, має наступну структуру HTTP:

- рядок запиту — визначає тип;
- заголовок запиту — параметри запиту;
- повідомлення — це поле є оптимальним.

У відповідь HTTP надсилає таку структуру:

- рядок стану — код стану і повідомлення;
- заголовок відповіді;
- додаткове повідомлення;

Працюючи з рядками стану, можна легко опрацьовувати винятки помилок. Наприклад, код 200 — означає, що сервер працює, запит було сформовано правильно і виконано; код 404 — запит користувача неможливо знайти в мережі, він відсутній.

Таким чином, RESTful - це будь-яка мережа, яка дотримується трьох обмежень: клієнт-сервер, відсутність стану, уніфікований інтерфейс.

Підсумовуючи даний розділ:

1. Описано вибір апаратних компонентів для системи.
2. Розглянуто характеристики вибраних мікроконтролера ESP8266 та давачів DHT22/YL-83.
3. Представлено схему з’єднання елементів.
4. Запрограмовано мікроконтролер.
5. Описано технологію API, за допомогою якої з’єднується уся система.

РОЗДІЛ 4. ПРЕДСТАВЛЕННЯ РОЗРОБЛЕНОГО ІНТЕРФЕЙСУ ТА БАЗИ ДАНИХ

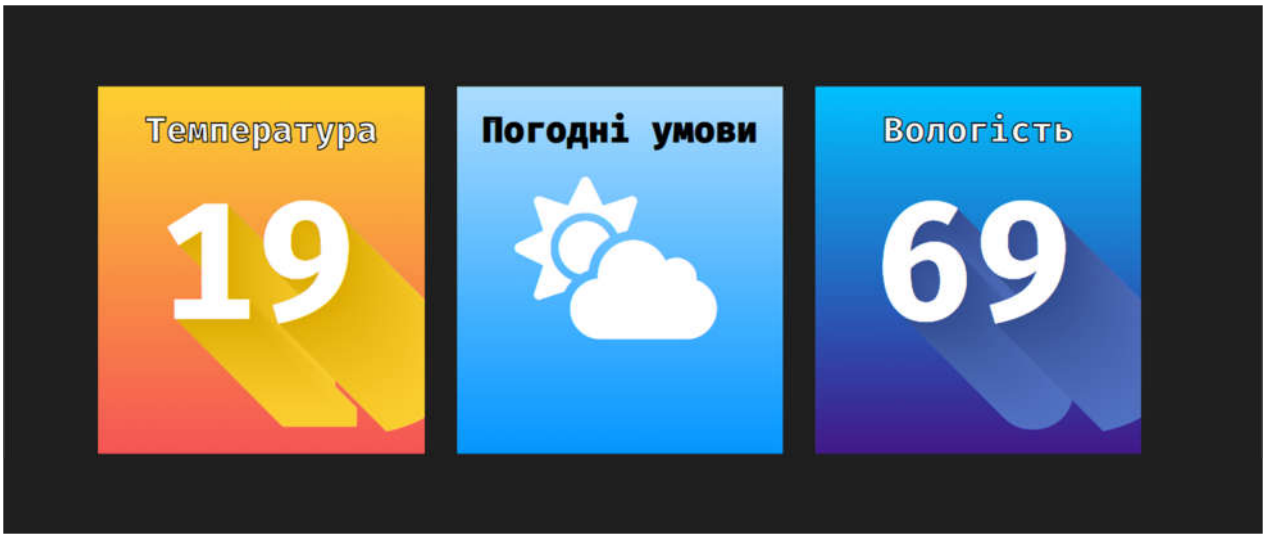

Рисунок 4.1. - Представлення розробленого веб-застосунку: сонячно.

Як можна побачити із рисунку вище, весь інтерфейс є цілком інформативним, та складається тільки із трьох блоків, на яких відповідно зображені дані: температури (DHT22), погодні умови (YL-83), вологість (DHT22). В даному випадку на давачі YL-83 відсутня вологість, тому на другому блоці представлене відповідне зображення. На третьому блоці представлено покази вологості.

На рисунку 4.2 показано варіант, при якому на давач YL-83 потрапила рідина, тому відповідно зображення одразу змінюється. Усі підключення: від апаратного забезпечення до бази і від бази до інтерфесу відбуваються за допомогою технології API.

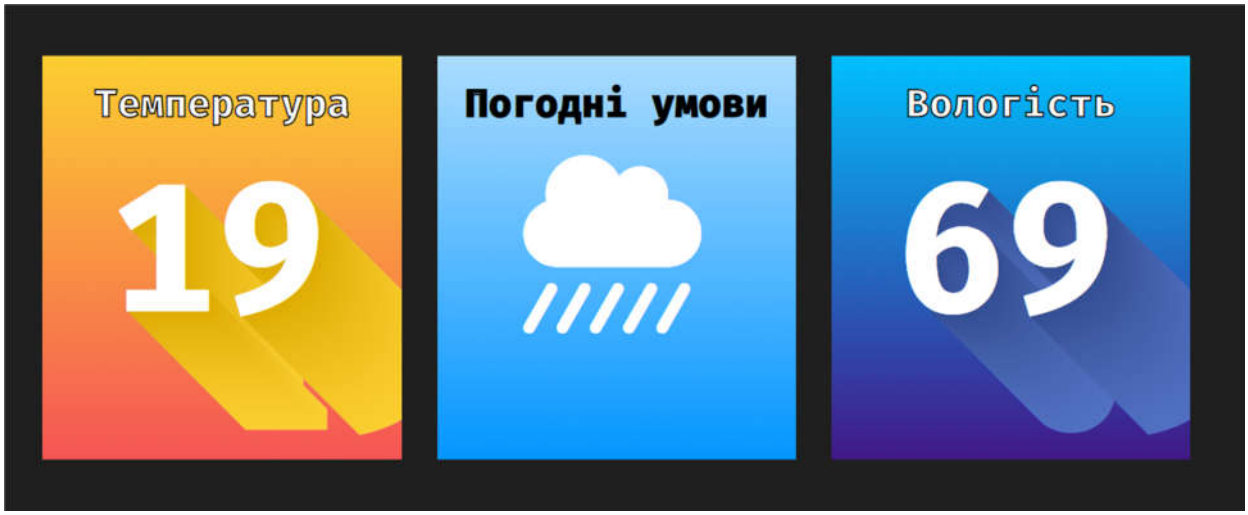

Рисунок. 4.2. - Представлення інтерфейсу користувача: дощить, всі інші дані в нормі.

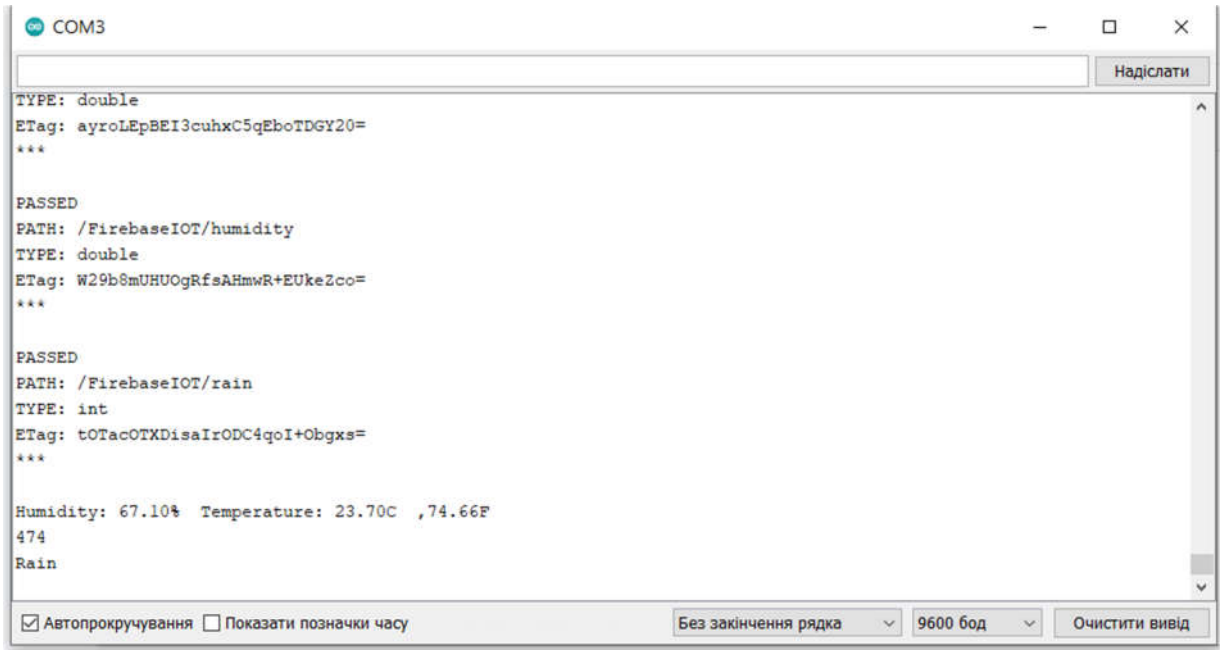

Рисунок 4.3. - Монітор послідовного плоттера.

На рисунку 4.3 представлено моніторинг роботи передачі даних із мікроконтролера до бази даних. При виявленні помилок буде виводитись відповідне повідомлення.

При роботі із базою даних був розглянутий і використаний варіант, при якому дані в базі не добавляються, а переписуються уже наявні. Очевидною перевагою є економія місця в сховищі, а недоліком – відсутність можливості аналізу даних.

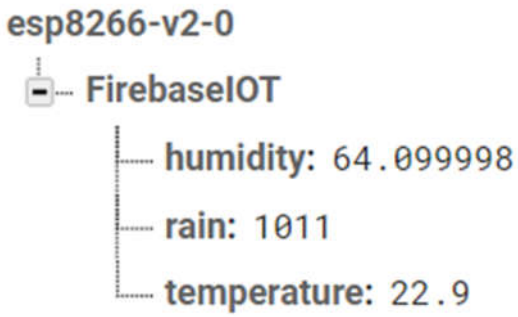

Рисунок 4.4. - Представлення даних в хмарній базі даних.

На рисунку 4.4. представлено модель даних ключ-значення, при якій ключі humidity, rain, temperature завжди будуть залишатись сталими, а числові значення будуть змінюватись за вставновленим інтервалом в мікроконтролері ESP8266 NodeMCU.

РОЗДІЛ 5. ЕКОНОМІЧНА СКЛАДОВА СТВОРЕНОЇ СИСТЕМИ

Оцінюючи даний проєкт варто оцінити економічний чинникк. Тобто розглянути затратність фінансів на створення даного проєкту.

В проєкті застосовувались різні технології для різних частих проєкту, кожна із технологій є з відкритим кодом (open-source). А це означає, що ці технології є безплатними, дозволяється вільне використання та розповсюдження. При реалізації бази даних було використано хмарне середовище. Збільшення фізичної пам’яті реалізується купівлею віртуального серверу, ціни залежать від різних чинників: представника послуг, розміщення серверів, додаткові послуги: шардинг, відновлення даних . Основні затрати були на апаратне забезпечння, які складають: 280 грн = ~10$ та відповідні технічні витрати на створення та обслуговування.

Технічні витрати:

- витрати робочого часу, що вимірюється в людино-годинах;
- витрати на електроенергію;
- на етапі реалізації даного проєкту витрати на доменне ім’я.

ВИСНОВКИ

1. Розглянуто сучасні технології створення веб-сайтів та веб-застосунків.
2. Проведено порівняльний аналіз популярних та вибраних технологій.
3. Вибрано та розроблено хмарну модель бази даних, для зручніших операцій запису-зчитування.
4. Представлено опис апаратного забезпечення, використаного в розробці проєкту.
5. Показано схему роботи системи.
6. Створено систему, що дозволяє слідкувати за метео-даними із будь-якого пристрою, підключеного до мережі.
7. Проведено економічний аналіз затрат на створення даної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гунда Ю. О. Створення масштабованого веб-застосування з використанням MongoDB, 2019. 69 с.

2. Кэмпбелл Л., Мейджорс Ч. Базы данных. Инжиниринг надежности / перевод на русский ООО «Питер». СПб.: Питер, 2020. 304 с.

3. Лоре А. Проектирование веб-API / перевод с англ. Д. А. Беликова. Москва: ДМК Пресс, 2020. 440 с.

4. Мартін Р. Чиста архітектура / переклад з англ. І.Бондар-Терещенко. Харків: Фабула, 2019. 368 с.

5. Тиленс М. Т. React в действии / перевод на русский ООО «Питер». СПб.: Питер, 2019. 368 с.

6. Handson Technology: User Manual V1.2 - ESP8266 NodeMCU WiFi Devkit, 2017. 22 с.

7. NoSQL databases — An Introduction. URL: https://medium.com/analytics-vidhya/no-sql-databases-an-introduction-eb9706fbe3

8. Saving Data. URL: https://firebase.google.com/docs/database/admin/save-data

9. What exactly is Node.js? URL: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/

10. What is a Database? Definition, Types and Components. URL: https://www.edureka.co/blog/what-is-a-database/

11. What RESTful actually means. URL: https://codewords.recurse.com/issues/five/what-restful-actually-means

ДОДАТКИ

Database.js

```
import React, { Component } from "react";
import firebase from "firebase";
import "./Database.css";

const config = {
  apiKey: "___Ваш_персональний_ключ__",
  authDomain: "esp8266-v2-0.firebaseapp.com",
  databaseURL: "https://esp8266-v2-0.firebaseio.com",
  projectId: "esp8266-v2-0",
  storageBucket: "esp8266-v2-0.appspot.com",
  messagingSenderId: "685633748521",
  appId: "1:685633748521:web:c347209df2e95c08bc2638",
};

firebase.initializeApp(config);
const tempSrc = firebase.database().ref("FirebaseIOT/temperature");
const humSrc = firebase.database().ref("FirebaseIOT/humidity");
const rainSrc = firebase.database().ref("FirebaseIOT/rain");
export default class Database extends Component {
  constructor(props) {
    super(props);
    this.state = {
      temp: 0,
      hum: 0,
      rain: 0,
    };
  }
```

```
componentDidMount() {
// функції вибірки даних із бази даних Firebase
    tempSrc.on("value", (snapshot) => {
      this.setState({
        temp: snapshot.val(),
      });
    });
    humSrc.on("value", (snapshot) => {
      this.setState({
        hum: snapshot.val(),
      });
    });
    rainSrc.on("value", (snapshot) => {
      this.setState({
        rain: snapshot.val(),
      });
    });
// перевірка виводу даних давача дощу
    console.log(this.state.rain);
  }
  render() {
// округлення даних
    const fixedTemp = Math.floor(this.state.temp);
    const fixedHum = Math.floor(this.state.hum);
    const rain = this.state.rain;

    return (
      <div className="father-container">
        <h1>Представлення даних із Firebase</h1>
        <ul>
```

```
          <li class="card">
           <div class="card__flipper">
            <div class="card__front">
             <p class="card__name">
              <br />
              Температура <br />{" "}
             </p>
             <p class="card__num">{fixedTemp}</p>
            </div>
            <div class="card__back">
             <p className="device-name">DHT22</p>
             <span className="backside-info">
              Діапазон вимірювання <br /> -40 ~ +80 □<br /> Точність 0.1
             </span>
            </div>
</div>
          </li>
          <li class="card">
           <div class="card__flipper">
            <div class="card__front">
             <p class="card__name yl-83"><br />Погодні умови</p>
             <p class="card__num">
              <i
               className={
                rain > 1000 ? "fas fa-cloud-sun" : "fas fa-cloud-showers-heavy" }
              ></i>
             </p>
            </div>
            <div class="card__back">
             <svg height="180" width="180"></svg>
```

```
            <span>
              Виміри відбуваються за допомогою давача
              <p class="hardware-name">YL-83</p>
            </span>
          </div>
        </div>
      </li>
      <li class="card">
        <div class="card__flipper">
          <div class="card__front">
            <p class="card__name">
              <br />
              Вологість <br />{" "}
            </p>
            <p class="card__num">{fixedHum}</p>
          </div>
          <div class="card__back">
            <p className="device-name">DHT22</p>
            <span className="backside-info">
              Діапазон вимірювання 0-100%
              <br /> Похибка ±2%{" "}
            </span>
          </div>
        </div>
      </li>
    </ul>
  </div>
  );
 }
}
```

App.js

```
import React, { useEffect } from "react";
import $ from "jquery";
import "./App.css";
import Database from "./Database";

function App() {
  useEffect(() => {
    var Flipper = (function() {
      var card = $('.card');
      var flipper = card.find('.card__flipper');
      var win = $(window);
      var flip = function() {
        var thisCard = $(this);
        var thisFlipper = thisCard.find('.card__flipper');
        var offset = thisCard.offset();
        var xc = win.width() / 2;
        var yc = win.height() / 2;
        var docScroll = $(document).scrollTop();
        var cardW = thisCard.outerWidth() / 2;
        var cardH = thisCard.height() / 2;
        var transX = xc - offset.left - cardW;
        var transY = docScroll + yc - offset.top - cardH;
        if (win.width() <= 700) transY = 0;
        if (card.hasClass('active')) unflip();
        thisCard.css({'z-index': '3'}).addClass('active');
        thisFlipper.css({
          'transform': 'translate3d(' + transX + 'px,' + transY + 'px, 0) rotateY(180deg)
scale(1)',
          '-webkit-transform': 'translate3d(' + transX + 'px,' + transY + 'px, 0)
rotateY(180deg) scale(1)',
```

```
        '-ms-transform': 'translate3d(' + transX + 'px,' + transY + 'px, 0)
rotateY(180deg) scale(1)'
      }).addClass('active');
      return false;
    };
   var unflip = function(e) {
     card.css({'z-index': '1'}).removeClass('active');
    flipper.css({
      'transform': 'none',
      '-webkit-transform': 'none',
      '-ms-transform': 'none'
     }).removeClass('active');
    };
var bindActions = function() {
     card.on('click', flip);
     win.on('click', unflip);
    }
    var init = function() {
     bindActions();
    };
    return {
     init: init
    };
  }());
  Flipper.init();
 });
 return <Database />;
}
export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './assets/main.css';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Server.js

```
const express = require("express");
const path = require("path");
const port = process.env.PORT || 3001;

const app = express();
app.use(express.static(__dirname));
app.use(express.static(path.join(__dirname, "build")));
app.get("/ping", function (req, res) {
  return res.send("pong");
});
app.get("/*", function (req, res) {
  res.sendFile(path.join(__dirname, "build", "index.html"));
});
app.listen(port);
```

ESP8266.c

```
#include "FirebaseESP8266.h"
#include <ESP8266WiFi.h>
```

```
#include <DHT.h>
#define FIREBASE_HOST "esp8266-v2-0.firebaseio.com/" //Домен бази даних
#define FIREBASE_AUTH "СЕКРЕТНИЙ_КЛЮЧ_БД"
#define WIFI_SSID "ІМ’Я" // ім'я точки доступу
#define WIFI_PASSWORD "ПАРОЛЬ" //пароль до точки доступу
#define DHTPIN 5
#define DHTTYPE    DHT22
DHT dht(DHTPIN, DHTTYPE);
FirebaseData firebaseData;
      FirebaseJson json;
      void setup()
      {
       Serial.begin(9600);
       dht.begin();
       WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
       Serial.print("Connecting to Wi-Fi");
       while (WiFi.status() != WL_CONNECTED)
       {
        Serial.print(".");
        delay(2000); // затримка 2 сек
       }
       Serial.println();
       Serial.print("Connected with IP: ");
       Serial.println(WiFi.localIP());
       Serial.println();
       Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
       Firebase.reconnectWiFi(true);
      }
      void sensorUpdate(){
       float h = dht.readHumidity(); // Зчитуємо температуру в Цельсіях
```

```
    float t = dht.readTemperature();
    // Зчитуємо температуру в Фарангейтах (якщо буде потрібно)
    float f = dht.readTemperature(true);
    int rain = analogRead(A0);
    // Перевірка правильності даних
    if (isnan(h) || isnan(t) || isnan(f)) {
      Serial.println(F("Failed to read from DHT sensor!"));
      return;
  }
   Serial.print(F("Humidity: "));
    Serial.print(h);
    Serial.print(F("%  Temperature: "));
    Serial.print(t);
    Serial.print(F("C  ,"));
    Serial.print(f);
    Serial.println(F("F  "));
    Serial.println(rain);
    Serial.println("Rain");

    if (Firebase.setFloat(firebaseData, "/FirebaseIOT/temperature", t))
    {
      Serial.println("PASSED");
      Serial.println("PATH: " + firebaseData.dataPath());
      Serial.println("TYPE: " + firebaseData.dataType());
      Serial.println("ETag: " + firebaseData.ETag());
      Serial.println("***");
      Serial.println();
    }
   else
    {
```

```
Serial.println("FAILED");
  Serial.println("REASON: " + firebaseData.errorReason());
  Serial.println("------------------------");
  Serial.println();
 }
 if (Firebase.setFloat(firebaseData, "/FirebaseIOT/humidity", h))
 {
  Serial.println("PASSED");
  Serial.println("PATH: " + firebaseData.dataPath());
  Serial.println("TYPE: " + firebaseData.dataType());
  Serial.println("ETag: " + firebaseData.ETag());
  Serial.println("***");
  Serial.println();
 }
 else
 {
  Serial.println("FAILED");
  Serial.println("REASON: " + firebaseData.errorReason());
  Serial.println("------------------------------------");
  Serial.println();
 }
if (Firebase.setFloat(firebaseData, "/FirebaseIOT/rain", rain))
 {
  Serial.println("PASSED");
  Serial.println("PATH: " + firebaseData.dataPath());
  Serial.println("TYPE: " + firebaseData.dataType());
  Serial.println("ETag: " + firebaseData.ETag());
  Serial.println("***");
  Serial.println();
 }
```

```
  else
  {
    Serial.println("FAILED");
    Serial.println("REASON: " + firebaseData.errorReason());
    Serial.println("------------------------------------");
    Serial.println();
  }
}

void loop() {
  sensorUpdate();
  delay(2000);
}
```