МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Завідувач кафедри, д.е.н., доц.

_____С.І. Левицький

# КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА
# СТВОРЕННЯ СИСТЕМИ РОЗУМНОГО ОСВІТЛЕННЯ З ІНТЕРФЕЙСОМ КОРИСТУВАЧА

Виконав
ст. гр. КІ – 228 _____ Ю.О. Кузьменко

Науковий керівник
доцент _____ О.А. Жеребцов

Запоріжжя
2023

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри,
д.е.н., доц.
_____ С.І. Левицький
___.___._____ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ

Студенту гр. КІ-228, спеціальності 123« Комп'ютерна інженерія»
Кузьменко Юлії Олексіївні

1.Тема: Створення системи розумного освітлення з інтерфейсом користувача затверджена наказом по інституту № 02-10 від 27.01.2023 р.

2. Термін здачі студентом закінченої роботи: 16.06.2023 р.

3. Перелік питань, що підлягають розробці:

1. Провести аналіз предметної області з метою визначення основних параметрів майбутньої системи;
2. Здійснити огляд аналогів за темою дослідження;
3. Здійснити аналіз вимог до майбутньої системи;
4. Здійснити вибір та обґрунтування програмно-апаратного стеку розробки та використовуємих технологій;
5. Здійснити проектування системи;
6. Здійснити програмування системи;
7. Здійснити впровадження системи, налаштування та тестування створеного програмного продукту;
8. Оформити звітню документацію за результатами роботи.

4. Календарний графік підготовки кваліфікаційної бакалаврської роботи

| № етапу | Зміст | Терміни виконання | Готовність по графіку %, підпис керівника | Підпис керівника про повну готовність етапу, дата |
|---|---|---|---|---|
| 1. | Збір практичного матеріалу за темою кваліфікаційної бакалаврської роботи | 16.01.23-11.02.23 | | |
| 2. | І атестація<br>І розділ кваліфікаційної бакалаврської роботи | 30.03.23 | | |
| 3. | ІІ атестація<br>ІІ розділ кваліфікаційної бакалаврської роботи | 27.04.23 | | |
| 4. | ІІІ атестація<br>ІІІ розділ кваліфікаційної бакалаврської роботи, висновки та рекомендації, додатки, реферат | 25.05.23 | | |
| 5. | Перевірка кваліфікаційної бакалаврської роботи на оригінальність | 15.05.23-12.06.23 | | |
| 6. | Доопрацювання кваліфікаційної бакалаврської роботи, підготовка презентації, отримання відгуку керівника і рецензії | 29.05.23-12.06.23 | | |
| 7. | Попередній захист кваліфікаційної бакалаврської роботи | 15.06.23 | | |
| 8. | Подача кваліфікаційної бакалаврської роботи на кафедру | за 3 дні до захисту | | |
| 9. | Захист кваліфікаційної бакалаврської роботи | 23.06.23 | | |

Дата видачі завдання: ___.___._____ р.

Керівник кваліфікаційної

бакалаврської роботи _____ О. А. Жеребцов

Завдання отримав до виконання _____ Ю.О. Кузьменко

# РЕФЕРАТ

Кваліфікаційна бакалаврська робота містить 138 сторінок, 80 рисунків, дев’ять лістингів, один додаток, 17 бібліографічних посилань.

Метою роботи є створення системи розумного освітлення з інтерфейсом користувача з використанням сучасних веб-технологій під мікроконтролерним керуванням.

Об’єктом дослідження є розумні системи, застосування мікроконтролеру та веб технологій для створення сучасних підсистем розумного дому, зокрема системи освітлення.

Предметом дослідження є додаток для контролю та дистанційного управління освітленням будинку.

У роботі проведено детальний аналіз предметної області та огляд сучасних аналогів. Виявлено, що розробка системи розумного освітлення для житлових будинків та громадських будівль є доцільною, а тема актуальною. Здійснено вибір програмно-апаратного комплексу. Розроблено схему майбутньої системи, здійснено проектування моделі предметної області, програмування сутностей та алгоритмів на базі клієнт-серверної архітектури. Проект реалізовано за допомогою таких засобів, як HTML, CSS, JavaScript, React, Node.js, Express.js, і MySQL. А також задіяно мікроконтролер ESP32 з використанням бездротової мережевої технології Wi-Fi.

Програмний продукт є зручним у використанні, має привабливий та інтуїтивно зрозумілий інтерфейс користувача. Система дозволяє керувати освітленням будинку дистанційно. Дає можливість планувати увімкнення та вимкнення освітлення у кімнатах в заданий час, а також переглядати історію та статистичні дані щодо тривалості використання світла у певний період.

WEB APP, SMART LIGHTING SYSTEM, IoT, MCU ESP32, JAVASCRIPT, HTML, CSS, REACT, NODE.JS, EXPRESS.JS, MYSQL

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

| Скорочення | Повна назва | Пояснення/переклад |
|---|---|---|
| БД | База даних | |
| ІІС | Інтелектуальна інформаційна система | |
| ІС | Інтегральна схема | |
| МК | Мікроконтролер | |
| МП | Мікропроцесор | |
| СУБД | Система управління базами даних | |
| ADC | Analog to Digital Converter | Аналогово-цифровий перетворювач |
| AJAX | Asynchronous JavaScript and XML | Асинхронний обмін даними між веб-браузером і веб-сервером. |
| API | Application Programming Interface | Інтерфейс прикладного програмування |
| CPU | Central Processing Unit | Центральний процесор |
| CSS | Cascading Style Sheets | Каскадні таблиці стилів |
| CISC | Complex Instruction Set Computer | Процесор з повним набором команд |
| DAC | Digital to Analog Converter | Цифро-аналоговий перетворювач |
| DNS | Domain Name System | Протокол для перетворення доменних імен в IP-адреси |
| EEPROM | Electrically Erasable and Programmable ROM | Енергонезалежна пам’ять даних |

| | | |
|---|---|---|
| GPIO | General Purpose Input/Output pins | Вхідні/вихідні контакти загального призначення |
| HTML | HyperText Markup Language | Мова розмітки гіпертексту |
| HTTP | Hypertext Transfer Protocol | Протокол передачі гіпертексту |
| IDE | Integrated development environment | Інтегроване середовище розробки |
| I/O Ports | Input/Output Ports | Паралельні порти вводу виводу даних |
| IP | Internet Protocol | Інтернет протокол (протокол передачі даних) |
| ISA | Instruction Set Architecture | Архітектура набору інструкцій (архітектура системи команд) |
| IoT | Internet of Things | Інтернет речей |
| LAN | Local Area Network | Локальна мережа |
| LED | Light-emitting diode | Світлодіод |
| MAN | Metropolitan Area Network | Міська мережа |
| PAN | Personal Area Network | Персональна мережа |
| PWM (ШІМ) | Pulse Width Modulation | Широтно-імпульсна модуляція |
| RAM (ОЗП) | Random Access Memory | Пам’ять з довільним доступом (Оперативний запам’ятовуючий пристрій) |
| RGB | Red, Blue, Green color model | Колірна модель (червоний, синій, зелений) |
| RISC | Reduced Instruction Set Computer | Процесор зі скороченим набором команд |

| | | |
|---|---|---|
| ROM (ПЗП) | Read Only Memory | Пам’ять тільки для читання (Постійно запам’ятовуючий пристрій) |
| SBC | Single-board computer | Одноплатний комп’ютер |
| Serial I/O | Serial Input/Output | Послідовні порти вводу виводу даних |
| SoC | System-on-a-chip | Система на кристалі (на чіпі) |
| SPI | Serial Peripheral Interface | Послідовний периферійний інтерфейс (Синхронні інтерфейси обміну даними) |
| SQL | Structured query language | Мова структурованих запитів |
| TCP | Transmission Control Protocol | Протокол керування передачею даних |
| UART | Universal Asynchronous Receiver / Transmitter | Універсальний асинхронний приймач/передавач (Асинхронні інтерфейси обміну даними) |
| URL | Uniform Resource Locator | Уніфікований локатор ресурсів або адреса ресурсу |
| WAN | Wide Area Network | Глобальна мережа |
| WebSocket | WebSocket protocol | Протокол для обміну інформацією між браузером та веб-сервером у режимі реального часу |
| Wi-Fi | Wireless Fidelity | Технологія бездротової передачі даних |
| WLAN | Wireless Local Area Network | Бездротова локальна мережа |
| WWW | World Wide Web | Всесвітня мережа |

# ВСТУП

У наш час на житлові та громадські будівлі припадає близько сорока відсотків світового споживання електроенергії, причому значну частину в енергоспоживанні становить саме енергія на освітлення [1]. Пріоритетного значення у період глобальної економічної кризи набуває питання стосовно збереження енергії в освітлювальних установках та управління джерелами освітлення. Очевидно, що для вирішення цієї проблеми є потреба у вдосконаленні систем енергозбереження при мінімізації використання матеріалів та/або електроенергії.

Скоротити витрати електроенергії можливо двома основними шляхами: перший – це знизити номінальну потужність освітлення, другий – це зменшити час використання світильників. Щоб зменшити встановлену потужність освітлення, в першу чергу треба перейти до більш ефективних джерел світла, які надають потрібний світловий потік, при цьому споживають значно менше електроенергії (наприклад, використання LED-ламп замість ламп-розжарювання). Досягнути зменшення часу використання світильників можна впровадженням сучасних систем управління і контролю освітлення. Одна із таких систем – це автоматизована система розумного освітлення.

Енергозбереження у житлових та громадських будівлях істотно впливає на витрату електроенергії, тому проблеми її якості і раціональні методи експлуатації є надзвичайно важливими, а тема автоматизації управління внутрішнім освітленням є актуальною.

# РОЗДІЛ 1
# ОГЛЯД ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Загальні відомості про інтелектуальні системи

В останні роки відбувся швидкий розвиток технологій, що призвело до появи різноманітних інтелектуальних та взаємопов’язаних систем. Ці системи, широко відомі як розумні системи, революціонізували спосіб взаємодії та сприйняття навколишнього середовища [1, 2].

Розумні системи (англ. – Smart Systems, or Intelligent Systems) відносяться до класу інтелектуальних і взаємопов’язаних систем, які об’єднують передові технології, такі як штучний інтелект (AI), Інтернет речей (IoT), машинне навчання (ML), аналітика даних і автоматизація. Ці системи розроблені для збору, аналізу та інтерпретації величезних обсягів даних із різних джерел у режимі реального часу, що дозволяє їм приймати обґрунтовані рішення та вживати автономних дій [3]. Використовуючи потужність передових технологій, розумні системи підвищують ефективність, оптимізують процеси та покращують загальну якість життя. Тобто, ключовим аспектом розумних систем є їх здатність збирати дані з різних джерел, аналізувати їх у режимі реального часу та приймати розумні рішення або надавати цінну інформацію. Ці системи підвищують зручність, продуктивність і стійкість, створюючи взаємопов’язані екосистеми, де пристрої та програми спілкуються та співпрацюють для досягнення кращих результатів.

Розумні системи охоплюють широкий спектр застосувань. Їх можна знайти в багатьох областях та класифікувати за кількома категоріями на основі їх застосування та функцій. Нижче наведено деякі загальновизнані типи розумних систем:

Розумні будинки (Smart Homes): ці системи об’єднують різні прилади та пристрої (такі як термостати, датчики, освітлення, системи безпеки та домашні

помічники) для покращення домашньої автоматизації, управління енергією та безпеки. Вони включають автоматизоване освітлення, контроль температури, управління енергією, безпеку та спостереження, системи розваг, тощо.

Розумні мережі (Smart Grids): це вдосконалені системи електропостачання, які використовують цифрові технології зв'язку та керування для оптимізації виробництва, розподілу та споживання енергії. Ці системи забезпечують двонаправлений зв'язок між постачальниками електроенергії та споживачами, сприяючи ефективному управлінню енергією, реагуванню на попит та інтеграції відновлюваних джерел енергії.

Розумні міста (Smart Cities): Розумні міста мають на меті покращити міське життя за допомогою інтеграції технологій, даних та інфраструктури. Ці системи включають пристрої IoT, датчики та аналітику даних для оптимізації міської інфраструктури, управління трафіком, розподілу енергії, управління відходами та громадських послуг. Тобто, вони використовують інтелектуальні системи для оптимізації транспорту, підвищення енергоефективності, управління ресурсами, моніторингу навколишнього середовища та підвищення громадської безпеки. Розумні міста використовують взаємопов'язані мережі та аналітику даних, щоб забезпечити ефективне управління та підвищити якість життя мешканців.

Промислова автоматизація (Industrial Automation): Промислова автоматизація передбачає застосування інтелектуальних систем для покращення виробничих процесів, підвищення продуктивності та ефективності роботи. Інтелектуальні системи промислової автоматизації включають робототехніку, алгоритми машинного навчання, аналітику даних і сенсорні мережі. Ці системи забезпечують прогнозне обслуговування, моніторинг у реальному часі та автономне прийняття рішень на виробничих підприємствах. Простими словами, у промислових умовах інтелектуальні системи забезпечують автоматизацію, моніторинг і контроль виробничих процесів, що сприяє підвищенню продуктивності, якості та безпеки.

Охорона здоров'я (Health care). Розумні системи використовуються в

охороні здоров’я для віддаленого моніторингу пацієнтів, переносних пристроїв, телемедицини, електронних медичних записів (EHR) і аналізу даних для покращення догляду за пацієнтами, діагностики та лікування.

Транспорт (Transportation): розумні системи покращують транспорт за допомогою таких технологій, як інтелектуальне керування дорожнім рухом, зв’язок між транспортними засобами, автономні транспортні засоби та навігаційні системи, що призводить до підвищення ефективності, безпеки та зменшення заторів.

Системи управління енергією (Energy management systems): розумні мережі та системи керування енергією використовують аналітику даних та Інтернет речей для оптимізації споживання енергії, моніторингу відновлюваних джерел енергії та зменшення втрат.

Сільське господарство (Agriculture): розумні системи допомагають у точному землеробстві, використовуючи датчики, дрони, штучний інтелект та аналіз даних для оптимізації зрошення, моніторингу здоров’я культур і боротьби зі шкідниками, що призводить до підвищення врожайності та ефективного використання ресурсів, тощо.

### 1.1.1 Система розумного дому

Одним з найпопулярніших типів розумних систем є система розумного дому.

Розумні будинки (анг. «intelligent building») – це житлові приміщення, обладнані інтелектуальними пристроями та системами, які підвищують комфорт, зручність і безпеку. Ці системи використовують датчики, приводи та підключення для створення згуртованого та чутливого життєвого середовища [3].

Система розумного дому складається з кількох взаємопов’язаних підсистем, які працюють разом, щоб забезпечити автоматизацію, контроль і функціональність у будинку.


Рисунок 1.1 – Образ системи розумного будинку

Надалі наведемо декілька поширених підсистем системи розумного будинку.

Система безпеки та охорони (Security and Safety System): підсистема безпеки забезпечує спостереження, контроль доступу та виявлення вторгнень. Вона включає такі компоненти, як розумні замки, датчики дверей/вікон, детектори руху, камери спостереження та системи сигналізації. Користувачі можуть відстежувати та віддалено контролювати безпеку свого будинку, отримувати сповіщення про несанкціонований доступ та інтегруватися з професійними службами моніторингу. Це підвищує загальну безпеку та забезпечує спокій.

Розважальна та мультимедійна система (Entertainment and Multimedia System): підсистема розваг покращує мультимедійні враження від дому. Система включає смарт-телевізори, потокові пристрої, аудіосистеми та налаштування домашнього кінотеатру. Користувачі можуть контролювати та транслювати медіаконтент, регулювати параметри звуку та створювати персоналізовані розваги.

Система HVAC: підсистема HVAC (від англ. слів «Heating, Ventilation, and Air Conditioning», переклад: опалення, вентиляція та кондиціонування

повітря) зосереджена на контролі клімату в будинку. Вона включає розумні термостати, розумні вентиляційні отвори та датчики температури. Користувачі можуть дистанційно контролювати та регулювати параметри температури, вологості та вентиляції та створювати персоналізовані графіки енергоефективності.

Система контролю приладів (Appliance Control System): ця підсистема зосереджена на контролі та моніторингу побутових приладів. Підсистема включає розумні розетки, розумні вимикачі та розумну техніку (холодильники, духовки, пральні машини). Користувачі можуть дистанційно керувати своїми приладами та контролювати їх, встановлювати таймери та отримувати повідомлення про стан приладів.

Система освітлення (Lighting System): підсистема освітлення дозволяє користувачам контролювати та автоматизувати освітлення вдома. Вона включає розумні лампочки, вимикачі, диммери та панелі керування освітленням. Користувачі можуть регулювати яскравість, колір і планувати освітлення залежно від зайнятості чи вподобань.

Система енергоменеджменту (Energy Management System): підсистема енергоменеджменту оптимізує використання енергії та ефективність у будинку. Вона включає такі компоненти, як пристрої моніторингу енергії, розумні лічильники та системи інтеграції відновлюваної енергії. Користувачі можуть контролювати споживання енергії, відстежувати моделі використання та отримувати рекомендації щодо дій щодо енергозбереження.

Система поливу та управління водою (Irrigation and Water Management System): ця підсистема використовується для ефективного використання води та поливу вдома. Він включає розумні спринклерні системи, детектори витоку води та пристрої моніторингу споживання води. Користувачі можуть контролювати та автоматизувати графіки поливу залежно від погодних умов і отримувати сповіщення про витоки води або ненормальне використання води, тощо.

Підсистеми розумного дому працюють разом, щоб створити комплексну систему розумного дому, яка пропонує автоматизацію, контроль та інтеграцію різних пристроїв і функцій у будинку. Однак ці підсистеми можуть використовуватися і окремо, в залежності від конкретних потреб і переваг користувача. Кожна підсистема може функціонувати незалежно, щоб забезпечити певні функції в будинку. Великий попит у наш час є саме на систему освітлення.

### 1.1.2 Підсистеми освітлення

Останнім часом все частіше власники квартир і будинків звертають увагу на автоматизацію освітлення. Все більше і більше людей вирішує впровадити автономну систему освітлення для керування та автоматизації освітлення свого дому, не інтегруючи її з іншими підсистемами розумного дому [3].

Автоматична система освітлення – це комплекс засобів, за рахунок якого можна управляти освітлювальними приладами дистанційно. Система розроблена для автоматичної роботи без необхідності ручного керування. Вона використовує різні технології та датчики для виявлення та реагування на зміни в навколишньому середовищі чи уподобаннях користувача, відповідно регулюючи освітлення. Наприклад, вмикати та вимикати світло в будинку, використовуючи смартфон, або за допомогою різних пристроїв: датчик руху чи датчик рівня освітленості. Ці датчики будуть вмикати та вимикати світло, якщо в кімнаті буде знаходитися людина,  рівень світла відповідатиме чітко встановленому у налаштуваннях значенню. Основне призначення системи автоматичного освітлення – забезпечити зручність, енергоефективність і розширену функціональність. Ось деякі основні функції та компоненти, які зазвичай містяться або можуть міститися у будь-якій системі автоматичного освітлення:

Датчики руху (Motion Sensors): датчики руху використовуються для виявлення присутності або руху людей у певній зоні. У разі виявлення руху

система освітлення вмикається, забезпечуючи освітлення у відповідь на наявність людей. Коли рух більше не виявляється, система може автоматично вимкнути або приглушити світло для економії енергії.

Датчики світла (Light Sensors): Датчики світла, також відомі як фотоелементи або датчики навколишнього освітлення, вимірюють кількість природного освітлення в кімнаті або на вулиці. На основі виявлених рівнів освітлення система освітлення може відповідним чином налаштувати штучне освітлення. Наприклад, за достатнього природного освітлення система може затемнювати або вимикати світло для економії енергії.

Таймери та розклади (Timers and Schedules): автоматичні системи освітлення часто містять таймери та функції планування. Користувачі можуть запрограмувати певний графік автоматичного ввімкнення або вимкнення світла. Це може допомогти імітувати зайнятість, коли мешканців немає, покращити безпеку та створити приємну атмосферу після прибуття.

Виявлення зайнятості та присутності (Occupancy and Presence Detection): окрім датчиків руху, деякі системи автоматичного освітлення містять передові технології виявлення присутності. Ці системи можуть виявляти та розрізняти нерухомих та рухомих мешканців, забезпечуючи більш точне керування освітленням. Наприклад, якщо людина сидить у певній зоні протягом тривалого часу, система може відповідно регулювати рівні освітлення.

Інтеграція та контроль (Integration and Control): системи автоматичного освітлення можна інтегрувати з іншими пристроями. Ними можна керувати віддалено за допомогою мобільних додатків або голосових помічників, що дозволяє користувачам налаштовувати налаштування, створювати сцени або автоматизувати дії з освітленням на основі своїх уподобань, тощо.

Сьогодні розумні системи освітлення по праву вважаються одним з найбільш затребуваних рішень, адже до їхніх основних переваг відносяться:

Енергоефективність: можливість енергозбереження приваблює екологічно свідомих споживачів і тих, хто хоче зменшити свої рахунки за електроенергію.

Зручність і контроль: інтелектуальні системи освітлення надають користувачам зручні можливості керування. За допомогою мобільних додатків або голосових команд користувачі можуть дистанційно керувати та автоматизувати своє освітлення, створювати персоналізовані сцени освітлення. Цей рівень контролю підвищує зручність і покращує загальний досвід проживання.

Економічна ефективність та масштабованість (Cost-Effective and Scalable): вартість на розумні системи освітлення останнім часом знизилася, що зробило їх доступними для ширшого кола споживачів. Крім того, ці системи є масштабованими, що дозволяє користувачам починати з малого, від кількох розумних лампочок і поступово розширювати мережу освітлення за потреби.

Отже, системи освітлення дуже затребувані, тому що вони дозволяють знизити витрати на електроенергію, а також забезпечують високий рівень комфорту (в тому числі за рахунок функцій дистанційного керування світлом).

## 1.2 Ключові компоненти та технології що використовуються в автоматизованих системах

Для автономного виконання завдань і покращення своїх функціональних можливостей, розумні системи використовують різноманітні технології та компоненти, інтеграція яких забезпечує ефективний зв’язок, безперебійне підключення, обмін даними, вдосконалений моніторинг та контроль в інтелектуальних системах. До основних належать: веб технології, технології IoT, мікроконтролери, локальні мережі та мережі Wi-Fi [4].

Технології IoT формують основу інтелектуальних систем, з’єднуючи пристрої та датчики, дозволяючи їм безперешкодно спілкуватися та обмінюватися даними.

Веб-технології відіграють важливу роль у забезпеченні віддаленого доступу, контролю та моніторингу розумних систем через веб-інтерфейси або мобільні додатки.

Локальні мережі та мережі Wi-Fi забезпечують інфраструктуру для з'єднання пристроїв у межах локалізованої області, уможливлюючи моніторинг і контроль у реальному часі.

Мікроконтролери, часто інтегровані в інтелектуальні пристрої, можна підключити до локальної мережі та використовувати веб-протоколи, такі як HTTP або WebSocket, для зв'язку з веб-інтерфейсами. Це дозволяє користувачам контролювати та керувати системою через веб-браузери або спеціальні мобільні програми з будь-якого місця, якщо вони мають доступ до Інтернету. Тобто, мікроконтролери діють як мозок системи, локально обробляють дані, приймають рішення та керують пристроями, виконуючи команди на основі попередньо визначених правил або введення користувача.

Використовуючи ці передові технології, розумні системи можуть досягти покращеної автоматизації, енергоефективності та загальної оптимізації. Розглянемо кожну складову більш детально.

### 1.2.1 Основні поняття про IoT технології. Їх роль в інтелектуальних системах

Інтернет речей (IoT) революціонізував спосіб нашої взаємодії з навколишнім світом. Підключаючи повсякденні об'єкти до Інтернету та дозволяючи їм збирати та обмінюватися даними, технології IoT проклали шлях для розвитку інтелектуальних систем [3].

Термін "Інтернет речей" (від англ. *Internet of Things*, скорочено – *IoT*) вперше було сформульовано у 1999 році. Згідно з найбільш поширеним формулюванням, – це концепція мережі, яка складається із взаємозв'язаних фізичних пристроїв (предметів, «*речей*»). Вони мають різні вбудовані датчики, а також програмне забезпечення, яке дозволяє в автоматичному режимі здійснювати передачу і обмін даними між фізичним світом і комп'ютерними

системами, використовуючи стандартні протоколи зв'язку (протоколів передавання даних, такі як Wi-Fi, Bluetooth, Zigbee, тощо). Окрім датчиків, мережа може мати виконавчі пристрої, вбудовані у фізичні об'єкти і пов'язані між собою через дротові чи бездротові мережі. Ці взаємопов'язані пристрої мають можливість зчитування та приведення в дію, функцію програмування та ідентифікації, а також дозволяють виключити необхідність участі людини, за рахунок використання інтелектуальних інтерфейсів. Іншими словами, технології IoT стосуються апаратних і програмних компонентів, які дозволяють пристроям підключатися до Інтернету та спілкуватися один з одним. Пристрої IoT можна знайти в різних місцях, включаючи будинки, міста, фабрики та транспортні системи. Ці пристрої оснащені датчиками, які збирають дані і передають їх через мережу іншим пристроям або серверам [4].

Архітектура IoT технологій залежить від функціональності та реалізації в різних системах. Однак, існує базовий процес, на основі якого будується IoT. Як правило, загальна фундаментальна архітектура IoT технологій складається із 4х компонентів, тобто IoT має 4-етапну архітектуру (рис. 1.2).

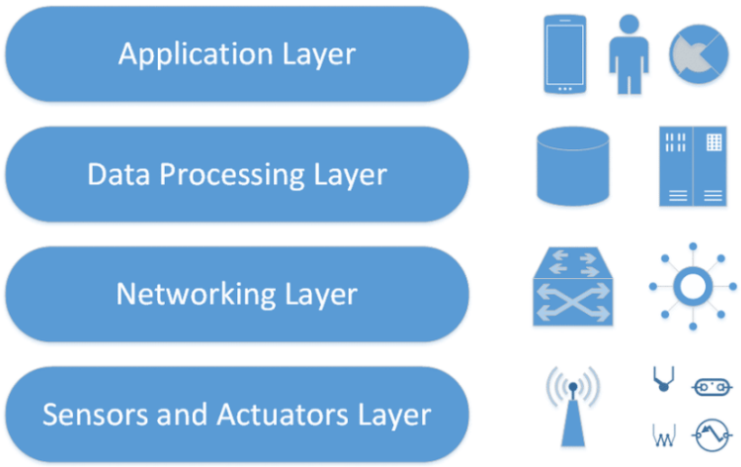


Рисунок 1.2 – Архітектура IoT технологій

1. Сенсорний рівень:

Сенсорний рівень є першим рівнем архітектури IoT, що відповідає за збір даних з різних джерел. Цей рівень містить датчики та виконавчі механізми (Sensors and Actuators).

Датчики є основними будівельними блоками систем IoT. Вони відповідають за збір даних із навколишнього середовища, і можуть вимірювати такі параметри, як температура, вологість, інтенсивність світла, рух, тиск, близькість тощо. Ці датчики перетворюють фізичні сигнали в електричні сигнали, які потім можна обробляти та аналізувати.

Актуатори, з іншого боку, дозволяють системам IoT взаємодіяти з фізичним світом. Актуатори (від англ. “Actuation” – приведення в дію; дія, яка змушує машину або пристрій працювати.) – це пристрої, які можуть керувати фізичними об’єктами або системами або маніпулювати ними на основі даних, зібраних датчиками. Приклади приводів включають двигуни, клапани, перемикачі, реле, сервоприводи, тощо. Поєднуючи датчики та виконавчі механізми, пристрої IoT можуть збирати дані та реагувати на оточення. Ці пристрої підключаються до мережевого рівня за допомогою дротових або бездротових протоколів зв’язку.

2. Мережевий рівень (Network layer):

Мережевий рівень архітектури IoT відповідає за забезпечення зв’язку та підключення між пристроями в системі IoT. Цей рівень також відомий як рівень передачі (Transmission layer). Він діє як міст, який переносить і передає дані, зібрані від фізичних об’єктів через датчики. Для встановлення зв’язку в системах IoT використовуються різні комунікаційні протоколи та технології, які дозволяють пристроям підключатися та спілкуватися один з одним і з широким Інтернетом. До них належать:

Wi-Fi: технологія Wi-Fi дозволяє пристроям IoT підключатися до локальних мереж (LAN) та Інтернету, забезпечуючи високу швидкість передачі даних.

Bluetooth: технологія Bluetooth забезпечує бездротовий зв’язок на короткій відстані між пристроями IoT, що зазвичай використовується для підключення смартфонів, переносних пристроїв та інших пристроїв, розташованих поблизу.

Zigbee і Z-Wave: ці протоколи бездротового зв’язку з низьким енергоспоживанням розроблені спеціально для домашньої автоматизації та інтелектуальних енергетичних програм.

Стільникові мережі. Пристрої IoT також можуть підключатися до стільникових мереж (2G, 3G, 4G і 5G), щоб забезпечити глобальне підключення, зокрема для таких програм, як відстеження активів, моніторинг транспортних засобів і розгортання розумних міст.

NFC (Near Field Communication): технологія NFC полегшує зв’язок між пристроями IoT на невеликій відстані, часто використовується для безконтактних платежів і обміну даними між смартфонами та іншими пристроями.

Крім того, мережевий рівень може включати шлюзи та маршрутизатори, які діють як посередники між пристроями та широким Інтернетом, а також можуть містити функції безпеки, такі як шифрування та автентифікація для захисту від несанкціонованого доступу.

3. Рівень обробки даних (Data Processing layer):

Рівень обробки даних відноситься до програмних і апаратних компонентів, які відповідають за збір, аналіз та інтерпретацію даних з пристроїв IoT. Цей рівень відповідає за отримання даних із пристроїв, їх збереження, обробку та надання доступу для подальшого аналізу чи дії. Він включає різноманітні технології та інструменти, різні аналітичні методи, що використовуються для отримання значущої інформації з даних і прийняття рішень на основі цих даних. Це включає системи керування даними, аналітику в реальному часі, пакетну обробку, алгоритми машинного навчання, тощо. Аналітика даних допомагає визначати закономірності, аномалії, тенденції та кореляції, забезпечуючи прийняття обґрунтованих рішень, оптимізацію та можливості прогнозування.

4. Прикладний рівень (рівень програм, Application layer):

Прикладний рівень архітектури IoT — це найвищий рівень, який безпосередньо взаємодіє з кінцевим користувачем. Він відповідає за надання

зручних інтерфейсів і функцій, які дозволяють користувачам отримувати доступ до пристроїв IoT і керувати ними. Цей рівень включає різноманітне програмне забезпечення та програми, такі як мобільні програми, веб-портали та інші інтерфейси користувача, які призначені для взаємодії з основною інфраструктурою IoT. Він також включає служби проміжного програмного забезпечення, які дозволяють різним пристроям і системам Інтернету речей безперебійно спілкуватися та обмінюватися даними. Рівень додатків також може включати аналітику та можливості обробки, напрклад, інструменти візуалізації даних та інші розширені аналітичні можливості.

Як бачимо, технології IoT були розроблені для вдосконалення та автоматизації різноманітних процесів, вони дозволяють здійснювати автоматизований контроль і фізичні маніпуляції на основі даних, зібраних із датчиків і проаналізованих системою, саме тому вони відіграють вирішальну роль в розумних системах, забезпечуючи їх функціональність та підключення.

### 1.2.2 Веб-технології та їхня роль в автоматизований системах

Веб-технології відіграють важливу роль в інтелектуальних системах. Вони забезпечують основу для веб-інтерфейсів, візуалізації даних та інтеграції з іншими системами. За допомогою веб-технологій можна створювати веб-сайти, мобільні додатки та онлайн-сервіси, які можуть бути доступними на різних пристроях і платформах [5]. Це надає можливість віддаленого доступу, забезпечує інтерактивний досвід користувача та ефективний обмін даними в системі. Саме веб-інтерфейси дозволяють користувачам віддалено моніторити, контролювати та оптимізовувати різні аспекти автоматизованої системи з будь-якого місця, де є підключення до Інтернету. Тобто, за допомогою інтуїтивно зрозумілих графічних інтерфейсів, використовуючи веб-браузер чи мобільні додатки, користувачі можуть отримувати доступ до даних, налаштовувати параметри, планувати операції та отримувати зворотний зв’язок у режимі реального часу. Також веб-технології полегшують візуалізацію та аналіз даних, зібраних розумними системами. Адже завдяки

інтерактивним діаграмам, графікам і інформаційним панелям користувачі можуть отримати уявлення про продуктивність системи, енергоспоживання та інші відповідні показники.

Взагалі, веб-технологія відноситься до набору інструментів, протоколів і різних мов, які використовуються для розробки та полегшення функціонування веб-сайтів, веб-додатків та інших онлайн-сервісів, тобто для створення та роботи програм і послуг у Всесвітній павутині (WWW). Це охоплює широкий спектр технологій, які працюють разом, щоб забезпечити спілкування, обмін даними та взаємодію через Інтернет. До основних компонентів веб-технологій належать [5]:

Мова розмітки гіпертексту (HTML): HTML — це стандартна мова розмітки, яка використовується для структурування та представлення вмісту в Інтернеті. Вона визначає структуру та макет веб-сторінок, містить елементи для тексту, зображень, посилань, форм, мультимедіа та інших засобів масової інформації.

Каскадні таблиці стилів (CSS): CSS — це мова таблиць стилів, яка описує представлення та візуальний вигляд елементів HTML, тобто використовується для опису представлення документів HTML. Це дозволяє розробникам визначати шрифти, кольори, макети та інші аспекти дизайну, забезпечуючи узгоджені та візуально привабливі веб-сторінки.

JavaScript: JavaScript — це мова сценаріїв, яка забезпечує інтерактивність і динамічну поведінку веб-сторінок. Вона надає можливість маніпулювати елементами HTML, обробляти події користувача, виконувати обчислення та спілкуватися з веб-серверами. Тобто JavaScript широко використовується для створення сценаріїв на стороні клієнта, де він запускається безпосередньо у веб-браузері та може маніпулювати елементами HTML, обробляти взаємодію користувачів і спілкуватися з веб-серверами.

Веб-фреймворки: клієнтські фреймворки та бібліотеки (такі як React, Angular або Vue.js) — це попередньо розроблені, так звані програмні «каркаси», які надають готові компоненти і інструменти для оптимізації веб-

розробки, та забезпечують структуроване середовище для розробки веб-додатків. Вони покращують продуктивність, спрощують і полегшують розробку інтерфейсу користувача та складних веб-додатків, тощо.

Технології на стороні сервера: Технології на стороні сервера, такі як PHP, Java або Python, використовуються для обробки запитів на веб-сервері та створення динамічного вмісту. Вони керують зберіганням даних, бізнес-логікою та взаємодією з базами даних.

Веб-сервери: веб-сервери — це програмні додатки, які доставляють веб-контент клієнтам (веб-браузерам) за запитом. Вони обробляють HTTP-запити та відповіді, керують зберіганням даних і виконують сценарії на сервері. Іншими словами, веб-сервери — це програми, які обробляють HTTP-запити від клієнтів (веб-браузерів) і обслуговують веб-сторінки та ресурси у відповідь. Вони зберігають і доставляють веб-вміст, наприклад файли HTML, зображення, відео та інші файли, клієнтам через Інтернет.

Веб-служби: веб-служби забезпечують зв’язок і обмін даними між різними програмами через Інтернет за допомогою стандартних протоколів, таких як HTTP. Вони дозволяють системам взаємодіяти та обмінюватися інформацією, полегшуючи інтеграцію та взаємодію між різними програмними додатками.

Веб-браузери: веб-браузери – це програми, які інтерпретують і відображають веб-вміст. Вони отримують файли HTML, CSS і JavaScript із веб-серверів і відтворюють їх як веб-сторінки, які користувачі можуть переглядати та взаємодіяти з ними. Тобто це програми, які отримують і відображають веб-вміст із веб-серверів, інтерпретують код HTML, CSS і JavaScript для відтворення веб-сторінок і забезпечення взаємодії користувача з веб-додатками.

Веб-інтерфейси API (англ. – Application Programming Interface, переклад – інтерфейси прикладного програмування) — це набори правил і протоколів, які дозволяють різним програмам програмного забезпечення взаємодіяти між собою. Вони дозволяють веб-службам відкривати функції та дані, до яких

можуть отримати доступ інші програми через Інтернет. Тобто, веб-інтерфейси надають набір правил і протоколів для створення та інтеграції веб-додатків. Вони дозволяють розробникам отримувати доступ і використовувати певні функції або дані з інших програм або служб, наприклад отримувати дані про погоду, отримувати доступ до платформ соціальних мереж, тощо.

Це лише деякі з основних компонентів веб-технологій. Веб-екосистема постійно розвивається, з'являються нові технології та стандарти для покращення веб-розробки, безпеки, продуктивності та взаємодії з користувачем.

### 1.2.3 Загальні відомості та основні поняття про мікроконтролери

Мікроконтролери служать центрами управління, «мозком» інтелектуальних систем, забезпечуючи необхідну обчислювальну потужність і надаючи можливості керування. Ці невеликі обчислювальні пристрої вбудовані в розумні пристрої або датчики, що дозволяє локально обробляти дані, приймати рішення та контролювати. Вони отримують дані від датчиків, обробляють їх і виконують команди або запускають дії на основі попередньо визначеної та запрограмованої логіки. Вони забезпечують оперативне реагування в реальному часі, що робить їх ідеальними для критичних за часом додатків в автоматизованих системах [6, 7].

За своєю суттю, мікроконтролери — це невеликі інтегральні схеми, які поєднують ядро мікропроцесора з пам'яттю та периферійними пристроями введення/виведення. Це невеликі програмовані пристрої, призначені для керування та моніторингу електронних систем, для виконання конкретних завдань в рамках інтелектуальної системи.

Мікроконтролери зазвичай містять центральний процесор (CPU), пам'ять, порти введення/виведення (I/O) та інші периферійні пристрої, такі як таймери, аналого-цифрові перетворювачі та комунікаційні інтерфейси. Їх можна програмувати за допомогою різних мов, включаючи мову асемблера, C і C++, і часто програмують за допомогою спеціальних програмних засобів.

Вони бувають різних форм і розмірів, починаючи від невеликих малопотужних пристроїв, призначених для додатків з живленням від батареї, до більш потужних пристроїв, які можуть виконувати складні завдання, швидко реагувати та адаптуватися до мінливих умов і середовища [7].

### 1.2.3.1 Історія виникнення мікроконтролерів

Історія мікроконтролерів починається з 1960-х років, коли розроблялися перші мікропроцесори. Однак потреба в невеликих інтегрованих обчислювальних системах, які можна було б запрограмувати для виконання конкретних завдань, призвела до появи мікроконтролерів наприкінці 1970-х років.

До мікроконтролерів найпоширенішими обчислювальними системами були мейнфрейми та міні-комп’ютери, які були великими та дорогими. Розвиток мікропроцесорів, які були невеликими та недорогими, проклав шлях до створення мікроконтролерів.

Перший мікроконтролер був представлений компанією Texas Instruments у 1971 році. Він називався TMS1802NC і використовувався в калькуляторах та інших невеликих пристроях. Однак першим комерційно успішним мікроконтролером став Intel 8048 (MCS-48), який був представлений у 1976 році. Він мав 1 КБ ПЗУ, 64 байти оперативної пам’яті та 27 контактів вводу/виводу, і використовувався в широкому діапазоні додатків, у тому числі промислових. засоби керування, медичні пристрої та побутова електроніка [7].

Поява мікроконтролерів стала можливою завдяки прогресу в напівпровідникових технологіях, які дозволили інтегрувати кілька компонентів на одному чіпі. Ця інтеграція зробила мікроконтролери маленькими, недорогими та енергоефективними, що зробило їх ідеальними для використання у вбудованих системах.

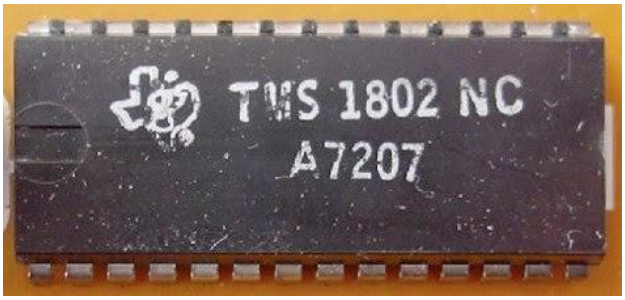


Рисунок 1.3 – Перший мікроконтроллер від компанії Texas Instruments

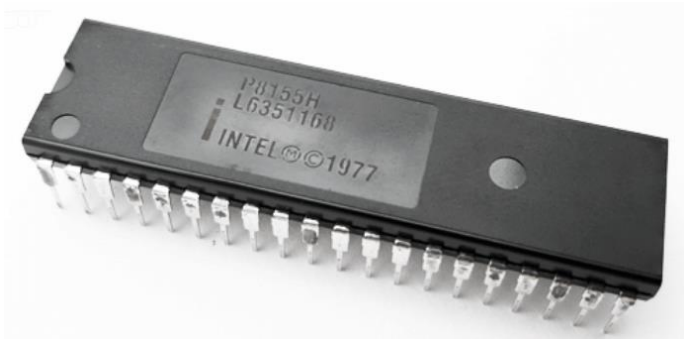


Рисунок 1.4 – Мікроконтролер компанії Intel

1.2.3.2 Основні відмінності мікроконтролеру від мікропроцесору

Основна відмінність між мікроконтролером і мікропроцесором полягає в тому, що мікроконтролер є повною обчислювальною системою на одному чіпі, тоді як мікропроцесор містить лише блок обробки обчислювальної системи [8].

Мікропроцесор призначений лише для обробки даних і інструкцій і покладається на інші компоненти, такі як пам’ять, периферійні пристрої введення/виведення та інтерфейси зв’язку. Мікропроцесор зазвичай використовується в програмах, де потужність процесора є основною проблемою, і де за потреби можна додати додаткові компоненти, необхідні для створення повної обчислювальної системи.

Тобто, на відміну від мікропроцессору, мікроконтролер зазвичай включає не тільки ядро мікропроцесора, але також пам'ять, периферійні пристрої введення/виведення та інше спеціалізоване обладнання, таке як аналого-цифрові перетворювачі, таймери та комунікаційні інтерфейси. Така

інтеграція дозволяє мікроконтролеру виконувати широкий спектр завдань без додаткових компонентів.

Ще однією ключовою відмінністю мікроконтролерів від мікропроцесорів є їх енергоспоживання. Мікроконтролери, як правило, розроблені для споживання дуже мало енергії, що робить їх ідеальними для додатків, що живляться від батарейок, та інших систем з низьким енергоспоживанням. Мікропроцесори, з іншого боку, часто є енергоємнішими та потребують додаткових компонентів для керування енергоспоживанням.

Загалом, основні відмінності між мікроконтролерами та мікропроцесорами полягають у рівні інтеграції та споживанні енергії. Мікроконтролери — це завершені обчислювальні системи на одному кристалі, тоді як мікропроцесори містять лише процесор обчислювальної системи. Мікроконтролери також зазвичай розроблені для споживання дуже мало енергії, тоді як мікропроцесори часто є більш енергоємними. Відмінність між мікропроцесорними системами та мікроконтролерами схематично показано на рисунках 1.5 та 1.6.

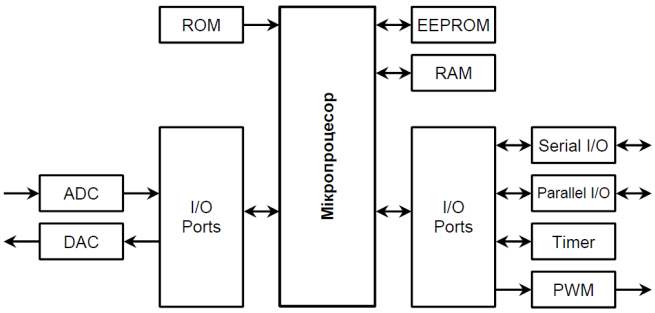


Рисунок 1.5 - Конфігурація мікропроцесора

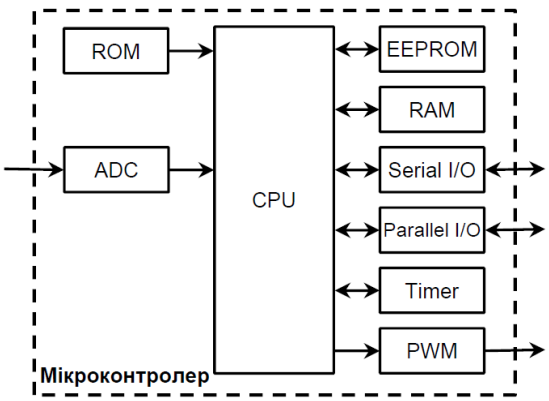


Рисунок 1.6 - Конфігурація мікроконтролера

CPU (Central Processing Unit) – центральний процесор.

RAM (Random Access Memory) – пам’ять з довільним доступом, чи оперативний запам’ятовуючий пристрій. RAM призначена для зберігання проміжних результатів та інших тимчасових даних протягом виконання програми.

ROM (Read Only Memory) – пам’ять тільки для читання, чи постійно запам’ятовуючий пристрій. ROM зберігає програмні інструкції (програму виконання) та таблиці з довідковими даними. У сучасних МК вона реалізована у вигляді flesh-пам’яті.

EEPROM (Electrically Erasable and Programmable ROM) – енергонезалежна пам’ять даних. У сучасних МК вона представляє собою flash-пам’ять. Основна відмінність від flash-пам’яті програм (ROM) – це можливість вибіркового програмування окремих байтів, на відміну від поблокового програмування flash-пам’яті програм.

I/O Ports (input/output) – паралельні порти вводу виводу надають інтерфейс між МК та периферійними пристроями вводу/виводу, такими як: клавіатура, дисплей тощо. Ніжки портів можуть бути як двонаправленими, так і однонаправленими, або лише на вхід, або на вихід.

Serial I/O – послідовні порти для обміну даними, що реалізовують асинхронні (UART) або синхронні (SPI) інтерфейси обміну даними. Асинхронний інтерфейс використовує протокол зі стартовим та стоповим бітами для передачі та прийому. Стартовий та стоповий біти вбудовані у кожен байт даних. Синхронні інтерфейси використовують синхронізуючі імпульси для кожного біта.

Timer/Counter (таймер/лічильник) – використовують для відліку часу або/та меж часових інтервалів між подіями, підрахунку кількості подій та генерації швидкості передачі даних (у бодах) для послідовних портів. Таймери також можуть керувати певними I/O- ногами у МК, наприклад, здійснювати підрахунок кількості імпульсів, що поступають на його вхід, чи навпаки, виводити певні послідовності імпульсів.

PWM (Pulse Width Modulation, широтно-імпульсна модуляція, ШІМ) – це спосіб кодування аналогового сигналу шляхом зміни ширини (тривалості) прямокутних імпульсів несучої частоти. Найбільш часто PWM використовують для керування моторами різних типів та активним навантаженням, наприклад, лампою розжарювання.

ADC (Analog to Digital Converter) – аналогово-цифровий перетворювач забезпечує інтерфейс для роботи з аналоговими пристроями, наприклад, з давачами, що видають аналогові електричні еквіваленти для фактичних фізичних параметрів, які ми хочемо контролювати.

DAC (Digital to Analog Converter) – цифро-аналоговий перетворювач забезпечує інтерфейс для роботи з виконавчими пристроями.

### 1.2.3.3 Класифікація мікроконтролерів та їх архітектура

Мікроконтролери можна класифікувати на основі їх архітектури, набору інструкцій і організації пам’яті. Основні типи мікроконтролерів [8]:

8-розрядні мікроконтролери: це найпростіший тип мікроконтролерів з обмеженим об’ємом пам’яті та обчислювальною потужністю. Вони мають 8-розрядну шину даних, що означає, що вони можуть обробляти дані 8-розрядними фрагментами. Зазвичай використовуються в малопотужних і недорогих програмах та в простих додатках, які не вимагають великої обчислювальної потужності, таких як побутова техніка, пульти дистанційного керування, іграшки.

16-розрядні мікроконтролери: ці мікроконтролери мають 16-розрядну шину даних, тобто можуть обробляти дані 16-розрядними фрагментами. Вони мають більше пам’яті та обчислювальної потужності, ніж 8-розрядні мікроконтролери. Це робить їх придатними для використання у більш складних програмах, таких як автомобільні системи, медичні пристрої та промислові системи керування.

32-розрядні мікроконтролери: ці мікроконтролери мають 32-розрядну шину даних, тобто можуть обробляти дані 32-розрядними фрагментами. Вони

мають більше пам’яті та обчислювальної потужності, ніж 16-розрядні мікроконтролери. Тому їх часто використовують у високопродуктивних програмах, таких як ігрові консолі, мультимедійні пристрої та мережеве обладнання, або, наприклад, у побутовій електроніці високого класу та медичних пристроях.

Архітектура мікроконтролера стосується способу організації та взаємозв’язку його компонентів. До найпоширеніших відносяться дві фундаментальні архітектури, які використовуються процесорами для доступу до пам’яті: це архітектура Фон-Неймана (Von Neumann architecture, також відома ще як Прінстонська), та Гарвардська архітектура (Harvard architecture).

Мікроконтролери Гарвардської архітектури (рис. 1.7): у мікроконтролерах цього типу пам’ять програм (програмних інструкцій) і пам’ять даних розділені та мають різні шини для передачі даних. Тобто, ця архітектура має окремі шини для даних та інструкцій, а також окрему пам’ять для кожної, що забезпечує більш швидке виконання інструкцій, оскільки можна отримати доступ до пам’яті програм і даних одночасно. Це робить їх швидшими та ефективнішими, ніж інші архітектури. Більшість сучасних МК використовують саме Гарвардську архітектуру пам’яті. Приклади мікроконтролерів гарвардської архітектури: мікроконтролери Atmel AVR і PIC.

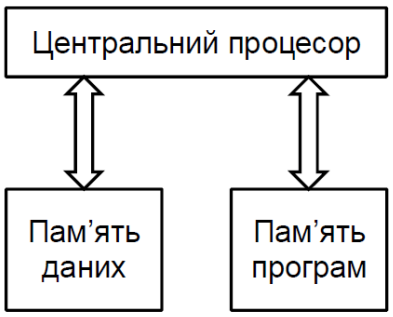


Рисунок 1.7 – Гарвардська архітектура

Мікроконтролери архітектури фон Неймана (рис. 1.8): у мікроконтролерах цього типу пам’ять програм і пам’ять даних використовують одну шину. Тобто, програмні інструкції та дані зберігаються в одному просторі пам’яті. Таких підхід, з одного боку, спрощує та здешевлює

виробництво мікроконтролерів, дана архітектура є простішою та економічнішою, ніж Гарвардська, але з іншого боку, це може призвести до зниження продуктивності, тобто може обмежуватися швидкість виконання. Тому зазвичай вона використовується в комп'ютерах загального призначення, і рідше в мікроконтролерах. Приклади мікроконтролерів, що побудовані на архітектурі фон Неймана – це мікроконтролери 8051 і ARM.

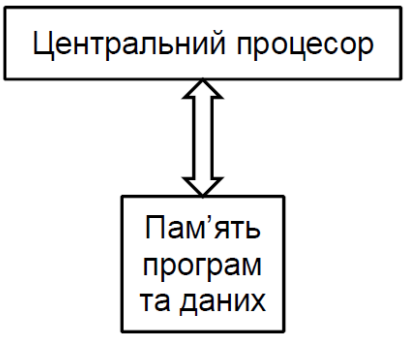


Рисунок 1.8 – Архітектура фон Неймана

Архітектура команд CISC і RISC:

CISC (Complex Instruction Set Computing) і RISC (Reduced Instruction Set Computing) — це два різні підходи до розробки архітектури набору інструкцій (ISA) комп'ютера або мікропроцесора. Основна відмінність між архітектурами CISC і RISC полягає в складності інструкцій, які вони використовують.

Архітектури CISC використовують складні інструкції, які можуть виконувати кілька операцій в одній інструкції. Ці інструкції можуть містити такі операції, як доступ до пам'яті, арифметичні та логічні операції та умовні переходи. Перевага архітектури CISC полягає в тому, що вона може зменшити кількість інструкцій, необхідних для виконання завдання, що може призвести до більш ефективного коду та швидшого часу виконання.

Архітектури RISC, з іншого боку, використовують простіший набір інструкцій, які виконують одну операцію. Ці інструкції зазвичай виконуються за один такт, що робить їх швидшими, ніж більш складні інструкції, що використовуються в архітектурах CISC. Однак, оскільки інструкції RISC

простіші, для виконання завдання може знадобитися більше їх, що може призвести до збільшення розміру коду.

Ось деякі ключові відмінності між архітектурами CISC і RISC:

Розмір інструкції: інструкції CISC, як правило, більші та складніші, ніж інструкції RISC, які зазвичай менші та простіші.

Час виконання: інструкції CISC можуть виконуватися довше, оскільки вони складніші, тоді як інструкції RISC зазвичай швидше, оскільки вони простіші.

Розмір коду: Інструкції RISC, як правило, компактніші, ніж інструкції CISC, що може призвести до меншого розміру коду.

Доступ до пам’яті: архітектури CISC часто мають складні режими адресації пам’яті, що може бути корисним у деяких ситуаціях, але також може сповільнити час виконання.

Вибір між архітектурами CISC і RISC залежить від конкретних потреб програми. Архітектури CISC краще підходять для додатків, які вимагають складних інструкцій і можуть виграти від зменшеного розміру коду, тоді як архітектури RISC краще підходять для додатків, які вимагають швидкого виконання та можуть терпіти більший розмір коду.

Майже усі сучасні мікроконтролери побудовані за RISC-архітектурою. До мікроконтролерів побудованих за CISC-архітектурою відносяться мікроконтролери фірми Intel з ядром MCS-51, які на сьогодні за ліцензією випускаються цілим рядом інших виробників.

Загалом вибір архітектури мікроконтролера залежатиме від конкретних вимог програми, включаючи продуктивність, енергоспоживання та вартість.

Отже, як бачимо, мікроконтролери відіграють життєво важливу роль у різних галузях промисловості та застосуваннях. Ці невеликі програмовані універсальні пристрої переважно використовуються в інтелектуальних системах автоматизації та контролю, забезпечуючи ефективне та надійне керування різноманітними приладами, функціями та процесами. Вони

компактні, малопотужні і економічно вигідні, саме це і робить їх ідеальними для використання в розумних системах, де необхідні контроль і автоматизація.

### 1.2.4 Використання мереж в інтелектуальних системах

Щоб забезпечити дистанційний моніторинг і керування пристроями, інтелектуальні системи управління та автоматизації часто покладаються на комунікаційні мережі. Комунікаційні мережі можуть мати різні форми, включаючи дротові та бездротові мережі, локальні мережі (LAN), глобальні мережі (WAN) та Інтернет. Ці мережі можна використовувати для передачі даних у режимі реального часу або для зберігання та пересилання даних для подальшої обробки. Для обміну даними та інформацією, а також для забезпечення взаємодії між компонентами системи, дані системи контролю використовують різні протоколи та стандарти передачі даних [6].

#### 1.2.4.1 Класифікація комп'ютерних мереж

Комп’ютерна мережа — це група взаємопов’язаних пристроїв, таких як комп’ютери, сервери, маршрутизатори, комутатори, принтери та інші пристрої, які спілкуються та обмінюються даними один з одним. Мережа може бути розташована в одному фізичному місці або охоплювати кілька місць, наприклад офіси, будівлі чи міста. Для зв’язку один з одним комп’ютери використовують загальні протоколи зв’язку через цифрові з’єднання. Ці взаємозв’язки складаються з технологій телекомунікаційних мереж, заснованих на фізично-дротових, оптичних і бездротових радіочастотних методах, які можуть бути організовані в різноманітних мережевих топологіях [9, 10].

Комп’ютерні мережі можна класифікувати за різними критеріями, такими як розмір, географічне покриття, топологія та протокол зв’язку, область застосування та призначення, тощо. Одним із найпоширеніших критеріїв класифікації є їх розмір. Ось основні три типи мереж за областю дії:

LAN (Local Area Network, переклад – локальна мережа) – це мережа, яка охоплює невелику територію чи географічну область, як правило, одну будівлю, офіс або кампус. Зазвичай нею володіє та керує одна організація, і вона використовується для спільного використання таких ресурсів, як принтери, файли та доступ до Інтернету.

MAN (Metropolitan Area Network, у перекладі – міська мережа або мережа мегаполісів): – це мережа, яка охоплює більшу географічну територію, ніж локальна мережа, як правило, це місто чи селище. Вона може складатися з кількох взаємопов'язаних локальних мереж.

WAN (Wide Area Network, у перекладі – глобальна мережа): – це мережа, яка охоплює велику географічну територію, наприклад місто, країну чи навіть увесь світ. Зазвичай нею володіють і керують кілька організацій і використовується для обміну даними та ресурсами між географічно рознесеними місцями.

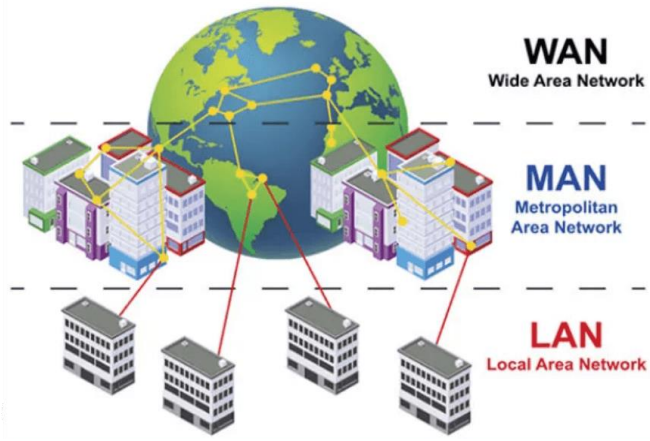


Рисунок 1.9 – Класифікація комп'ютерних мереж за областю дії

До деяких інших типів комп’ютерних мереж належать:

PAN (Personal Area Network, переклад – персональна мережа): – це найменший тип мережі, що використовується для зв’язку між пристроями на невеликій відстані (зазвичай у межах 10 метрів). Інколи її ще називають мережою, яка побудована «навколо» людини. Дана мережа використовується для об'єднання персонального електронного обладнання користувача,

наприклад, смартфони, ноутбуки, бездротові навушники, тощо. Такими стандартизованими мережами в наш час є Bluetooth, ZigBee, Piconet.

WLAN (Wireless Local Area Network - бездротова локальна мережа): це локальна мережа, яка використовує технологію бездротового зв'язку, наприклад Wi-Fi, для підключення пристроїв до мережі.

Cloud Computing Network (Мережа хмарних обчислень): – це віртуальна мережа, яку використовують для надання послуг і програм через Інтернет за допомогою постачальника хмарних послуг. Хмарні мережі дозволяють користувачам зберігати та отримувати доступ до даних і програм із віддалених серверів і пристроїв, що знаходяться у різних місцях.

1.2.4.2 Особливості використання локальних мереж та мереж WiFi в автоматизованих системах

Локальні мережі та мережі Wi-Fi — це два типи мереж, які зазвичай використовуються в інтелектуальних системах, але вони відрізняються за обсягом і можливістю підключення [6].

Локальні мережі: — це мережі, які з'єднують пристрої в межах обмеженої географічної області. Для встановлення зв'язку між пристроями в локальних мережах використовуються дротові з'єднання, такі як кабелі Ethernet або оптоволокно. Зазвичай вони забезпечують високу швидкість і надійність передачі даних, що робить їх придатними для критичних за часом програм, де потрібна низька затримка та детермінована поведінка. Прикладами технологій локальної мережі є Ethernet, системи польових шин (наприклад, Modbus або Profibus) і промислові варіанти Ethernet (такі як EtherNet/IP або PROFINET).

Мережі Wi-Fi, з іншого боку, є бездротовими мережами, які дозволяють пристроям підключатися та спілкуватися без потреби у фізичних кабелях. Мережі Wi-Fi використовують радіочастоти для встановлення бездротових з'єднань між пристроями та точками доступу. Мережі Wi-Fi зазвичай розгортаються в певній зоні, наприклад у домі, офісі чи громадському місці, і

пропонують мобільність і гнучкість підключення пристроїв.

Мережі Wi-Fi забезпечують бездротовий доступ до Інтернету та дозволяють пристроям підключатися до локальної мережі та отримувати доступ до таких ресурсів, як принтери, спільні файли або підключене до мережі сховище. Мережі Wi-Fi використовують стандарт IEEE 802.11 і працюють у неліцензійних діапазонах частот. Вони пропонують зручне підключення, дозволяючи користувачам підключати свої смартфони, ноутбуки та інші пристрої до мережі без необхідності фізичного підключення.

Тобто, як бачимо, локальні мережі та мережі Wi-Fi не є взаємовиключними, вони можуть доповнювати одна одну в розумній системі. Мережі Wi-Fi можна розглядати як розширення локальної мережі, що забезпечує бездротове підключення в межах території, охопленої інфраструктурою локальної мережі. Мережі Wi-Fi дозволяють пристроям підключатися до локальної мережі та отримувати доступ до ресурсів так само, як пристрої, підключені через дротове з’єднання.

Наприклад, у розумному домі такі пристрої, як інтелектуальні термостати, системи освітлення або камери спостереження, можна підключити до локальної мережі за допомогою кабелів Ethernet. У той же час мобільні пристрої, ноутбуки та інші портативні пристрої можуть підключатися до однієї локальної мережі через Wi-Fi, забезпечуючи бездротове керування та доступ до системи розумного дому.

У цьому сценарії локальна мережа забезпечує зв’язок і контроль, а мережа Wi-Fi забезпечує гнучкість і мобільність для пристроїв користувачів. Обидві мережі працюють разом, щоб забезпечити безперебійне підключення, обмін даними та контроль у розумній системі.

Варто зазначити, що вибір між локальними мережами та мережами Wi-Fi залежить від конкретних вимог програми. Локальні мережі надають перевагу в промислових умовах або в ситуаціях, коли зв’язок у реальному часі та детермінована поведінка є критичними. Мережі Wi-Fi більше підходять для програм, де потрібна мобільність, зручність і бездротове підключення.

1.2.4.3 Типи мережевих протоколів і їх призначення

Мережевий протокол — це набір правил і процедур, які регулюють обмін даними між різними пристроями в мережі. Він визначає, як дані передаються, приймаються та обробляються, забезпечуючи ефективний зв’язок пристроїв один з одним [10].

Модель OSI (англ. Open Systems Interconnection Basic Reference Model, базова еталонна модель взаємодії відкритих систем): це концептуальна основа, яка стандартизує зв’язок між різними системами та протоколи зв’язку, що використовуються в мережі. Простими словами, це абстрактна модель для комунікацій і розробки мережевих протоколів. Вона складається із 7-ми рівнів. Кожен має свій унікальний набір функцій і протоколів та відповідає за певний аспект комунікаційного процесу, від фізичної передачі до послуг на рівні програми. Модель OSI корисна розуміння того, як різні компоненти мережі взаємодіють один з одним, і для усунення проблем мережі. Та потрібна, щоб забезпечити стандартну модель для мережевого зв’язку, яка допоможе забезпечити взаємодію та сумісність між різними пристроями та протоколами.

| Тип даних | Рівень | Функції |
|---|---|---|
| Дані | 7. Прикладний (Application) | Доступ до мережевих служб |
| | 6. Представницький (Presentation) | Представлення і шифрування даних |
| | 5. Сеансовий (Session) | Управління сеансом звязку |
| Сегменти | 4. Транспортний (Transport) | Прямий звязок між кінцевими пунктами та надійність |
| Пакети | 3. Мережевий (Network) | Визначення маршруту та логічна адресація |
| Кадри | 2. Канальний (Data Link) | Фізична адресація |
| Біти | 1. Фізичний (Physical) | Робота з середовищем передачі, сигналами та двійковими даними |

Рисунок 1.10 – Модель OSI: функції та типи даних

| Рівень OSI | Протоколи |
|---|---|
| прикладний | HTTP, gopher, Telnet, DNS, DHCP, SMTP, SNMP, CMIP, FTP, TFTP, SSH, IRC, AIM, NFS, NNTP, NTP, SNTP, XMPP, FTAM, APPC, X.400, X.500, AFP, LDAP, SIP, IETF, RTP, RTCP, ITMS, Modbus TCP, BACnet IP, IMAP, POP3, SMB, MFTP, BitTorrent, e2k, PROFIBUS<br>Це всього лише кілька найрозповсюдженіших протоколів прикладного рівня, яких існує неймовірно велика кількість. Всі їх неможливо описати в рамках даної статті. |
| представлення | ASN.1, XML, TDI, XDR, NCP, AFP, ASCII, Unicode |
| сеансовий | ASP, ADSP, DLC, Named Pipes, NBT, NetBIOS, NWLink, Printer Access Protocol, Zone Information Protocol, SSL, TLS, SOCKS, PPTP |
| транспортний | TCP, UDP, NetBEUI, AEP, ATP, IL, NBP, RTMP, SMB, SPX, SCTP, DCCP, STP, TFTP, RTP |
| мережевий | IPv4, IPv6, ICMP, IGMP, IPX, NWLink, NetBEUI, DDP, IPSec, SKIP |
| канальний (Ланки даних) | ARCnet, ATM, DTM, SLIP, SMDS, Ethernet, ARP, FDDI, Frame Relay, LocalTalk, Token Ring, PPP, PPPoE, StarLan, WiFi, PPTP , L2F, L2TP, PROFIBUS |
| фізичний | RS-232, RS-422, RS-423, RS-449, RS-485, ITU-T, RJ-11, T-carrier (T1, E1), модифікації стандарту Ethernet: 10BASE-T, 10BASE2, 10BASE5, 100BASE-TX, 100BASE-FX, 100BASE-T, 1000BASE-T, 1000BASE-TX, 1000BASE-SX |

Рисунок 1.11 – Модель OSI: рівні та протоколи

Ось короткий огляд кожного рівня та деяких загальних протоколів, які використовуються на кожному рівні:

Фізичний рівень (англ. Physical Layer) : цей рівень стосується фізичної передачі даних через мережу. Протоколи, які використовуються на цьому рівні, включають Ethernet, Wi-Fi і Bluetooth.

Канальний рівень (відомий ще як рівень каналу даних, англ. Data Link Layer): цей рівень відповідає за встановлення та підтримку зв’язку між суміжними вузлами мережі. Протоколи, які використовуються на цьому рівні, включають ARP (Address Resolution Protocol), PPP (Point-to-Point Protocol).

Мережевий рівень (англ. Network Layer): цей рівень відповідає за маршрутизацію пакетів даних між різними мережами. Протоколи, які використовуються на цьому рівні, включають IP (Internet Protocol, забезпечує зв’язок між пристроями в мережі), ICMP (Internet Control Message Protocol).

Транспортний рівень (Transport Layer): цей рівень відповідає за надійну передачу даних між програмами на різних пристроях. Протоколи, які використовуються на цьому рівні, включають TCP (Transmission Control Protocol, протокол керування передачею, забезпечує надійну та ефективну передачу даних в мережі), UDP (User Datagram Protocol, протокол дейтаграм користувача), і т.д.

Сеансовий рівень (рівень сеансу, англ. Session Layer): цей рівень відповідає за встановлення, підтримку та завершення сеансів зв’язку між програмами. Протоколи, які використовуються на цьому рівні, включають NetBIOS (Network Basic Input / Output System, базова мережева система введення / виведення), RPC (Remote Procedure Call, віддалений виклик процедур).

Презентаційний рівень (рівень представлення, англ. Presentation Layer): цей рівень відповідає за перетворення даних у формат, зрозумілий прикладному рівню, за перетворення протоколів і кодування/декодування даних. Отримані з мережі дані перетворює у формат, зрозумілий для додатків та застосунків. Протоколи, які використовуються на цьому рівні, включають SSL/TLS (Secure Sockets Layer/Transport Layer Security, рівень захищених сокетів/безпеку транспортного рівня), MIME (Multipurpose Internet Mail Extensions , багатоцільові розширення Інтернет-пошти).

Прикладний рівень (рівень додатків, англ. Application Layer): цей рівень відповідає за надання мережевих послуг додаткам, забезпечує взаємодію мережі й користувача. Протоколи, які використовуються на цьому рівні, включають HTTP (Hypertext Transfer Protocol , протокол передачі гіпертексту, використовується для передачі веб-сторінок через Інтернет), FTP (File Transfer Protocol , протокол передачі файлів, використовується для передачі файлів між пристроями в мережі), DNS (Domain Name System, протокол для перетворення доменних імен в IP-адреси), SMTP (Simple Mail Transfer Protocol, протокол передачі пошти, використовується для надсилання та отримання повідомлень електронної пошти), тощо.

Це лише кілька прикладів мережевих протоколів, до рівнів яких вони належать у моделі OSI. На кожному рівні використовується багато інших протоколів, кожен з яких призначений для певної мети та забезпечує зв’язок між пристроями в мережі.

Загалом, комунікаційні мережі та мережеві протоколи є критично важливими компонентами інтелектуальних систем керування та

автоматизації. Адже вони забезпечують безперебійну інтеграцію та координацію різних компонентів системи, забезпечують підключення та обмін даними, що необхідні для оптимальної продуктивності та ефективної роботи системи у цілому.

## 1.3 Огляд аналогів та існуючих систем розумного освітлення

На ринку вже існують готові системи управління освітленням та розумні освітлювальні пристрої. Кожен із них має свої переваги та недоліки. Серед популярних продуктів виділяються розумні лампи, настільні світильники, різні смартрозетки, світлодіодні стрічки, датчики руху, тощо. Розглянемо деякі із них.

LED лампи компанії «Yangcsl» (Рисунок 1.12). Це лампочка зі зміною кольору RGB. Вона має 6 режимів роботи (Flash, Strobe, Smooth, Warm, Fresh, Romantic) та функцію таймера: світло автоматично вмикатиметься та вимикатиметься щодня у заданий час. Кожна лампа комплектується своїм дистанційним пультом керування. І тільки через нього здійснюється управління цією лампою.


Рисунок 1.12 – LED лампи компанії «Yangcsl».

LED лампа компанії «SIOLAXEN» (Рисунок 1.13), як і попередня, зі зміною кольору RGB. Вона має вбудований динамік, що дає можливість відтворювати музику через неї. Посередині лампи є дискотечна куля, що

проектує різнокольорові вогні. Це дозволяє зробити домашні вечірки та дитячі свята веселими. Управління здійснюється за допомогою дистанційного пульта керування або за допомогою мобільного смартфону, підключення здійснюється лише через Bluetooth.

Рисунок 1.13 – LED лампа компанії «SIOLAXEN».

Система управління освітленням компанії «Brilliant» (Рисунок 1.14) відноситься до систем розумного дому, оскільки вона дозволяє керувати не тільки освітленням, а й температурою та безпекою будинку (замками, камерами, домофоном).


Рисунок 1.14 – Система управління освітленням компанії «Brilliant».

Система має вбудовану функцію Alexa, щоб надавати команди голосом. Також є можливість вмикати музику. Управління здійснюється через додаток, встановлений на смартфон, або за допомогою сенсорної панелі. Панель керування вмонтовується в стіну замість звичайних перемикачів. Вартість такої системи на момент написання дипломної роботи становить від 350 до 600$ (в залежності від кількості перемикачів на панелі).

Отже, до основних переваг розумних освітлювальних пристроїв можна віднести:

- невисоку доступну ціну;

- вони не вимагають складного монтажу.

Проте з цього випливають (з'являються) і їхні недоліки:

- багато розумних ламп та світильників комплектуються рідним пультом, і можна керувати всіма можливостями з нього. Тобто дистанційно управляти освітленням можна лише в одній кімнаті, не по всій квартирі. До того ж такі пристрої підключити безпосередньо до смартфону або планшету для отримання розширеного набору можливостей не вдасться;

- якщо і є можливість керувати розумним пристроєм через смартфон, то зазвичай додаток містить дуже примітивний функціонал без розширених налаштувань;

- щоб підключитися до пристрою за допомогою смартфона на Android або iOS, більшість пристроїв використовують Bluetooth-модуль, вбудований в телефон (не WiFi);

- якщо пристрій і використовує WiFi для підключення до смартфону, то ціна його буде на порядок вища. До того ж із відгуків користувачів, часто при підключенні пристрою в додатку у них виникають труднощі (додаток не виявляє пристрій, бо потрібен дуже гарний сигнал WiFi).

Готові системи управління освітленням мають такі переваги:

- автоматизація. Освітлювальні пристрої можуть бути підключені до єдиної системи «розумного освітлення». Відповідно це надає можливість

керувати та управляти ними в автоматичному режимі. До того ж це значно економить час та витрати за комунальні рахунки;

- вся система управляється єдиним пристроєм, і найчастіше це мобільний телефон;

- підключення до сматрфону відбувається в основному за допомогою бездротової мережі типу Wi-Fi, що дозволяє керувати освітленням через інтернет, навіть на віддаленій відстані, коли людина знаходиться не вдома;

Але не дивлячись на всі ці переваги, все ж таки системи управління освітленням не так сильно поширені. Тому що мають наступні недоліки:

- зависока ціна (коштують такі рішення недешево);

- вимагають складного монтажу;

- зазвичай ці системи багатофункціональні, і мають дуже високу ступінь гнучкості налаштувань. Більшість із цих функцій виявляються «непотрібними» користувачу, а висока ступінь налаштувань робить інтерфейс корисувача заплутаним і не зручним у використанні.

Саме тому, провівши аналіз предметної області та здійснивши огляд існуючих систем і аналогів, було прийнято рішеня створити та розробити власну систему розумного освітлення, яка буде мати простий та user-friendly інтерфейс користувача з мінімальним набором базових функцій, які задовольнять наші потреби. До того ж її кінцева ціна буде не високою.

## 1.4 Основні параметри та вимоги майбутньої системи

Загальний функціонал майбутньої системи повинен забезпечувати можливість дистанційно керувати освітленням будинку через WiFi, використовуючи Web-інтерфейс.

Система повинна зберігати дані. Для зберігання та керування даними система має використовувати базу даних. Дані повинні передаватися по бездротовій мережі WiFi.

Програмний продукт повинен бути зручним у використанні, мати привабливий та інтуїтивно зрозумілий інтерфейс користувача.

Інтерфейс користувача має бути доступним через браузер на комп’ютері та/або на мобільному пристрої.

Веб-інтерфейс повинен надавати користувачу можливість керувати світлом дистанційно, встановлювати графік, планувати увімкнення та вимкнення освітлення у кімнатах в заданий час, а також переглядати історію та статистичні дані щодо тривалості використання електроенергії у певний період та у певній кімнаті.

Система керування (центральний блок керування) повинен бути реалізований на базі мікроконтроллера, який забезпечить зв’язок між освітлювальними приладами та іншими пристроями в системі. Також мікроконтроллер повинен забезпечувати вбудовані можливості WiFi, щоб безперешкодно інтегрувати його в бездротову мережу. Адже зв'язок між пристроєм та сервером а також передача даних буде здійснюватися через бездротову мережу WiFi.

Отже, основні функції майбутньої системи наступні:

а) дистанційне керування світлом в кожній кімнаті;

б) планування і встановлення автоматичного включення або виключення світла за допомогою прив’язки до часу доби (увімкнення та вимкнення світла за графіком);

в) збереження даних та перегляд історії і статистичних даних.

## 1.5 Висновки за розділом

У даному розділі було здійснено поглиблений огляд та аналіз предметної області інтелектуальних систем. Було детально описано автоматизовані системи, зокрема системи розумного будинку та підсистеми освітлення. Було досліджено та розглянуто ключові компоненти і технології, що використовуються в автоматизованих системах, їх інтеграція та ролі. Було

пояснено фундаментальні концепції технологій IoT, описано складові веб-технологій, представлено мікроконтролери, їх класифікацію та архітектурні аспекти. Також було досліджено використання комп’ютерних мереж та мережевих протоколів в інтелектуальних системах, описано особливості та взаємозв’язок локальних і WiFi мереж в розумних системах.

Також було проведено та представлено огляд аналогів та існуючих систем розумного освітлення. Визначено основні параметри та вимоги майбутньої системи. Проектування та розробка власної інтелектуальної системи освітлення буде здійснюватися з урахуванням визначених параметрів і вимог.

# РОЗДІЛ 2
# ВИБІР ТА ОБҐРУНТУВАННЯ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

## 2.1 Вибір програмних засобів розробки проекту

Для створення додатку і інтерфейсу користувача було вирішено використовувати наступні технології та інструменти: мову розмітки гіпертексту HTML, каскадні таблиці стилів CSS, мову програмування JavaScript, фронтенд бібліотеку React, середовище виконання Node.JS, бекенд фреймворк Express.js, реляційну систему управління базами даних MySQL, мову структурованих запитів SQL, графічний інтерфейс адміністрування систем управління базами даних DBeaver, програмну платформу Docker, та інтегроване середовище розробки Visual Studio Code.

### 2.1.1 Мова гіпертекстової розмітки HTML

HTML (Hypertext Markup Language) — стандартна мова розмітки, яка використовується для створення та структурування веб-сторінок. Вона є основою Всесвітньої павутини та дозволяє створювати документи, які можна переглядати на різних пристроях і веб-браузерах. Іншими словами, це стандартизована мова розмітки документів для перегляду веб-сторінок у браузері. Веб-браузери отримують HTML документ від сервера за протоколами HTTP/HTTPS або відкривають з локального диска, далі інтерпретують код в інтерфейс, який відображатиметься на екрані монітора [5].

HTML вперше був представлений Тімом Бернерсом-Лі в 1991 році як засіб для обміну науковою та технічною інформацією між дослідниками. Ідея гіпертекстової системи полягає у тому, що юзер (користувач) має можливість переглядати документи (сторінки тексту) не послідовно, як це прийнято при читанні книг, а у найбільш зручному для себе порядку. Це досягається шляхом

створення спеціального механізму пов`язуваня різних сторінок тексту за допомогою гіпертекстових посилань.

Мова розмітки надає набір тегів і атрибутів, які визначають структуру, вміст і представлення елементів у веб-документі. Вона дозволяє визначити структуру електронного документа з поліграфічним рівнем оформлення. Результуючий документ може містити різноманітні елементи: ілюстрації, аудіо і відео фрагменти. Мова HTML включає розвинені засоби для визначення кількох рівнів заголовків, шрифтових виділень, різних груп об’єктів та багато інших можливостей. Тобто, HTML впроваджує засоби для:

- створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації зі Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

HTML використовує тегову модель як основу моделі розмітки документів. Тегова модель описує документ як сукупність контейнерів (блоків). Кожен блок починається і закінчується тегами. Вони прості і зрозумілі використанні. Теги створені за допомогою загальновживаних слів англійської мови, мають зрозумілі скорочення і позначення. Найчастіше HTML-теги складаються з початкового і кінцевого компонентів, між якими розміщуються текст та інші елементи документа. Ім’я кінцевого тега ідентичне імені початкового тегу, але перед ім’ям кінцевого тегу ставиться коса риска (/). Елементи (базові компоненти розмітки HTML) мають дві основні властивості: атрибути та вміст (контент). Атрибути містять додаткову інформацію про елемент, що не показується у фактичному контенті. Більшість з атрибутів елемента являє собою пару «назва-значення», які розділені між собою знаком рівності, та записані одразу після назви елемента у початковому тегу. На Рисунках 2.15 та 2.16 показано приклад елементу <p> (абзац) та атрибуту. В даному випадку, class – це назва атрибута, а editor-note – його

значення. Атрибут class впроваджує засіб об'єднання схожих елементів у класи.

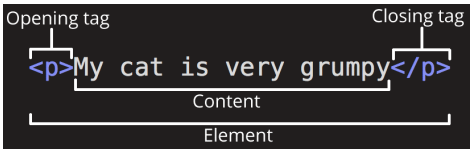


Рисунок 2.15 – Елемент абзацу


Рисунок 2.16 – Атрибут елементу

Документ HTML складається з 4х основних елементів (нижче на рисунку 2.17 показано основну структуру документу HTML):

Оголошення типу документа (<!DOCTYPE>): у ньому вказується версія HTML, яка використовується. Для HTML5 декларація doctype виглядає наступним чином: <!DOCTYPE html>.

Елемент HTML: кореневим елементом документа HTML є <html>. Він охоплює весь вміст документа.

Елемент Head: Елемент <head> містить метаінформацію про документ, таку як назва документа, кодування символів, пов’язані таблиці стилів і сценарії. Він не відображає жодного видимого вмісту на веб-сторінці.

Елемент Body: Елемент <body> містить видимий вміст веб-сторінки, включаючи текст, зображення, посилання, форми та інші елементи.

```
<!DOCTYPE html>
    <html>

        <head>
            <title> Title here </title>
        </head>

        <body>
            Web page content goes here.
        </body>

    </html>
```

Рисунок 2.17 – структура HTML документу

### 2.1.2 Каскадні таблиці стилів CSS

CSS (від англ. Cascading Style Sheets, переклад - каскадні таблиці стилів) - це так звана «мова стилів». За допомогою неї описують зовнішній вигляд документів (як і де відображати елементи веб-сторінки). CSS, як і HTML та JavaScript, є основною технологією всесвітньої павутини. Каскадні таблиці стилів використовуються для запису оформлення сторінок (документів), які розмічаються мовою HTML, XHTML або XML. Проте, найчастіше CSS комбінується саме з мовою розмітки HTML [5].

Дана мова стилів прийшла на заміну табличному верстанню вебсторінок. Поява CSS стало революцією в світі web-дизайну. Використовуючи CSS, можна змінювати колір тексту, відстань між параграфами, стиль шрифтів, розміри та розташування колонок (блоків), використовувати фонові зображення, макети дизайну, і безліч інших ефектів. Каскадні таблиці стилів також дозволяють адаптувати контент сторінки до різних умов відображення (в залежності чи це екран монітора, мобільного пристрою, телевізора, чи документ у роздрукованому вигляді, тощо.). Головна перевага блочного верстання - це те, що є можливість розділити зміст сторінки (дані, контент) від її візуальної презентації (зовнішнього вигляду документу) . Це забезпечує потужний контроль над поданням HTML-документа.

CSS легко освоїти і зрозуміти. Мова стилів має простий синтаксис. Для найменування різних складових стилю, вона використовує англійські слова. Стилі складаються зі списку правил, кожне має один або більше селектор (англ. selector) та блок визначення (блок оголошення, від англ. – declaration block), який складається з оточеного фігурними дужками списку властивостей. Властивості у списку правил оформлюються у наступному вигляді:
{ назва властивості : значення ; } .

Тобто, кожне правило починається з покажчика (селектора), який вказує на html-елементи, до яких буде застосуватися певне css-правило. Найцікавіше відбувається у блоці оголошень – саме тут встановлюється правила

відображення обраних елементів, визначається їх властивості, такі як розмір, колір, положення на екрані, тощо.

Нижче, на рисунку 2.18 зображено структуру CSS правила.

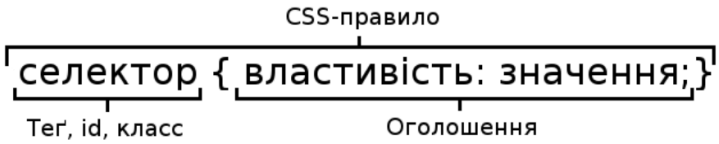


Рисунок 2.18 – Структура CSS правила

2.1.3 Мова програмування JavaScript

JavaScript (скорочено JS) – це динамічна та об'єктно-орієнтована прототипна мова програмування. Реалізована стандартом ECMAScript. Найчастіше вона використовується для створення сценаріїв веб-сторінок. Це надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд вебсторінки.

JavaScript класифікується як прототипна (підмножина об'єктно-орієнтованої), скриптова мова програмування з динамічною типізацією. Звісно, JavaScript підтримує і інші парадигми програмування (імперативну та частково функціональну). Вона має деякі відповідні архітектурні властивості, такі як: динамічна та слабка типізація, автоматичне управління пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

У перші роки існування, JavaScript використовували зазвичай програмісти-аматори, так як більшість професійних програмістів скептично ставилися до цієї мови програмування. Але поява технології AJAX змінила ситуацію та привернула увагу професійної спільноти до JavaScript, а її подальші модифікації за стандартами ES6+ внесли багато корисних можливостей. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза

браузером. Наразі вона є однією із найпопулярніших мов програмування (рис.2.19) [5].

Мову програмування JavaScript використовують для:

- для написання скриптів веб-сторінок та для надання їм інтерактивності;

- для створення односторінкових та прогресивних вебзастосунків (використовуючи React, AngularJS, Vue.js);

- для програмування на боці сервера (Node.js (Express.js));

- для написання десктопних та мобільних додатків (Electron, NW.js, React Native), тощо.

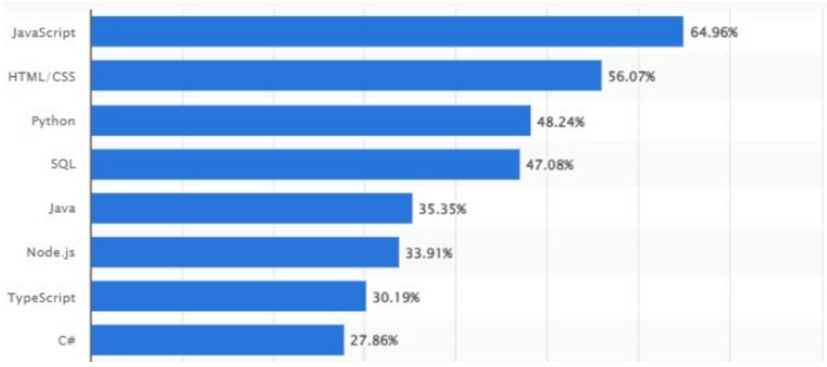


Рисунок 2.19 – Статистичні дані про найпопулярні мови програмування

### 2.1.4 Середовище виконання Node.JS

Node.js – це середовище виконання мови JavaScript. Іншими словами – це платформа, що використовується для виконання високопродуктивних мережевих застосунків, які були написані мовою програмування JavaScript. Це платформа з відкритим кодом, засновником якої є Раян Дал (англ. Ryan Dahl). Раніше мова JavaScript застосовувалася лише для обробки даних на стороні браузера користувача. Та платформа Node.js надала можливість виконувати скрипти, написані мовою JavaScript на стороні серверу, а також відправляти користувачеві результат їхнього виконання. Node.js перетворила мову JavaScript на мову загального використання. Тепер вона має велику спільноту розробників [12, 13].

Платформа Node.js має наступні властивості:

- асинхронну одно-нитеву (одно-потокову) модель виконання запитів;

- неблокуючий ввід/вивід (ініціація запитів паралельно);

- систему JavaScript модулів CommonJS;

- програму-рушій JavaScript від Google V8 («двигун»);

Отже, дана платформа використовується для створення клієнтських та серверних програм, а не лише для роботи із серверними скриптами для веб-запитів. Node.js надає можливість JavaScript підключатися до різних пристроїв введення-виведення (наприклад, камерам, мікрофонам, тощо), до різних бібліотек на різних мовах програмування. І це розширює можливості програми. Node.js має велику колекцію підготовлених модулів, щоб розширити функціональні можливості застосунків. Завдяки їм можна реалізацувати HTTP, FTP, DNS, IMAP, POP3 сервери і клієнти, зробити інтеграцію з різними веб-фреймворками; є модулі-обробники WebSocket і AJAX, конектори до різноманітних СУБД (такі як MySQL, PostgreSQL, SQLite, MongoDB), шаблонізатори, CSS-рушії, XML-парсери, тощо. Для керування модулями використовується пакетний менеджер npm (node package manager). NPM (менеджер пакетів) призначений для спрощення встановлення, оновлення та видалення бібліотек. Він полегшує програмістам публікацію та обмін початковим кодом бібліотек Node.js.

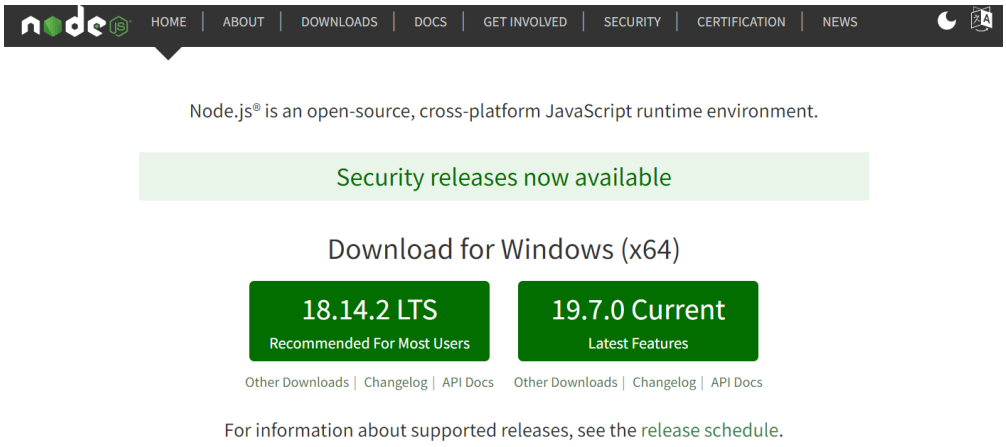


Рисунок 2.20 – Головна сторінка офіційного вебсайту Node.js

Найчастіше платформа Node.js застосовується як web сервер. Web-застосунок, побудований з використанням платформи Node.js, буде

продуктивним і не вимогливим до ресурсів. Використання мови javascript в серверній і клієнтській частинах – це унікальне рішення, яке прискорить розробку, а також гарантує масштабованість і легку підтримку проєкту.

### 2.1.5 Бібліотека React

React (попередні назви: React.js, ReactJS) – це декларативна, ефективна і гнучка JavaScript-бібліотека, призначена щоб створювати інтерфейси користувача. Відмінність декларативних інтерфейсів від імперативних у тому, що декларативні роблять код більш передбачуваним, такий код набагато легше налагоджувати. Розробникам потрібно лише описати, як кожна частина інтерфейсу виглядає в різних станах (states) додатку, а бібліотека буде міняти їх при отриманні потрібної команди [15].

Дану бібліотеку створив у 2011 році Джордан Волк (Jordan Walke), програміст з Facebook. У березні 2015-го, проект був представлений як відкрите програмне забезпечення. Нині вона розробляється і розвивається компанією Meta (раніше назва Facebook) та спільнотою індивідуальних розробників.

ReactJS не розрахована виключно на web. Її можна використовувати для створення програм віртуальної реальності, для розробки серверних додатків, наприклад, разом з Node.js, для розробки мобільних додатків, разом з React Native, і для створення програмного забезпечення, яке добре працює як на Android, так і на iOS платформі.

React надає можливість компонувати невеликі окремі частини коду (так звані «компоненти»), у складні інтерфейси. Тобто, основна концепція React.js – це багаторазові компоненти. Програміст пише невеликі частини коду, які потім можна використовувати як самостійні елементи інтерфейсу, або об'єднувати їх та формувати із них більші частини коду. Найголовніше в цій концепції те, що і великі, і маленькі частини коду (компоненти) можна використовувати повторно і в поточному, і в новому проекті. На рисунку 2.21 у загальних рисах зображено концепцію React-компонентів.

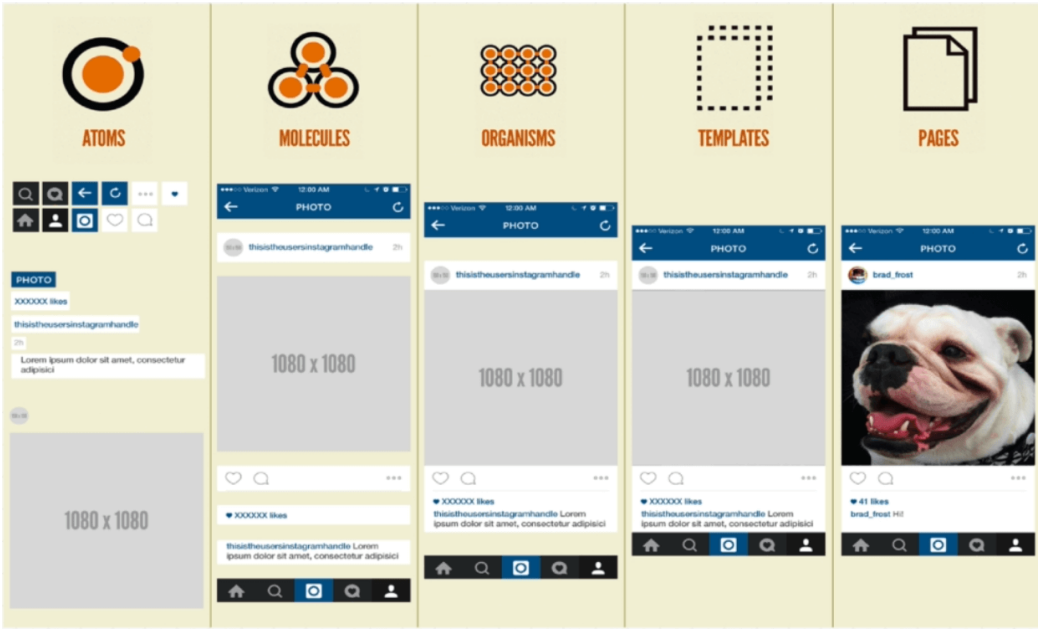


Рисунок 2.21 – Концепція React-компонентів у загальних рисах

React дозволяє визначати компоненти як класи чи функції. Тобто інсує Class-based та Function-based компоненти. Вони приймають довільні вхідні дані (так звані «пропси») і повертають React-елементи, які описують те, що повинно з’явитися на екрані при зміні стану компоненту. Щоб описати як повинен виглядати інтерфейс користувача, бібліотека використовує JSX. Це розширення синтаксису для JavaScript, яке дозволяє декларативно створювати компоненти користувальницького інтерфейсу. JSX розширює синтаксис JavaScript, щоб HTML-подібний код міг існувати разом з ним. Тобто завдяки JSX, код розмітки розташований там же, де і програмний код компонента.

На Рисунках 2.22 та 2.23 наведено приклади класового та функціонального компонентів, створенних з використанням JSX.

```
function Welcome(props) {
  return <h1>Привіт, {props.name}</h1>;
}
```

Рисунок 2.22 – Function-based компонент

```
class Welcome extends React.Component {
  render() {
    return <h1>Привіт, {this.props.name}</h1>;
  }
}
```

Рисунок 2.23 – Class-based компонент

2.1.6 Фреймворк Express.js

Express.js, або просто Express, — це популярний бекенд фреймворк для Node.js, розроблений для створення веб-додатків та RESTful API. Він випущений як безкоштовне програмне забезпечення з відкритим вихідним кодом за ліцензією MIT. Його де-факто називають стандартним серверним каркасом для Node.js. Адже він забезпечує мінімалістичний і гнучкий підхід до веб-розробки, дозволяючи розробникам швидко створювати надійні та масштабовані програми [13, 14].

Express.js був створений автором, відомим під ім'ям TJ Holowaychuk (Головайчук) і випущений у листопаді 2010 року. Його надихнули інші веб-фреймворки, такі як Sinatra (Ruby) і Connect (фреймворк проміжного програмного забезпечення для Node.js). Express.js мав на меті забезпечити простішу та гнучкішу альтернативу існуючим веб-фреймворкам для Node.js на той час.

До ключових переваг використання даного фреймворку відносяться:

Мінімалістичний і «невимушеність» (від англ. Unopinionated, синонім: open-minded): Express.js дотримується мінімалістичного і «невимушеного» (відкритого, вільного) підходу, забезпечуючи полегшену структуру, яка дозволяє розробникам вибирати інструменти та бібліотеки, яким вони віддають перевагу. Він надає набір основних функцій, залишаючи простір для налаштування та інтеграції додаткових компонентів.

Підтримка проміжного програмного забезпечення: Express.js створено на основі Connect, який є проміжним програмним забезпеченням для Node.js. Функції проміжного програмного забезпечення в Express.js обробляють

запити та відповіді, дозволяючи розробникам додавати функціональність у модульний спосіб. Express.js включає різноманітні вбудовані функції проміжного програмного забезпечення, а також дозволяє розробникам створювати власні.

Маршрутизація: Express.js забезпечує просту та виразну систему маршрутизації. Розробники можуть визначати маршрути для різних методів HTTP (GET, POST, PUT, DELETE тощо) і вказувати відповідні обробники для обробки запитів. Параметри маршруту, параметри запиту та символи підстановки можна використовувати для гнучкого зіставлення URL-адрес.

Інтеграція механізму шаблонів: Express.js дозволяє розробникам інтегрувати різні механізми шаблонів, такі як EJS, Pug (раніше Jade) і Handlebars, для відтворення динамічних шаблонів HTML. Це дозволяє створювати динамічний вміст, який надсилається клієнтам на основі даних із сервера.

Надійна обробка запитів і відповідей (Request and Response Handling): Express.js надає ряд методів і проміжного програмного забезпечення для ефективної обробки об’єктів запитів і відповідей. Це спрощує такі завдання, як розбір тіл запиту, обробка файлів cookie, налаштування заголовків і надсилання відповідей клієнтам.

Розширюваність та екосистема: Express.js має багату екосистему проміжного програмного забезпечення та розширень, наданих спільнотою. Ці розширення покращують функціональність Express.js, надаючи такі функції, як автентифікація, керування сеансами, перевірка тощо. Це дозволяє розробникам використовувати існуючі бібліотеки та компоненти для прискорення розробки.

Інтеграція з іншими інструментами: Express.js легко інтегрується з різними інструментами та бібліотеками в екосистемі Node.js. Він добре працює з такими базами даних, як MongoDB, MySQL і PostgreSQL, завдяки використанню бібліотек для баз даних або фреймворків об’єктно-реляційного відображення (ORM, Object-Relational Mapping). Він також інтегрується з

бібліотеками WebSocket, такими як Socket.IO, для спілкування в реальному часі.

З роками Express.js набув значної популярності та визнання в спільноті Node.js завдяки своїй простоті, гнучкості та надійності. Він продовжує розвиватися і залишається одним із найпоширеніших фреймворків для створення веб-додатків і API за допомогою Node.js.

### 2.1.7 Система управління базами даних MySQL

Розглянемо систему управління базами даних MySQL, мову структурованих запитів SQL та графічний інтерфейс адміністрування СУБД DBeaver.

MySQL – це вільна реляційна система керування реляційними базами даних. Вона була розроблена компанією «TcX», щоб підвищити швидкість обробки великих баз даних. У реляційній базі даних дані зберігаються в окремих таблицях, а не всі скопом. Завдяки цьому досягається виграш в швидкості і гнучкості. Таблиці зв'язуються між собою за допомогою відносин. Це забезпечує можливість об'єднувати дані з декількох таблиць при виконанні запиту [17].

Дана система управління базами даних (СУБД MySQL) була створена як альтернатива комерційним системам. Спочатку вона була дуже схожа на mSQL, але з часом вона розширювалася, і зараз MySQL – це одна із найпоширеніших систем керування базами даних. В першу чергу, її використовують для створення динамічних вебсторінок, тому що вона має чудову підтримку з боку різноманітних мов програмування.

MySQL вважається прийнятливим рішенням для малих і середніх застосунків. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

Сервер MySQL має наступні можливості:

- вона проста у встановленні та використанні;

- підтримує необмежену кількість користувачів, які одночасно працюють із БД;

- кількість рядків у таблицях може досягати 50 млн;

- має високу швидкість виконання команд;

- у наявності є проста та ефективна системи безпеки.

СУБД MySQL містить наступні технічні можливості:

MySQL являє собою систему клієнт-сервер, що містить багато-SQL-сервер, забезпечуючи підтримку різних обчислювальних машин баз даних. Також вона містить декілька різних клієнтських програм і бібліотек, засоби адміністрування і широкий спектр програмних інтерфейсів (API). Сервер MySQL також постачається у вигляді багато поточної бібліотеки, яку можна підключити до призначеного для користувача додатка і отримати компактний, більш швидкий і легкий в управлінні продукт.

Основним об'єктом для зберігання інформації в реляційній базі даних є таблиця, яка складається з рядків і стовпців, займає в базі даних фізичний простір і може бути постійною або тимчасовою. Кожна таблиця бази даних повинна містити хоча б один стовпець, за яким буде закріплений певний тип даних. Цю частину таблиці у реляційній базі даних ще називають «полем». Рядок даних – це запис в таблиці бази даних, він включає поля, що містять дані з одного запису таблиці.

Створювати базу даних можна використовуючи мову запитів SQL, або через контексте меню програми - графічний інтерфейс адміністрування систем управління базами даних, наприклад DBeaver.

SQL (англ. Structured Query Language, переклад – мова структурованих запитів) – це декларативна мова програмування, яку використовують щоб користувач міг взаємодіїяти з базами даних. Мову запитів застосовують для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою керування базами даних, ані окремим програмним продуктом. На відміну від дійсних мов програмування (C або

Pascal), SQL може формувати інтерактивні запити або, бувши вбудованою в прикладні програми, виступати як інструкції для керування даними. Окрім цього, стандарт SQL містить функції для визначення зміни, перевірки та захисту даних [16].

Простими словами, SQL як частина системи MySQL можна охарактеризувати як мову структурованих запитів. Під запитом мається на увазі звернення до бази даних, яке необхідне для управління інформацією в ній (видалення, додавання або зміна), здійснюване за допомогою системи управління базами даних (СКБД). Це найбільш поширена стандартна мова, яка використовується для доступу до баз даних.

DBeaver – це SQL клієнт та інструмент управління базами даних. Для реляційних баз даних DBeaver використовує програмний інтерфейс JDBC API, котрий взаємодіє з базами даних через драйвер JDBC. Для інших баз даних (не SQL) він використовує власні драйвери баз даних. DBeaver має функції завершення коду та підсвічування синтаксису, що забезпечує краще сприйняття. Він забезпечує архітектуру плагінів (засновану на архітектурі плагінів (платформі) Eclipse), яка дозволяє змінювати більшу частину роботи програми, для підтримки специфічних для баз даних функцій, незалежних від бази даних.

Іншими словами, DBeaver - це програмне забезпечення, яке діє як універсальний інструмент баз даних, призначений для розробників та адміністраторів баз даних. DBeaver має добре розроблений користувальницький інтерфейс, платформу, засновану на фреймворці з відкритим кодом, що дозволяє писати кілька розширень, а також бути сумісним з будь-якою базою даних.

DBeaver випускається в двох версіях:

1. DBeaver Comnunity Edition (скорочено DBeaver CE) – безкоштовний комп’ютерий додаток, написаний на Java. Дана версія має відкритий вихідний код, та поширюється під ліцензією Apache.

2. DBeaver Enterprise Edition (скорочено DBeaver EE) – платний додаток. Має розширені можливості роботи з noSQL базами даних (MongoDB і т.п.). Для розробки системи освітлення та створення додатку я використовую безкоштовну версію Comnunity Edition. На Рисунку 2.24 показано графічний інтерфейс програми DBeaver та створення таблиць через контекстне меню.

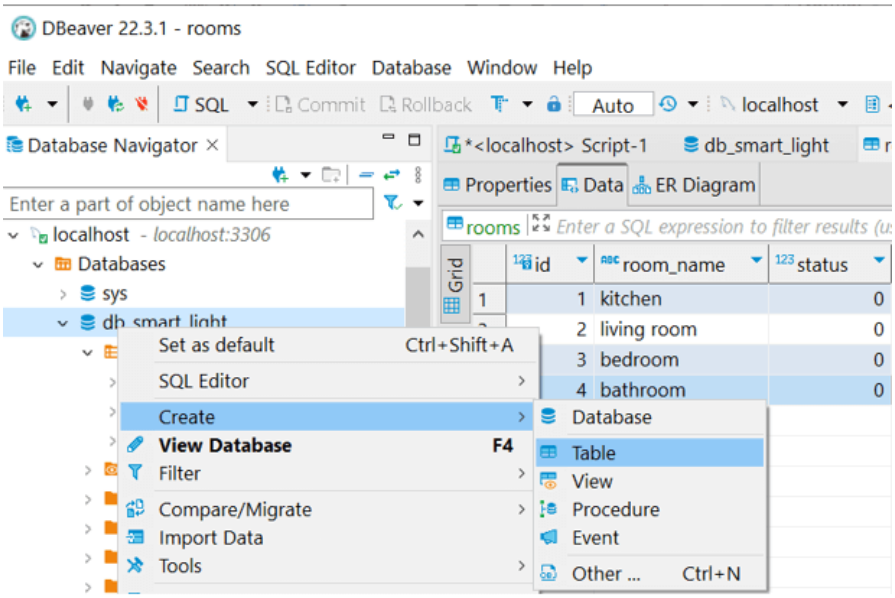


Рисунок 2.24 – Графічний інтерфейс програми DBeaver

### 2.1.8 Програмна платформа Docker

Docker — це платформа з відкритим кодом, яка дозволяє створювати, розгортати та керувати програмами за допомогою технології контейнеризації. Docker забезпечує легке та портативне рішення для упаковки програмного забезпечення та його залежностей у стандартизовані одиниці, які називаються контейнерами.

Контейнерізація — це метод віртуалізації, який дозволяє додаткам працювати в ізольованих середовищах, які називаються контейнерами, без необхідності використання окремої операційної системи для кожного контейнера. Контейнери забезпечують ізоляцію на рівні програми, забезпечуючи послідовну роботу програми та її залежностей у різних середовищах.

У контексті докеру, контейнер — це самодостатнє ізольоване середовище, яке включає код програми, середовище виконання, системні інструменти, бібліотеки та конфігурацію. Він інкапсулює всі залежності, необхідні для роботи програми, роблячи її незалежною від базової хост-системи. Контейнери Docker використовують ядро та ресурси головної операційної системи, забезпечуючи ізольоване середовище виконання. Кожен контейнер працює незалежно, з власною файловою системою, мережею та процесом. Контейнери легкі, швидко запускаються, їх можна легко тиражувати (replicate) та масштабувати (scale) за потреби.

Docker був розроблений однойменною компанією Docker Inc., та вперше був запущений у 2013 році. До основних його переваг відносяться:

Портативність: контейнери забезпечують узгоджене та портативне середовище, що дозволяє програмам безперебійно працювати в різних операційних системах, інфраструктурі та хмарних платформах. Контейнер інкапсулює програму та її залежності, усуваючи проблеми сумісності.

Ефективність: контейнери мають невелику вагу, оскільки вони спільно використовують ядро та ресурси хост-системи (головної операційної системи), що забезпечує швидший час запуску та ефективне використання ресурсів. Кілька контейнерів можуть працювати одночасно на одному хості без значних витрат на продуктивність.

Ізоляція: контейнери забезпечують ізоляцію на рівні процесу, запобігаючи взаємодії програм один з одним. Ця ізоляція підвищує безпеку та стабільність, гарантуючи, що зміни, зроблені в одному контейнері, не впливають на інші.

Керування залежностями: контейнери інкапсулюють усі залежності, необхідні для програми, включаючи бібліотеки, інструменти та середовища виконання. Це спрощує керування залежностями та дозволяє уникнути конфліктів між різними програмами або версіями однієї програми.

Масштабованість: контейнери дозволяють легко масштабувати програми. Розробники можуть створювати кілька екземплярів контейнерів для

обробки збільшених робочих навантажень або розподіляти програму між кількома хостами, забезпечуючи горизонтальну масштабованість.

Нижче на рисунку 2.25 наведена схема архітектури Docker та описано її ключові компоненти.

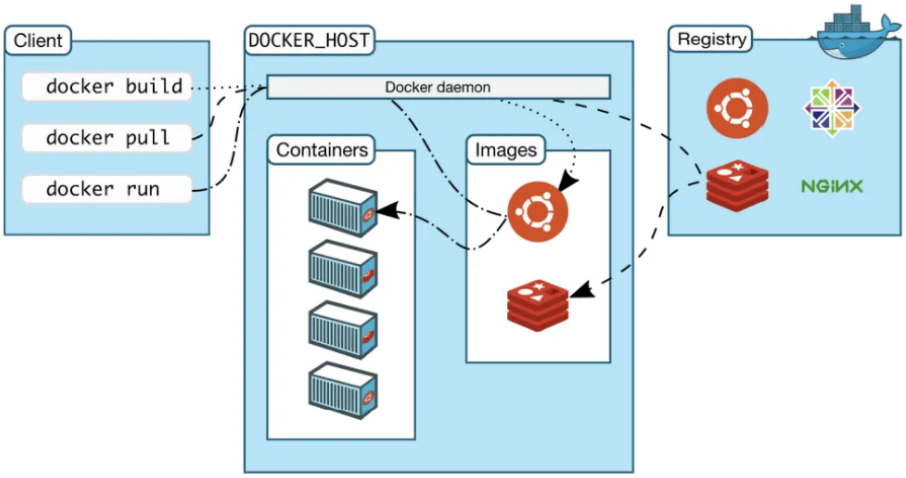


Рисунок 2.25 – Архітектура Docker

Основна частина усієї системи – це Docker Engine. Docker Engine – це додаток (програма), яка слідує клієнт-серверній архітектурі. Він встановлений на хост-машині. У Docker Engine є три компоненти:

Сервер (Server): це демон docker, так званий dockerd. Він може створювати та керувати образами, контейнерами, мережею тощо. Тобто він виконує важку роботу зі створення, запуску та розповсюдження контейнерів Docker.

Rest API: використовується для вказівки демону docker, тобто вказує docker демону, що робити, які інструкції виконувати.

Інтерфейс командного рядка (CLI, Command Line Interface): це клієнт, який використовується для введення команд Docker.

Docker клієнт: Клієнт Docker є ключовим компонентом системи Docker, який використовується користувачами для взаємодії з Docker-ом, він забезпечує інтерфейс командного рядка (CLI). Коли ми запускаємо команди докеру, клієнт надсилає ці команди демону «dockerd», щоб створити, запустити та зупинити програму.

Docker Host: Докер-хост — це тип машини, яка відповідає за роботу кількох контейнерів. Він складається з демона Docker, зображень, контейнерів, мереж і сховища.

Docker Registry: Docker реєстр – це місце, де зберігаються образи (images) Docker. Docker Hub і Docker Cloud є загальнодоступними піблічними реєстрами, яким може користуватися кожен.

Як бачимо, Docker і контейнеризація забезпечують гнучкий і ефективний підхід до розгортання додатків і керування ними. Вони сприяють послідовності, відтворюваності та переносимості, полегшуючи розробку, тестування та розгортання програмного забезпечення в різних середовищах.

### 2.1.9 Інтегроване середовище розробки Visual Studio Code

Visual Studio Code (скорочено VS Code) — це безкоштовний редактор вихідного коду з відкритим кодом, розроблений компанією Microsoft. Він був вперше випущений у квітні 2015 року та здобув значну популярність серед розробників завдяки своїй легкості, широким можливостям налаштування та широкому спектру підтримуваних мов програмування та розширень. Це середовище розробки стало першим крос-платформовим продуктом у лінійці Visual Studio. Він поширюється в безкоштовному доступі і підтримується всіма актуальними операційними системами: Windows, Linux і macOS.

Ключові особливості середовища розробки VS Code:

Кросплатформна підтримка: VS Code доступний для Windows, macOS і Linux, дозволяючи розробникам працювати з улюбленою операційною системою.

Інтуїтивно зрозумілий інтерфейс користувача: він має чистий і простий у використанні інтерфейс із бічною панеллю для навігації файлами, вбудованим терміналом і потужним редактором із підтримкою таких функцій, як підсвічування синтаксису, фрагменти коду та інтелектуальне завершення коду.

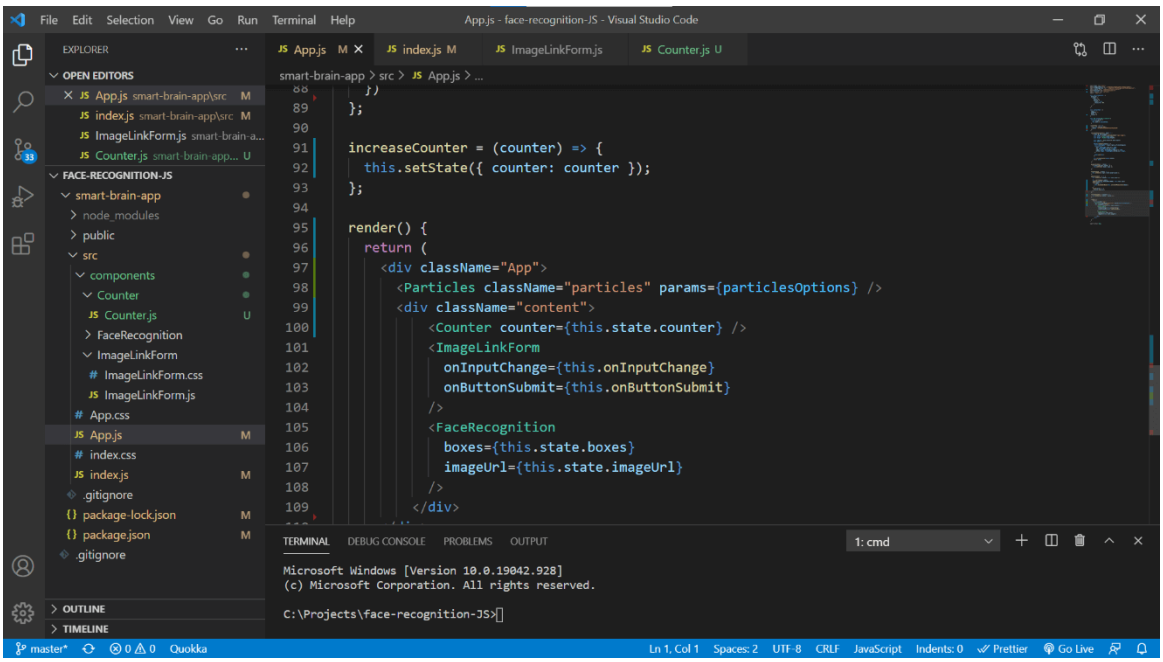


Рисунок 2.26 – Інтерфейс програми VS Code

Екосистема розширень: VS Code надає багату екосистему розширень, які покращують її функціональність. Ці розширення надані спільнотою та охоплюють широкий спектр випадків використання, включаючи підтримку мови, налагодження, контроль версій тощо.

Інтегрований термінал: містить інтегрований термінал у редакторі, що дозволяє розробникам запускати інструменти командного рядка та сценарії без переходу до окремого вікна терміналу.

Інтеграція контролю версій: VS Code інтегрується з такими популярними системами контролю версій, як Git, забезпечуючи такі функції, як візуальні відмінності, керування розгалуженнями та вбудовані анотації коду.

Підтримка налагодження: пропонує вбудовану підтримку для налагодження різних мов програмування, дозволяючи розробникам встановлювати контрольні точки, перевіряти змінні та покроково виконувати код.

IntelliSense: редактор забезпечує інтелектуальне завершення коду, навігацію по коду та підказки щодо параметрів на основі мови, що використовується, покращуючи продуктивність і зменшуючи помилки.

VS Code, інтерфейс якого показано на рисунку 2.26, підтримує широкий спектр мов програмування, включаючи JavaScript, TypeScript, Python, C#, Java, Go та багато інших. Щоб покращити функціональність середовища, Microsoft регулярно випускає оновлення з новими функціями, покращеннями та виправленнями помилок. Редактор можна розширити, встановивши розширення з Visual Studio Code Marketplace. VS Code також пропонує широкі можливості налаштування, що робить його кращим вибором для редагування та розробки коду на різних платформах.

## 2.2 Вибір апаратного обладнання

Для реалізації апаратної частини системи розумного оствітлення було вирішено використовувати мікроконтролер ESP32 та середовище розробки Arduino IDE. Також нам знадобиться 4х канальний реле модуль, який буде служити інтерфейсом (посередником) між мікроконтролером і LED лампами.

У даному підрозділі проведено детальний огляд популярних мікроконтролерів і середовищ розробки, вказано їхні переваги та недоліки, та обгрунтовано вибір. А також описано загальні відомості про реле модуль, його будову та основні функції які він виконує.

### 2.2.1 Вибір мікроконтролеру

На ринку доступно багато популярних мікроконтролерів і налагоджувальних плат (плати розробки), які є ідеальним рішенням залежно від конкретного проекту і потреб системи, яку проектують. До найбільш широко використовуваних відносяться Raspberry Pi, Arduino, ESP32.

#### 2.2.1.1 Мікроконтролер Raspberry Pi

Raspberry Pi — це популярний одноплатний комп’ютер (SBC, single-board computer), який використовується для різноманітних проектів, починаючи від домашньої автоматизації, медіацентрів і робототехніки. Він

був розроблений Raspberry Pi Foundation у Великій Британії в 2012 році та широко доступний у всьому світі. Raspberry Pi популярні серед виробників, педагогів і любителів. Вони невеликі, розміром з кредитну картку, та можуть працювати з різними операційними системами, включаючи Linux. Raspberry Pi має процесор Broadcom ARM із тактовою частотою від 700 МГц до 1,4 ГГц і поставляється з різним обсягом оперативної пам'яті залежно від моделі.

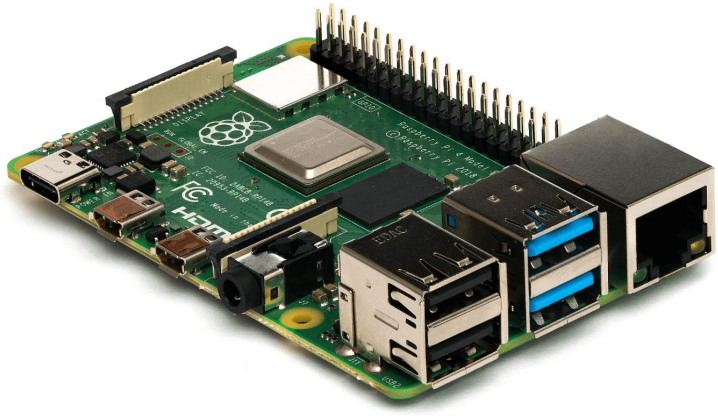

Рисунок 2.27 – Плата Raspberry Pi 4 Model B вид збоку


Рисунок 2.28 – Плата Raspberry Pi 4 Model B вид зверху

Raspberry Pi 4 Model B – це остання версія Raspberry Pi, випущена в 2019 році. Ось деякі з ключових особливостей Raspberry Pi 4 Model B:

Процесор Broadcom BCM2711: Raspberry Pi 4 містить чотирьохядерний процесор ARM Cortex-A72 з тактовою частотою 1,5 ГГц, який значно швидший за процесор попередніх моделей Raspberry Pi.

RAM: Raspberry Pi 4 доступний з 1 ГБ, 2 ГБ або 4 ГБ оперативної пам'яті

(RAM), що значно більше, ніж у попередніх моделей Raspberry Pi.

Вбудований бездротовий зв’язок: Raspberry Pi 4 включає вбудований Wi-Fi і Bluetooth 5.0, що полегшує підключення до бездротових мереж і периферійних пристроїв.

Порт Gigabit Ethernet: Raspberry Pi 4 містить порт Gigabit Ethernet, який можна використовувати для підключення до дротової мережі на вищій швидкості, ніж у попередніх моделях Raspberry Pi.

Подвійні порти micro-HDMI (Dual micro-HDMI ports): Raspberry Pi 4 містить два порти micro-HDMI, які можна використовувати для підключення до двох моніторів або телевізорів одночасно.

Порти USB: Raspberry Pi 4 містить два порти USB 2.0 і два порти USB 3.0, які можна використовувати для підключення різноманітних периферійних пристроїв, включаючи клавіатури, миші та зовнішні жорсткі диски.

Слот для картки microSD: Raspberry Pi 4 містить слот для картки microSD, який використовується для зберігання операційної системи та даних користувача.

GPIO pins: Raspberry Pi 4 містить набір із 40 контактів GPIO, які можна використовувати для підключення до різноманітних датчиків, приводів та інших пристроїв.

Raspberry Pi 4 Model B може працювати зі спеціалізованими операційними системами, такими як RetroPie (дистрибутив, орієнтований на ігри) або Pi-hole (мережевий блокувальник реклами).

Переваги мікроконтролера Raspberry Pi:

Доступність: Raspberry Pi є відносно недорогим мікроконтролером (вартість на 2023 рік становить від 25$), що робить його доступним для широкого кола користувачів, включаючи любителів, студентів і викладачів.

Універсальність: Raspberry Pi можна використовувати для широкого спектру проектів, від простих експериментів з електронікою до складних робототехнічних проектів.

Велика спільнота: Raspberry Pi має велику спільноту користувачів, а це означає, що в Інтернеті доступно багато ресурсів, зокрема навчальні посібники, форуми та проекти з відкритим кодом.

Настроюваність: Raspberry Pi дуже легко налаштовується, з різними операційними системами та доступним програмним забезпеченням, що робить його придатним для широкого спектру програм.

Висока продуктивність: Raspberry Pi має потужний процесор і пристойний обсяг оперативної пам’яті, що дозволяє йому працювати зі складнішими програмами, ніж інші мікроконтролери.

Недоліки мікроконтролера Raspberry Pi:

Енергоспоживання: Raspberry Pi може споживати багато енергії, що може викликати занепокоєння для проектів, що живляться від батареї.

Складність: Raspberry Pi складніший за деякі інші мікроконтролери, що може стати проблемою для новачків, які тільки починають знайомство з електронікою.

Обмежені контакти вводу/виводу: хоча Raspberry Pi має пристойну кількість контактів GPIO, цього може бути недостатньо для великих проектів, які вимагають багато датчиків або приводів.

Обмежені можливості реального часу: Raspberry Pi не є мікроконтролером реального часу, що означає, що він може не підходити для деяких програм, які вимагають точного часу або низької затримки.

Питання безпеки: Оскільки Raspberry Pi є комп’ютером загального призначення, він може бути більш вразливим до загроз безпеці, ніж деякі інші мікроконтролери, розроблені для спеціальних програм.

Можливі проблеми сумісності програмного забезпечення: Хоча Raspberry Pi може працювати з різними операційними системами та програмним забезпеченням, можуть виникати проблеми сумісності з деяким програмним або апаратним забезпеченням.

2.2.1.2 Мікроконтролер Arduino

Arduino — це платформа з відкритим кодом, яка використовується для створення електронних проектів, що складається як з апаратних, так і з програмних компонентів. За своєю суттю плата Arduino — це мікроконтролер, який можна запрограмувати на взаємодію з різними компонентами та датчиками для виконання певних завдань. Апаратне забезпечення складається з простої друкованої плати з входами та виходами для підключення інших електронних компонентів [11].

Плати Arduino розроблені таким чином, щоб бути зручними для користувача, що дозволяє людям, які не мають досвіду в електроніці, легко створювати власні електронні проекти. Мова програмування Arduino заснована на C++, і є багато доступних бібліотек, які спрощують процес програмування плати. Arduino IDE (інтегроване середовище розробки) — це програмний додаток, який можна використовувати для написання, компілювання та завантаження коду на плату.

Доступно багато різних плат Arduino, кожна з яких має свої функції, різні можливості та особливості. Деякі призначені для конкретних завдань, таких як керування двигунами або зв’язок з іншими пристроями, тоді як інші мають більш загальне призначення. Найпопулярнішою та найбільш широко використовуваною вважається плата Arduino Uno.

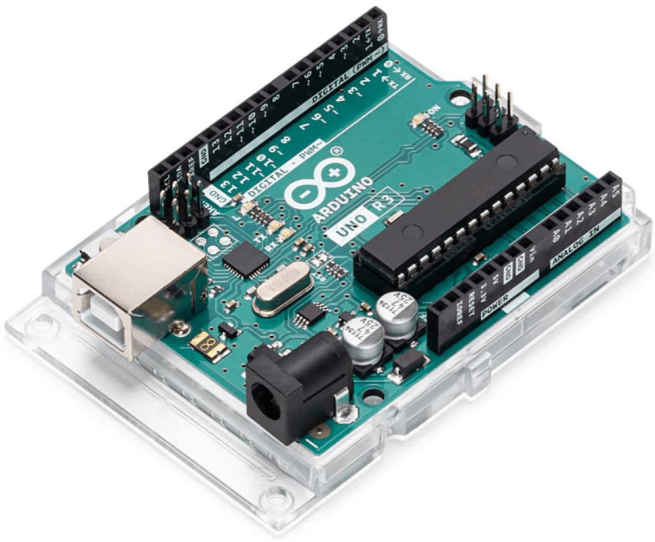


Рисунок 2.29 – Плата Arduino Uno R3 вид збоку

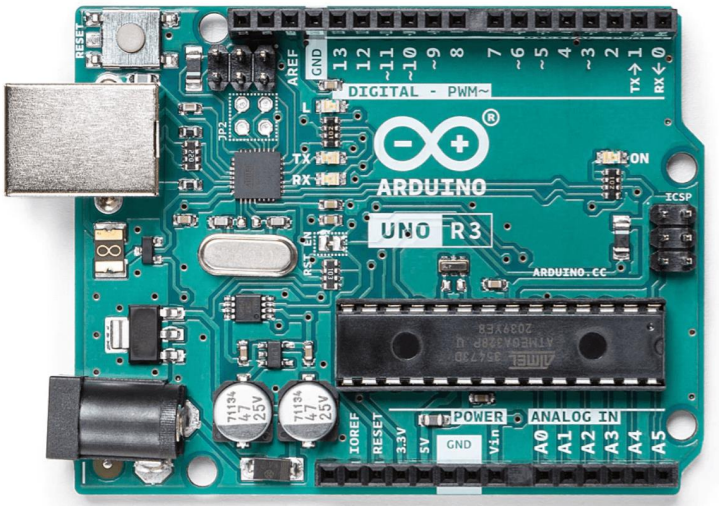


Рисунок 2.30 – Плата Arduino Uno R3 вид зверху

Arduino Uno заснована на мікроконтролері ATmega328. Вона має 14 контактів цифрових входів/виходів, 6 із яких можна використовувати як PWM виходи (від англ. Pulse Width Modulation, переклад – Широтно-імпульсна модуляція), 6 аналогових входів, керамічний резонатор 6 МГц, кварцовий кристал 16МГц, робочу напругу 5В, 32 КБ флеш-пам'яті, 2KB SRAM, 1KB EEPROM, USB-з’єднання, роз’єм живлення (a power jack), та кнопку скидання.

Популярність плати Arduino Uno пояснюється її низькою вартістю (ціна на момент написання дипломної роботи становить 25$). В інтернеті для плати Arduino Uno доступно багато навчальних посібників, прикладів і бібліотек коду. Саме наявність обширної документації робить плату Arduino Uno хорошим і привабливим вибором для новачків і початківців, які хочуть вивчати електроніку та програмування, оскільки він простий у використанні та має велику спільноту користувачів.

Arduino Uno R3 — остання версія плати Arduino Uno. Вона схожа на свого попередника з точки зору апаратних характеристик. Arduino Uno R3 заснована та базується на тому ж мікроконтролері ATmega328P і має 14 цифрових входів/виходів, шість аналогових входів та інші функції, такі як порт USB, роз’єм живлення та кнопку скидання, тощо.

Одна з головних змін в Arduino Uno R3 полягає в тому, що вона використовує нову мікросхему USB-інтерфейсу Atmega16U2, яка забезпечує

швидший зв’язок з комп’ютером і робить плату сумісною з більшою кількістю операційних систем. Arduino Uno R3 також має нову схему скидання, що робить плату більш надійною. Вона являється зворотно сумісною з оригінальною платою Arduino Uno, і це означає, що користувачі можуть без проблем оновити плату (замінити стару) на нову, використовуючи той самий код і екрани, що і на старій платі попередньої версії.

Переваги мікроконтролера Arduino Uno:

Простий у використанні: Arduino Uno розроблено таким чином, щоб його було легко використовувати навіть новачкам, для тих, хто має обмежений досвід програмування. Він поставляється з простою мовою програмування , та зі зручним інтегрованим середовищем розробки (IDE), яке спрощує процес програмування.

Доступний: Arduino Uno має відносно низьку вартість. Він недорогий порівняно з іншими мікроконтролерами з подібними можливостями. Це робить його доступним та ідеальним для ширшої аудиторії.

Велика спільнота: Arduino Uno має велике та активне співтовариство користувачів і розробників. Це означає, що в мережі Інтернет доступно багато ресурсів, зокрема навчальні посібники, приклади коду та форуми, які допоможуть користувачам усунути проблеми, що виникають під час програмування мікроконтролеру, та знайти натхнення для нових проектів.

Відкритий вихідний код (Open Source): Arduino Uno має відкритий вихідний код, що означає, що користувачі мають доступ до схем, мікропрограми та програмного забезпечення, що використовується в платі. Це дозволяє користувачам легко модифікувати плату відповідно до своїх конкретних потреб.

Розширюваність: Arduino Uno можна розширити за допомогою ряду екранів і модулів. Це дозволить користувачу (розробнику) додати до проекту додаткову функціональність без необхідності розробляти власну схему.

Універсальність: Arduino Uno можна використовувати для широкого

спектру застосувань та проектів, від простих світлодіодних дисплеїв, включаючи домашню автоматизацію та пристрої IoT, до складної робототехніки.

Недоліки мікроконтролера Arduino Uno:

Обмежена обчислювальна потужність: Arduino Uno має мікроконтролер ATmega328P, який використовується в Arduino Uno, має тактову частоту 16 МГц і обмежену обчислювальну потужність. Це означає, що він може бути обмеженням для більш складних проектів, може не підійти для програм, які вимагають великої обчислювальної потужності.

Обмежена пам’ять: Arduino Uno має обмежену пам’ять (32 КБ Flash і 2 КБ SRAM), що може обмежити кількість складних програм, які можна запускати на ньому. Це може бути обмеженням для великих проектів.

Обмежена кількість вхідних/вихідних контактів: Arduino Uno має обмежену кількість контактів входу/виводу (I/O, 14 цифрових контактів і 6 аналогових контактів), що може обмежити його використання для складних програм, для більш складних проектів.

Обмежені можливості підключення: Arduino Uno має обмежені можливості підключення, що означає, що він може не підходити для програм, які потребують високошвидкісного зв’язку або розширених мережевих можливостей.

Не підходить для великих проектів: Arduino Uno розроблений для малих і середніх проектів і може не підійти для великомасштабних проектів, які потребують більшої обчислювальної потужності та пам’яті, оскільки йому не вистачає деяких функцій, необхідних для надійної та надійної роботи в суворих умовах.

Підсумовуючи, Arduino Uno — це популярна й універсальна плата мікроконтролера, проста у використанні та доступна за ціною, що робить її чудовим вибором для любителів і студентів. Однак його обмежена

обчислювальна потужність, пам’ять і можливості підключення можуть зробити його менш придатним для більш складних програм і великих проектів.

### 2.2.1.3 Мікроконтролер ESP32

ESP32 – це потужний і універсальний мікроконтролер, розроблений у 2015 році компанією Espressif Systems, що розташована в Шанхаї, Китай. Він заснований на двоядерному процесорі Xtensa LX6 з тактовою частотою 240МГц і включає велику кількість інтегрованих периферійних пристроїв, таких як Wi-Fi і Bluetooth, аналого-цифрові перетворювачі, цифро-аналогові перетворювачі, ємнісні сенсорні датчики та багато іншого. ESP32 є наступником ESP8266 і широко використовується в різноманітних додатках і проєктах, таких як пристрої Інтернету речей (IoT), домашня автоматизація, робототехніка, промислова автоматизація, тощо, завдяки низькому енергоспоживанню, малому форм-фактору, економічній ефективності, та значній потужності обробки.

Мікроконтролер ESP32 доступний у різноманітних варіантах та платах розробки, кожна з яких має власний набір функцій і можливостей. Найбільш широко використовуваним вважається ESP-WROOM-32 (Рисунок 2.31).

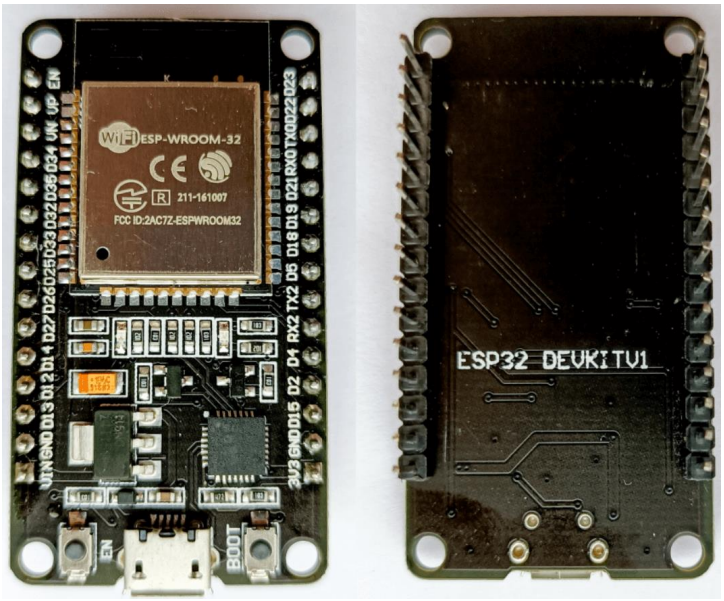


Рисунок 2.31 – Мікроконтролер ESP-WROOM-32,
плата ESP32 DEVKIT V1 вид зверху та знизу

ESP-WROOM-32 – це популярний варіант мікроконтролера ESP32, розроблений спеціально для використання в програмах, що потребують використання Wi-Fi і Bluetooth мережі. Даний мікроконтролер включає вбудовану антену та підсилювач потужності, що дозволяє легко створювати бездротові пристрої без необхідності використання зовнішніх компонентів. Серед популярних розробних плат ESP-WROOM-32 виділяють DEVKIT V1.

ESP-WROOM-32 — це модуль мікроконтролера з підтримкою Wi-Fi і Bluetooth, який базується на чіпі ESP32 від Espressif Systems. ESP32 DEVKIT V1 – це плата розробки на основі модуля ESP-WROOM-32. Він забезпечує зручний спосіб прототипування та розробки додатків, які використовують чіп ESP32. Нижче на Рисунку 2.32 зображено архітектуру даної плати розробки.

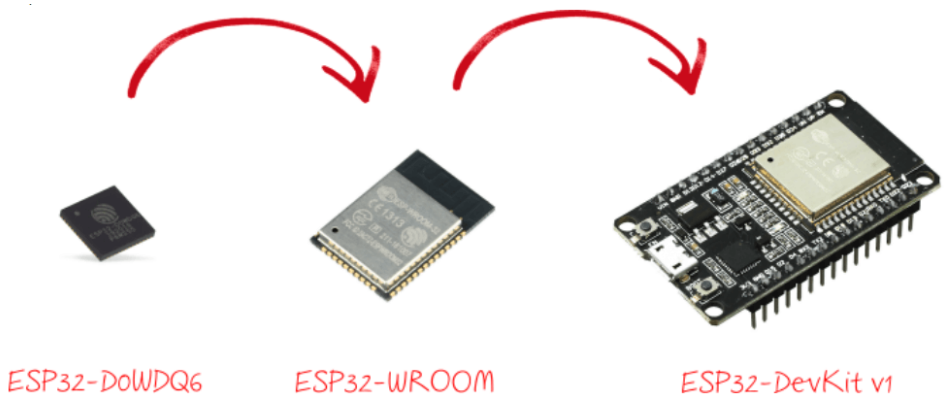


Рисунок 2.32 – Архітектура плати ESP32 DEVKIT V1

Ось деякі ключові особливості плати ESP32 DEVKIT V1:

Мікроконтролер: чіп ESP32 виконаний за технологією SoC (від англ. System-on-a-Chip, переклад – система на кристалі) – це двоядерний мікроконтролер із тактовою частотою до 240 МГц. В нього входить 32-бітний процесор Tensilica Xtensa LX6 з блоками пам'яті ROM на 448 КБ та SRAM на 520 КБ. Це дозволяє ESP32 виконувати більш складні завдання, ніж ESP8266. У кристалі розташовані бездротові модулі Wi-Fi/Bluetooth, датчик Холла та сенсор температури.

Wi-Fi: чіп ESP32 підтримує мережі Wi-Fi на частотах 2,4 ГГц і 5 ГГц і може працювати як точка доступу або станція.

Bluetooth: чіп ESP32 також підтримує Bluetooth 4.2 і Bluetooth Low Energy (BLE).

Аналогові входи: Мікросхема ESP32 має 12-розрядні АЦП, які можна використовувати для читання аналогових сигналів із датчиків.

Живлення: Плату розробки можна живити за допомогою кабелю USB, акумулятора або зовнішнього джерела живлення.

USB-UART перетворювач: Перетворювач USB-UART на мікросхемі забезпечує зв'язок модуля ESP32-WROOM із USB-портом комп'ютера. При підключенні до ПК платформа ESP32 DevKit визначається як віртуальний COM-порт.

Налагодження (Debugging): плата розробки має перетворювач USB-UART, який можна використовувати для налагодження та програмування.

Програмування: плату розробки можна програмувати за допомогою Arduino IDE, ESP-IDF (Espressif IoT Development Framework) та інших середовищ програмування.

GPIO pins (англ. General Purpose Input/Output pins – Вхідні/вихідні контакти загального призначення): плата розробки має 30 контактів GPIO, які можна використовувати для взаємодії з датчиками, приводами та іншими пристроями. На Рисунку 2.33 зображено розпінування даної плати розробки.

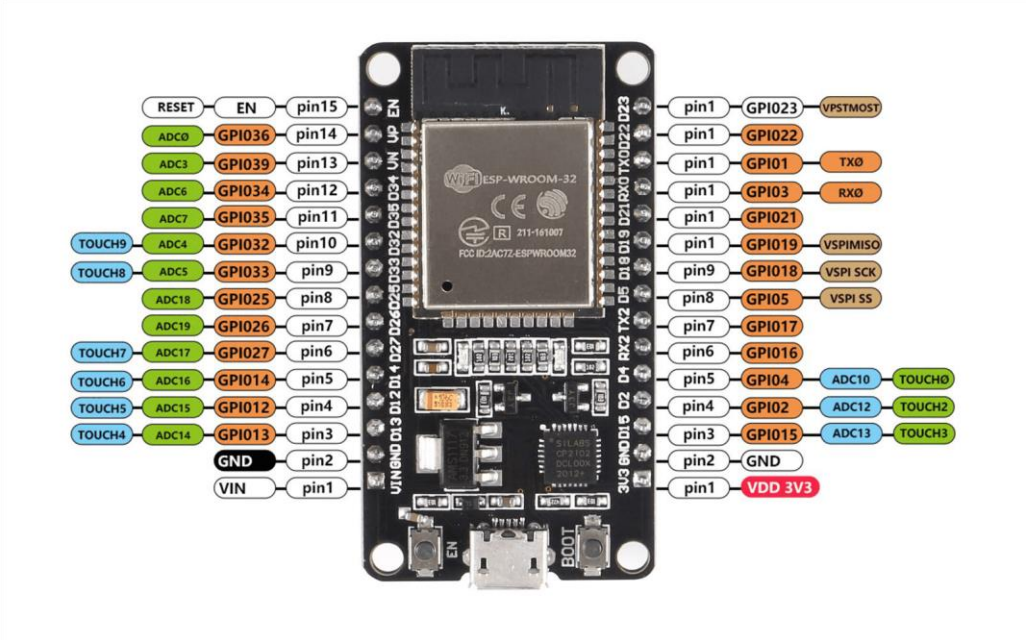


Рисунок 2.33 – Розпінування плати ESP32 DEVKIT V1

Переваги плати ESP32 DEVKIT V1 (мікроконтролера ESP32):

Висока потужність обробки: SoC ESP32 забезпечує потужні можливості обробки, включаючи двоядерні процесори із тактовою частотою до 240 МГц.

Багатий набір периферійних пристроїв: ESP32 містить широкий набір периферійних пристроїв, включаючи АЦП, ЦАП, I2C, SPI, UART тощо, що робить його універсальним і придатним для широкого спектру застосувань.

Доступність: плата ESP32 DEVKIT V1 має низьку вартість. Він найдешевший порівняно з іншими мікроконтролерами. Ціна на 2023 рік становить приблизно 5$. Це робить його дуже доступним та ідеальним для малобюджетних проектів.

Легка розробка: плата ESP32 DEVKIT V1 містить вбудовані компоненти, які спрощують розробку та тестування, такі як регулятори напруги, USB-порти та вбудований налагоджувач.

Низьке енергоспоживання: ESP32 розроблено для дуже низького споживання енергії, що робить його ідеальним для пристроїв, що живляться від акумулятора чи батареї, яким необхідно працювати протягом тривалого часу. До того ж він має кілька режимів енергозбереження, включаючи режим глибокого сну, який може знизити споживання електроенергії до 10 мкА.

Breadboard-friendly (Зручний для макетної плати): DEVKIT V1 розроблено для встановлення на макетну плату, що полегшує створення прототипу та експериментування.

Більше контактів вводу/виводу: плата розробки мікроконтролеру ESP32 має більше контактів вводу/виводу, порівняно з іншими мікроконтролерами.

Параметри підключення: ESP32 пропонує кілька варіантів підключення, включаючи Wi-Fi, Bluetooth і Ethernet, що полегшує підключення до Інтернету та інших пристроїв.

Недоліки плати ESP32 DEVKIT V1 (мікроконтролера ESP32):

Обмежений обсяг пам’яті: як і будь-які інші мікроконтролери, ESP32 має обмежений обсяг флеш-пам’яті та RAM (оперативної пам’яті). Це може

стати обмеженням для проектів, які потребують великого обсягу зберігання даних, та обробки великих обсягів даних або багатозадачності.

Підсумовуючи, ESP32 — це потужний і універсальний мікроконтролер, який має вбудований модуль Wi-Fi і Bluetooth, що дозволяє спілкуватися з іншими пристроями через бездротову мережу. Він підтримує всі поширені протоколи Wi-Fi, такі як WPA, WPA2 і WEP, та добре підходить для широкого спектру додатків, особливо тих, які потребують низького енергоспоживання та бездротового підключення. Саме тому для реалізації проекту було обрано мікроконтролер ESP32 та плату розробки ESP32 DEVKIT V1.

### 2.2.2 Вибір мови програмування та середовища розробки для мікроконтролеру

Мікроконтролер ESP32 можна програмувати за допомогою різних мов програмування, включаючи C++, MicroPython і Arduino. Він також підтримується рядом інструментів розробки, таких як Arduino IDE, Espressif IDF і MicroPython IDE. Розглянемо кожну більш детально.

#### 2.2.2.1 Espressif IDF

Espressif IDF (IoT Development Framework) — це платформа розробки програмного забезпечення з відкритим кодом, розроблена китайською компанією з виробництва електроніки Espressif Systems. IDF призначений для розробки додатків Інтернету речей (IoT) на лінійці бездротових мікроконтролерів (MCU) Espressif, включаючи ESP32 і ESP8266.

IDF містить повний набір бібліотек, інструментів і документації для розробки додатків IoT, таких як мережеві протоколи Wi-Fi, Bluetooth і TCP/IP, а також драйвери для звичайних периферійних пристроїв, таких як датчики та дисплеї. IDF також включає систему збірки та інтерфейс командного рядка (CLI) для компіляції та розгортання додатків на MCU Espressif.

IDF має широкі можливості налаштування та підтримує низку середовищ розробки, включаючи Arduino IDE, Eclipse та Visual Studio Code.

Він також містить симулятор для тестування програм без потреби у фізичному обладнанні.

Espressif IDF випущено за дозволеною ліцензією Apache 2.0, яка дозволяє розробникам вільно змінювати та поширювати структуру за потреби. IDF має активне співтовариство розробників, які роблять внесок у розвиток фреймворку та надають підтримку через форуми, групи чатів та інші канали.

Espressif IDF використовує мову програмування C для програмування мікроконтролера ESP32. Хоча IDF також підтримує C++, більшість кодової бази IDF і документації написані на C.

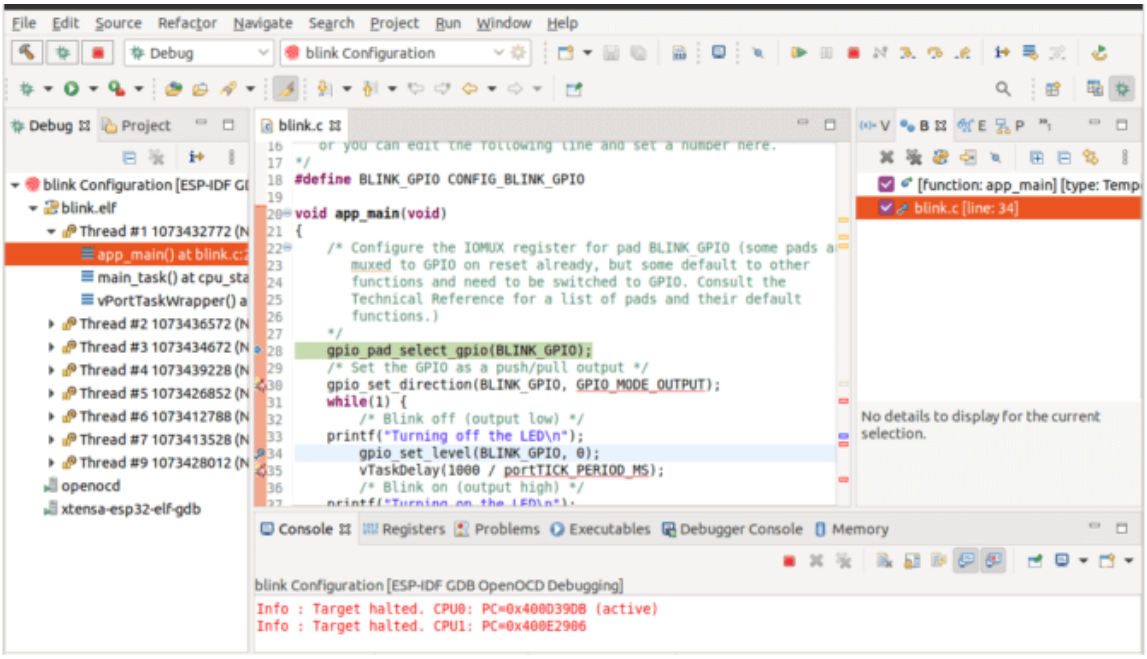


Рисунок 2.34 – Зображення робочого поля програмування у програмі «Espressif IDF» (IoT Development Framework)

Переваги Espressif IDF:

Open-source: Espressif IDF є фреймворком з відкритим кодом, що означає, що він вільний для використання та може бути змінений відповідно до ваших потреб.

Комплексні бібліотеки: IDF містить широкий спектр бібліотек для Wi-Fi, Bluetooth і мережевих протоколів, а також драйвери для звичайних периферійних пристроїв, таких як датчики та дисплеї.

Симулятор: IDF містить симулятор для тестування програм без потреби

у фізичному обладнанні.

Активна спільнота: IDF має велике та активне співтовариство розробників, які роблять внесок у розвиток фреймворку та надають підтримку через форуми, групи чату та інші канали.

Недоліки Espressif IDF:

Steep learning curve (Крута крива навчання): для початківців може бути важко освоїти IDF, оскільки він вимагає хорошого розуміння програмування та вбудованих систем.

Обмежена апаратна підтримка: IDF в основному розроблено для використання з лінійкою бездротових мікроконтролерів Espressif, таких як ESP32 і ESP8266.

Обмежена документація: хоча IDF містить документацію, деякі користувачі можуть вважати її недостатньою в деталях і ясності.

Обмежена підтримка сторонніх розробників: оскільки IDF є відносно новою структурою, вона може не мати такого рівня підтримки сторонніми розробниками, як більш усталені IDE, такі як Arduino.

Ресурсомісткий: IDF може бути ресурсомістким, для ефективної роботи може знадобитися більш потужний комп’ютер.

2.2.2.2 MicroPython IDE

MicroPython — популярна реалізація мови програмування Python, яка оптимізована для мікроконтролерів та інших вбудованих систем. Існує кілька IDE (інтегрованих середовищ розробки), доступних для мови MicroPython, які можна використовувати для програмування мікроконтролерів. Ось деякі з популярних IDE MicroPython:

Thonny: популярна IDE MicroPython, проста у використанні та має інтуїтивно зрозумілий інтерфейс. Він надає такі функції, як підсвічування синтаксису, доповнення коду, налагодження та вбудований провідник файлів.

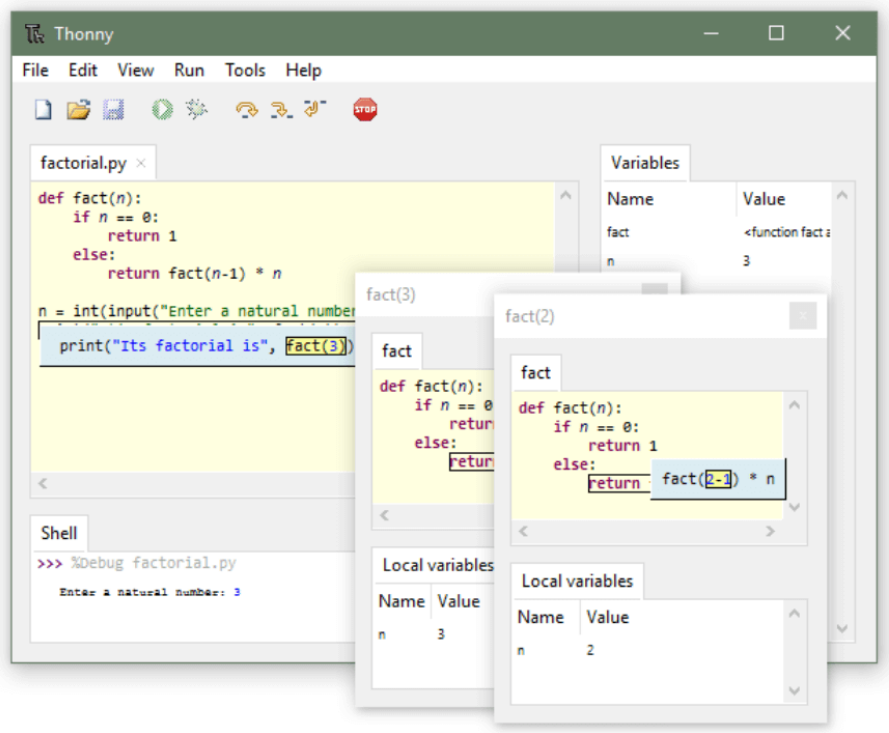


Рисунок 2.35 – Зображення робочого поля програмування у програмі «Thonny»

Mu: проста та легка IDE, розроблена для початківців. Він надає такі функції, як підсвічування синтаксису, автоматичний відступ і консоль REPL (Read-Eval-Print Loop).

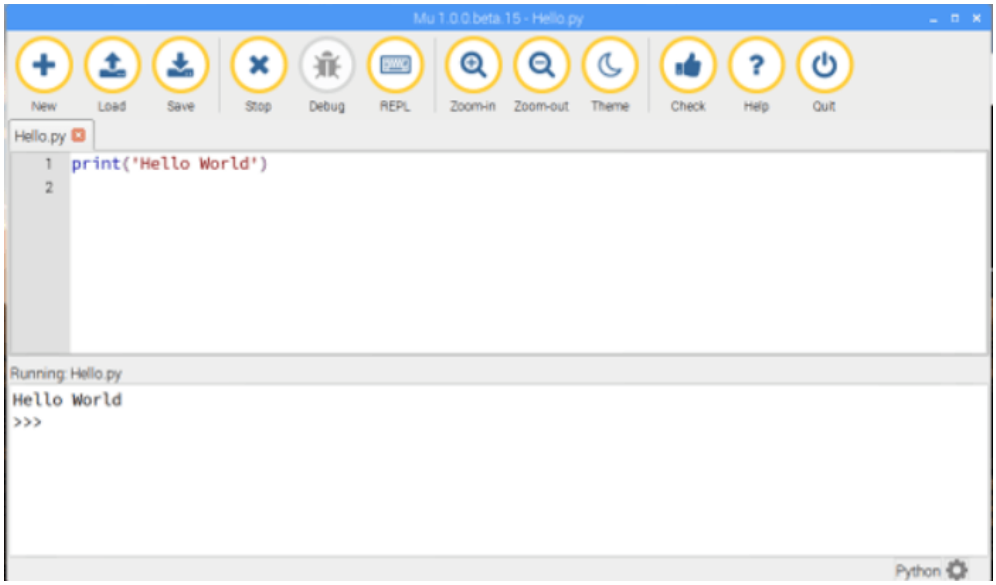


Рисунок 2.36 – Зображення робочого поля програмування у програмі «Mu»

PyCharm: це повнофункціональна IDE, яка підтримує розробку на Python

і MicroPython. Він надає такі функції, як налагодження, аналіз коду та інтеграція контролю версій. Він також має видання для спільноти, яким можна користуватися безкоштовно.

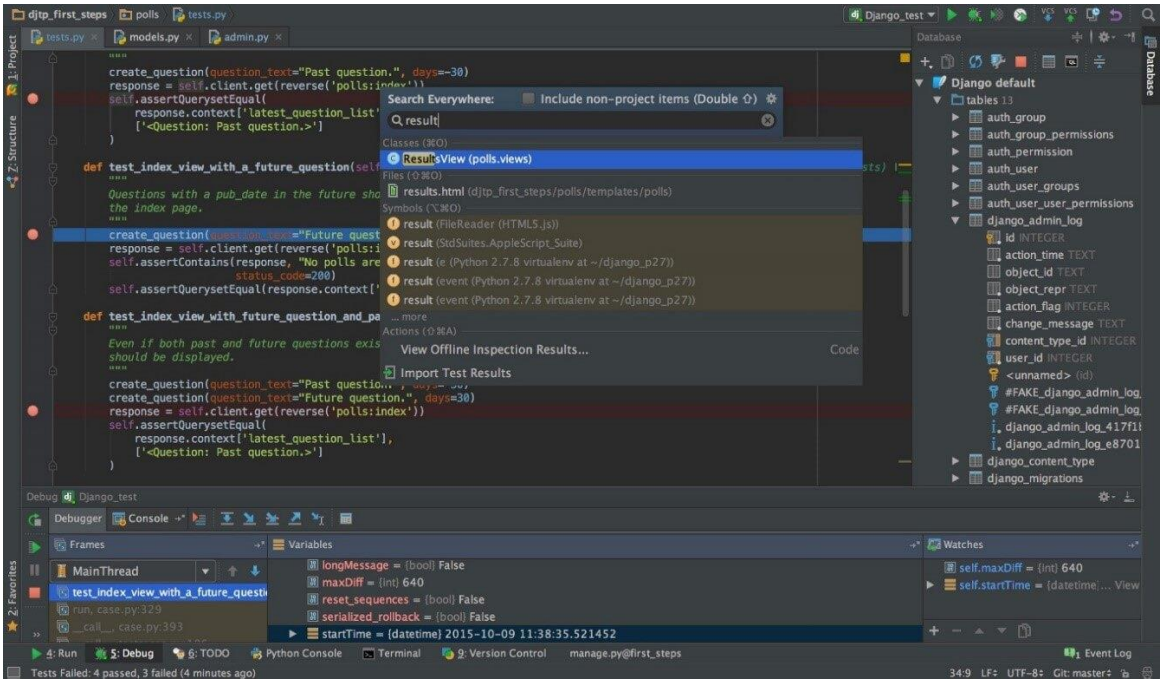

Рисунок 2.37 – Зображення робочого поля програмування у програмі «PyCharm»

Переваги використання MicroPython IDE для програмування мікроконтролерів:

Простий у вивченні та використанні: Python — це мова високого рівня з простим синтаксисом, що полегшує вивчення та використання початківцям.

Сумісність із різними платформами: MicroPython IDE сумісна з Windows, Linux і macOS, що робить її доступною для широкого кола користувачів.

Підтримка різноманітних мікроконтролерів: MicroPython підтримує низку мікроконтролерів, включаючи популярний ESP32.

Широка підтримка бібліотек: Python має величезну колекцію бібліотек, які можна використовувати з MicroPython, що полегшує додавання функцій до проектів мікроконтролерів.

Інтерактивна оболонка: MicroPython IDE містить інтерактивну оболонку, яка дозволяє розробникам тестувати та налагоджувати код на льоту.

Недоліки використання MicroPython IDE для програмування мікроконтролерів:

Повільніша швидкість виконання: код MicroPython інтерпретується, а не компілюється, що може призвести до нижчої швидкості виконання порівняно з скомпільованими мовами, такими як C або C++.

Обмежений доступ до апаратного забезпечення: Python є мовою високого рівня, і MicroPython не надає прямого доступу до апаратних функцій низького рівня, що може обмежити типи проектів, які можна розробляти.

Загалом, MicroPython використовує мову програмування Python для програмування мікроконтролерів, включаючи ESP32. Однак версія Python, яка використовується в MicroPython, оптимізована для роботи на мікроконтролерах з обмеженими ресурсами. Синтаксис і функціональні можливості MicroPython подібні до звичайного Python, але є деякі відмінності, наприклад відсутність підтримки певних бібліотек і використання модулів uPy для доступу до апаратних функцій.

### 2.2.2.3 Arduino IDE

Arduino IDE — це програма (інтегроване середовище розробки), яка використовується для програмування плат Arduino. Це програмне забезпечення з відкритим кодом, яке можна безкоштовно завантажити на веб-сайті Arduino. Програма базується на середовищі програмування Processing і використовує мову програмування C++, та доступна для операційних систем Windows, macOS і Linux [6, 11].

Дане середовище розробки надає простий і легкий у використанні інтерфейс для написання, компіляції та завантаження коду на плату, містить вбудований монітор послідовного порту для налагодження та тестування коду, а також різноманітні бібліотеки та приклади, які можна використовувати для початку програмування плати. Крім того, доступно багато бібліотек сторонніх

розробників, які можна завантажити та встановити, щоб додати додаткові функції до плати, яка програмується.

Arduino IDE можна використовувати для програмування широкого спектру мікроконтролерів, і не лише плат Arduino (Uno, Nano, Mega, тощо). Окрім цих офіційних плат, середовище розробки Arduino IDE також можна використовувати для програмування багатьох інших мікроконтролерів, які сумісні з екосистемою програмного забезпечення Arduino, зокрема: ESP8266, ESP32, STM32, плати на основі AVR (наприклад, ATtiny85, ATmega328P), плати на основі ARM (наприклад, SAM3X8E, SAMD21).

Проте, варто зазначити, що незважаючи на те, що Arduino IDE можна використовувати для програмування багатьох різних мікроконтролерів, для деяких плат може знадобитися встановлення чи налаштування додаткового програмного чи апаратного забезпечення. Крім того, деякі плати можуть вимагати модифікації основних бібліотек Arduino або пакетів підтримки плат для належної роботи з IDE.


Рисунок 2.38 – Сторінка завантаження Arduino IDE на офіційному веб-сайті


Рисунок 2.39 – Зображення робочого поля програмування у програмі «Arduino IDE»

Arduino IDE — популярне середовище розробки для програмування мікроконтролерів, таких як ESP32. Ось деякі переваги та недоліки використання Arduino IDE для програмування ESP32:

Переваги:

Простий у використанні: Arduino IDE дуже зручний і простий у використанні для початківців. Він має простий інтерфейс і містить багато вбудованих бібліотек, які полегшують програмування ESP32.

Кросплатформенність: Arduino IDE доступна для багатьох операційних систем, таких як Windows, Mac і Linux, що полегшує використання на будь-якій платформі.

Підтримка спільноти: Arduino має велику спільноту розробників, які активно сприяють його розвитку. Спільнота надає багато бібліотек і ресурсів, які можна використовувати для програмування ESP32.

Безкоштовна та Open Source: Arduino IDE можна безкоштовно завантажити та використовувати, а його вихідний код є відкритим. Це означає, що користувачі можуть змінювати та налаштовувати його відповідно до своїх потреб.

Сумісність: Arduino IDE сумісна з широким спектром мікроконтролерів,

включаючи ESP32.

Недоліки:

Обмежена підтримка налагодження: Arduino IDE надає обмежену підтримку налагодження, що може ускладнити виявлення та виправлення помилок у складних програмах.

Обмежена масштабованість: хоча Arduino IDE підходить для малих і середніх проектів, вона може бути не найкращим вибором для великомасштабних проектів через обмежену функціональність і підтримку налагодження.

Обмежене налаштування: незважаючи на те, що Arduino IDE є відкритим вихідним кодом, його може бути важко налаштувати, окрім додавання спеціальних бібліотек або зміни вихідного коду.

Обмеження пам’яті: Arduino IDE розроблено для роботи з мікроконтролерами з обмеженою пам’яттю, і це може бути обмеженням під час роботи зі складнішими програмами.

Обмежена функціональність: Arduino IDE — це базова IDE, яка надає обмежену функціональність порівняно з іншими IDE, такими як Visual Studio або Eclipse.

Загалом, Arduino IDE — чудовий вибір для початківців або любителів, які хочуть програмувати мікроконтролери ESP32. Він забезпечує простий у використанні інтерфейс, підтримку спільноти та сумісність між платформами. Однак для більш складних проектів або комерційних додатків інші IDE з більш розширеними функціями можуть бути більш придатними.

### 2.2.3 Загальні відомості про реле модуль. Його будова та функції

Реле — це електричний перемикач, яким можна керувати за допомогою малопотужного електричного сигналу, наприклад, вихідного сигналу мікроконтролера. Використовуючи мікроконтролер та релейний модуль, можна керувати високовольтними або потужними пристроями, такими як освітлення, двигуни та електроприлади, зі свого комп’ютера чи мобільного

пристрою. Реле складається з кількох ключових компонентів (Рисунок 2.40):

1. Котушка (Coil): котушка створює магнітне поле, коли через неї пропускають електричний струм, яке використовується для розмикання або замикання контактів перемикача.

2. Контакти: контакти є перемикаючими елементами реле і можуть бути нормально розімкненими (NO, normally open) або нормально замкнутими (NC, normally closed). Коли котушка знаходиться під напругою, магнітне поле притягує рухомий якір (armature), який розмикає або замикає контакти.

3. Якір (armature, арматура) — це рухомий компонент, який притягується магнітним полем, створюваним котушкою. Він розмикає або замикає контакти в залежності від стану котушки.

4. Рама (Frame): рама забезпечує механічну підтримку компонентів реле та захищає реле від зовнішніх пошкоджень.

5. Клеми (Terminals): Клеми забезпечують підключення реле до зовнішніх схем. Котушка підключається до схеми управління, а контакти підключаються до навантаження.

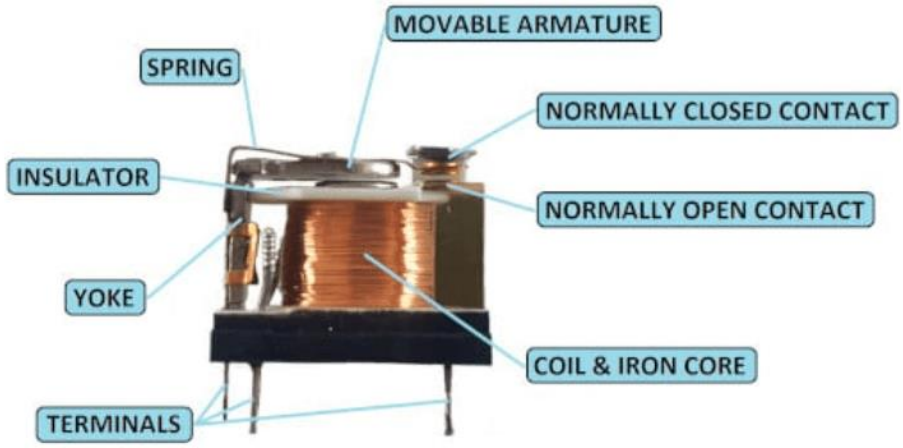


Рисунок 2.40 – Конструкція реле

В свою чергу, релейні модулі — це друковані плати, на яких розміщено одне або кілька реле. Вони бувають різних форм і розмірів, але найчастіше прямокутні з 2, 4 або 8 реле, встановленими на них. Крім блоку реле, модулі реле містять і інші компоненти, такі як індикаторні світлодіоди, захисні діоди, транзистори, резистори та інші деталі, для підвищення їх продуктивності та

надійності.

Вхідна напруга релейного модуля зазвичай становить постійний струм (DC). Однак електричне навантаження, яким керуватиме реле, може бути змінним або постійним струмом (AC or DC), але, по суті, в межах граничних рівнів, для яких розроблено реле. Релейний модуль доступний у низці номінальних значень вхідної напруги: це може бути релейний модуль на 3,2 В або 5 В для перемикання малої потужності, або це може бути релейний модуль на 12 або 24 В для важких систем.

Функція релейного модуля в основному полягає в увімкненні та вимкненні електричних пристроїв і систем. Він також служить для ізоляції ланцюга керування від пристрою чи системи, якою керують. Це важливо, оскільки дозволяє використовувати мікроконтролер або інший малопотужний пристрій для керування пристроями зі значно вищою напругою та струмом.

Інша мета релейного модуля полягає в посиленні керуючого сигналу, щоб він міг перемикати більш високі струми, використовуючи лише невелике споживання живлення від мікроконтролера.

Тобто, у розумній системі освітлення релейний модуль служить інтерфейсом між мікроконтролером і лампочками або іншими освітлювальними пристроями. Це потрібно для:

Ізоляції напруги: релейний модуль забезпечує електричну ізоляцію між мікроконтролером і ланцюгом освітлення високої потужності. Оскільки мікроконтролер працює при нижчій напрузі (наприклад, 3,3 В або 5 В), релейний модуль гарантує, що потужна схема, яка зазвичай працює при напрузі мережі (наприклад, 110 В або 220 В), не впливає на схему мікроконтролера.

Перемикання живлення: релейний модуль діє як перемикач, який керує живленням лампочок. Підключивши лампочки до контактів реле, можна керувати їх ввімкненням/вимкненням за допомогою мікроконтролера. Це дозволяє автоматизувати систему освітлення, дистанційно вмикати та вимикати світло або програмувати їх реагування на певні події чи умови.

Релейні модулі доступні з конфігураціями перемикача з нормально відкритим (NO) або нормально замкнутим (NC). Розпінування релейного модуля 5 В складається з з’єднань на вхідній стороні, де він отримує сигнал запуску, і на вихідній стороні, де він контролює навантаження (Рисунок 2.41).

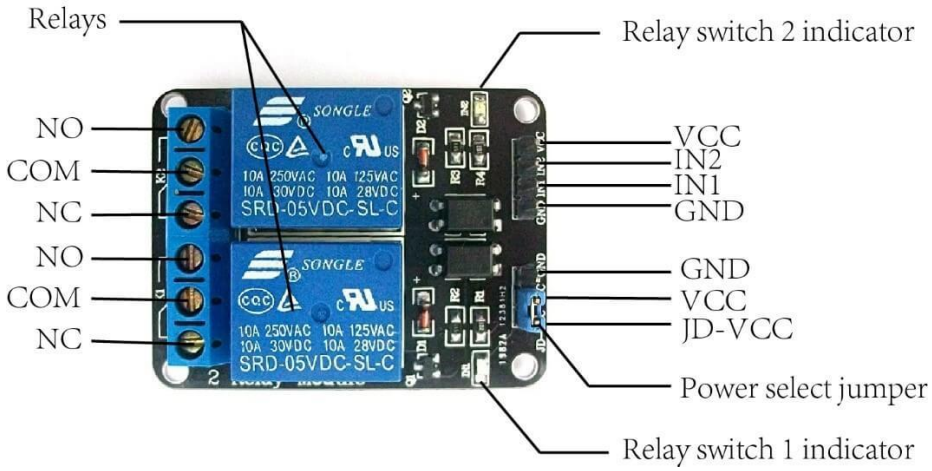


Рисунок 2.41 – 2х канальний реле модуль

Вхідна сторона релейного модуля має 3 або 4 з’єднання: вони перераховані та пояснені нижче.

VCC – це підключення живлення. Він подає 5 В постійного струму на модуль і зазвичай підключається до позитивної клеми джерела живлення.

GND – це заземлення. Він підключається до мінусової клеми джерела живлення.

IN1, IN2 – це входи, на які подається тригерний сигнал. IN1 призначений для одноканального релейного модуля, тоді як IN2 для двоканального релейного модуля. Вивід IN (вхід) підключається до виходу мікроконтролера, датчика або логічного пристрою.

Вихідна сторона релейного модуля має три підключення:

NO (Normally Open, нормально відкритий) – це підключення навантаження, коли реле ввімкнено. Коли реле вимкнено, NO підтримує відкрите з'єднання з COM. Тобто перемикач NO відкритий, коли електромагніт не активований, і закритий, коли він активований.

COM (Common, загальний) – з’єднання модуля реле, позначене «COM»,

є загальним з’єднанням як для контактів NO, так і для NC (нормально замкнутих).

NC (Normally Closed, нормально закритий) – це підключення навантаження. Він підключається до терміналу COM за замовчуванням або коли реле вимкнено. Тобто NC релейний перемикач залишається замкнутим за замовчуванням і відкривається лише тоді, коли реле активовано.

## 2.4 Висновки за розділом

У даному розділі здійснено вибір та обґрунтування програмно-апаратного комплексу. Зокрема, було описано загальні відомості про реле модуль, його будову та функції. Проведено огляд та аналіз існуючих мікроконтролерів. Було розглянуто програмні засоби розробки проекту.

Для реалізації апаратної частини системи розумного оствілення було вирішено використовувати мікроконтролер ESP32 та плату розробки ESP32 DEVKIT V1, що розроблена на основі модуля ESP-WROOM-32 з підтримкою Wi-Fi. У якості інтегрованого середовища розробки було обрано Arduino IDE. Також знадобиться 4х канальний реле модуль.

Для створення додатку і інтерфейсу користувача було вирішено використовувати наступні технології та інструменти: мову розмітки гіпертексту HTML, каскадні таблиці стилів CSS, мову програмування JavaScript, фронтенд бібліотеку React, середовище виконання Node.JS, бекенд фреймворк Express.js, реляційну систему управління базами даних MySQL, мову структурованих запитів SQL, графічний інтерфейс адміністрування систем управління базами даних DBeaver, програмну платформу Docker, та інтегроване середовище розробки Visual Studio Code.

# РОЗДІЛ 3
# РОЗРОБКА СИСТЕМИ. РЕАЛІЗАЦІЯ ПРОЕКТУ.

## 3.1 Проектування та моделювання системи

Проектування – це перетворення вимог до розроблення у послідовність проектних рішень щодо способів їх реалізації, а саме: формування загальної архітектури програмної системи та принципів її прив’язки до конкретного середовища функціонування; визначення детального складу модулів кожної з архітектурних компонент. Іншими словами, проектування - це етап життєвого циклу розроблення програмних систем, наступний після визначення вимог. Завданням цього етапу є перетворення вимог у простір проектних рішень, подання їх у вигляді моделей.

Моделювання – це процес створення точного опису системи; це метод пізнання, створенння і дослідження моделей. Моделювання полегшує вивчення об'єкта з метою його створення, подальшого перетворення і розвитку. Воно використовується щоб дослідити існуючу систему, і тоді, коли реальний експеримент проводити недоцільно через значні фінансові і трудові витрат, а також тоді, коли необхідно провести аналіз проектованої системи, яка ще фізично не існує.

Для проектування систем, використовують інформаційні моделі. Вони представляють об'єкти і процеси у формі різноманітних рисунків, схем, таблиць, формул, текстів і т.п. Для реалізації та проектування проекту (від етапу аналізу до створення програмного коду) існує велика кількість інструментальних засобів, які можна використовувати. Окремо виділяють так звану UML-модель.

UML (від англ. Unified Modeling Language, у перкладі українською – уніфікована мова моделювання) – це мова позначень або побудови діаграм, призначена для визначення, візуалізації і документування моделей, які

зорієнтовані на об’єкти систем програмного забезпечення. UML не являється мовою програмування. Це відкритий стандарт графічного опису програмного забезпечення та загальноприйнятий засіб моделювання в програмній інженерії.

Конструкції UML створюються з багатьох модельних елементів, які позначають різні частини системи програмного забезпечення. Ці елементи UML мови використовуються для побудови діаграм, що відповідають певній частині системи або точці зору на систему.

В мові UML є 12 типів діаграм:

- 4 типи діаграм представляють статичну структуру додатку;

- 5 типів представляють поведінкові аспекти системи;

- 3 представляють фізичні аспекти функціонування системи (діаграми реалізації).

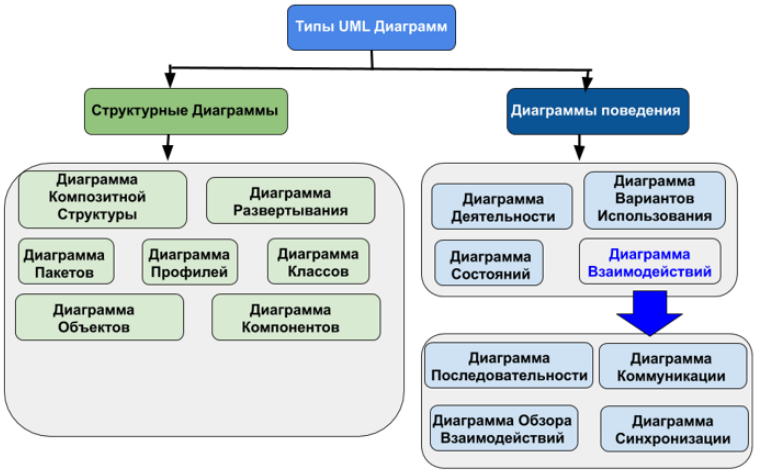


Рисунок 3.43 – Типи UML діаграм

Деякі з видів діаграм специфічні для певної системи і додатку. Найбільш популярними та широко використовуваними з них є:

- Діаграма прецедентів (Use-case diagram);

- Діаграма активностей (Activity diagram);

- Діаграма послідовності (Sequence diagram);

Діаграма прецедентів (Use-case diagram, також її ще називають діаграмою варіантів використання) – вона дозволяє уявити типи ролей та їх

взаємодію із системою. Звісно, це не показує порядок виконання кроків, а всього лише зображує функціональні вимоги системи(те, що система може зробити) з точки зору користувача. Тобто діаграма use-case показує людей або інших користувачів системи (дієвих осіб), сценарії використання системи (випадки використання) та взаємодію між ними.

Діаграма активності (діаграма діяльності) візуалізує процес використання системи, ілюструє потік повідомлень від однієї дії до іншої, та зміни дій, які є наслідком подій, що сталися у певній частині системи. Показує цілісну роботу системи. Іншими словами, ця діаграма у вигляді блок-схеми описує динамічні аспекти поведінки системи, відображає бізнес-процеси, логіку процедур і потоки робіт — перехід від однієї дії до іншої. По суті, зображується алгоритм дій (логіка поведінки) системи або взаємодії кількох систем.

Діаграма послідовності (англ. Sequence Diagram) – це діаграма що показує часові особливості передачі і прийому повідомлень об’єктами. ЇЇ не треба плутати з часовими діаграмами! Під «впорядкованістю за часом» мається на увазі послідовність дій. Дана діаграма описує поведінкові аспекти системи та використовується для уточнення діаграм прецедентів. Вона відображає взаємодію об'єктів в динаміці, в часі. При цьому сама інформація набуває вигляду повідомлень, а взаємодія об'єктів передбачає обмін цими повідомленнями в рамках сценарію.

Різниця між діаграмою діяльності (активності) та послідовності в тому, що діаграми послідовності показують порядок виконання дій або їх послідовність, а діаграма діяльності показує перехід від однієї діяльності до іншої, і зазвичай потрібна для опису роботи всієї системи.

Згідно вимогам та функціям, описаним у першому розділі, було спроектовано концептуальну схему роботи системи розумного освітлення та побудовано UML діаграми (Рисунки 3.44 – 3.47).

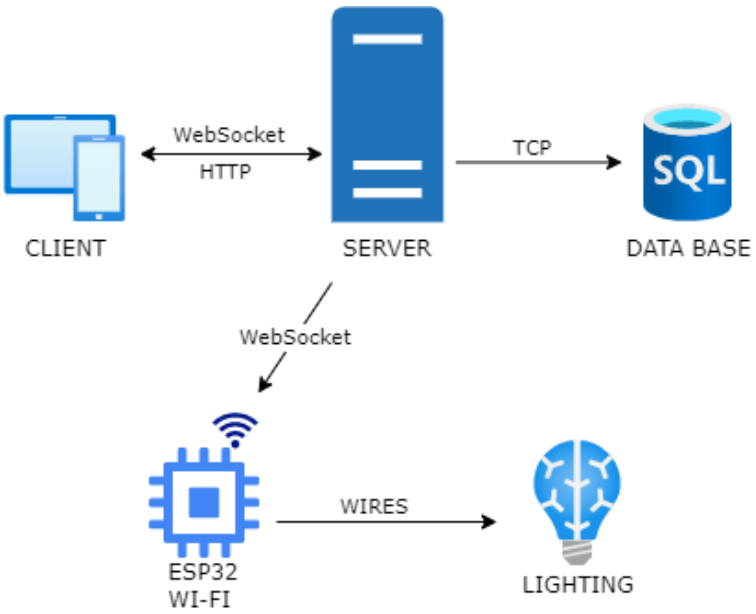


Рисунок 3.44 – Загальна схема роботи системи

Система розумного освітлення має клієнт-серверну архітектуру.

Клієнт з'єднується з сервером через WebSocket протокол для того щоб отримувати інформацію про останній стан освітлювальних приладів (лампочок), або відправляти інформацію про змінений (оновлений) стан лампочки. Тут використовується протокол WebSocket, тому що до системи можуть бути підключені декілька клієнтів (смартфон 1, смартфон 2, планшет, ноутбук, тощо). І якщо користувач змінює стан лампочок на одному із клієнтів, то ці змінення повинні відобразитися і на інтерфейсах інших клієнтів, підключених до системи (в режимі реального часу).

Також клієнт (UI, інтерфейс користувача) використовуючи HTTP протокол, відправляє запит на бекенд для наступних функцій:

- для отримання списку запланованих задач (розкладу) по увімкненню та ввимкненню освітлення в певний час та в певній кімнаті;

- редагування вже запланованої задачі у розкладі з можливістю зміни часу, кімнати, статусу світла;

- додавання нової задачі в розклад, або видалення вже існуючої;

- для отримання списку історії з увімкнення та вимкнення освітлення в запланований час та у певній кімнаті з можливістю відфільрувати історію по даті та кімнаті;

- для отримання статистичних данних щодо тривалості використання електроенергії у певний період та у певній кімнаті з можливістю відфільрувати статистику по окремій кімнаті, періоду (рік, місяць, тиждень, день) та виміру відповідно (назви місяців, дні місяця, дні тижня, години дня).

У свою чергу, сервер надсилає запит до бази даних через TCP протокол для збережання (створення), редагування, та\або видалення даних.

Мікроконтролер в системі виступає у ролі клієнту. Він підключений до локальної мережі (до маршрутизатору) у режимі «станція». Цей режим використовується для підключення мікроконтролеру до існуючої мережі Wi-Fi як клієнтський пристрій. Для обробки та передачі даних між мікроконтролером та сервером використовується WebSocket протокол.

Мікроконтролер завжди отримує повідомлення від серверу, і на основі отриманої інформації фізично змінює стан освітлювальних приладів (тобто подає сигнали, керує освітленням, вмикає або вимикає лампочки).

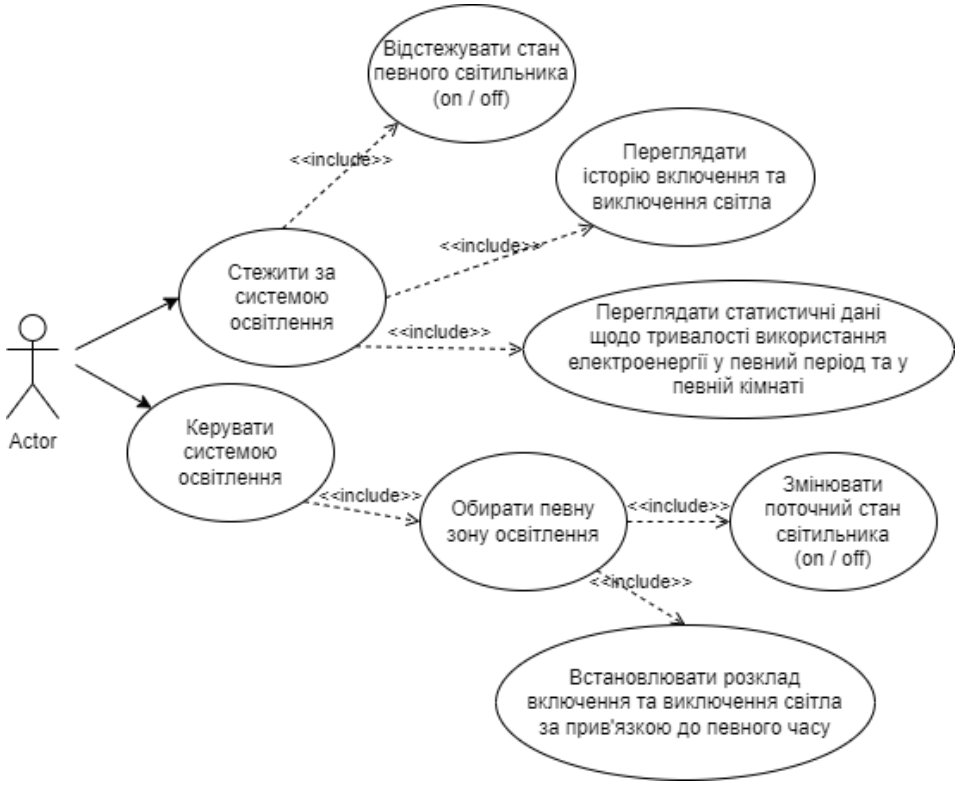


Рисунок 3.45 – Діаграма прецедентів (use-case)

Існує лише один актор додатку – користувач. Він може керувати світлом у кожній зоні освітлення (кімнаті), планувати та встановлювати розклад увімкнення та вимкнення освітлення, щоб світло автоматично включалося чи вимикалося у певній кімнаті в заданий час. Також має можливість стежити за системою освітлення, переглядати історію та статистичні дані щодо тривалості використання електроенергії у певний період та у певній кімнаті.

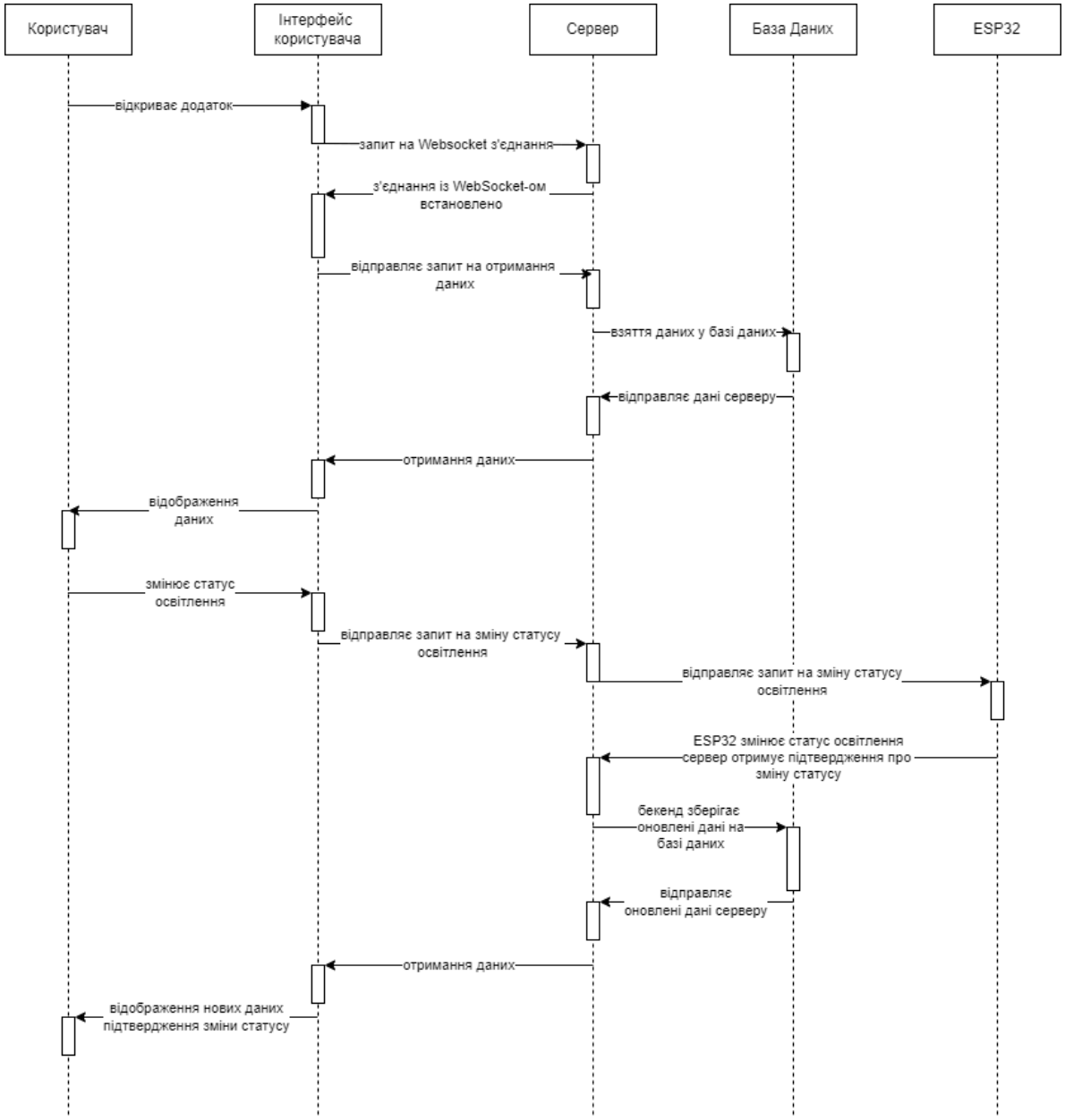


Рисунок 3.46 – Діаграма послідовності (sequence diagram)

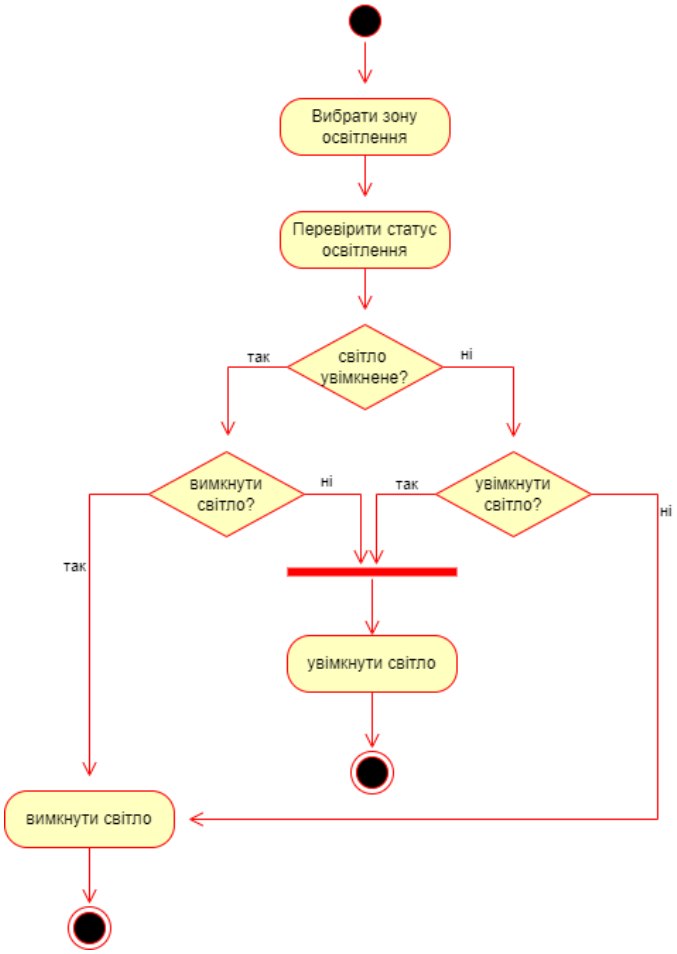


Рисунок 3.47 – Діаграма активності (activity diagram)

## 3.2 Програмна реалізація проекту

Веб-додаток системи «Smart Light» відноситься до клієнт-серверної архітектури. Тому розробка програми розділена на 2 частини: клієнтська і серверна. Взаємодія між клієнтом та сервером показана на Рисунку 3.48.

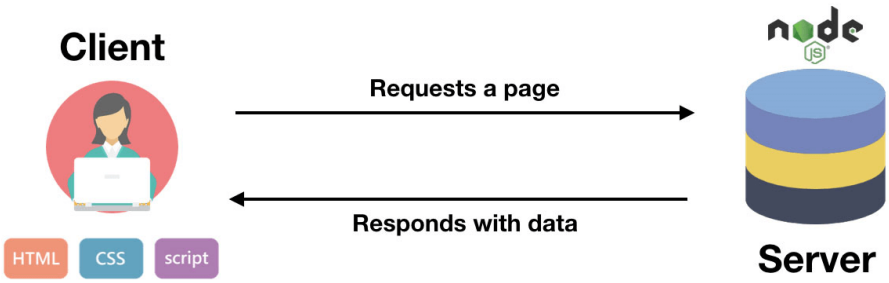


Рисунок 3.48 – Архітектура «клієнт - сервер»

Архітектура клієнт-сервер — це обчислювальна модель, яка визначає взаємозв’язок між двома об’єктами: клієнтом і сервером. Це широко використовуваний підхід для проектування та реалізації розподілених програм і систем, де можливості обробки та зберігання розподілені між кількома машинами.

У цій архітектурі клієнт — це пристрій або програмне забезпечення, яке ініціює запит на послугу або ресурс. Сервер, з іншого боку, є спеціалізованим комп’ютером або програмним додатком, який відповідає на запити клієнта та надає запитувану послугу чи ресурс.

Основна концепція клієнт-серверної архітектури полягає в розподілі праці та відповідальності між клієнтом і сервером. Клієнт відповідає за ініціювання запитів і надання інтерфейсу користувача для взаємодії, тоді як сервер відповідає за обробку запитів, виконання обчислень і керування спільними ресурсами.

Зв'язок між клієнтом і сервером зазвичай відбувається за моделлю запит-відповідь (request-response). Клієнт надсилає запит серверу, який у свою чергу обробляє запит і надсилає клієнту відповідь із запитаними даними або результатом запитаної операції. Цей зв’язок може відбуватися через різні мережеві протоколи, наприклад HTTP, TCP/IP або WebSocket.

### 3.2.1 Клієнтська частина додатку

Інтерфейс користувача було створено в середовищі розробки Visual Studio Code за допомогою мови розмітки гіпертексту HTML, каскадних таблиць стилів CSS, мови програмування JavaScript, фронтенд бібліотеки React.

Дизайн користувальницького інтерфейсу виконано у темній кольоровій гамі. Таке рішення є комфортним для зору очей, особливо у нічний період часу.

Інтерфейс користувача складається із 4х вкладок: rooms, scheduler, history, statistics (відповідно переклад: кімнати, розклад, історія, статистика).

На сторінці rooms (Рисунок 3.49) відображається список кімнат у вигляді блоків, кожен з якої має інформацію про назву кімнати, та містить тригер-кнопку увімкнути/вимкнути світло (toggle button).

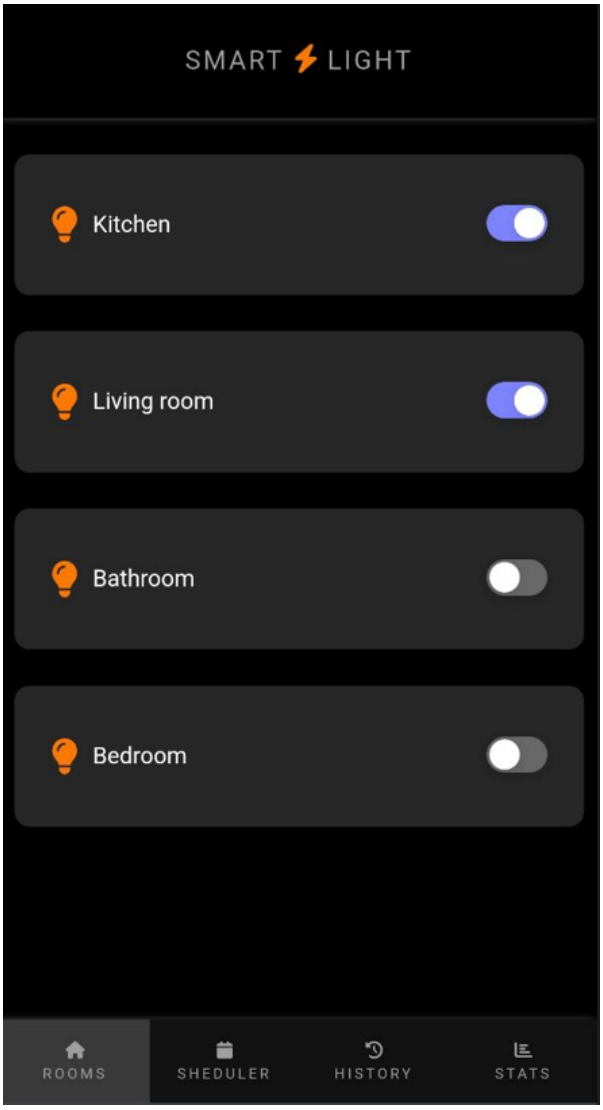


Рисунок 3.49 – Головна сторінка інтерфейсу «rooms»

На сторінці scheduler (Рисунок 3.50 та 3.51) користувач може переглядати список запланованих задач (розклад) по увімкненню та ввимкненню освітлення в запланований час та в певній кімнаті. У верхньому правому кутку є кнопка додавання нової задачі в розклад. При додавані нової задачі обирається кімната, статус, дата та час, коли і у якій кімнаті лампочка повинна увімкнутися чи вимкнутися автоматично. При редагуванні існуючої задачі є можливість змінити дату, час, статус освітлення та саму кімнату, або взагалі видалити задачу.

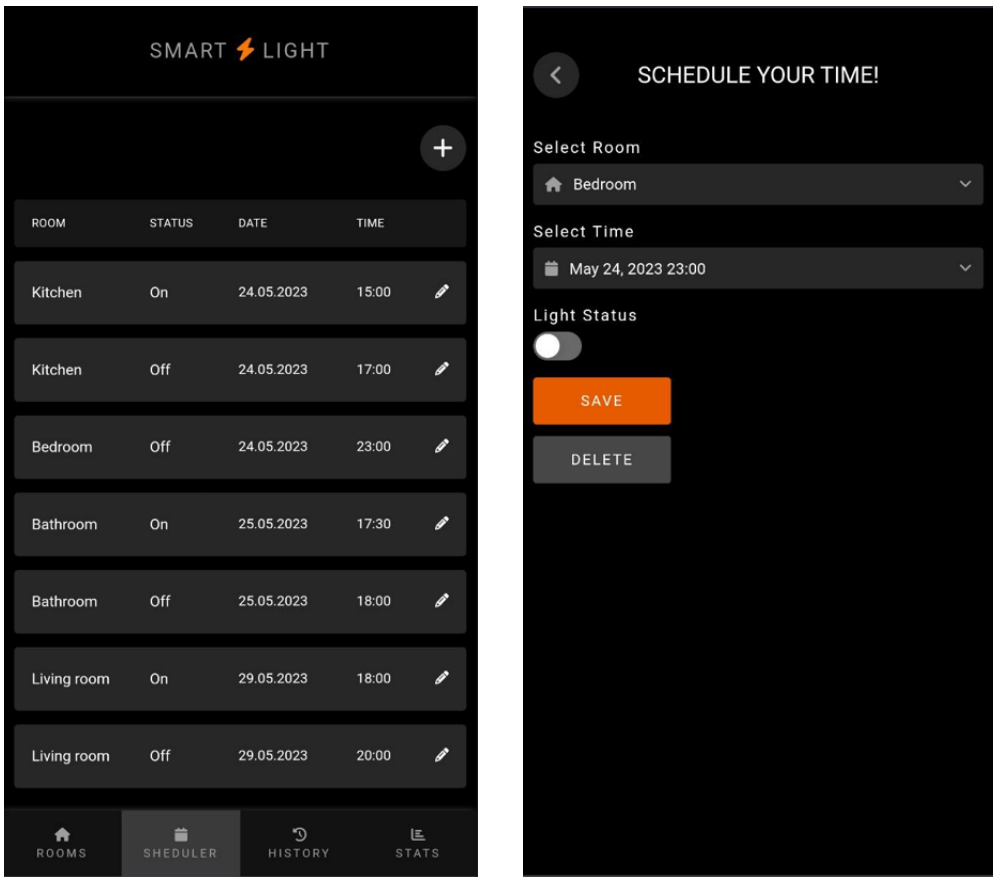


Рисунок 3.50 – Сторінка «scheduler».

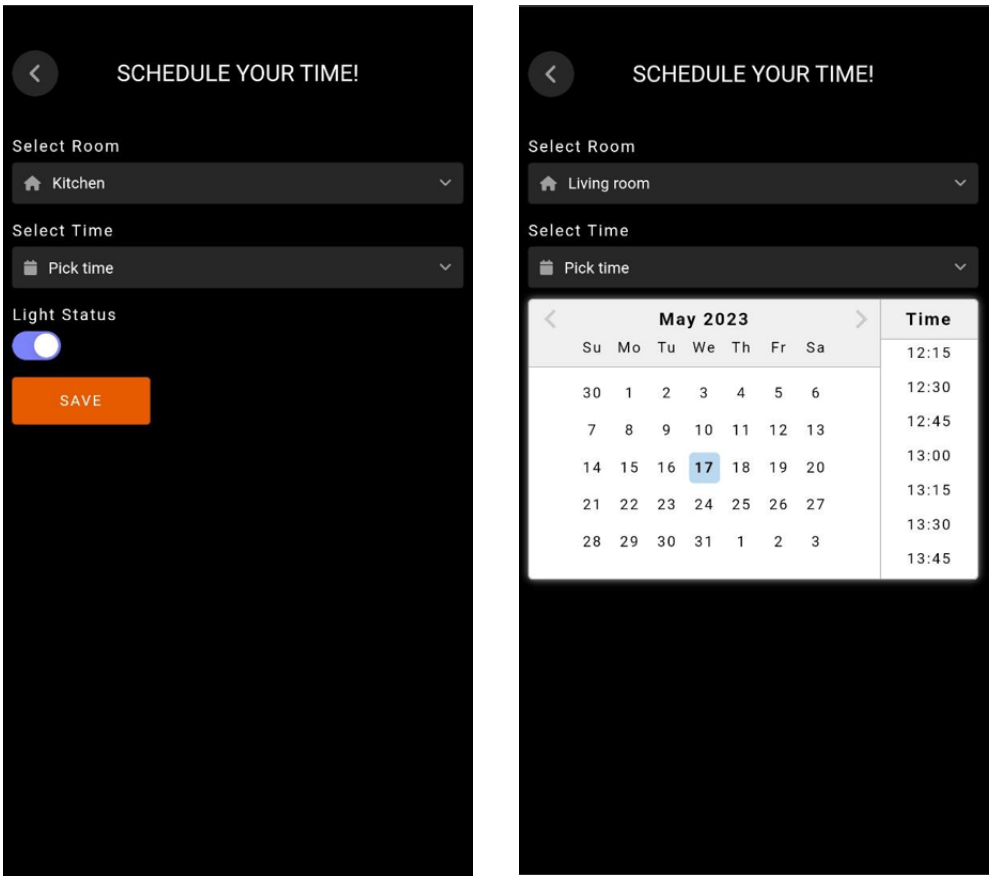


Рисунок 3.51 – Сторінка «scheduler». Нова задача, вибір дати та часу

На сторінці history (Рисунок 3.52) користувач може передивлятися історію, коли було автоматично увімкнено бо вимкнено освітлення згідно розкладу, в запланований час та у певній кімнаті. Є можливість відфільрувати історію по даті та кімнаті.

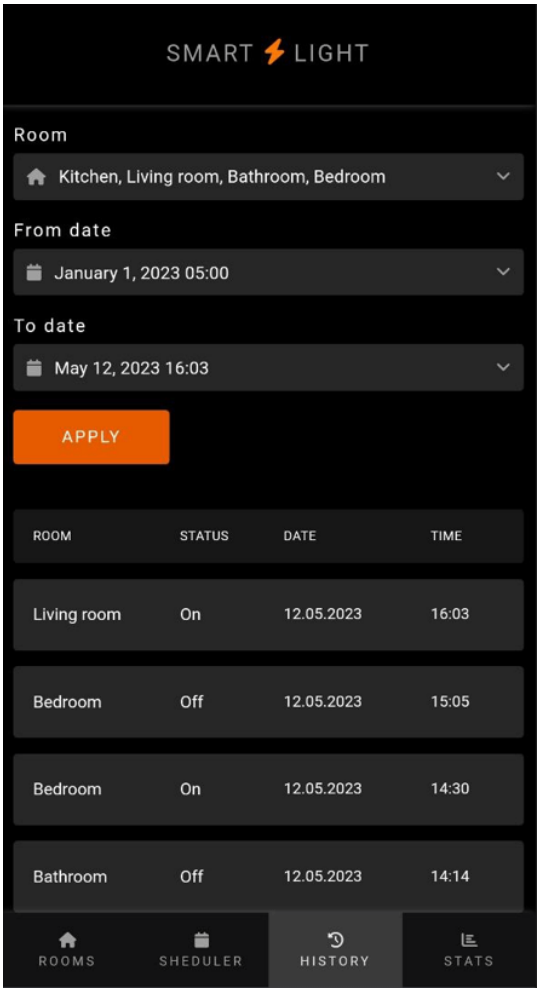


Рисунок 3.52 – Сторінка «history»

На сторінці statistics (скорочено – stats, Рисунок 3.53) користувач може передивлятися статистику, скільки часу було увімкнене або вимкнене світло у певній кімнаті (або у всіх кімнатах) протягом певного періоду часу. Є можливість відфільтрувати статистику по окремій кімнаті, періоду (рік, місяць, тиждень, день) та виміру відповідно (назви місяців, дні місяця, дні тижня, години дня). Відображення даних подано у вигляді графіка.

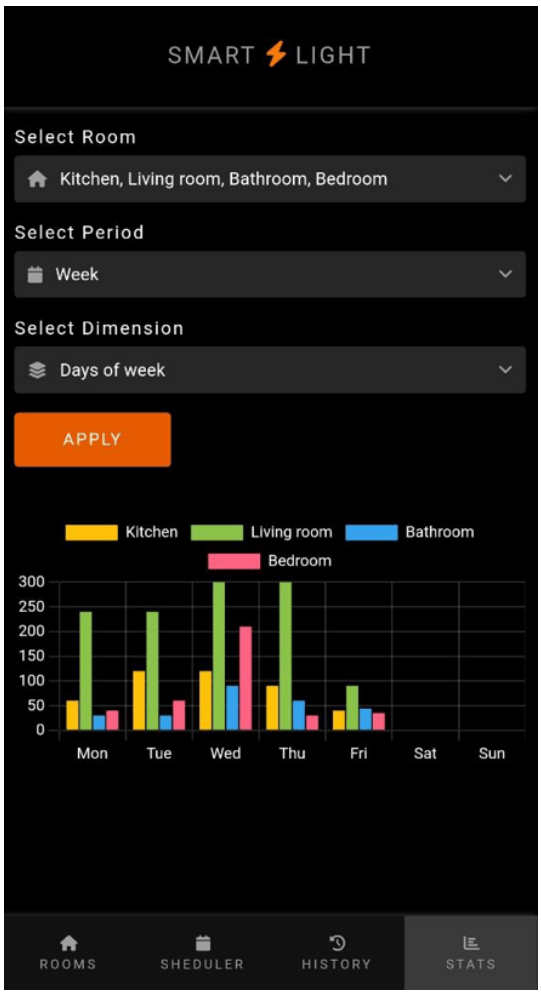


Рисунок 3.53 – Сторінка «statistics»

На Рисунку 3.54 зображено діаграму класів клієнської частини проекту.

Діаграма класів клієнтської частини вийшла доволі великою. Це трапилось через те, що у своїй роботі я використовую React.js. Це не фреймворк у повноцінному розумінні, а бібліотека зі своїми властивостями та параметрами. Все це оформлено у велику кількість класів, які наслідуються один від одного. Але мені потрібен лише клас Component (React.Component), який я розширюю та додаю до нього різні компоненти, особливі методи та атрібути.

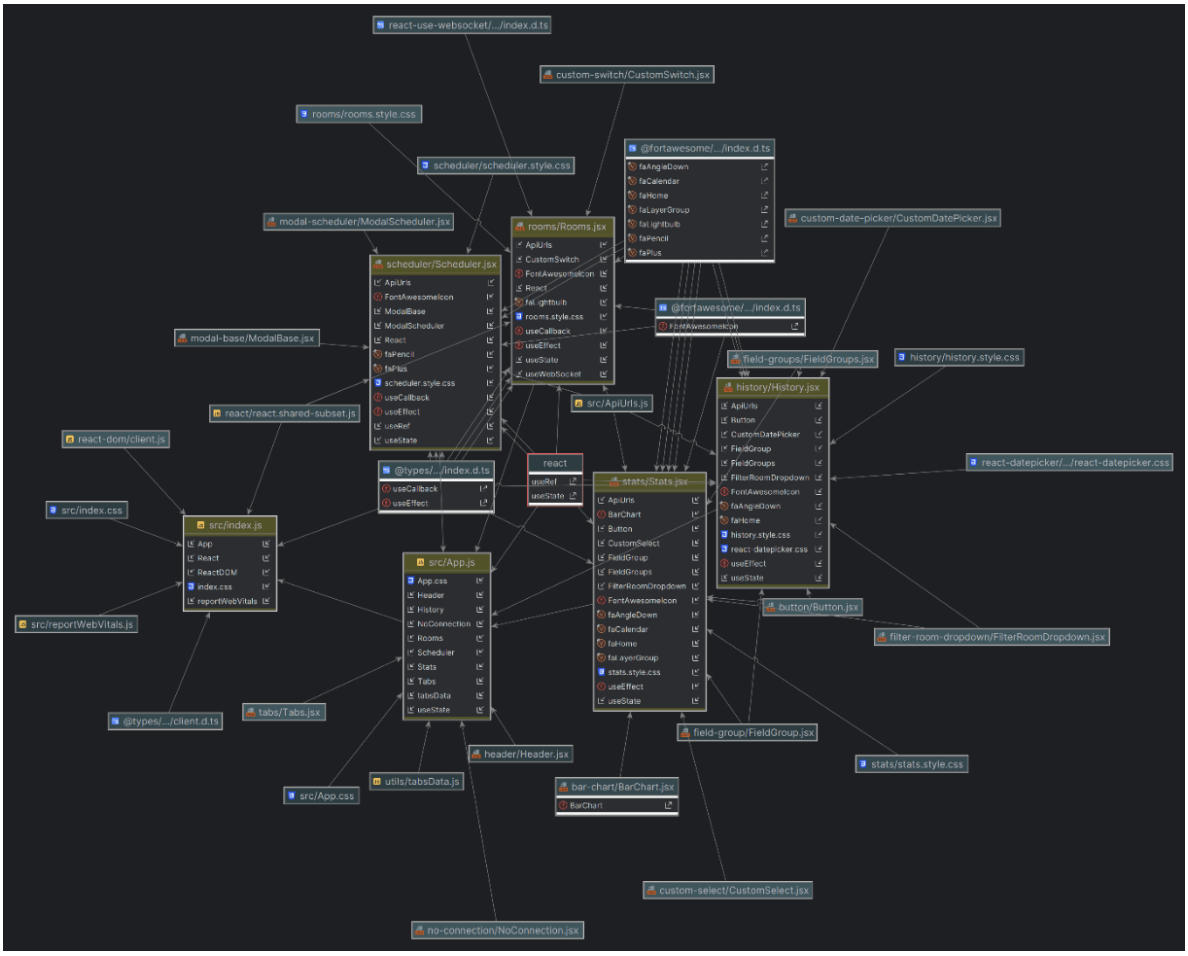


Рисунок 3.54 – Діаграма класів (клієнт)

Розглянемо структуру клієнтської частини веб-додатку, що зображена нижче на рисунку 3.55.

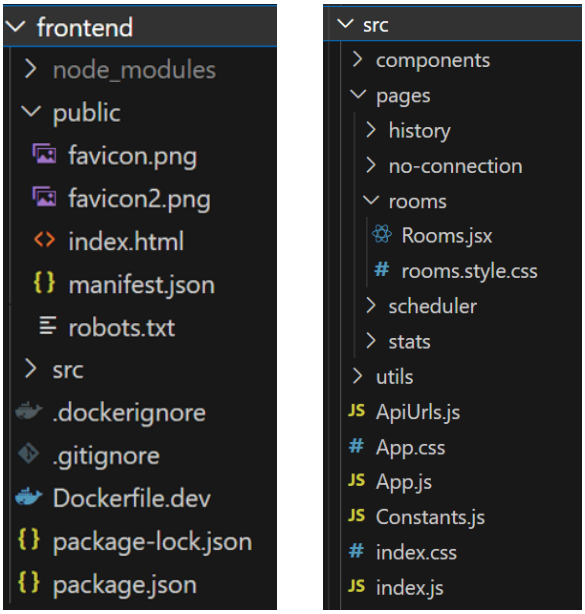


Рисунок 3.55 – Структура клієнтської частини додатку

Папка «node_modules» — це всі встановлені модулі проекту. Сюди потрапляють файли встановлених сторонніх бібліотек, фреймворків, модулів. Тобто у цій директорії знаходяться всі модулі, які було встановлено через npm.

Файл «index.html» у папці Public є «точкою входу» для браузера. Код файлу «index.html» предоставлено нижче у Лістингу 3.1.

Файл «favicon.png» – це фавіконка сайту (маленький значок, який відображається на відкритій вкладці браузера).

Src – основна папка проекту. Вона містить файли і папки, що становлять саме React-додаток. Іншими словами, тут знаходяться всі вихідні тексти проекту.

Файл «index.js» у папці src — точка входу в додаток клієнта, в якому ініціалізуються інстанси додатку з усіма параметрами, і відбувається рендеринг головного компоненту проекта – <App /> component, який у свою чергу відповідає за функціональність та рендеринг інших компонентів, який відправляється в файл index.html.  Код файлу «index.js» предоставлено нижче у Лістингу 3.2.

Components та Pages – це папки для створених react компонентів. В кожному модулі знаходитися своя власна папка, яка зазвичай має два файли: один файл містить компонент (наприклад, «Rooms.jsx»), а інший – стилі (відповідно, «Rooms.style.css»). Компоненти, які не перетворюють вхідні дані і не використовують всередині себе інші компоненти, називаються «dumb component». Протилежні їм – «smart component».

Файл «package.json» містить у собі інформацію про проект, таку, як назва проекту, версія, ліцензія, автор, опис, а також залежності проекта та скрипти. Бібліотеки і пакети, що встановлені, повинні бути вказані тут.

Лістинг 3.1 – Скрипт файлу «index.html»

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon2.png" />
```

```
    <meta name="viewport"
      content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon"
      href="%PUBLIC_URL%/favicon.png" />
    <link rel="manifest"
      href="%PUBLIC_URL%/manifest.json" />
    <title>Smart Light</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <div id="modal-root"></div>
  </body>
</html>
```

Лістинг 3.2 – Скрипт файлу «index.js» клієнської частини додатку

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App />
);
```

### 3.2.2 Програмування серверної частини додатку та створення БД

Для створення серверної частини додатку було використано наступні основні залежності: Express, Nodemon, Knex, Ws.

Express — це мінімалістичний та гнучкий фреймворк для веб-застосунків, побудованих на Node.js, що надає широкий та великий набір функціональності (функцій) для мобільних і веб-додатків. Він також використовується для швидкого створення "скелету" додатка. Простими словами, Express – це програмний каркас розробки серверної частини веб-застосунків для Node.js.

Nodemon – це утиліта інтерфейсу командної строки, яка відстежує файлову систему додатку Node і автоматично перезапускає процес, якщо є така необхідність. Якщо говорити простіше, то Nodemon – це пакет, що зауважує всі зміни в коді та перезапускає сервер. Це дуже зручно, тому що немає необхідності зупиняти і запускати сервер кожен раз, як будуть внесені будь-які зміни в код.

Knex – це популярний конструктор запитів для Node.js, який дозволяє розробникам створювати SQL-запити за допомогою JavaScript. Він забезпечує чистий та інтуїтивно зрозумілий синтаксис для створення запитів до бази даних, полегшуючи взаємодію з реляційними базами даних. За допомогою Knex можна підключитися до різних систем баз даних, таких як MySQL, PostgreSQL, SQLite та Oracle. Він підтримує такі функції, як побудова запитів, міграція даних і об’єднання з’єднань.

Ws (скорочено від WebSocket) – це проста у використанні клієнтська та серверна реалізація WebSocket. Це популярна бібліотека (модуль) для Node.js, яка забезпечує простий спосіб реалізації серверів і клієнтів WebSocket із двустороннім з’єднанням. Він пропонує простий і ефективний API для роботи з WebSocket протоколом, що дозволяє створювати програми в реальному часі, сервери чату, інструменти для спільної роботи тощо.

Нижче на рисунку 3.56 зображено діаграму класів серверної частини проекту.

На діаграмі класів серверної частини зображено лише ті класи та функції, які було створено мною. Інші допоміжні бібліотеки я не долучала до цієї схеми, щоб не перезавантажувати її. Головна точка входу – це файл «index.js», який знаходиться у кореневій папці «backend > app».

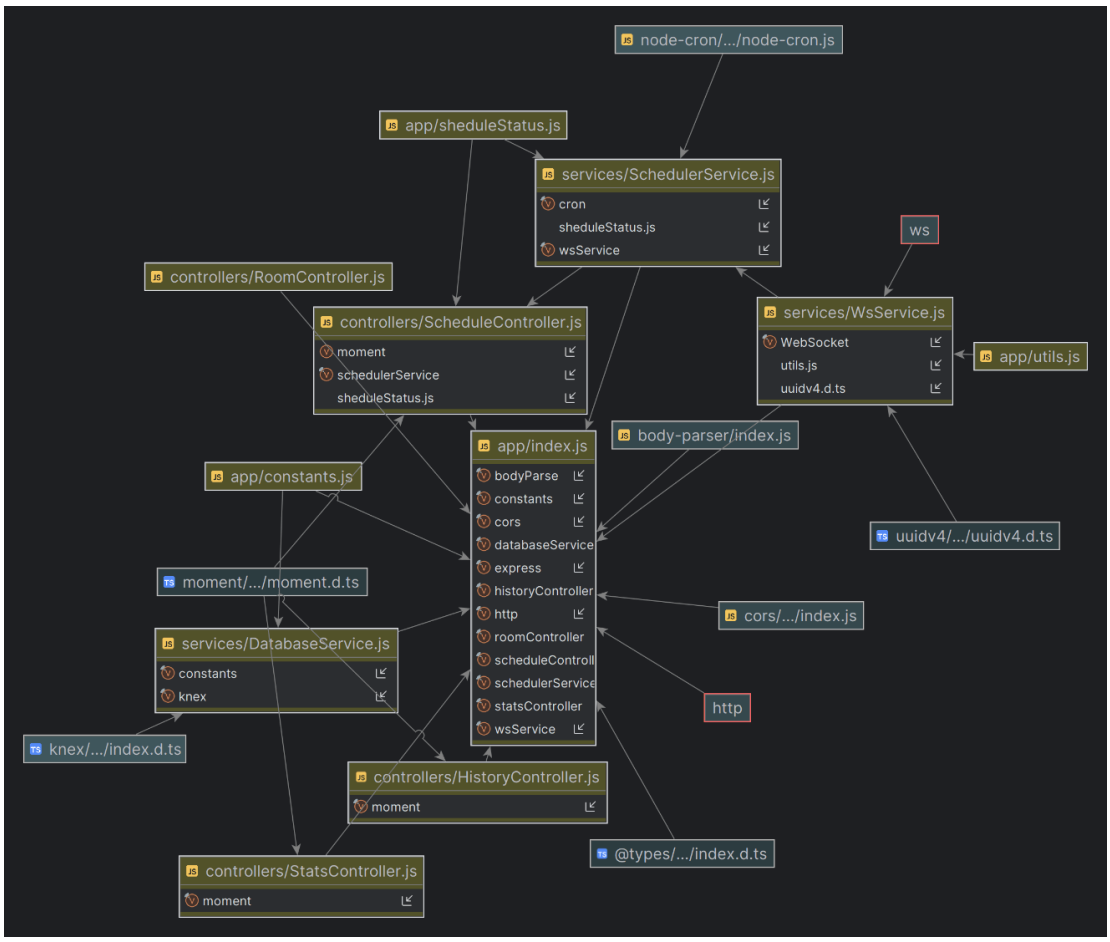


Рисунок 3.56 – Діаграма класів (сервер)

Розглянемо структуру серверної частини веб-додатку, що зображена на Рисунку 3.57.

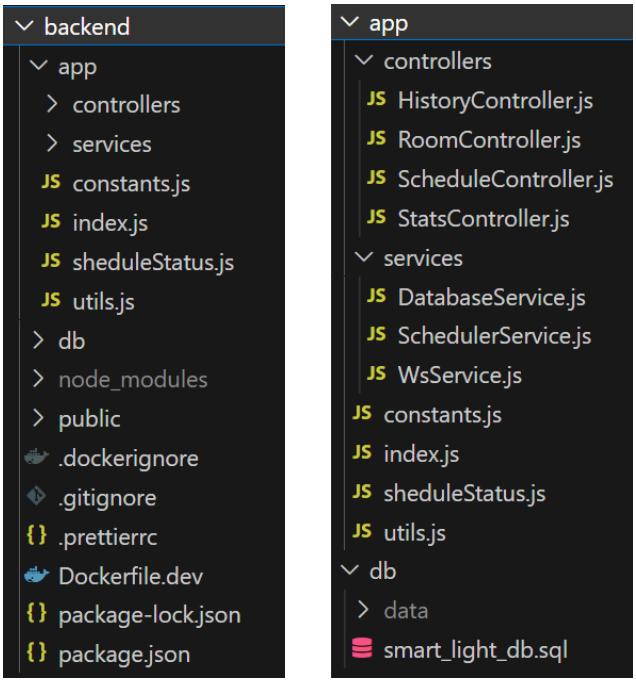


Рисунок 3.57 – Структура серверної частини додатку

Папка node_modules (як і в клієнтській частині) — це всі встановлені модулі проекту. Сюди потрапляють файли встановлених сторонніх бібліотек, фреймворків, модулів.

Controllers – це папка що містить файли «контролери». Контролер відповідає за отримання запиту від клієнта та його обробку. В даному проекті їх 4 шт: HistoryController, RoomController, ScheduleController, StatsController.

HistoryController.js відповідає за отримання історії та фільтрів, щоб мати можливість фільтрувати історію за назвою кімнати та проміжком часу (від-до).

RoomController.js відповідає за отримання списку кімнат на сторінці розкладу при додавані нової задачі в розклад чи редагуванні існуючої.

ScheduleController.js відповідає за отримання списку всіх запланованих задач на сторінці розкладу; за отримання даних вже запланованої задачі при її редагувані; за створення (додавання) нової задачі у розклад; за редагування та\або видалення існуючої задачі в рокладі.

StatsController.js відповідає за отримання статистичних даних про тривалість освітлення у певній кімнаті за певний період. А також за отримання фільтрів, щоб мати можливість фільтрувати інформацію за певними показниками, такі як назва кімнати, періоду (рік, місяць, тиждень, день) та виміру відповідно (назви місяців, дні місяця, дні тижня, години дня).

Services – це допоміжна папка, що зберігає наступні 3 файли: DatabaseService, SchedulerService, WsService.

DatabaseService.js відповідає за встановлення з’єдання з базою даних, використовуючи Knex бібліотеку. Дане з’єднання буде використовуватися для того щоб маніпулювати даними бази даних.

SchedulerService.js відповідає за роботу розкладу, точніше за так звані «джоби» (від англ. job). У даному контексті термін означає певну задачу в розкладі, яку потрібно виконати. У нашому випадку це заплановане автоматичне увімкнення чи ввимкненя світла у кімнаті.

WsService.js відповідає за запуск WebSocket серверу; за встановлення з’єднання з клієнтами. При першому запусці веб-сокет серверу кожен клієнт в

системі, який має з’єднання з бекендом, отримує останній стан лампочок та відображає його на своїй стороні. Сюди входять як UI клієнти (інтерфейси користувачів), так і сам мікроконтролер (бо в нашій системі освітлення він виступає у ролі клієнту).

Папка «db» – це база даних нашого проекту. Про неї поговоримо трішки пізніше (див. на сторінці 114). Код файлу «smart_light_db.sql» предоставлено у Лістингу 3.5 на тій же сторінці.

Файл «index.js» – точка входу в додаток, ініціалізація і запуск сервера. Він робить з’єднання з базою даних для того щоб маніпулювати даними (зберігати, змінювати, видаляти їх). Визначає роути (маршрути), які на основі URL та методу запиту (GET, POST, PUT, DELETE) визначають який треба використовувати контролер для обробки цього запиту. Також створює HTTP сервер та WebSocket сервер, та запускає їх (сервери). Код файлу «index.js» предоставлено нижче у Лістингу 3.3.

Лістинг 3.3 – Скрипт файлу «index.js» серверної частини додатку

```
const constants = require('./constants')
const express = require('express');
const http = require('http');
const bodyParse = require('body-parser');
const cors = require('cors');
const scheduleController =
 require('./controllers/ScheduleController');
const roomController =
 require('./controllers/RoomController');
const historyController =
 require('./controllers/HistoryController');
const statsController =
 require('./controllers/StatsController');
const schedulerService =
 require('./services/SchedulerService');
const wsService = require('./services/WsService');
const databaseService =
 require('./services/DatabaseService');

const corsOptions = {
  origin: '*'
};

const app = express();
app.use(cors(corsOptions));
```

```
app.use(bodyParse.json());
app.use(express.static('public'))

const router = express.Router();
app.use('/api', router);

const db = databaseService.createConnection();

router.get('/schedules',
scheduleController.getSchedules(db));
router.get('/schedules/:scheduleId',
scheduleController.getSchedule(db));
router.post('/schedules',
scheduleController.addSchedule(db));
router.put('/schedules/:scheduleId',
scheduleController.editSchedule(db));
router.delete('/schedules/:scheduleId',
scheduleController.deleteSchedule(db));
router.post('/history', historyController.getHistory(db));
router.get('/history/filters',
historyController.getHistoryFilters(db));
router.get('/rooms', roomController.getRooms(db));
router.post('/stats', statsController.getStats(db));
router.get('/stats/filters',
statsController.getStatsFilters(db));

const server = http.createServer(app);

wsService.runWsServer(server, db);

schedulerService.runScheduler(db);

server.listen(constants.serverPort, () => {
  console.log(`Listening on port ${constants.serverPort}`);
});
```

Вище було розглянуто структуру клієнтської та серверної частин (тобто папки проекту «backend» та «frontend»). До загальної структури проекту також відноситься файл «Docker-compose.yml» (Рисунок 3.58).

Docker-compose.yml – це файл за допомогою якого ми запускаємо, «піднімаємо» всі контейнери проекту. Тобто тут ми визначаємо всі сервіси (контейнери). У моєму випадку файл містить три сервіси: backend, frontend, data base. Код файлу предоставлено у Лістингу 3.4.

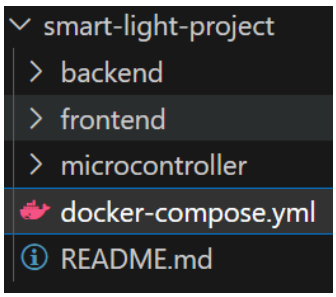


Рисунок 3.58 – Загальна структура проекту

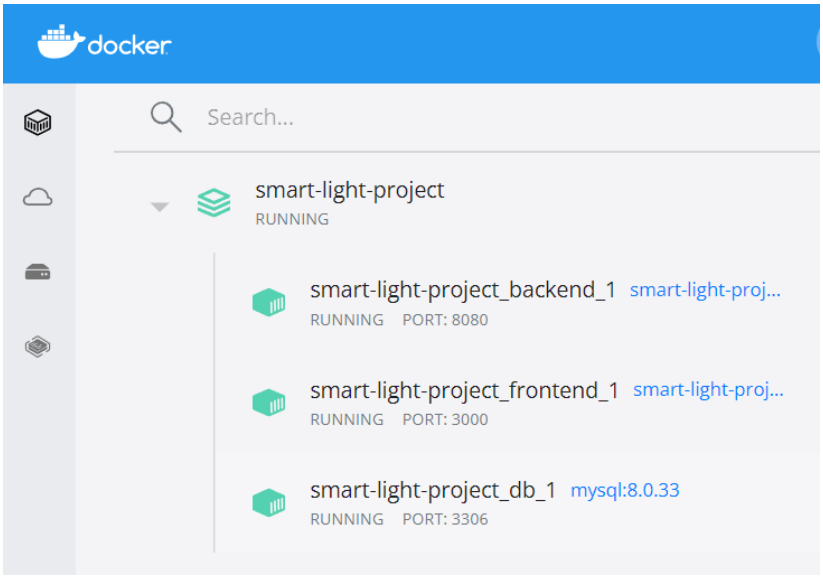


Рисунок 3.59 – Сервіси (контейнери) в докері

Лістинг 3.4 – Скрипт файлу «Docker-compose.yml»

```
version: '3.1'
services:
  frontend:
    build:
      dockerfile: Dockerfile.dev
      context: ./frontend
    restart: always
    environment:
      CHOKIDAR_USEPOLLING: 'true'
      WATCHPACK_POLLING: 'true'
      FAST_REFRESH: 'false'
      WDS_SOCKET_PORT: 3000
    volumes:
      - /app/node_modules
      - ./frontend:/app
    ports:
      - 3000:3000

  backend:
    build:
      dockerfile: Dockerfile.dev
      context: ./backend
```

```
      restart: on-failure
      volumes:
        - /app/node_modules
        - ./backend:/app
      depends_on:
        - db
      environment:
         SERVER_PORT: 8080
         DB_HOST: db
         DB_PORT: 3306
         DB_USER: root
         DB_PASS: root
         TZ: America/Chicago
      ports:
        - 8080:8080

    db:
      image: mysql:8.0.33
      command:
        --default-authentication-plugin=mysql_native_password
      restart: on-failure
      volumes:
        - ./backend/db/data:/var/lib/mysql
        -  ./backend/db/smart_light_db.sql:/docker-entrypoint-
initdb.d/smart_light_db.sql
      environment:
        MYSQL_ROOT_PASSWORD: root
        TZ: America/Chicago
      ports:
        - 3306:3306
```

Повертаємося до бази даних. Вона зберігається у докері (Docker). Папка «db», що знаходиться у серверній частині проекту, містить файл «smart_light_db.sql» (див. рисунок 3.57 на сторінці 115). Даний файл – це скрипт для створення структури бази даних, та наповнення даними таблиці «кімнати». Він виконується лише один раз – при першому запусці докер-контрейнеру для бази даних.

Лістинг 3.5 – Скрипт файлу «smart_light_db.sql»

```
CREATE DATABASE smart_light;
USE smart_light;

CREATE TABLE `rooms` (
  `room_id` int NOT NULL AUTO_INCREMENT,
  `room_name` varchar(100) NOT NULL,
  `light_status` tinyint(1) NOT NULL DEFAULT '1',
```

```
  `bulb_pin` int NOT NULL,
  PRIMARY KEY (`room_id`)
) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `rooms`
     VALUES (1,'Kitchen',1,14),
            (2,'Living room',1,27),
            (3,'Bathroom',1,26),
            (4,'Bedroom',1,25);

CREATE TABLE `history` (
  `history_id` int NOT NULL AUTO_INCREMENT,
  `room_id` int NOT NULL,
  `light_status` tinyint(1) NOT NULL,
  `scheduled_time` datetime NOT NULL,
  PRIMARY KEY (`history_id`),
  KEY `history_FK` (`room_id`),
  CONSTRAINT `history_FK` FOREIGN KEY (`room_id`)
REFERENCES `rooms` (`room_id`)
) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `scheduler` (
  `schedule_id` int NOT NULL AUTO_INCREMENT,
  `room_id` int NOT NULL,
  `light_status` tinyint(1) NOT NULL DEFAULT '1',
  `scheduled_time` datetime NOT NULL,
  `status` varchar(100) NOT NULL,
  PRIMARY KEY (`schedule_id`),
  KEY `scheduler_FK` (`room_id`),
  CONSTRAINT `scheduler_FK` FOREIGN KEY (`room_id`)
REFERENCES `rooms` (`room_id`)
) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Нижче на Рисунку 3.60 показано концептуальну схему бази даних, її ER-модель "сутність – зв'язок" (Entity-relationship model):

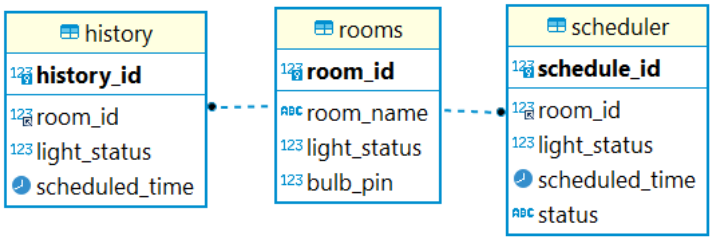


Рисунок 3.60 – ER-модель розробленої бази даних

Дані до таблиць баз даних можна додавати або за допомогою мови запитів SQL, або через графічний інтерфейс адміністрування систем управління базами даних (я використовую DBeaver). На рисунках 3.61 – 3.63 зображено таблиці, наповнені даними.

Properties | Data | ER Diagram

rooms | Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text

| | room_id | room_name | light_status | bulb_pin |
|---|---|---|---|---|
| 1 | 1 | Kitchen | 1 | 14 |
| 2 | 2 | Living room | 0 | 27 |
| 3 | 3 | Bathroom | 1 | 26 |
| 4 | 4 | Bedroom | 0 | 25 |

Рисунок 3.61 – Зміст таблиці «rooms»

Properties | Data | ER Diagram

scheduler | Enter a SQL expression to filter results (use Ctrl+Space)

| | schedule_id | room_id | light_status | scheduled_time | status |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 2023-05-12 00:38:00 | COMPLETED |
| 2 | 2 | 2 | 1 | 2023-05-12 16:03:00 | COMPLETED |
| 3 | 3 | 2 | 0 | 2023-05-17 13:15:00 | COMPLETED |
| 4 | 4 | 3 | 1 | 2023-05-21 19:00:00 | COMPLETED |
| 5 | 5 | 3 | 0 | 2023-05-21 21:00:00 | COMPLETED |
| 6 | 6 | 1 | 1 | 2023-05-23 07:00:00 | COMPLETED |

Рисунок 3.62 – Зміст таблиці «scheduler»

history | Enter a SQL expression to filter results (use Ctrl+Space)

| | history_id | room_id | light_status | scheduled_time |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2023-01-01 05:00:00 |
| 2 | 2 | 1 | 0 | 2023-01-01 07:00:00 |
| 3 | 3 | 2 | 1 | 2023-01-01 10:00:00 |
| 4 | 4 | 2 | 0 | 2023-01-01 15:00:00 |
| 5 | 5 | 3 | 1 | 2023-01-01 19:00:00 |

Рисунок 3.63 – Зміст таблиці «history»

Також для спрощення тестування та перевірки працездатності системи було розроблено емулятор, зовнішній вигляд якого показано на рисунку 3.64. Код емулятору (файл «index.html» що знаходиться у папці «backend > public > emulator») предоставлено у Лістингу 3.6 без урахування стилів.

За своєю суттю, емулятор – це інструмент, що дозволяє імітувати програмне або апаратне забезпечення на іншій платформі, дозволяючи

розробникам тестувати свої програми, не покладаючись на фактичне цільове апаратне чи програмне середовище.

У нашому випадку ми маємо систему розумного освітлення, що складається із 4х кімнат (тобто 4 лампочки). Якщо світло увімкнене – то лампочка має помаранчевий колір, якщо ввимкнене – то сірий.

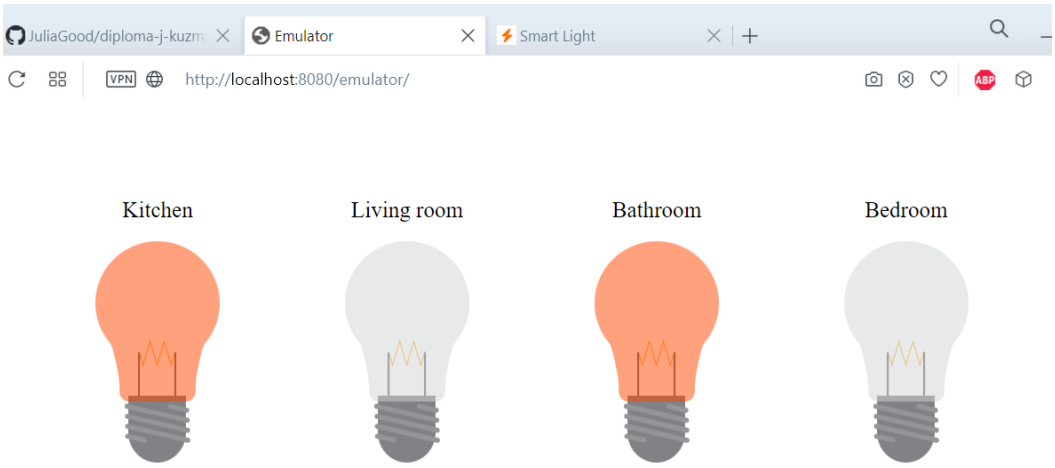


Рисунок 3.64 – Емулятор системи освітлення

Лістинг 3.6 – Скрипт емулятору

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport"
   content="width=device-width, initial-scale=1.0">
  <title>Emulator</title>
</head>

<body>
  <div id="rooms-render"></div>
</body>

<script>
  const socket = new WebSocket('ws://localhost:8080');

  function getBulb(color) {
    return `<svg>…</svg>`;
  }

  function getBulbDiv(room) {
    const isLightOn = room.lightStatus;
```

```
    const bulbColor = isLightOn ? '#ff4600' : '#d1d3d';

    return `<div class="room">
              <div class="room-name">${room.roomName}</div>
              <div class="room-bulb ${isLightOn ? "on":""}">
                  ${getBulb(bulbColor)}
              </div>
            </div>`;
  }

  const roomsDiv = document.getElementById('rooms-render');
  const bulbsDiv = {};

  socket.addEventListener('message', function (event) {
    console.log(event.data);

    const data = JSON.parse(event.data);

    if (data && Array.isArray(data.rooms)) {
      data.rooms.forEach((room) => {
        bulbsDiv[room.roomId] = getBulbDiv(room);
      });

    } else {
      const room = data;
      bulbsDiv[room.roomId] = getBulbDiv(room);
    }

    roomsDiv.innerHTML = `<div class="rooms">
      ${Object.values(bulbsDiv).join('')}
    </div>`;
  });
</script>

</html>
```

### 3.3 Налагодження апаратного комплексу системи

На рисунку 3.65 зображено принципову схему апаратного комплексу системи (схематично показано як мікроконтролер, реле модуль та 4 лампочки повинні бути підключені один до одного).

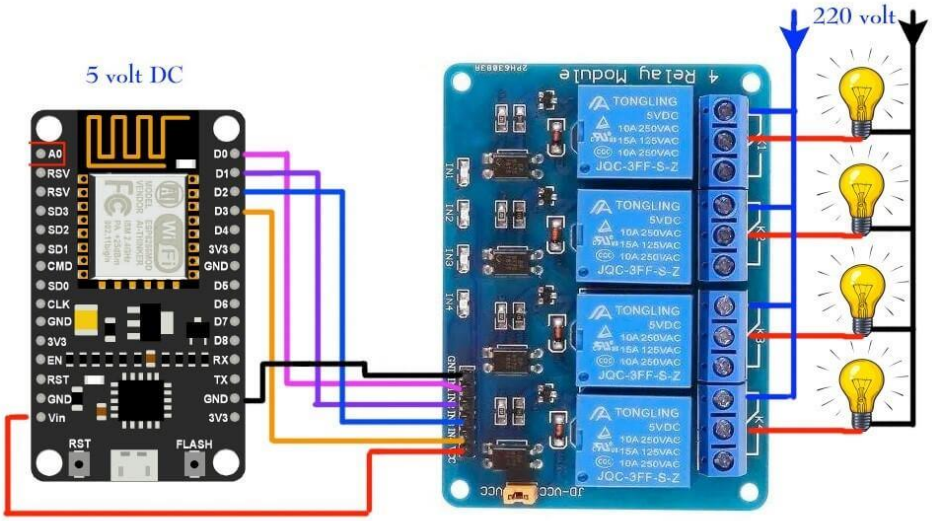


Рисунок 3.65 – Схема підключення апаратної частини системи

Використовуючи цю схему, підключаємо всі комплектуючі системи разом. Нам знадобиться: патрони для лампочок (light bulb socket/fixture), макетна плата (breadboard), чорний та червоний кабель, а точніше тонкі провода (red and black copper wiring), вилка (plug), з'єднувальні дроти «мама-тато» (male-female jumper wires), та безпосередньо сам мікроконтролер і мікро USB кабель (micro USB cable).

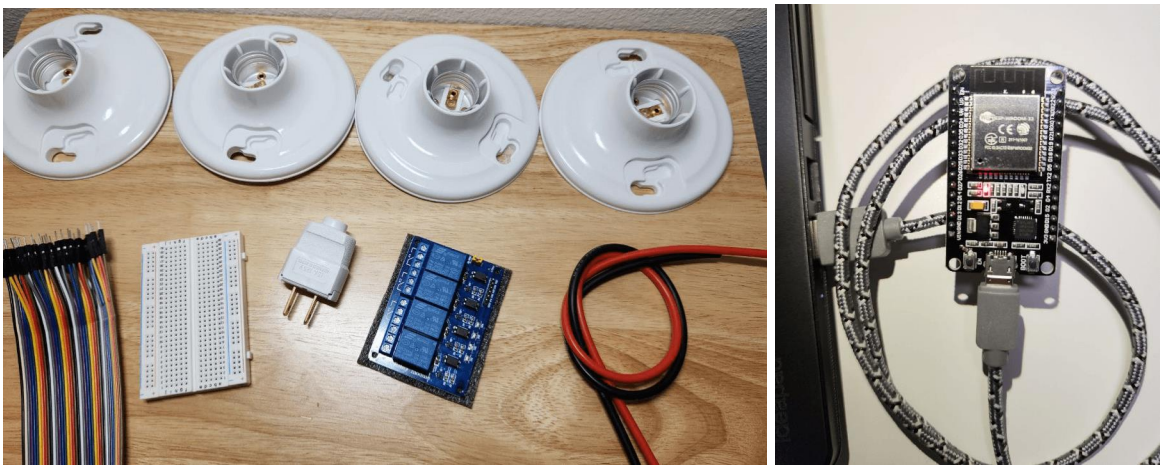

Рисунок 3.66 – Апаратні комплектуючі системи

Мікроконтролер встановлюється на макетну плату та підключається (з’єднується) до реле модулю за допомогою дротів «мама-тато». Червоний колір проводу використовуємо для живлення, а чорний – для заземлення. Підключаємо їх у відповідні для цього піни:

GND – від слова «ground», що означає «заземлення»;

VIN – від англ. «voltage input», у перекладі означає «вхідна напруга»;

VCC – англ. «voltage collector supply», «напруга живлення колектора».

Для ламчпочок у мікроконтролері було обрано наступні піни (контакти): D14, D27, D26, D25. Ці контакти підключаємо до відповідного вхідного керуючого сигналу реле модулю (IN – input control signal). Жовтий колір проводу – для першої лампочки, кімнати «кухня»; зелений провід – для другої лампочки, кімнати «living room»; синій – для ванної кімнати «bathroom»; фіолетовий – для кімнати «bedroom» (спальня).

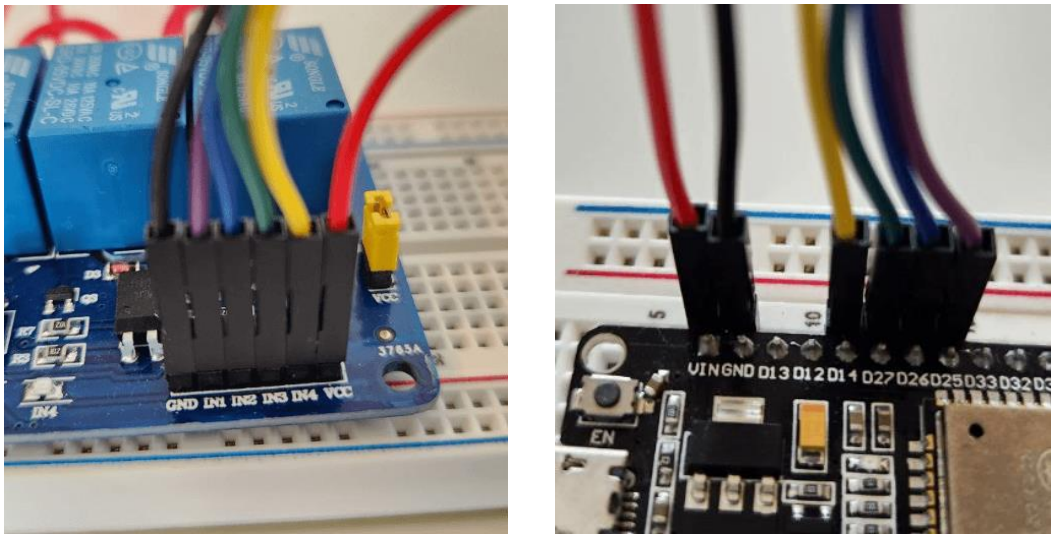


Рисунок 3.67 – Підключення мікроконтролеру до реле модулю

Використовуючи чорний та червоний кабель (тонкі провода) підключаємо реле модуль до вилки та патронів, у які будуть вкручуватися лампочки (по схемі що була зображена вище на рисунку 3.65). І потім сам мікроконтролер підключаємо до комп’ютеру за допомогою USB. Результат підключення усіх апаратних комплектуючих системи показано на рис. 3.69.

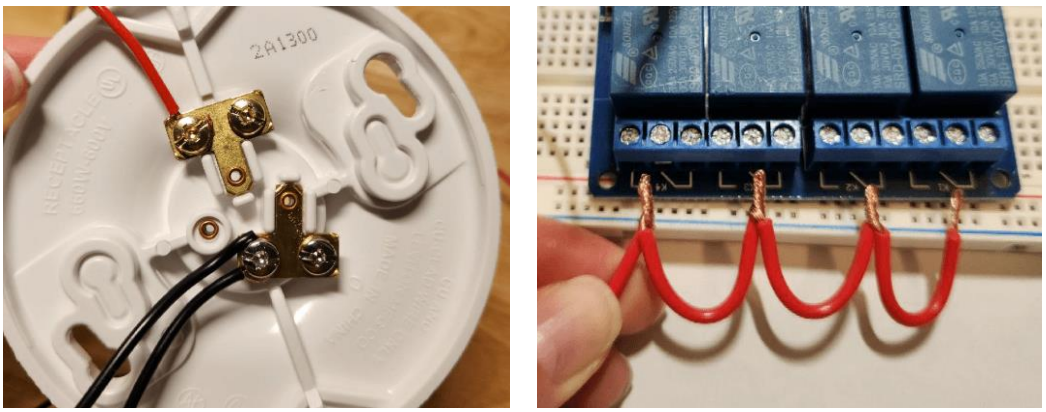

Рисунок 3.68 – Підключення патронів до реле модулю

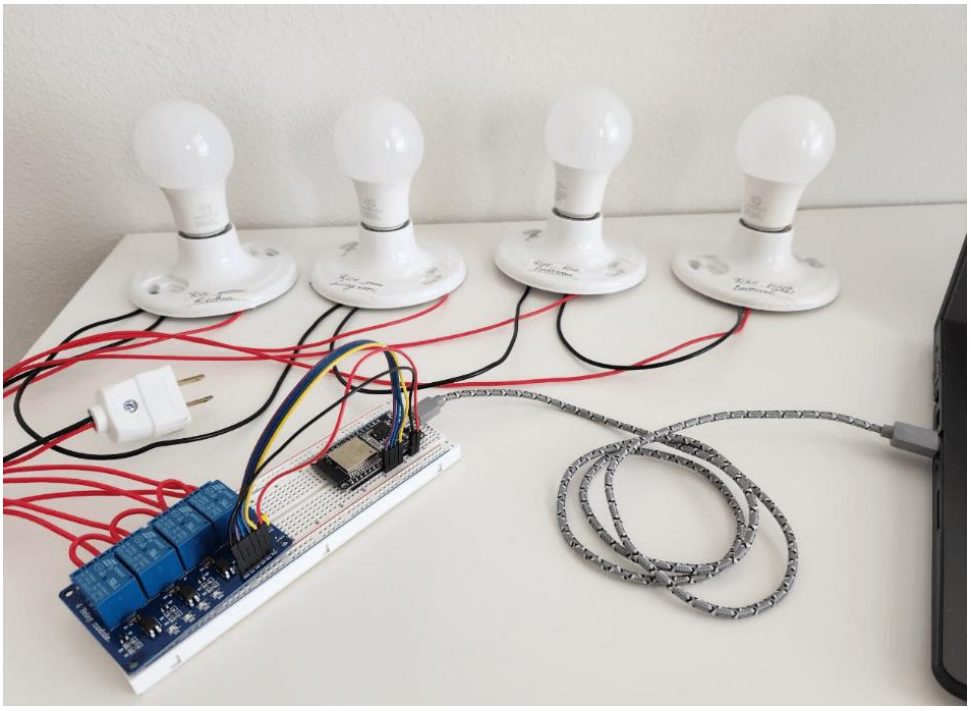

Рисунок 3.69 – Результат підключення всіх апаратних комплектуючих

### 3.3.1 Налаштування мікроконтролеру

Як вже було сказано у попередньому підрозділі, мікроконтролер підключаємо до комп’ютеру за допомогою micro-USB кабелю. Далі треба завантажити (скачати) спеціальні драйвери. Я завантажила їх з офіційного веб-сайту для OS Windows (посиланнння на веб-сайт знаходилося в мануальній інструкції, що входила до мікроконтролеру):

https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads

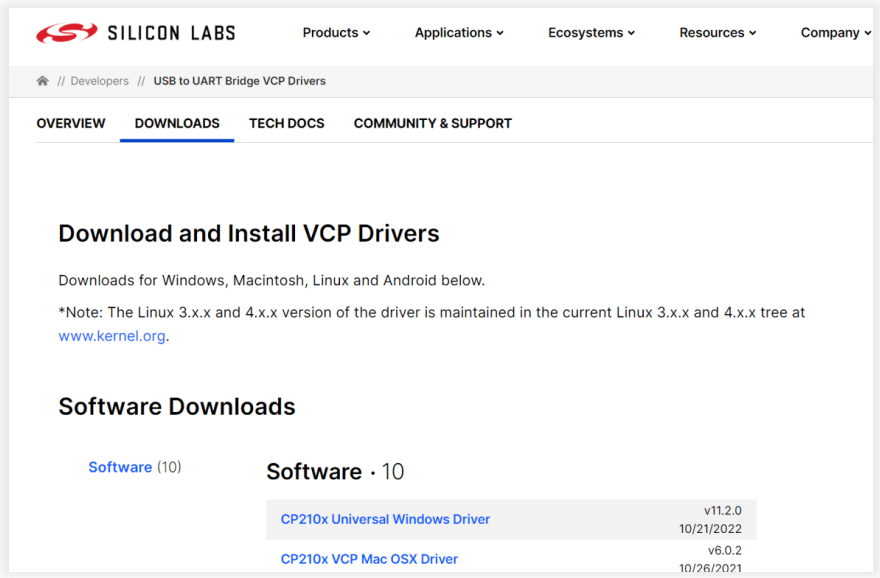


Рисунок 3.70 – Веб-сторінка для завантаження драйверів

Далі перейшла у Windows Device Manager (диспетчер пристроїв) та встановила ці драйвери (переглянула апаратне забезпечення комп’ютера).

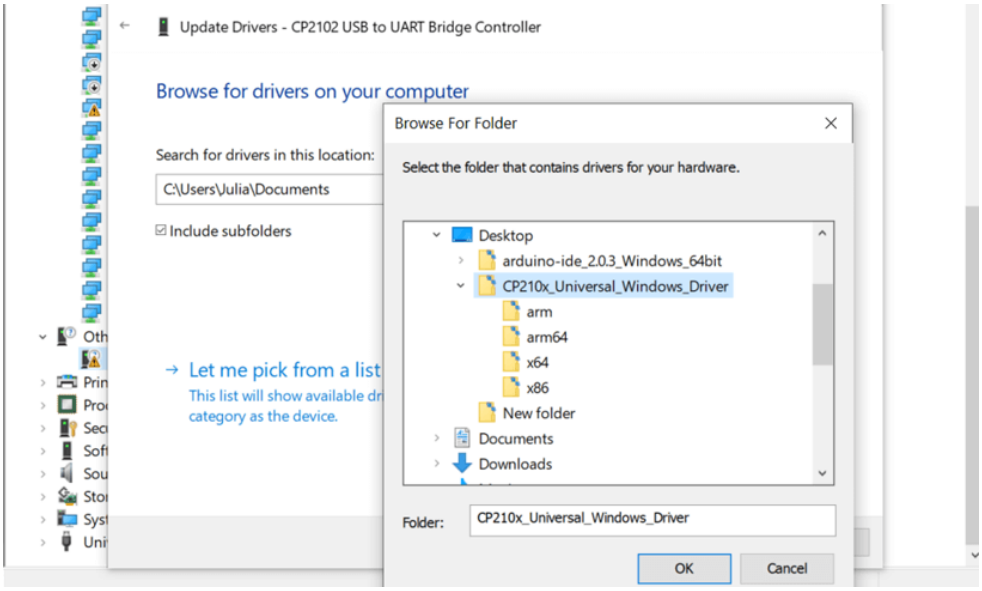


Рисунок 3.71 – Встановлення драйверів у Windows Device Manager

Переконалася, що драйвери встановилися та ПК бачить пристрій:

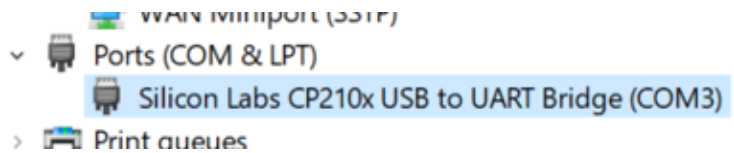


Рисунок 3.72 – Потрібні драйвери встановлені

Відкрила середовище розробки Arduino IDE, в налаштуваннях встановила ESP32 (менеджер плат), потім обрала потрібну плату та порт.

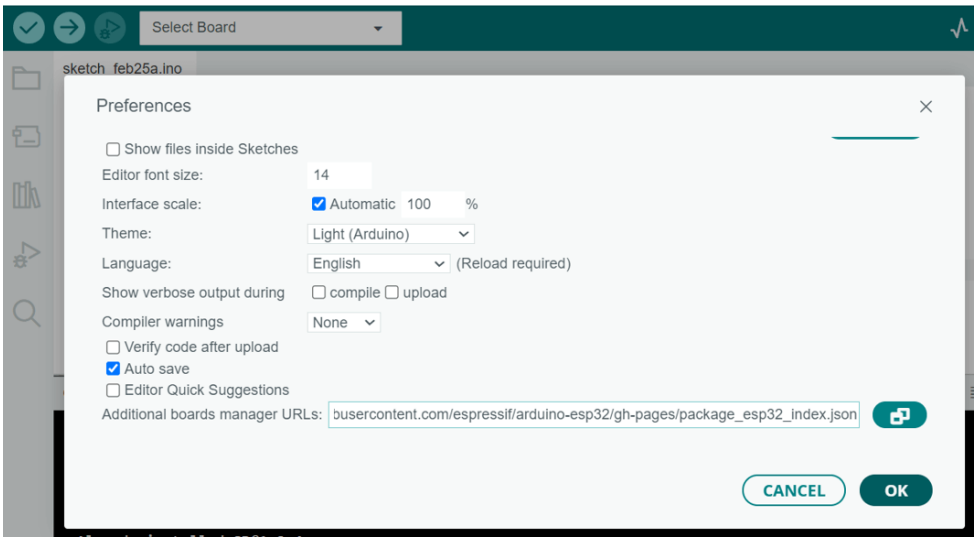


Рисунок 3.73 – Встановлення ESP32 у середовищі Arduino IDE

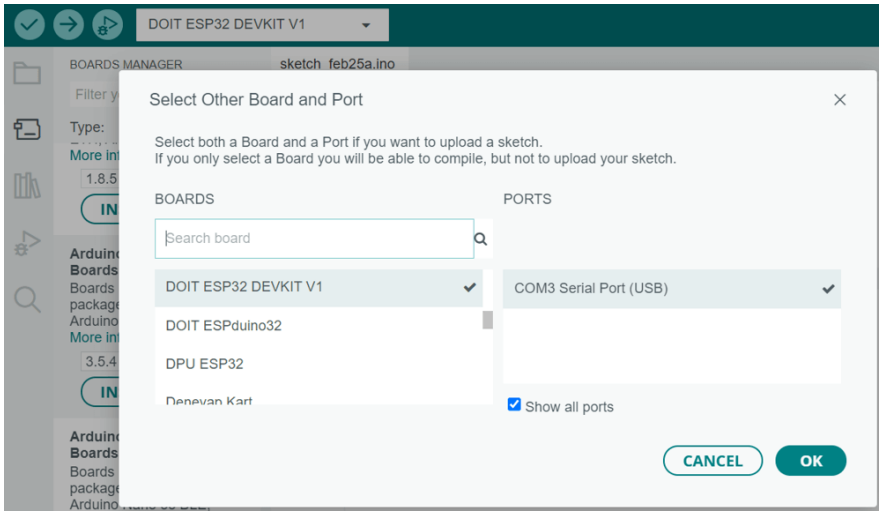


Рисунок 3.74 – Вибір плати розробки у середовищі Arduino IDE

Мікроконтролер встановлено та налаштовано. Переходимо до його програмування.

### 3.3.2 Програмування мікроконтролеру

Даний розділ поділяється на 2 частини: підключення мікроконтролеру до мережі Wi-Fi, та підключення мікроконтролеру до серверу і налаштування контактів. Повний вихідний код надано у Лістингу 3.9 (на сторінці 131).

3.3.2.1 Підключення мікроконтролеру до мережі Wi-Fi

Перед тим як програмувати мікрокотролер та підключати його до мережі Wi-Fi, пропоную розглянути режими роботи Wi-Fi мікроконтролеру ESP32.

Мікроконтролер ESP32 підтримує мережі Wi-Fi на частотах 2,4 ГГц і 5 ГГц і може працювати як точка доступу (Access Point), як станція (вузол мережі, Station), або і як станція і як точка доступу одночасно.

Щоб встановити режим Wi-Fi, треба використовувати функцію WiFi.mode() і як аргумент передати потрібний режим:

WiFi.mode(WIFI_STA) – режим «станція» (англ. station mode).

WiFi.mode(WIFI_AP) – режим «точка доступу» (англ. access point mode).

WiFi.mode(WIFI_AP_STA) – режим «точка доступу + станція».

Режим станції (рисунок 3.75): у цьому режимі ESP32 підключається до існуючої мережі Wi-Fi як клієнтський пристрій. ESP32 діє як клієнт Wi-Fi і може підключатися до маршрутизатора або точки доступу (AP). Він може отримати IP-адресу від маршрутизатора та отримати доступ до Інтернету або спілкуватися з іншими пристроями в мережі. Цей режим зазвичай використовується, коли ESP32 потрібно підключитися до існуючої мережі для доступу до ресурсів або зв’язку зі службами.

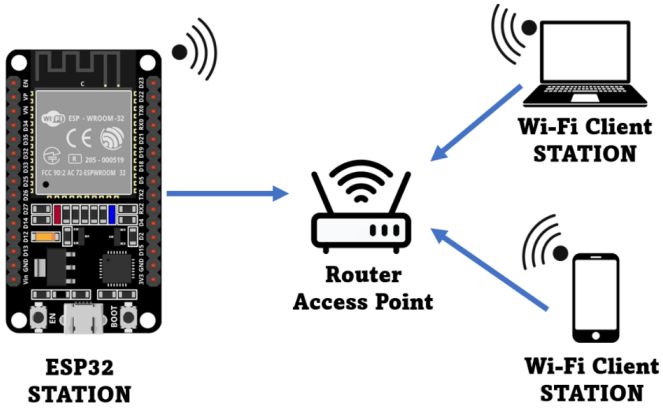


Рисунок 3.75 – ESP32 встановлений як станція

Режим точки доступу (рисунок 3.76): у цьому режимі ESP32 діє як точка доступу Wi-Fi. ESP32 створює власну мережу Wi-Fi, що дозволяє іншим пристроям (наприклад, смартфонам або комп'ютерам) підключатися до неї. ESP32 діє як маршрутизатор, надаючи IP-адреси підключеним пристроям. Цей режим корисний, коли треба створити локальну мережу, де пристрої можуть спілкуватися безпосередньо з ESP32.

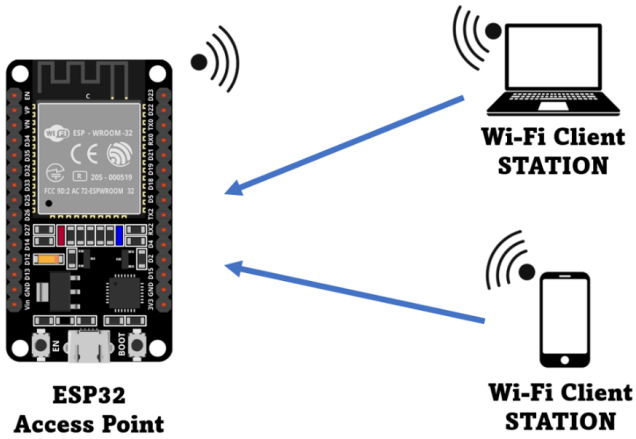


Рисунок 3.76 – ESP32 встановлений як точка доступу

Режим «Станція + точка доступу» (рисунок 3.77): Цей режим поєднує в собі одночасно функції точки доступу Wi-Fi і клієнта Wi-Fi. ESP32 діє і як точка доступу, що дозволяє іншим пристроям підключатися до нього, і як клієнт, одночасно підключаючись до існуючої мережі Wi-Fi. Цей режим корисний, якщо треба, щоб ESP32 діяв як міст між пристроями, підключеними до його точки доступу, та пристроями в існуючій мережі Wi-Fi.

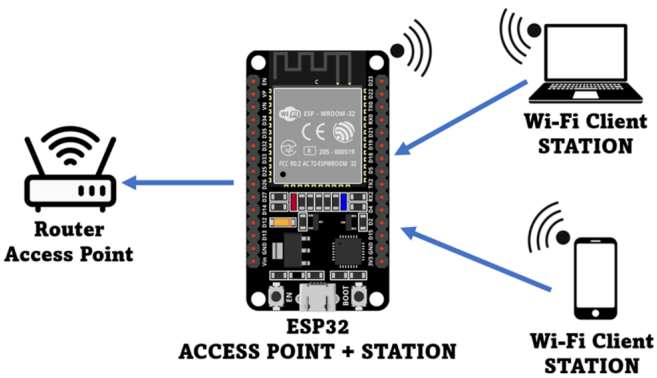


Рисунок 3.77 – ESP32 встановлений як станція + точка доступу

У даній дипломній роботі буде здійснюватися програмування мікроконтролеру ESP32, який зможе працювати як станція (вузол мережі), тому що він у нашій системі розумного освітлення виступає у ролі клієнту. То ж переходимо безпосередньо до його програмування.

Середовище розробки Arduino IDE містить різноманітні вбудовані приклади та ескізи для мікроконтролеру ESP32, доступ до яких можна отримати через меню «Файл» > «Приклади» ("File" > "Examples"). Ці приклади є чудовою відправною точкою для новачків, які вивчають, як використовувати ESP32 з Arduino IDE. Дані ескізи надають базове розуміння того, як працювати з контактами GPIO ESP32, можливостями Wi-Fi, Bluetooth і т.д. Користувачі можуть змінювати ці приклади відповідно до конкретних вимог проекту або створювати нові ескізи з нуля.

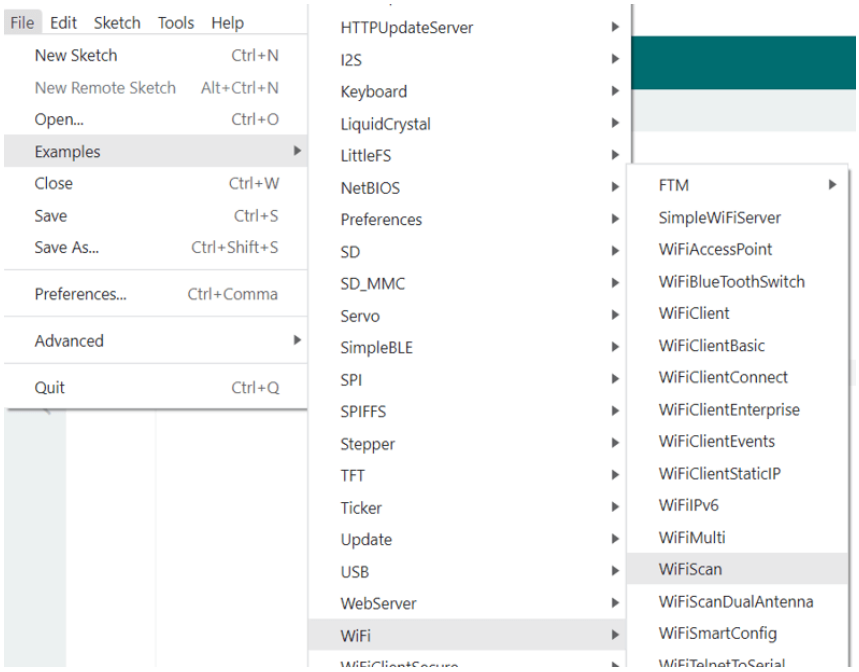


Рисунок 3.78 – Вбудовані приклади для ESP32 у середовищі Arduino IDE

Для підключення мікроконтролеру до мережі Wi-Fi, як станція, я буду використовувати ескіз WiFiScan. Адже перед тим як підключати мікроконтролер до мережі Wi-Fi, бажано «просканувати» мережу, щоб знайти свою мережу, свій роутер. Саме у прикладі WiFiScan показано, як робити

сканування мережі Wi-Fi поблизу та відображати інформацію про них на моніторі послідовного порту (Serial monitor).

Лістинг 3.7 – сканування та перегляд доступних WiFi мереж:

```
#include "WiFi.h"
void setup()
{
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);
    Serial.println("Setup done");
}
void loop()
{
    Serial.println("Scan start");

    // WiFi.scanNetworks returns the number of networks found.
    int n = WiFi.scanNetworks();
    Serial.println("Scan done");
    if (n == 0) {
        Serial.println("no networks found");
    } else {
        Serial.print(n);
        Serial.println(" networks found");
        Serial.println("Nr | SSID | RSSI | CH | Encryption");
        for (int i = 0; i < n; ++i) {
            // Print SSID and RSSI for each network found
            Serial.printf("%2d", i + 1);
            Serial.print(" | ");
            Serial.printf("%-32.32s", WiFi.SSID(i).c_str());
            Serial.print(" | ");
            Serial.printf("%4d", WiFi.RSSI(i));
            Serial.print(" | ");
            Serial.printf("%2d", WiFi.channel(i));
            Serial.print(" | ");
            switch (WiFi.encryptionType(i))
            {
            case WIFI_AUTH_OPEN:
                Serial.print("open");
                break;
            case WIFI_AUTH_WEP:
                Serial.print("WEP");
                break;
            case WIFI_AUTH_WPA_PSK:
                Serial.print("WPA");
                break;
            case WIFI_AUTH_WPA2_PSK:
                Serial.print("WPA2");
                break;
            case WIFI_AUTH_WPA_WPA2_PSK:
                Serial.print("WPA+WPA2");
                break;
            case WIFI_AUTH_WPA2_ENTERPRISE:
                Serial.print("WPA2-EAP");
```

```
                break;
            case WIFI_AUTH_WPA3_PSK:
                Serial.print("WPA3");
                break;
            case WIFI_AUTH_WPA2_WPA3_PSK:
                Serial.print("WPA2+WPA3");
                break;
            case WIFI_AUTH_WAPI_PSK:
                Serial.print("WAPI");
                break;
            default:
                Serial.print("unknown");
            }
            Serial.println();
            delay(10);
        }
    }
    Serial.println("");
    // Delete the scan result to free memory for code below.
    WiFi.scanDelete();
    // Wait a bit before scanning again.
    delay(5000);
}
```

Натискаю кнопку "Завантаження" ("Upload") та чекаю декілька секунд, поки код компілюється та завантажується у плату. Відкриваю вікно Serial Monitor (послідовного COM-порту) та налаштовую швидкість передачі даних на 115 200 бод. І далі на самій платі розробки (ESP32 DEVKIT v1) натискаю кнопку «EN» (Enable), і після чого бачу мережі, доступні для ESP32. На Рисунку 3.79 показано список мереж, серед яких моєю є «VK-WIFI».


Рисунок 3.79 – Список мереж доступних для ESP32

Підключити мікроконтролер ESP32 до певної мережі Wi-Fi як вузол мережі (режим «станція») досить просто: треба знати її SSID (тобто назву, Service Set Identifier) та пароль. Ці дані треба передати у якості аргументів у функцію WiFi.begin("SSID", "Password").

Лістинг 3.8 – Підключення мікроконтролеру до мережі Wi-Fi

```
#include <WiFi.h>
const char* ssid     = "VK-WIFI";
const char* password = "wifi_password";

void setup(){
    Serial.begin(115200);
    delay(10);
    // We start by connecting to a WiFi network
    WiFi.mode(WIFI_STA);
    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected.");
}
```

У коді створюю змінні (константи) ssid і password, що містять назву і пароль мережі, до якої я хочу підключитися. Потім, щоб підключитися до мережі, користуюся вбудованою функцією WiFi.begin(), в яку передаю аргументи SSID мережі та її пароль. Коли ESP32 підключиться до мережі Wi-Fi, то на Serial Monitor буде відображатися повідомлення «Wi-Fi connected», що в перекладі означає «Wi-Fi підключено».

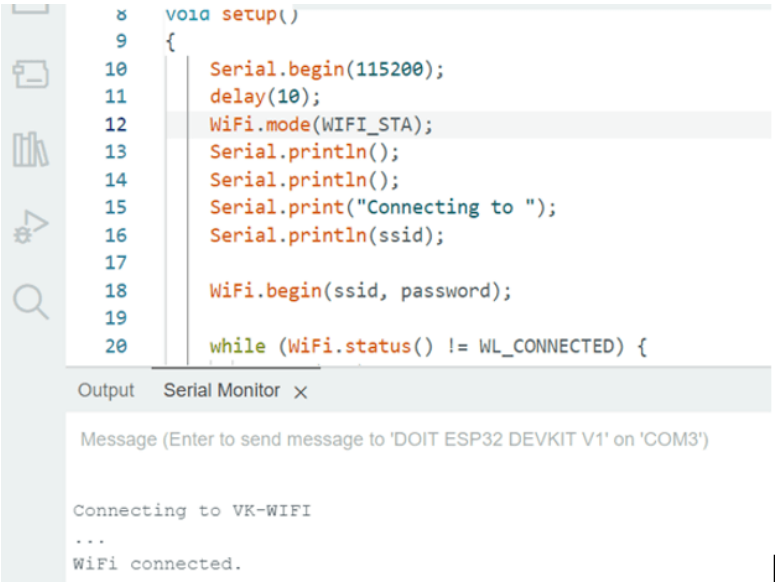


Рисунок 3.80 – Підключення ESP32 до мережі Wi-Fi

3.3.2.2 Підключення мікроконтролеру до серверу та налаштування контактів плати

Для підключення мікроконтролеру до серверу, я буду використовувати популярну бібліотеку <WebSocketsClient.h>, яка забезпечує функціональність WebSocket клієнта, та дозволяє мікроконтролеру встановлювати WebSocket з'єднання і спілкуватися з WebSocket серверами (тобто дозволяє встановлювати з'єднання з сервером через WebSocket протокол).

Щоб почати використовувати бібліотеку, треба створити екземпляр класу WebSocketsClient. (У моєму коді це `WebSocketsClient webSocket;`).

Для обробки WebSocket подій треба застосовувати обробники подій (event handlers), такі як webSocket.onOpen(), webSocket.onMessage(), webSocket.onClose() і webSocket.onError(). Ці обробники дозволяють визначати поведінку під час встановлення з'єднання WebSocket, отримання повідомлень, закриття з'єднання та виникнення помилок відповідно. Для обробки подій треба викликати webSocket.loop() у функції loop(). Для ініціалізації та запуску клієнта WebSocket використовується функція webSocket.begin(). А щоб надсилати повідомлення на сервер, треба використовати функцію webSocket.send() або webSocket.sendPing().

Щоб керувати зовнішніми пристроями, підключеними до

мікроконтролеру (у нашому випадку це лампочки), та маніпулювати рівнем напруги зазначеного контакту на платі, треба використоввати вбудовані функції pinMode() та digitalWrite() відповідно.

Функція pinMode() використовується для налаштування режиму певного контакту (піну, від англ. «pin») на платі, вказуючи, чи буде він використовуватися як вхід чи вихід. Функція приймає 2 параметри: pin і mode.

1) pin: номер контакту, який потрібно налаштувати. Це можна вказати як ціле число (наприклад, 14, 27, 26, 25) або за допомогою попередньо визначених міток контактів (predefined pin labels). У моєму коді це «`bulbPin`».

2) mode: режим, який потрібно встановити для піну (для контакту). Це може бути одна з двох констант:

INPUT: налаштовує контакт як вхід, що дозволяє зчитувати рівень напруги, що подається на контакт.

OUTPUT: конфігурує контакт як вихід, що дозволяє встановити рівень напруги для керування зовнішніми пристроями, підключеними до контакту.

Функція digitalWrite() використовується для встановлення рівня напруги певного контакту на платі. Функція приймає 2 параметри: pin і value.

1) pin: номер контакту, яким треба керувати. Вказується як ціле число або за допомогою попередньо визначених позначок (predefined pin labels).

2) value: рівень напруги, який потрібно подати на контакт. Це може бути одна з двох констант:

HIGH: встановлює для контакту логічний рівень напруги HIGH (високий), зазвичай 5 В на більшості плат.

LOW: встановлює для контакту логічний рівень напруги LOW (низький), зазвичай 0 В або заземлення (ground).

Нижче у Лістингу 3.9 надано повний вихідний код мікроконтролеру.

Лістинг 3.9 – Повний програмний код мікроконтролеру

```
#include <WiFi.h>
#include <WebSocketsClient.h>
#include <ArduinoJson.h>
```

```
const char* ssid = "VK-WIFI";
const char* password = "wifi-password";

// Initialize the WebSocket client
WebSocketsClient webSocket;

void handleWebSocketEvent(WStype_t type, uint8_t* payload, size_t
length) {
  switch (type) {
    case WStype_DISCONNECTED:
      Serial.println("Disconnected from websockets server");
      break;
    case WStype_CONNECTED:
      Serial.println("Connected to websockets server");
      break;
    case WStype_TEXT:
      Serial.print("Received message: ");
      Serial.println((char*)payload);

      // Parse the JSON message
      DynamicJsonDocument json(1024);
      DeserializationError error = deserializeJson(json,
payload);
      if (error) {
        Serial.print(F("deserializeJson() failed: "));
        Serial.println(error.c_str());
        return;
      }
      // Check if this is a first connection.
      // In this case we should consider the payload as an array
in order to update the initial status of the bulbs
      bool isFirstConnection =
json["firstConnection"].as<bool>();

      if (isFirstConnection) {
        // Iterate through the array and update the pins
        for (JsonVariant value : json["rooms"].as<JsonArray>()) {
          // Update the pinMode, analogWrite, and digitalWrite
values for the current pin
          updateBulb(value.as<JsonObject>());
```

```
            }
          } else {
            // Handle a non-array payload
            // Update the pinMode, analogWrite, and digitalWrite
values for the single pin
            updateBulb(json.as<JsonObject>());
          }
          break;
      }
    }
    void updateBulb(JsonObject data) {
      int bulbPin = data["bulbPin"].as<int>();
      int lightStatus = data["lightStatus"].as<int>();

      if (bulbPin > 0) {
        pinMode(bulbPin, OUTPUT);
        Serial.print("bulbPin: ");
        Serial.println(bulbPin);

        digitalWrite(bulbPin, lightStatus ? LOW : HIGH);
        Serial.print("lightStatus: ");
        Serial.println(lightStatus);
      }
    }
    void setup() {
      // Initialize the serial port
      Serial.begin(115200);

      // Connect to the Wi-Fi network
      WiFi.begin(ssid, password);
      while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to Wi-Fi...");
      }
      Serial.println("Connected to Wi-Fi");

      // Connect to the remote WebSocket server
      webSocket.begin("192.168.1.178", 8080, "/", "ws");
      webSocket.onEvent(handleWebSocketEvent);
      Serial.println("WebSocket client started");
    }
```

```
    void loop() {
      // Handle incoming WebSocket messages
      webSocket.loop();

      // Send a ping message to the server every 10 seconds to keep
the connection alive
      static unsigned long pingTime = millis();
      if (millis() - pingTime > 10000) {
        webSocket.sendPing();
        pingTime = millis();
      }
    }
```

## 3.4 Висновки за розділом

У даному розділі було успішно реалізовано інтелектуальну систему освітлення. Зокрема було виконано проектування та моделювання системи, зроблено та детально описано концептуальну (загальну) схему роботи системи, побудовано UML діаграми, зображено принципову схему апаратного комплексу системи. Здійснено програмну реалізацію проекту: створено інтерфейс користувача, запрограмовано серверну частину веб-додатку, розроблено базу даних та наповнено її даними. Було налагоджено апаратний комплекс системи: з’єднано усі комплектуючі системи разом, запрограмовано мікроконтролер, зокрема підключено його до мережі Wi-Fi, встановлено з’єднання з сервером, здійснено налаштування контактів плати. Також було проведено тестування системи. Для спрощення тестування та перевірки працездатності системи було розроблено емулятор.

# ВИСНОВКИ

У процесі виконання та написання кваліїікаційної бакалаврської роботи було здійснено детальний огляд предметної області. Він виявив доцільність застосування автоматичного керування освітленням через можливість енергозаощадження, економічної вигоди та зручності для користувачів.

Зібрано та проаналізовано інформацію про інтелектуальні системи, зокрема про системи розумного будинку та підсистеми освітлення. Виявлено, що розробка власної інтелектуальної системи освітлення є доцільною, а тема актуальною. Було визначено основні вимоги та функції системи, а саме розробка веб-застосунку на базі тришарової клієнт-серверної архітектури з можливістю моніторингу стану та керуванням пристроями освітлення через мікроконтролерну систему у локальній бездротовій мережі.

Програмно-апаратний комплекс було запроєктовано, запрограмовано та успішно реалізовано, що у підсумку ми маємо систему з інтерфейсом користувача у веб-застосунку та можливістю керування IoT пристроями на прикладі підсистеми освітлення.

Даний програмно-апаратний продукт може використовуватися не лише для власного будинку. ЇЇ можна впроваджувати у громадські, навчальні, адміністративні і виробничі приміщення. Впровадження інтелектуальної системи освітлення значно впливає на енергозбереження. Застосовуючи енергозберігаючі методи за допомогою системи розумного освітлення, можна досягти значної економії енергії за рахунок зменшення непотрібного використання освітлення та оптимізації споживання енергії. А це не тільки зменшує рахунки за електроенергію, але й знижує загальний попит на енергоресурси.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kerry James Hinton, Robert Ayre, Jeffrey Cheong: Modeling the Power Consumption and Energy Efficiency of Telecommunications Networks, 2021.

2. Angelo Baggini, Andreas Sumper: Electrical Energy Efficiency, Technologies and Applications, Edition 1. – 2012.

3. Miller Michael, Internet of Things, The: How Smart TVs, Smart Cars, Smart Homes, and Smart Cities Are Changing the World. 1st Edition, 2015

4. J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, "Internet of Things (loT): A vision architecctural elements and future directions", Future Generation Computer Systems, 2013. – 45 p.

5. N. P. Gopalan, t. A. Adikesavan: Web technology. A developer’s perspective. – 348 p. – 2014.

6. Adeel Javed: Building Arduino Projects for the Internet of Things: Experiments with Real-World Applications.

7. Поджаренко В.О., Кучерук В.Ю., Севастьянов В.М. Основи мікропроцесорної техніки. Навчальний посібник. Вінниця: ВНТУ, 2006. 226 с.

8. Програмування мікроконтролерів систем автоматики: конспект лекцій для студентів базового напряму 050201 “Системна інженерія” / Укл.: А.Г. Павельчак, В.В. Самотий, Ю.В. Яцук – Львів: Львівська політехніка. – 2012. – 143 с.

9. Таненбаум Э. Компьютерные сети. [Текст] / Э. Таненбаум, Д. Уэзеролл –СПб.: Питер, 2014. – 870 с.

10. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы. Учебник для ВУЗов (изд: 6) [Текст] / В.Г. Олифер, Н.А. Олифер – СПб.: Питер, 2017. – 960 с.

11. Jeremy Blum. Exploring Arduino: Tools and Techniques for Engineering Wizardry. – 2013.

12. Tim Oxley, Nathan Rajlich, TJ Holowaychuk, Alex Young: Node.js in Action. – 392 p, 2017.

13. Ethan Brown: Web Development with Node and Express: Leveraging the JavaScript Stack 2nd Edition, O'Reilly Media, Inc. 2014.

14. Evan Hahn: Express in Action. Writing, Building, and Testing Node.js Applications. – 256 p. – 2016.

15. Alex Banks, Eve Porcello: Learning React. Functional Web Development with React and Redux. – 350 p. – 2017.

16. Alan Beaulieu, Learning SQL, 3rd Edition, O'Reilly Media, Inc. 2020

17. Seyed M.M. (Saied) Tahaghoghi, Hugh Williams, Learning MySQL: Get a Handle on Your Data, 2006

# ДОДАТКИ

## Додаток А Вихідний код програми

```
// file backend/app/index.js
const constants = require('./constants')
const express = require('express');
const http = require('http');
const bodyParse = require('body-parser');
const cors = require('cors');
const scheduleController = require('./controllers/ScheduleController');
const roomController = require('./controllers/RoomController');
const historyController = require('./controllers/HistoryController');
const statsController = require('./controllers/StatsController');
const schedulerService = require('./services/SchedulerService');
const wsService = require('./services/WsService');
const databaseService = require('./services/DatabaseService');

const corsOptions = {
  origin: '*'
};

const app = express();
app.use(cors(corsOptions));
app.use(bodyParse.json());
app.use(express.static('public'))

const router = express.Router();
app.use('/api', router);

// Create DB connection
const db = databaseService.createConnection();

// Controllers
router.get('/schedules', scheduleController.getSchedules(db));
router.get('/schedules/:scheduleId',
scheduleController.getSchedule(db));
router.post('/schedules', scheduleController.addSchedule(db));
router.put('/schedules/:scheduleId',
scheduleController.editSchedule(db));
router.delete('/schedules/:scheduleId',
scheduleController.deleteSchedule(db));
router.post('/history', historyController.getHistory(db));
```