Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)
Кафедра обчислювальної техніки
(повна назва кафедри)

Пояснювальна записка

до бакалаврського дипломного проекту
бакалавра
(освітньо-кваліфікаційний рівень)

на тему «Мікропроцесорна система Smart-годинника»

Виконав: студент 4 курсу, групи 1 СП-20б
спеціальності 123 — Комп'ютерна інженерія
Івахов П.В.
Керівник к.т.н., доц. каф. ОТ
Кадук О.В..
"12" 06 2024р.
Рецензент, к.т.н.,доц. каф. МБІС
Шиян А.А.
"19" 06 2024р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«20» 06 2024р.

ВНТУ – 2024 року

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 123 – «Комп'ютерна інженерія»
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри
обчислювальної техніки
проф., д.т.н. О. Д. Азаров
«6» лютого 2024 р.

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Івахову Павлу Віталійовичу

1 Тема проекту «Мікропроцесорна система Smart-годинника»

Керівник проекту Кадук Олександр Володимирович, к.т.н., доцент, затверджені наказом вищого навчального закладу від «11» березня 2024 р. №80

2 Строк подання студентом проекту 07.06.20.24 р.

3 Вихідні дані до проекту: технічні параметри мікропроцесорної системи Smart-годинника.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) вступ, аналіз системи Smart-годинника як об'єктів керування, вибір елементної бази для реалізації системи Smart-годинника, покращення функціональних можливостей та підвищення ефективності системи Smart-годинника, висновки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): системи фіксації часу перебування, модульність структури Spring, функціональна схема, діаграма взаємодії модулів.

6 Консультанти розділів роботи приведені в таблиці 1

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1– 3	Кадук О.В., к.т.н., доцент кафедри ОТ		

7. Дата видачі завдання 12.03.2024 р

Таблиця 2 — Календарний план

з/п	Назва етапів виконання бакалаврського проекту	Строк виконання етапів роботи	Примітка
1	Постановка задач проекту	27.02.24	вик
2	Огляд інформаційних джерел	06.03–20.03.24	вик
3	Аналіз системи системи Smart-годинника як об'єкта керування	27.03–7.04.24	вик
4	Вибір елементної бази для реалізації системи Smart-годинника	10.04–21.04.24	вик
5	Покращення функціональних можливостей та підвищення ефективності системи Smart-годинника	24.04–12.05.24	вик
6	Оформлення пояснювальної записки та ілюстративного матеріалу	15.05–26.05.24	вик
7	Аналіз виконання проекту. Висновки. Додатки	29.05–09.06.24	вик
8	Перевірка якості виконання бакалаврського проекту та усунення недоліків	06.06.24	вик

Студент Івахов П.В.
(підпис) (прізвище та ініціали)

Керівник роботи О.В. Кадук

АНОТАЦІЯ

УДК 004.9

Івахов П.В.

Мікропроцесорна система Smart-годинника. Бакалаврська дипломна робота зі спеціальності 123 — комп’ютерна інженерія, освітня програма — системне програмування. Вінниця: ВНТУ, 2024, 84 с.

На укр.мові. Бібліогр.: 16 назв, рис. 25, табл.3.

У проекті розглянуто створення мікропроцесорної системи smart-годинника, що забезпечить облік робочих годин працівників на підприємстві. Після аналізу об'єкта управління було прийнято рішення про поділ його на дві підсистеми. Кожна підсистема виконує своє завдання, спрямоване на реалізацію загальної функції системи обліку часу входу-виходу працівників з робочого місця.

Система повинна складатися з двох частин: термінала і сервера, що знизить навантаження на менш потужний термінал і зробить систему гнучкішою.

Ключові слова: мікропроцесорна система, smart-годинник, сервер, термінал.

ANNOTATION

Smart watch microprocessor system. Bachelor diploma project in the specialty 123 — computer engineering, educational program system programming. Vinnitsa, VTNU, 2024, 84 p.

In the Ukr.leng. Libr.name 18, figure 35, table 3

The project envisages the creation of a microprocessor system of a smart watch, which will ensure the accounting of the working hours of the company's employees. After analyzing the control object, a decision was made to divide it into two subsystems. Each subsystem performs its task, aimed at implementing the general function of the system for recording the time of entry and exit of employees from the workplace.

The system should consist of two parts: a terminal and a server, which will reduce the load on a less powerful terminal and make the system more flexible.

Keywords: microprocessor system, smart watch, server, terminal.

ЗМІСТ

ВСТУП

Актуальність проекту зумовлена необхідністю вивчення систем реєстрації та обліку доступу до різних виробничих приміщень. Такі системи мають безліч переваг. Вони не тільки точно фіксують робочий час для роботодавців, але й зберігають дані для кадрової служби, які можуть використовуватися під час розрахунку заробітної плати. Для працівників впровадження таких систем означає, що вони матимуть чітке уявлення про свій робочий день: час початку, закінчення та перерв. Якщо працівник працює понаднормово, роботодавець повинен забезпечити справедливу компенсацію. Чітке розмежування робочого часу забезпечує прозорість графіку та оплати праці, включаючи всі додаткові години.

Оптимізована система обліку часу дозволяє працівникам легко та ефективно реєструвати години перерв. Ручний облік перерв може спричинити плутанину, але автоматизована система документує вхід і вихід працівника, а також тривалість перерв. Це зменшує непорозуміння та дозволяє працівникам краще керувати своїм розкладом. Для роботодавців це також зручне рішення для моніторингу робочого часу без зайвих проблем.

Час є обмеженим ресурсом для будь-якої організації. Сучасні системи інтегруються з мобільними пристроями, збираючи та відстежуючи дані про вхід та вихід, повідомляючи керівництво про розбіжності. Багато програм мають функції для виявлення недоліків у відвідуваності, що допомагає керівництву вдосконалювати політики компанії.

У проекті буде розглянуто створення мікропроцесорної системи smart-годинника, що забезпечить облік робочих годин працівників на підприємстві. Після аналізу об'єкта управління було прийнято рішення про поділ його на дві підсистеми. Кожна підсистема виконує своє завдання, спрямоване на реалізацію загальної функції системи обліку часу входу-виходу працівників з робочого місця.

Мета створення системи полягає в забезпеченні реєстрації і аналізу часу входу і виходу працівників, обліку понаднормових, недопрацьованих та вихідних годин, що дозволить правильно нараховувати заробітну плату. Система повинна складатися з двох частин: термінала і сервера, що знизить навантаження на менш потужний термінал і зробить систему гнучкішою.

Для досягнення поставленої мети було вирішено такі завдання:

— проаналізувати загальна характеристику систем smart-годинника як об'єкта керування;

— обрати та обґрунтувати вибір мови програмування;

— обрати та обґрунтувати вибір бази даних для зберігання даних;

— обрати елементну базу для реалізації системи smart-годинника;

— обрати та обґрунтувати вибір хостингу;

— реалізувати метод обробки сигналів;

— промоделюватисистеми радіочастотної ідентифікації для smart-годинника.

В даному проекті була розроблена мікропроцесорна система Smart-годинника для обліку робочих годин працівників. У рамках проекту було створено серверну частину та функціональну схему терміналу, яка детально описує його особливості. Проведено аналіз існуючих систем обліку робочих годин та їх класифікацію, що дозволило вибрати оптимальні компоненти для проекту.

Основні компоненти серверної частини були обрані з урахуванням мови програмування, вебфреймворку та бази даних, що надійно підтримують вимоги проекту.

Для апаратної частини терміналу були детально обґрунтовані технічні характеристики, враховуючи фактори, такі як розмір, потужність, умови експлуатації, шум, вібрація та енергоспоживання.

Процес хостингу був організований з використанням AWS, що забезпечило надійність та безперебійну роботу системи.

Для зберігання та обробки даних був розроблений метод з використанням RFID тегів, що дозволив ефективно використовувати пам'ять.

Аналіз технологічного процесу та інформаційних алгоритмів показав необхідність вдосконалення системи обліку, що було враховано у проекті. Застосування математичного комплексу програмного забезпечення для обробки сигналів RFID дозволило знизити обсяг пам'яті та підвищити ефективність системи.

Розроблена система забезпечує надійну та якісну роботу, зокрема шляхом оперативного реагування на різні сценарії роботи та мінімізації навантаження на мережу. Це сприяє підвищенню надійності та забезпечує ефективне управління робочими процесами.

Об’єктом дослідження є робота комп'ютерно-інтегрованої системи обліку на підприємстві, принципи регулювання обліку та основні компоненти системи.

Предметом дослідження є засоби та компоненти автоматизованої системи контролю при необхідних параметрах ефективної роботи.

Практичне значення одержаних результатів полягає у запропонованих практичних рекомендаціях щодо проектування систем автоматизованого управління, що дозволяє підвищити їх надійність та ефективність.

1 АНАЛІЗ СИСТЕМИ SMART-ГОДИННИКА ЯК ОБ'ЄКТА КЕРУВАННЯ

1.1 Загальна характеристика системи

По-суті, в нашому випадку, мікропроцесорна система Smart-годинника призначена для обліку робочих годин, а її мета полягає в електронному контролі та реєстрації присутності працівників для точного нарахування заробітної плати. Ця система дозволяє компанії створювати детальні звіти, які містять інформацію про відпрацьовані години, відгул, понаднормову роботу, відсутності тощо. Це сприяє автоматизації розрахунку зарплат та забезпечує дотримання трудового законодавства. Система фіксує робочі години та понаднормовий час для ефективного моніторингу діяльності співробітників. На сьогоднішній день такі системи можна класифікувати наступним чином (рисунок 1.1):

— з використанням перфокарти;

— з біометричною ідентифікацією;

— цифрові системи;

— веб-системи;

— мобільні системи.

Системи з використанням перфокарти вважаються застарілими. Працівники отримують паперову картку обліку робочого часу, яку вони вставляють у спеціальний пристрій при вході та виході з робочого місця. Пристрій записує або перфорує картку, фіксуючи час початку та закінчення роботи. Ця система не підходить для віддалених працівників.

Системи з біометричною ідентифікацією використовують апаратні пристрої, зазвичай розташовані біля входу в офіс, для підтвердження особи за допомогою біометричних даних, таких як відбитки пальців, долонь, розпізнавання обличчя тощо, для реєстрації входу та виходу.

Цифрові системи застосовують картки з магнітною смугою для реєстрації дати та часу входу і виходу. Альтернативно, ці системи можуть

використовувати сенсорні панелі, де для реєстрації вводяться унікальні ідентифікатори, надані роботодавцями.

Рисунок 1.1 — Системи фіксації часу перебування

Веб-системи дозволяють працівникам реєструвати свою присутність через веб-браузер на комп'ютері за допомогою унікальних облікових даних для фіксації часу початку та закінчення робочого дня або перерви.

Останнім часом мобільні системи набули великої популярності. У світі, який змінила пандемія коронавірусу, де дистанційна робота стала нормою, мобільні системи отримали перевагу. Вони дозволяють співробітникам реєструвати свою присутність за допомогою мобільних пристроїв, таких як смартфони або планшети. Найкращі версії таких систем інтегровані з GPS, що дозволяє відстежувати місцезнаходження працівників [3]. Мобільні системи є незамінними для таких галузей, як ресторанний бізнес та роздрібна торгівля, де необхідно відстежувати переміщення численних співробітників доставки.

Переваги таких систем численні. Вони не лише точно відраховують робочий час для роботодавців, але й зберігають дані, які можна

використовувати відділом кадрів під час нарахування заробітної плати (рисунок 1.2). Для працівників впровадження такої системи на робочому місці означає чітке уявлення про свій робочий день, час початку, час закінчення та перерви. Якщо роботодавець вимагає від працівника понаднормової роботи, він повинен забезпечити справедливу компенсацію.

Рисунок 1.2 — Приклад системи ідентифікації працівників

Такі чіткі розмежування забезпечують працівникам прозорий графік та оплату праці, включаючи всі відпрацьовані додаткові години. Оптимізована система обліку часу дозволяє співробітникам легко й ефективно фіксувати години перерв. У випадках, коли заклад покладається на ручний облік перерв, може виникнути плутанина. Але автоматизована система реєструє вхід і вихід працівника, а також фіксує тривалість їхніх перерв на обід та інші перерви. Це зменшує непорозуміння та дозволяє працівникам краще управляти своїм розкладом. Крім того, це надає роботодавцям зручний спосіб моніторингу робочого часу співробітників без будь-яких проблем.

Час є одним із цінних ресурсів для будь-якої установи. Сучасні системи інтегруються з мобільними пристроями, збираючи та відстежуючи дані про час

роботи й перерви, а також повідомляючи керівництво про будь-які розбіжності. Більшість таких програм мають вбудовані функції для виявлення недоліків у відвідуваності співробітників, що допомагає керівництву коригувати політику відповідно до потреб компанії.

У таких галузях, як роздрібна торгівля чи ресторанний бізнес, де працівники повинні бути присутніми у визначений час, точне відстеження їхнього робочого часу є важливим для успішного функціонування закладу. Система фіксує неробочий час і гарантує, що роботодавці не оплачують працівникам години, які вони не відпрацювали. За певний період це може суттєво зекономити кошти закладу.

Коли працівники використовують систему для обліку робочого часу, пристрій автоматично створює записи, що спрощує відстеження робочих та понаднормових годин. Це вигідно як для працівників, так і для роботодавців. Працівники можуть справедливо вимагати оплату за відпрацьовані понаднормові години, а роботодавці можуть зменшити несанкціоновану понаднормову роботу та забезпечити дотримання законодавства.

Однією з найважливіших переваг системи є можливість постійного відстеження часу, що дозволяє роботодавцю справедливо визначати, кому з працівників потрібно платити більше. Загалом, такі системи не лише ведуть облік, але й дозволяють оцінити ефективність співробітників та розуміти, хто оптимально використовує свій робочий час. Це сприяє підвищенню продуктивності та прибутковості. На основі зібраних даних керівництво може приймати рішення щодо винагородження працьовитих співробітників.

1.2 Обґрунтування вибору мови програмування

Згідно з різними дослідженнями, Java має численні переваги, що робить її відмінним вибором для Інтернету речей (IoT). Однією з найбільших переваг є можливість написати код лише один раз і використовувати його на будь-яких пристроях. За даними Фонду Eclipse, проведеного опитування розробників IoT з усього світу у 2015 році, використання Java для IoT зростає, і згідно з

опитуванням 2018 року, Java очолила список найпопулярніших мов програмування для розробки IoT з вражаючим показником 66,5%. Ця революційна технологія зростає експоненціально, ставши після Інтернету найбільш вимаганим винаходом. Прогнози IDC показують, що до 2024 року кількість пристроїв IoT перевищить 35 мільярдів у всьому світі.

Кожен ринок і галузь мають широкі можливості для використання IoT з Java через її гнучкість та простоту використання. Java є ідеальним вибором для взаємодії та координації роботи багатьох пристроїв, що є характерним для IoT. Вона відносно менш складна та проста у використанні порівняно з іншими мовами програмування, що робить її бажаним вибором для розробників.

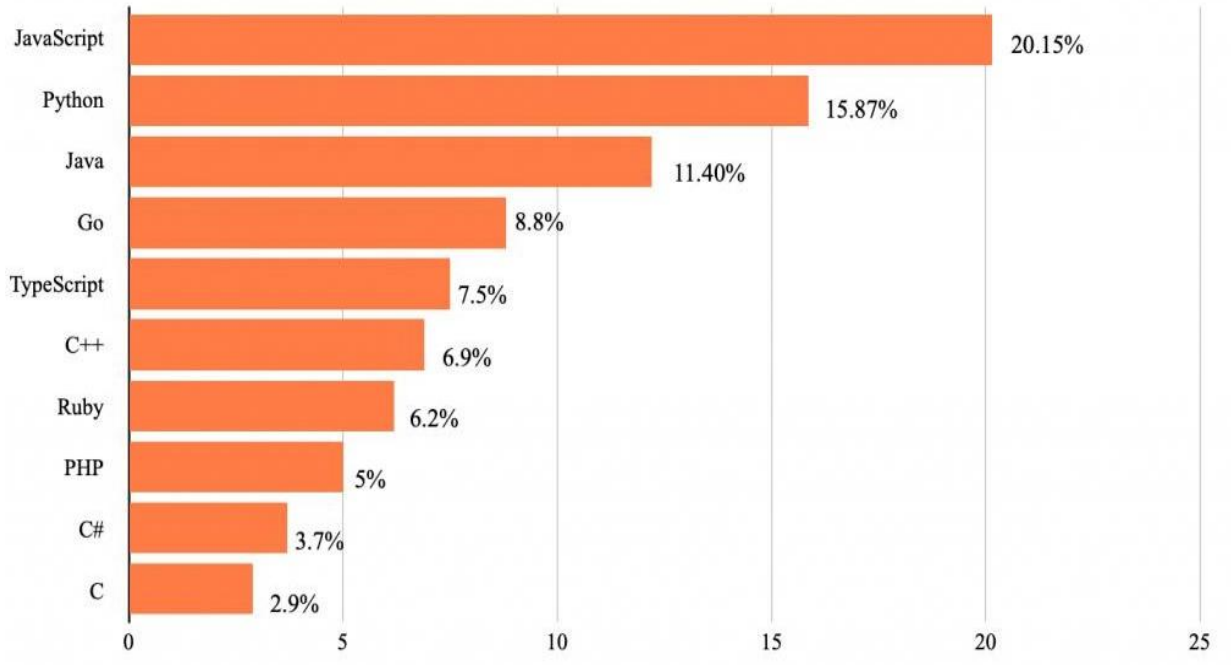

Рисунок 1.3 — Поширеність використання мов програмування

Ця мова програмування вирішує проблеми з безпекою даних, тому є ідеальним варіантом для Інтернету речей. Багато пояснень вибору Java як найкращого рішення для розробки IoT базуються на її універсальності. Під час розробки вбудованої програми необхідно враховувати операційну систему реального часу, процесор і різні протоколи для підключення пристроїв. Java, як мова незалежна від платформи, узагальнює всі ці аспекти, що дозволяє розробникам писати код лише один раз і використовувати його на будь-яких

пристроях. Розроблені додатки Java для IoT можуть працювати на різних апаратних пристроях і платформах без потреби змінювати код. Це полегшує розробку і робить її менш обтяжливою.

Гнучкість і доступність Java роблять її оптимальною платформою для Інтернету речей. Програми Java IoT легко переносяться на нову платформу, що зменшує ймовірність помилок і проблем. Java розроблена для роботи в середовищах з обмеженими ресурсами, що забезпечує мінімальне використання ресурсів і максимальну ефективність.

Java пропонує користувачам більше 4000 бібліотек, що охоплюють всі вимоги до програмування IoT. Програмування на Java дає можливість використовувати API з мінімальним переписуванням, що прискорює процес розробки і виконання коду. Java також має вбудовані заходи безпеки, такі як неявні покажчики, які зменшують ймовірність проблем з безпекою. Вона також легко вивчається та має широкий функціонал, включаючи масштабованість і можливість використання існуючих API.

Загалом, Java є ідеальним вибором для програмування в Інтернеті речей через свою гнучкість, універсальність і вбудовані заходи безпеки. Її успіх у світі IT базується на принципі "напиши один раз, запусти де завгодно", що робить її найкращим вибором для об'єднання всіх пристроїв в Інтернеті речей.

Інтернет речей (IoT) визначається як підключення пристроїв до Інтернету та їх взаємодія через мережу. Це велика мережа різноманітних обчислювальних пристроїв, об'єктів та машин, які обмінюються інформацією без прямого втручання людини. Від фітнес-трекерів до кухонних приладів, від безпілотних автомобілів до роботів, IoT стає все більш загальною реальністю.

Процес обміну інформацією між пристроями IoT відбувається через різні точки дотику, що створюються за допомогою вбудованих програм для кращого з'єднання. Технологія IoT вже не є просто модним трендом, вона стала невід'ємною частиною нашого життя, і компанії, включаючи розробників веб-додатків на Java, активно використовують її для створення нових інноваційних продуктів та послуг.

Використання технології IoT може перетворити досвід користувачів і перетворити цінності додатків, які раніше були неможливими. Уряди також використовують IoT для розбудови "розумних" міст з численними додатками для мешканців. Однак для успішного впровадження IoT в такі програми необхідні надійні програмні рішення, що відповідають стандартам даних та безпеки.

Java стала мовою програмування №1 для створення програм IoT, і однією з основних переваг її використання є надійність створеного на ній коду. Java має численні функції, які роблять її популярною, включаючи вбудовані заходи безпеки та здатність виконувати код на різних платформах. Крім того, Java дозволяє розробникам ефективно використовувати API для з'єднання пристроїв на рівні програми.

У підсумку, Java залишається живою та популярною мовою програмування, яка відмінно підходить для реалізації проектів IoT завдяки своїй надійності, портативності та здатності до з'єднання на рівні програми.

Подивимося на деякі аспекти, які роблять Java такою привабливою для розробників програмного забезпечення.

По-перше, Java має велику кількість спільного з іншими популярними мовами, такими як C, C++ і JavaScript, що спрощує перехід для розробників, які вже знайомі з цими мовами. Чіткі правила синтаксису і структури коду роблять навчання Java більш передбачуваним.

По-друге, Java забезпечує потужну мережу підтримки для програмістів на всіх етапах їхнього розвитку. Від онлайн-ресурсів до спільнот і форумів, таких як YouTube, StackOverflow і CodeRanch, новачки знайдуть відповіді на свої запитання і підтримку у вирішенні проблем.

Крім того, постійні оновлення і удосконалення, такі як можливості функціонального програмування з лямбда-функціями, свідчать про активний розвиток мови. Важливо також відзначити зворотну сумісність, яка дозволяє розробникам безпечно оновлювати свої програми.

Нарешті, екосистема інструментів Java, таких як Gradle, Maven і Jenkins, сприяє швидкому і ефективному процесу розробки. Ці інструменти допомагають спростити керування залежностями, збіркою та безперервною інтеграцією.

У підсумку, Java продовжує залишатися найпопулярнішою мовою програмування завдяки своїй простоті в навчанні, потужній мережі підтримки, постійним оновленням та розширенню функціональності, а також розвинутій екосистемі інструментів.

Android є найпопулярнішою операційною системою для мобільних телефонів у світі, а Java стала де-факто мовою програмування для розробки додатків для Android. Хоча версія Java для Android відрізняється від тієї, що знаходиться в JDK, Google фактично використовував більше 11 500 рядків коду з Java Standard Edition при створенні свого власного варіанту Java. Це означає, що розробники можуть очікувати, що Java на Android буде досить близькою до оригіналу. Таким чином, якщо ви володієте навичками програмування на Java для настільних або серверних програм, ви з легкістю зможете розпочати розробку для Android. Усі низькорівневі відмінності між JVM і Android Runtime будуть абстраговані від вас після короткого періоду навчання. Після того, як розробники оволодіють Java, вони матимуть доступ до всієї екосистеми Android. Java розвивається повільно, але постійно.

Програмування на Java є зручним та гнучким, що робить його популярним вибором для розробників веб-додатків та експертів з управління програмами. Незалежно від того, чи вам потрібна мова для чисельних обчислень, мобільних додатків чи настільних комп'ютерів, Java буде вам в нагоді. Код на Java легко зрозуміти та відлагоджувати. Крім того, Java працює на комп'ютерах Mac, Linux або навіть Windows, і JRE сумісний з мобільними телефонами, що робить її універсальною мовою. Вона ідеально підходить для таких пристроїв, як Raspberry Pi. Java працює як на великих, так і на малих проектах, що робить код стабільним та надійним. І хоча у Java немає обмежень,

ви можете навіть створити програмне забезпечення для перекладу, використовуючи цю мову.

Spring є відомим фреймворком для розробки корпоративних додатків на Java, підтримуваним великою громадою розробників по всьому світу. Він дозволяє створювати високопродуктивний код, який легко тестувати і повторно використовувати. Spring framework базується на Java з відкритим кодом і був розроблений Родом Джонсоном, вперше випущений під ліцензією Apache 2.0 у червні 2003 року. Основні можливості Spring Framework можна використовувати для будь-яких програм на Java, включаючи розширення для веб-програм на основі Java EE. Основна мета Spring - спростити розробку J2EE і популяризувати ефективні практики програмування, основані на моделі POJO.

Переваги використання Spring включають здатність розробки корпоративних програм за допомогою POJO, що уникає потреби в EJB і дозволяє використовувати надійні контейнери сервлетів, такі як Tomcat або інші комерційні рішення. Фреймворк організований за модульним принципом, що дозволяє використовувати тільки необхідні компоненти і ігнорувати зайві. Використання існуючих технологій, таких як ORM, фреймворки журналювання та таймерів JEE, сприяє ефективності розробки. Тестування програм, написаних з використанням Spring, просте завдяки принципу переносу коду, залежного від середовища, в структуру Spring. Також JavaBean-style POJO полегшує ін'єкцію залежностей для тестових даних.

Веб-фреймворк Spring є реалізацією популярного шаблону архітектури MVC для веб-додатків. Він представляє собою ефективну альтернативу іншим веб-фреймворкам, таким як Struts та інші менш популярні варіанти. Spring надає зручний API для перекладу специфічних для технологій винятків, таких як JDBC, Hibernate або JDO, що спрощує розробку персистентного шару додатків.

Контейнери IoC в Spring є досить легкими, особливо у порівнянні з контейнерами EJB, що робить їх ідеальними для розгортання програм на обмежених за ресурсами комп'ютерах. Spring забезпечує узгоджений інтерфейс

для керування транзакціями, який можна легко масштабувати від локальних до глобальних транзакцій. Наприклад, від простих транзакцій з однією базою даних до складних системних транзакцій з використанням JTA.

Основною технологією, з якою асоціюється Spring, є інверсія керування залежностями (DI). Інверсія керування (IoC) - це загальна концепція, яку можна реалізувати різними способами, і ін'єкція залежностей є одним із цих способів. У складних Java-програмах класи повинні бути максимально незалежними один від одного для полегшення повторного використання та тестування кожного класу окремо під час модульного тестування. Ін'єкція залежностей допомагає з'єднати ці класи, зберігаючи їхню незалежність. Це досягається шляхом передачі параметрів до конструктора або через методи встановлення після створення об'єкта. Ін'єкція залежностей є ключовим компонентом Spring Framework.

Одним із важливих компонентів Spring є аспектно-орієнтоване програмування (AOP). Функції, які стосуються кількох частин програми, називаються наскрізними проблемами і вони концептуально відокремлені від бізнес-логіки додатка. Приклади таких аспектів включають журналювання, декларативні транзакції, безпеку, кешування тощо. Якщо в об'єктно-орієнтованому програмуванні (ООП) основною одиницею модульності є клас, то в AOP цією одиницею є аспект. Інверсія керування залежностями (DI) дозволяє відокремити об'єкти програми один від одного, а AOP допомагає відокремити наскрізні проблеми від об'єктів, на які вони впливають. Модуль AOP у Spring Framework реалізує аспектно-орієнтоване програмування, дозволяючи визначати перехоплювачі методів і точки розрізу, щоб чітко відокремити код, який реалізує різні функціональні аспекти.

Spring має потенціал стати універсальною платформою для всіх корпоративних додатків. Завдяки модульній структурі (рисунок 1.4), Spring дозволяє розробникам вибирати лише ті модулі, які їм потрібні, без необхідності використовувати решту.

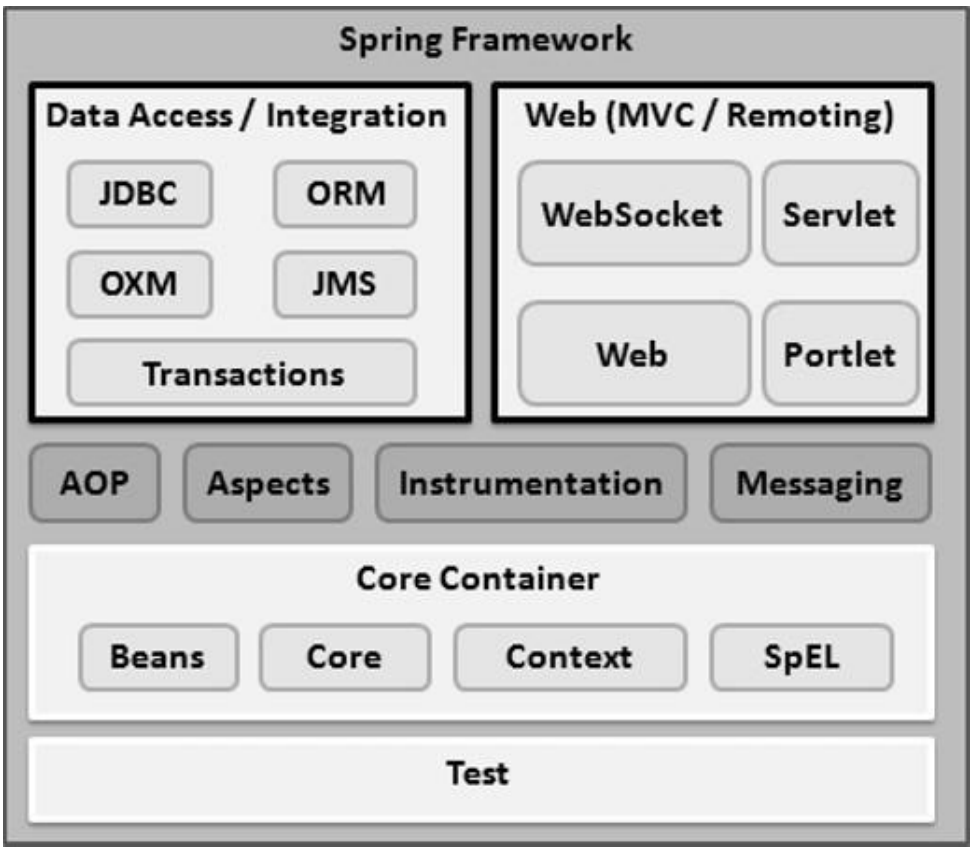

Рисунок 1.4 — Модульність структури Spring

Spring Framework пропонує близько 20 модулів, які можна використовувати відповідно до потреб програми. Основні модулі контейнера Core включають Core, Beans, Context та Expression Language:

— модуль Core забезпечує основні функції фреймворку, включаючи IoC та Dependency Injection;

— модуль Beans надає BeanFactory, що є вдосконаленою реалізацією фабричного шаблону;

— модуль Context побудований на основі модулів Core та Beans і забезпечує доступ до визначених і налаштованих об'єктів через інтерфейс ApplicationContext;

— модуль SpEL надає потужну мову виразів для запитів та маніпулювання графом об'єктів під час виконання.

Рівень інтеграції складається з модулів JDBC, ORM, OXM, JMS та Transaction:

— модуль JDBC пропонує рівень абстракції JDBC, усуваючи потребу у виснажливому кодуванні, пов'язаному з JDBC;

— модуль ORM забезпечує інтеграцію з популярними API об'єктно-реляційного відображення, такими як JPA, JDO, Hibernate та iBatis;

— модуль OXM підтримує реалізацію відображення Object/XML для JAXB, Castor, XMLBeans, JiBX та XStream;

— модуль JMS містить функції для створення та споживання повідомлень у службі обміну повідомленнями Java;

— модуль Transaction підтримує програмне та декларативне керування транзакціями для спеціальних інтерфейсів та POJO класів.

Веб-рівень складається з модулів Web, Web-MVC, Web-Socket та Web-Portlet;

— модуль Web надає основні функції веб-інтеграції, такі як завантаження файлів та ініціалізація контейнера IoC через слухачі сервлетів та контекст веб-додатків;

— модуль Web-MVC реалізує Model-View-Controller (MVC) для веб-додатків;

— модуль Web-Socket забезпечує підтримку двостороннього зв'язку на основі WebSocket між клієнтом і сервером;

— модуль Web-Portlet реалізує MVC для середовища портлетів, аналогічно модулю Web-Servlet.

Існує також кілька інших важливих модулів, таких як AOP, Aspects, Instrumentation, Web та Test:

— модуль AOP забезпечує аспектно-орієнтоване програмування, дозволяючи визначати перехоплювачі методів і точки розрізів;

— модуль Aspects інтегрується з AspectJ, потужною структурою AOP;

— модуль Instrumentation підтримує інструменти класів та реалізацію завантажувача класів для певних серверів додатків;

— модуль обміну повідомленнями підтримує STOMP як підпротокол WebSocket та забезпечує аннотаційний підхід для маршрутизації та обробки повідомлень STOMP від клієнтів WebSocket;

— модуль Test підтримує тестування компонентів Spring за допомогою JUnit або TestNG.

Останню версію SDK можна завантажити з сайту Oracle Java на сторінці Java SE Downloads. Інструкції щодо встановлення JDK знаходяться в завантажених файлах. Дотримуйтеся цих інструкцій, щоб правильно встановити та налаштувати параметри. Після встановлення необхідно встановити змінні середовища PATH і JAVA_HOME для посилання на каталоги, що містять файли java і javac, зазвичай це java_install_dir/bin і java_install_dir відповідно.

Для Windows NT/2000/XP натисніть правою кнопкою миші на "Мій комп'ютер", виберіть "Властивості", потім "Додатково" і "Змінні середовища". Оновіть значення PATH і натисніть "OK". Якщо ви використовуєте інтегроване середовище розробки (IDE), таке як Borland JBuilder, Eclipse, IntelliJ IDEA або Sun ONE Studio, скомпілюйте та запустіть просту програму, щоб переконатися, що IDE розпізнає встановлену Java. В разі необхідності, виконайте налаштування згідно з документацією IDE (рисунок 1.5).

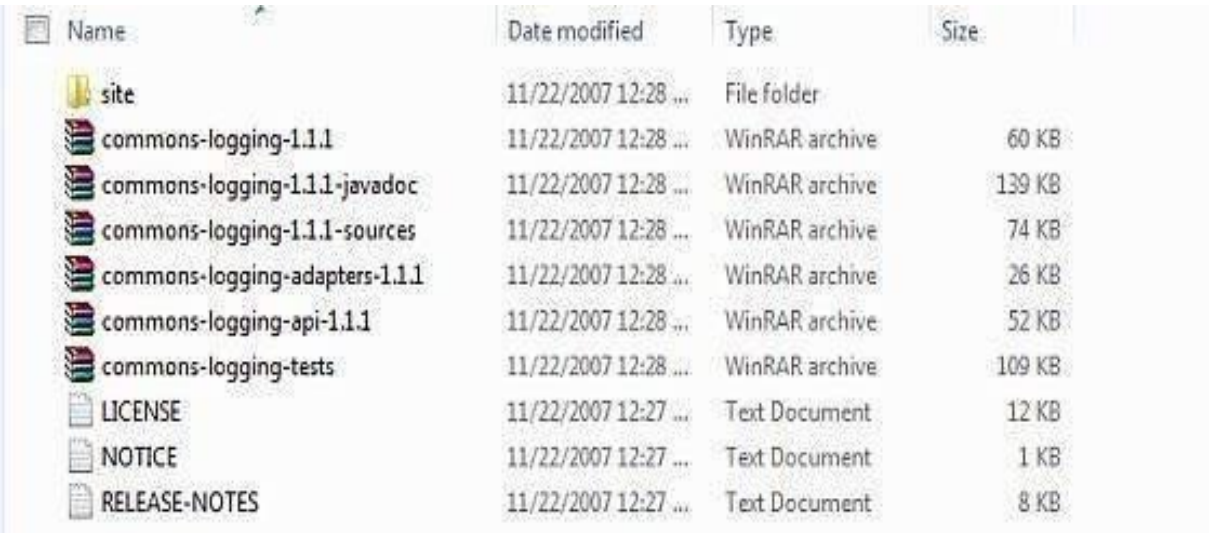

Рисунок 1.5 — Приклад пакету легування

Ви можете завантажити найновішу версію Apache Commons Logging API. Після завантаження розпакуйте двійковий дистрибутив у зручне для вас місце.

Цей каталог міститиме jar файли та інші супровідні документи. Переконайтеся, що правильно налаштували змінну CLASSPATH для цього каталогу, інакше під час роботи програми можуть виникнути проблеми.

Для встановлення Eclipse IDE завантажте останню версію Eclipse з офіційного сайту. Після завантаження розпакуйте двійковий дистрибутив у зручне місце та налаштуйте змінну PATH відповідно. Якщо все встановлено правильно, після запуску програми ви побачите очікуваний результат (рисунок 1.6).

Нижче наведено обґрунтування вибору фреймворку Spring:

— інтуїтивна функціональність;
— високий рівень безпеки;
— простота роботи з базами даних;
— модульний дизайн;
— можливість інтеграції з іншими фреймворками;
— ін'єкція залежностей;
— зручність тестування.

Spring легко освоїти та реалізувати завдяки модульній структурі, яка дозволяє писати програми з інтерфейсами та абстрактними класами, аналогічно до стандартних програм Java. Цей фреймворк спрощує інтеграцію компонентів, що полегшує роботу зі Spring, дозволяючи розробникам більше зосередитися на логіці програми, а не на її реалізації. Завдяки можливості вводити лише потрібні залежності, Spring допомагає зберігати ресурси, мінімізуючи непотрібне використання пам'яті.

Spring забезпечує високий рівень безпеки завдяки використанню Spring Security, який можна налаштувати для базової автентифікації та захисту від вразливостей. Шаблон MVC (Model-View-Controller) у Spring сприяє чіткому розділенню реалізації та бізнес-логіки, дозволяючи розробникам зосередитися на коді для підвищення продуктивності додатків. MVC допомагає систематично

обробляти запити: спочатку запити надходять у представлення, яке передає їх контролеру, далі до моделі, і нарешті, результат відображається в браузері. Такий підхід робить MVC-фреймворки, як Spring, ефективним рішенням для розробки програмного забезпечення.

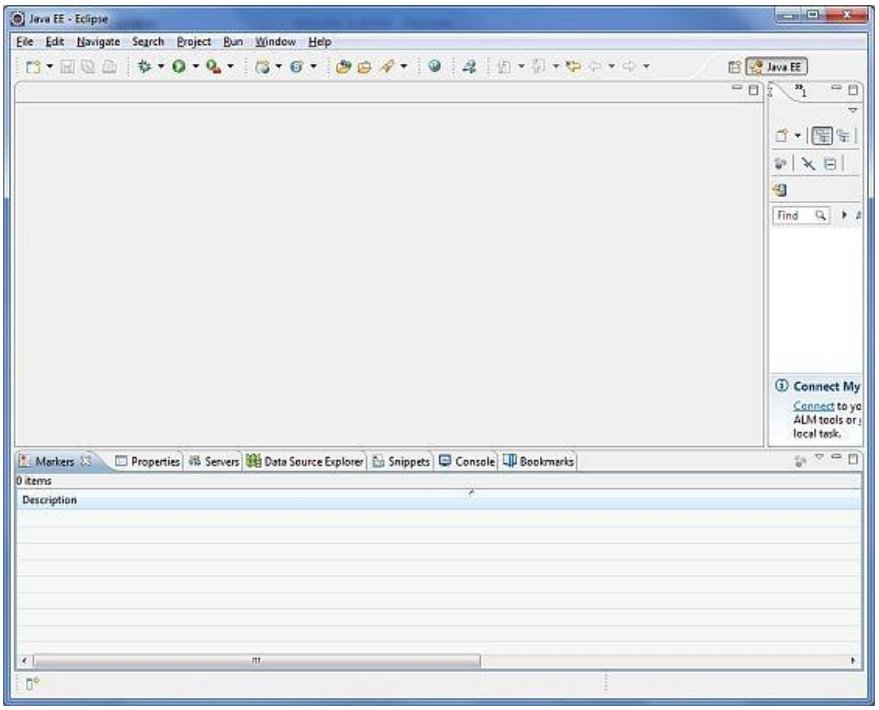

Рисунок 1.6 — Середовище розробки Eclipse IDE

Spring підтримує шаблон MVC для розробки проектів, що сприяє поділу завдань, що є однією з ключових характеристик якісного програмного забезпечення.

Бази даних є критично важливою частиною будь-якої програми, оскільки без плавного та легкого зв'язку з базою даних програма може стати складною та малоефективною. Spring забезпечує зручний і ефективний зв'язок із базами даних завдяки функціональним можливостям DAO (Data Access Object), призначеним для зчитування та запису даних у базу даних. Використання DAO у Spring спрощує роботу з базами даних за допомогою технологій, таких як Hibernate, JDBC та JPA, що значно полегшує взаємодію з базами даних . Наприклад, Hibernate у Spring дозволяє виконувати операції CRUD (створення,

читання, оновлення, видалення) з мінімальним обсягом коду, використовуючи прості функції для роботи з даними.

Однією з визначних особливостей Spring є його модульність. Це не монолітний фреймворк, який містить усе як тісно пов'язаний пакет, а набір різних JAR-файлів, які можна використовувати незалежно один від одного. Він розділений на компоненти, такі як основний контейнер, доступ/інтеграція даних, веб та тестування, які можуть працювати разом, але залишаються незалежними. Реалізація також може бути організована за шаблоном MVC (Model-View-Controller), де представлення взаємодіє з користувачем, отримує запити та надсилає відповіді, контролер обробляє запити та передає їх відповідним методам, а модель працює з базами даних і даними.

Така модульність дозволяє розробникам зосередитися на конкретних аспектах програми, спрощуючи її підтримку та розвиток. Це робить Spring потужним інструментом для створення масштабованих, ефективних та легко підтримуваних корпоративних додатків.

Аспектно-орієнтоване програмування (АОР) дозволяє структурувати програму по-новому, забезпечуючи модульність за рахунок поділу логіки на окремі частини, відомі як аспекти. Це допомагає розділити бізнес-логіку програми і підвищити її модульність. АОП виникло з парадигми об'єктно-орієнтованого програмування (ООП), де клас є ключем до модульності в ООП, а аспект — до модульності в АОП.

Тестування є важливою частиною розробки програмного забезпечення, і Spring сприяє цьому процесу завдяки функції ін'єкції залежностей, яка робить фреймворки більш придатними для тестування. Слабкий зв'язок між класами допомагає в модульному тестуванні, дозволяючи тестувати класи незалежно один від одного. Spring полегшує тестування складних проєктів, забезпечуючи ефективну перевірку їх функціональності.

Spring також ефективно працює з внутрішніми і зовнішніми ресурсами, такими як файли властивостей, зображення та файли XML, використовуючи інтерфейси Resource і ResourceLoader для їх обробки.

Spring Boot, частина фреймворку Spring, призначений для швидкого запуску програм Spring. Micronaut, платформа на основі JVM, створена для усунення деяких недоліків Spring/Spring Boot. Порівняння легкості створення нової програми, підтримки мов та інших параметрів конфігурації дозволяє обрати оптимальне рішення для розробки програмного забезпечення (рисунок 1.7).

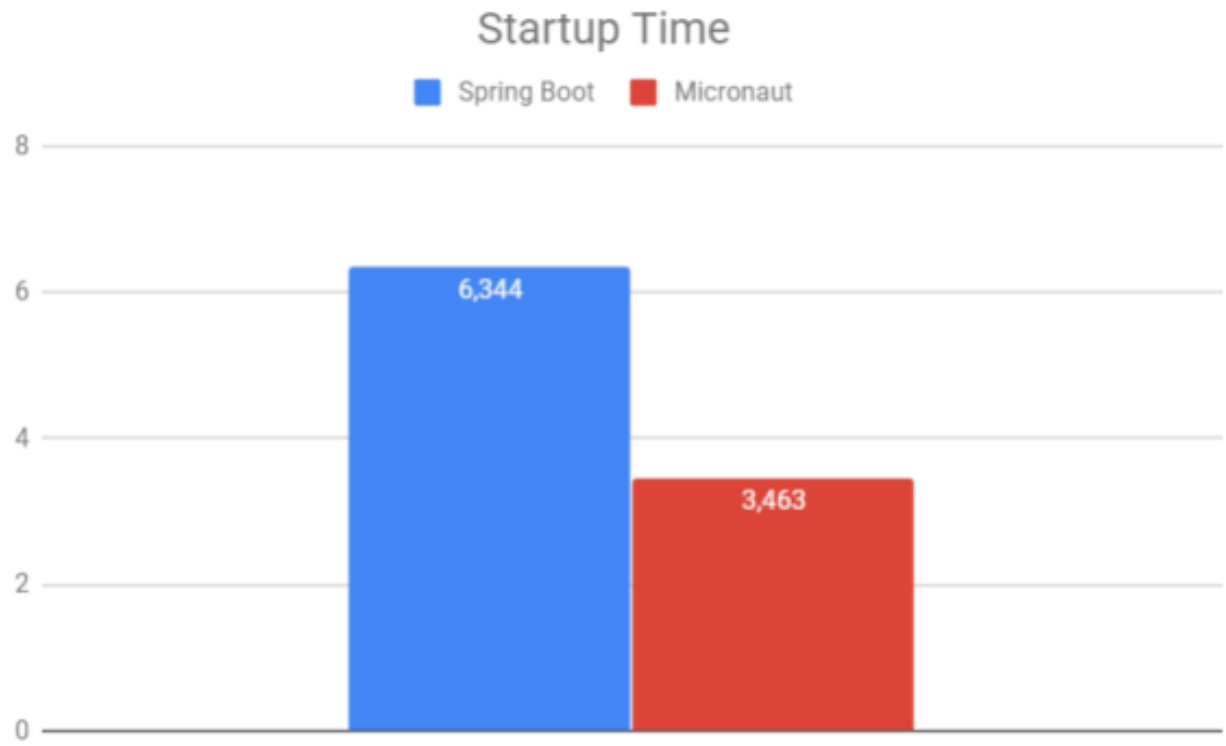

Рисунок 1.7 — Порівняльні характеристики часів запуску

Micronaut і Spring Boot пропонують зручні способи створення нових програм. Наприклад, можна скористатися командним рядком для створення нової програми, використовуючи будь-яку з цих структур. Також можна використати Spring Initializr для Spring Boot або Launch для Micronaut.

Щодо підтримки IDE, існують спеціальні плагіни для обох фреймворків. Spring Boot має плагіни для більшості популярних IDE, включаючи Eclipse Spring Tools Suite. Для IntelliJ є плагін Micronaut.

Щодо підтримки мов, обидва фреймворки майже однакові. Ви можете вибрати між Java, Groovy або Kotlin. Обидва підтримують Java 8, 11 і 17. Крім

того, обидва фреймворки підтримують використання Gradle або Maven як системи збірки.

За умовчанням програма, створена за допомогою Spring Boot, використовує Tomcat (рисунок 1.8). Проте можна налаштувати Spring Boot на використання Jetty або Undertow.

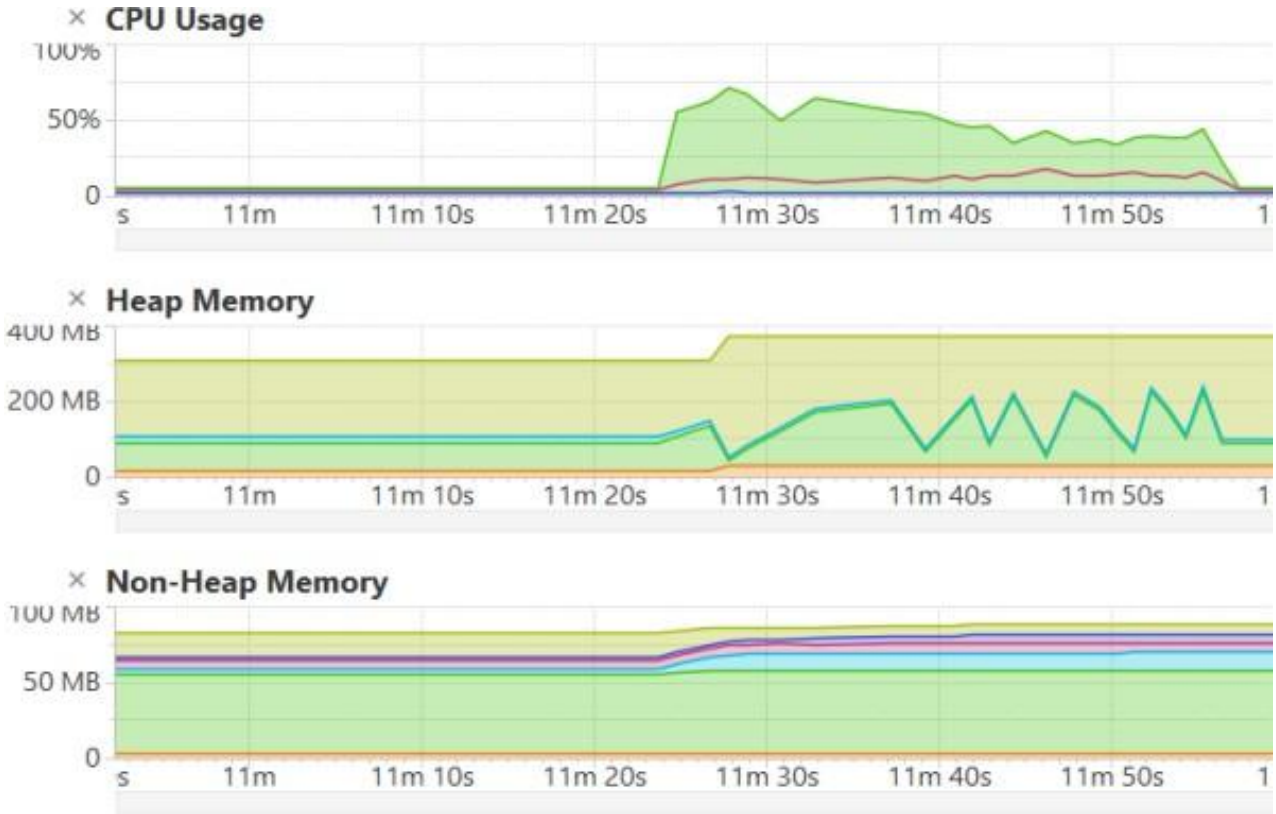

Рисунок 1.8 — Діаграми використання ресурсів Spring

Наші програми Micronaut за замовчуванням працюють на HTTP-сервері Netty. Проте ми можемо переключити їх на Tomcat, Jetty або Undertow. У Spring Boot ми визначаємо властивості в `application.properties` або `application.yml`. Для різних середовищ можна використовувати конвенцію `application-{env}.properties`. Також можна перевизначити ці властивості за допомогою системних властивостей, змінних середовища або атрибутів JNDI. У Micronaut можна використовувати `application.properties`, `application.yml` і `application.json` для файлів властивостей, змінюючи їх за допомогою системних властивостей або змінних середовища.

Для обміну повідомленнями у Spring Boot доступні Active MQ, Artemis, Rabbit MQ і Apache Kafka. У Micronaut підтримуються Apache Kafka, Rabbit MQ і Nats.io. Spring Boot пропонує п’ять стратегій авторизації: базову, авторизацію через форму, JWT, SAML і LDAP. Micronaut підтримує всі ці методи, крім SAML. Обидва фреймворки підтримують OAuth2. Для захисту методів в обох фреймворках використовуються анотації.

Обидва фреймворки дозволяють відстежувати різні показники та статистику. Вони також дозволяють створювати власні кінцеві точки і налаштовувати їхню безпеку. Spring Boot має вбудовані кінцеві точки, що спрощує моніторинг додатків.

Для створення повних додатків з інтерфейсом обидва фреймворки підтримують різні мови шаблонів. У Spring Boot можна використовувати Thymeleaf, Apache Freemarker, Mustache, Groovy та JSP (хоча це не рекомендується). У Micronaut доступні Thymeleaf, Handlebars, Apache Velocity, Apache Freemarker, Rocker, Soy/Closure та Pebbles.

Загалом, Spring Boot все ще лідирує завдяки своїй широкій підтримці та досвіду з мікросервісами. Однак Micronaut є перспективним і може скласти серйозну конкуренцію Spring Boot у майбутньому.

1.3 Обґрунтування вибору бази даних для зберігання даних

База даних є важливою структурованою системою для організації та зберігання даних відповідно до певних норм і правил. Вибір підходящої бази даних для проекту має вирішальне значення і вимагає урахування як характеристик самої системи, так і особливостей конкретного проекту, таких як обсяг даних, навантаження і час роботи [12].

Сучасне технологічне суспільство підкреслює важливість цілісності та безпеки даних, особливо в контексті великих проектів, які потребують надійного зберігання значної кількості інформації в одному місці [13]. Тому обрання відповідної СУБД для роботи є кроком, що вимагає докладного вивчення різних варіантів, їх переваг і недоліків.

В моєму проекті я використовую PostgreSQL як основну СУБД. Ця об'єктно-реляційна СУБД з відкритим кодом відома своєю здатністю безпечно зберігати великі обсяги складних даних. PostgreSQL допомагає розробникам створювати складні програми, виконувати адміністративні завдання і створювати надійні середовища.

Започаткована в 1986 році, PostgreSQL здобула велику популярність та зберігає свою актуальність завдяки регулярним оновленням і вдосконаленням функціональності. Порівнюючи її з іншими конкурентами, можна відзначити, що вибір між реляційними і нереляційними базами даних залежить від таких аспектів, як формат зберігання даних, масштабованість, безпека, доступність транзакцій і цілісність даних.

На цьому етапі обговорення виникає вибір між реляційними базами даних і нереляційними (NoSQL). PostgreSQL, як представник реляційних СУБД, використовує мову SQL, що є декларативною для маніпулювання даними в таблицях. SQL спрощує створення складних запитів і забезпечує стандартний підхід до роботи з даними у структурованій формі.

Насупроти цього, NoSQL представляє собою різноманітні підходи до зберігання даних, що не ґрунтуються на традиційній SQL. Такі бази даних, як MongoDB, CouchDB, Redis і Apache Cassandra, використовуються для зберігання та обробки нереляційних даних, де кожен тип системи має свої особливості:

Формат зберігання даних: Реляційні бази даних зберігають дані у вигляді таблиць з відносинами між ними, тоді як NoSQL бази даних використовують різноманітні формати, такі як документи, стовпці, ключ-значення, графи тощо.

Спосіб отримання даних: SQL бази даних використовують мову запитів SQL для отримання даних за допомогою складних JOIN операцій, в той час як NoSQL бази даних надають більш прості та гнучкі методи доступу до даних.

Нереляційні бази даних, як правило, використовують формат JSON для зберігання даних у вигляді ключ-значення. Вони дозволяють організовувати схожі файли в колекції. Структура NoSQL баз даних може бути побудована на

документах, які містять колекції ключ-значення, графів, або широких стовпців, і вони не мають стандартної схеми, тому їх також називають базами даних без зв'язків. Вони характеризуються горизонтальним масштабуванням, що дозволяє легко додавати нові сервери у інфраструктуру NoSQL бази даних для обробки великого обсягу трафіку. Це сприяє високій продуктивності для простих запитів, оскільки всі відповідні дані зберігаються в одному документі, що уникне операцій злиття даних.

Нереляційні бази даних ідеально підходять в наступних випадках:

— для роботи з великою кількістю даних будь-якої природи: структурованих, напівструктурованих або неструктурованих;

— коли потрібні локальні транзакції даних, які не потребують довгострокового збереження;

— для впровадження сучасних практик розробки програмного забезпечення;

— якщо потрібно працювати з об'єктно-орієнтованим програмуванням;

— якщо ви працюєте з даними без строгої схеми;

— для створення економічної та ефективної архітектури;

— якщо реляційна база даних не здатна масштабуватися в межах вашого бюджету.

У порівнянні з цим, реляційні системи зазвичай зберігають інформацію у вигляді таблиць, де кожна таблиця має фіксовану кількість стовпців і строго типізовані дані. Ця структура, хоч і менш гнучка, мінімізує ризик помилок.

У реляційних базах даних схема (як показано на рисунку 1.9) визначає таблиці і типи полів у цих таблицях. Це означає, що перед початком проекту необхідно спроектувати структуру бази даних, перш ніж визначати будь-яку бізнес-логіку. Після створення схеми внесення значних змін стає відносно складним процесом. У не реляційних системах таких обмежень немає: користувачі можуть вносити зміни в базу даних у будь-який момент. Без схеми база даних не вимагає попереднього жорсткого дизайну системи.

Реляційні СУБД використовують мову запитів SQL, що дозволяє легко отримувати необхідні дані з однієї чи декількох таблиць. У не реляційних системах відсутній стандартний інтерфейс мови запитів, тому всі операції потрібно виконувати вручну. Крім того, при роботі з не реляційними системами часто виникають проблеми з безпекою та функціональністю, які зазвичай виникають через недостатній рівень знань у розробника.

Щодо масштабованості, для реляційних систем це може бути проблематичним завданням. Проте при необхідності можна розглянути кластеризацію декількох сервісів навколо центрального сховища даних для подальшого розподілу пов'язаних даних. На відміну від цього, у не реляційних систем масштабування реалізується значно простіше. Вертикальне масштабування полягає у додаванні ресурсів, таких як пам'ять або потужніше процесори, до існуючого сервера.

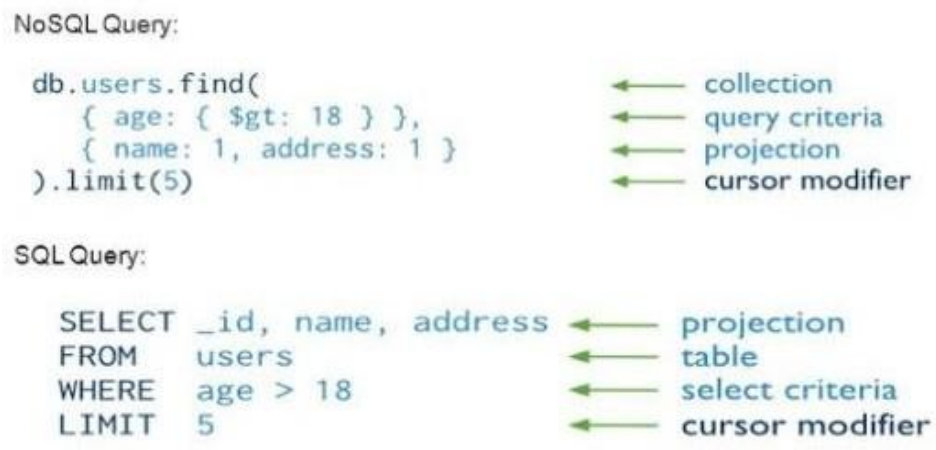

Рисунок 1.9 — Порівняння запитів

Горизонтальне масштабування означає додавання нових серверів (вузлів) до існуючої системи, що дозволяє збільшувати її робочу навантаження та пропускну здатність (див. рисунок 1.10).

```
CREATE TABLE IF NOT EXISTS public.terminal
(
    id bigint NOT NULL DEFAULT nextval('terminal_id_seq'::regclass),
    create_date_time timestamp without time zone,
    update_date_time timestamp without time zone,
    description character varying(255) COLLATE pg_catalog."default" NOT NULL,
    location character varying(255) COLLATE pg_catalog."default" NOT NULL,
    name character varying(255) COLLATE pg_catalog."default" NOT NULL,
    company_id bigint,
    CONSTRAINT terminal_pkey PRIMARY KEY (id)
);
```

Рисунок 1.10 — Процес створення таблиці

Цілісність і точність даних є важливими аспектами для реляційних систем баз даних, які використовують механізм транзакцій. У реляційних СУБД транзакції гарантують принцип "все або нічого", що означає, що всі оновлення або додатки до бази даних відбуваються або відхиляються як одне ціле. Це забезпечує точність і цілісність даних навіть у випадках системних збоїв чи втрати живлення. Такі гарантії безпеки описуються абревіатурою ACID: атомарність, консистенція, ізоляція та довговічність.

У відміну від цього, не реляційні системи не підтримують транзакції, тому кожне оновлення або додавання обробляється окремо. Це може призводити до проблем з точністю та цілісністю даних, оскільки не гарантується, що всі зміни будуть введені у базу даних атомарно. Відсутність механізму транзакцій у не реляційних системах є однією з їхніх основних відмінностей від традиційних реляційних баз даних.

Хоча обидва підходи до зберігання даних мають свої переваги і недоліки, вибір між ними часто залежить від конкретних потреб проекту та бізнес-вимог. Реляційні бази даних часто використовуються для проектів з складною структурою даних, де потрібна висока точність і надійність. З іншого боку, не реляційні системи можуть бути більш гнучкими і зручними для роботи з великими обсягами неструктурованих або напівструктурованих даних, де потрібна велика масштабованість і продуктивність.

Отже, вибір між реляційними та не реляційними базами даних не є чистою конкуренцією, а скоріше альтернативою залежно від конкретного сценарію використання і вимог проекту.

MongoDB є однією з найпоширеніших не реляційних баз даних, заснованою на документах, ідеально підходить для сучасних додатків. Ось основні характеристики MongoDB:

— база даних без схеми: MongoDB зберігає дані у вигляді документів JSON, що дозволяє зберігати дані різних типів у кожному документ,і це надає велику гнучкість і дозволяє легко змінювати структуру даних;

— документоорієнтована структура — дані в MongoDB зберігаються у вигляді документів, а не у вигляді таблиць з рядками і стовпцями, це дозволяє MongoDB бути дуже гнучкою системою для зберігання даних;

— індексація: MongoDB автоматично індексує дані за допомогою первинних і вторинних індексів, що робить операції пошуку і отримання даних швидшими і ефективнішими;

— масштабованість: MongoDB підтримує горизонтальну масштабованість з використанням шардингу, що дозволяє розподіляти дані на кілька серверів для підтримки великих обсягів даних та високої продуктивності;

— реплікація: функція реплікації MongoDB дозволяє створювати копії даних на різних серверах, забезпечуючи високу доступність та стійкість до відмов;

— Агрегація: MongoDB підтримує різні типи агрегацій, включаючи конвеєрну агрегацію, функції зменшення карти та одноцільову агрегацію. Ці функції дозволяють операціям з даними отримувати зведені результати.

Загалом, MongoDB є потужною і гнучкою системою для зберігання даних, яка підходить для широкого спектру використання в сучасних додатках.

Ось кілька переваг програмного забезпечення MongoDB:

— швидкість та ефективність: MongoDB відома своєю високою швидкістю та ефективністю, що робить її відмінним вибором для сучасних додатків, які вимагають швидкого доступу до даних;

— легкість у використанні: MongoDB використовує документи у форматі JSON, що спрощує роботу з даними порівняно з традиційними реляційними базами даних, вона також інтегрована з JavaScript, що полегшує розробку і взаємодію з базою даних для веб-розробників;

— простота управління: MongoDB дозволяє автоматично додавати та створювати структури даних під час введення даних, що уникне необхідності визначення схеми даних наперед, це робить управління базою даних більш простим і зручним;

— Масштабованість: MongoDB підтримує горизонтальне масштабування з використанням шардингу, що дозволяє розподіляти навантаження на декілька серверів, це робить її ідеальним вибором для великих обсягів даних і додатків з високими вимогами до продуктивності;

— динамічна схема: MongoDB має динамічну схему даних, що дозволяє зберігати різноманітні дані, від структурованих до неструктурованих, це особливо важливо для додатків, де структура даних може змінюватися з часом;

— реплікація і безпека: MongoDB підтримує реплікацію, що забезпечує збереження даних в безпеці в разі відмови сервера. Вона також забезпечує можливість робити резервні копії даних для забезпечення високої доступності та стійкості системи.

Загалом, MongoDB є потужним і сучасним рішенням для зберігання даних, яке відповідає вимогам сучасних додатків, надаючи швидкість, гнучкість і простоту у використанні.

Ось деякі недоліки MongoDB, які можуть вплинути на його використання:

— відсутність підтримки транзакцій: MongoDB не підтримує традиційні транзакції, які гарантують атомарність (все або нічого), консистентність, ізоляцію і довговічність, хоча MongoDB використовує механізми, подібні до

транзакцій (наприклад, для оновлення одного документа), вони не є стандартними транзакціями в реляційному значенні.

— можливе пошкодження даних при оновленні: оскільки MongoDB не має строгих транзакцій, при неочікуваному зупиненні або втраті підключення може виникнути ситуація, коли оновлення даних не збережуться коректно, що може призвести до пошкодження даних;

— високе споживання пам'яті: MongoDB може вимагати значний обсяг оперативної пам'яті, особливо при роботі з великими обсягами даних або при виконанні операцій з великою кількістю користувачів;

— обмежений розмір документів: кожен документ в MongoDB обмежений розміром до 16 МБ, це може стати проблемою для додатків, які потребують зберігання великих об'ємів даних в одному документі.

— продуктивність обмежена: хоча MongoDB може бути масштабованою, для високих навантажень і великої кількості користувачів можуть виникати проблеми з продуктивністю;

— проблеми з індексуванням: необережне створення і управління індексами в MongoDB може призвести до проблем з продуктивністю і надмірним використанням ресурсів. Індекси потребують уваги і оптимізації для ефективної роботи.

Ці недоліки важливо враховувати при виборі бази даних для проекту. Хоча MongoDB є потужним і гнучким рішенням для багатьох сценаріїв, він може не підходити для всіх випадків за рахунок цих обмежень і особливостей.

З іншого боку, ми маємо PostgreSQL. Починаючи з версії 9.4, PostgreSQL отримав ряд оптимізацій і оновлень, серед яких додавання HStore і Jsonb, бінарної версії для зберігання JSON. Це дозволило інтегрувати деякі функції NoSQL, хоча горизонтальне масштабування все ще відсутнє. Тим не менш, ця реалізація надає PostgreSQL перевагу, оскільки вона поєднує реляційні та нереляційні практики, дозволяючи зберігати набори даних ключ-значення в таблицях, що працює належним чином.

PostgreSQL здатний ефективно обробляти складні запити для найбільших компаній та установ, що перевірено роками. Команда розробників постійно вдосконалює систему, випускаючи нові версії, які включають поліпшення для неструктурованих даних. Вони постійно пропонують нові інструменти та функції для забезпечення актуальності даних. PostgreSQL забезпечує валідацію даних для будь-якого поля JSON, і введення некоректних даних призводить до помилки. На відміну від MongoDB, PostgreSQL працює з більшістю сучасних мов програмування, таких як Python, Groovy, JavaScript, Java, C, C++, Perl та іншими.

PostgreSQL тепер підтримує різні формати даних, включаючи JSON, ключ-значення та XML, що також підтримуються в MongoDB. Це робить PostgreSQL хорошою альтернативою деяким нереляційним базам даних. Крім того, PostgreSQL забезпечує взаємодію з іншими джерелами даних, такими як Oracle, MySQL, MongoDB, CouchDB, Redis, Neo4j, Twitter, LDAP, файли, Hadoop та інші. Це дозволяє зберігати дані в PostgreSQL та інтегрувати їх з іншими системами.

У загальному, Postgres NoSQL виявляється більш вигідним в порівнянні з іншими системами баз даних NoSQL, такими як MongoDB, завдяки своєму здатністю інтегрувати функціонал як SQL, так і NoSQL. У наступних абзацах буде розглянуто технічні відмінності між PostgreSQL та MySQL. Порівнюючи продуктивність, PostgreSQL значно випереджає будь-яку NoSQL базу даних. Незважаючи на це, MySQL також демонструє високу продуктивність, особливо при роботі з великими навантаженнями, хоча це не завжди було таким. У минулому MySQL здобула репутацію швидкої бази даних завдяки своїй паралельній обробці, тоді як Postgres, навпаки, зазвичай показує збалансовані результати, використовуючи паралельну обробку. Однак з останніми версіями обох систем усі ці відмінності стираються. Порівнюючи рівень паралелізму, PostgreSQL виходить переможцем, оскільки вона краще впорядковує паралельну обробку за допомогою Multiversion Concurrency Control. Постгрес також славиться своєю розширюваністю, підтримуючи широкий спектр типів

даних, включаючи геометричні/ГІС, мережеві адреси, JSONB, UUID та мітки часу з урахуванням часового поясу, що недоступні для MySQL. Щодо документації, PostgreSQL виграє в очах багатьох користувачів завдяки своїй детальній інформації, яку можна знайти в її документації, що порівняно з MySQL є значно кращою. Щодо адміністрування, MySQL пропонує простоту в управлінні, але це часто обмежується вузькими можливостями. У відміну від цього, PostgreSQL легко справляється зі складними запитами, хоча й вимагає більше часу для правильної настройки і налагодження. В кінцевому підсумку, обидві системи мають свої сильні та слабкі сторони, і вибір між ними залежатиме від конкретних потреб проекту.

Отже, висновок, що PostgreSQL є ідеальним вибором для даного проекту, можна зробити наступним чином:

— типи даних: PostgreSQL підтримує всі необхідні типи даних, включаючи документи, примітиви, геометрію, структури тощо;

— цілісність даних: Postgres забезпечує цілісність даних через обмеження та регулювання додаваних даних, що дозволяє уникнути недійсності чи втрати записів;

— продуктивність: PostgreSQL здійснює розпаралелювання запитів на читання, використовує потужні методи індексування та багатоверсійний контроль паралельності для оптимізації продуктивності;

— інтернаціоналізація та текстовий пошук: PostgreSQL підтримує міжнародні набори символів, використовує повнотекстовий пошук для швидшого пошуку та інтегрує сортування без урахування регістру та наголосу.

— підтримка не реляційних даних: PostgreSQL забезпечує підтримку документів JSON, XML, Hstore і Cstore, що значно розширює його функціонал до бази даних типу NoSQL.

Ці аспекти роблять PostgreSQL ідеальним вибором для проекту, оскільки база даних поєднує в собі властивості традиційних реляційних баз даних із здатністю ефективно опрацьовувати неструктуровані дані, що є суттєвим для сучасних додатків та сервісів.

2 ВИБІР ЕЛЕМЕНТНОЇ БАЗИ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ SMART-ГОДИННИКА

2.1 Обґрунтування вибору компонентів

Для описаної апаратної частини проекту, основне завдання якої полягає в зчитуванні даних з карток співробітників на прохідній (рисунок 2.1) і їх відправці на сервер для подальшої обробки і зберігання, потрібно врахувати наступні умови та вимоги:

1. Зчитування тегу RFID та відправка на сервер:

— пристрій має зчитувати тег RFID і відправляти його на сервер через HTTP-запити.

2. Відправка мітки часу у форматі UNIX та синхронізація часу:

— повинна підтримуватися відправка мітки часу у форматі UNIX;

— пристрій має синхронізувати свій час з сервером для точності.

3. Збереження тегів у випадку недоступності сервера:

— якщо сервер недоступний, теги повинні зберігатися в пам'яті пристрою;

— після відновлення мережевого зв'язку пристрій має відправити збережені теги на сервер;

— необхідно зберегти принаймні 300 записів у разі перерви зв'язку.

4. Підтримка статичних та динамічних IP-адрес:

— проект повинен підтримувати налаштування як статичних, так і динамічних IP-адрес для пристрою.

5. Характеристики станцій входу/виходу:

— станції повинні бути компактними і працювати від будь-якого джерела живлення 5 В (PowerBank або через USB);

— потужність, необхідна для роботи пристрою, становить 300 мА;

— корпус станції можна виготовити з акрилу, фанери чи інших матеріалів (рисунок 2.2).

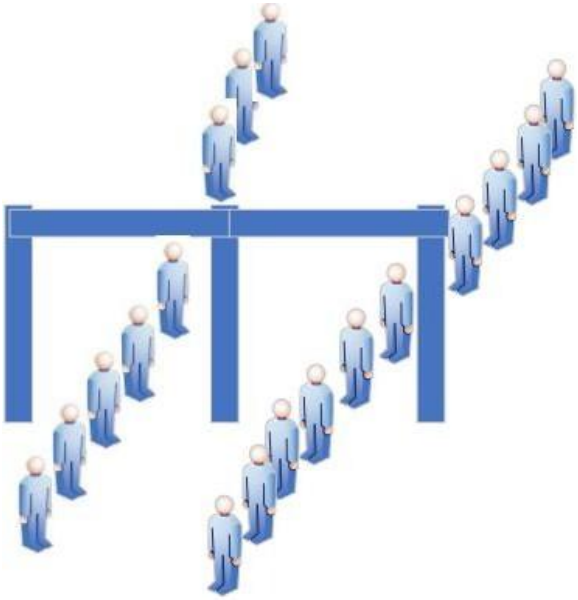

Рисунок 2.1 — Концепція прохідної

Рисунок 2.2 — Концепція корпусу станції

6. Використовувані компоненти:

— мікроконтролер: ATmega2560 (з серії Arduino Mega);

— Ethernet Shield: W5100 для забезпечення мережевого зв'язку;

— зчитувач RFID: Mifare RC522 (13.56MHz) або аналогічний;

— EEPROM: AT24C32 з інтерфейсом I2C для зберігання даних;

— динамік для аудіоіндикації.

Загальний зовнішній вигляд станції повинен бути зручним для щоденного використання і легким у виготовленні. Описані компоненти та вимоги забезпечують оптимальну функціональність і надійність системи для збору і передачі даних із зчитувачів RFID на сервер.

Atmega2560 зазвичай використовується в Arduino Mega 2560 як основний мікроконтролер (див. таблицю 2.1). Це мікроконтролер на базі AVR RISC, який виконує потужні команди за один такт [19]. Це дозволяє досягти гарного балансу між енергоспоживанням і швидкістю обробки.

Таблиця 2.1 — Характеристики Atmega2560

Параметр	Значення
Розмір програмної пам'яті (Кб)	256
Швидкість ЦП (MIPS/DMIPS)	16
SRAM (КБ)	8,192
Комунікаційна периферія	4-UART, 5-SPI, 1-I2C
Діапазон температур (°C)	-40 до 85
Діапазон робочої напруги (В)	1,8 до 5,5
Таймери	2x 8-біт, 4 x 16-біт

Переваги контролера Atmega2560 включають:

— низьке енергоспоживання при швидкому запуску;

— простота використання, оскільки 8-розрядний мікроконтролер менш складний, ніж 32/64-розрядні версії;

— використання qtouch suite для легкого дослідження, розробки та налагодження сенсорних програм.

Недоліки контролера Atmega2560 включають:

— обмежена кількість циклів запису флеш-пам'яті, що обмежує кількість разів перепрограмування, якщо програма часто змінюється;

— обмежена продуктивність порівняно з мікроконтролерами з вищими розрядами.

Arduino Mega 2560 (див. рисунок 2.3) відомий своїми можливостями для складних проектів, таких як 3D-принтери і робото-технічні системи. Він надає проектам багато простору і можливостей завдяки 54 цифровим входам/виходам, 16 аналоговим входам і значній обсяговій пам'яті.

Рисунок 2.3 — Платформа Arduino Mega 2560

Ось технічні характеристики платформи Arduino Mega 2560:

— робоча напруга: 5В;

— вхідна напруга (рекомендована): 7-12В;

— вхідна напруга (ліміти): 6-20В;

— цифрові контакти вводу/виводу: 54 (з яких 14 забезпечують вихід ШІМ);

— аналогові входи: 16;

— постійний струм введення/виведення на кожному контакті: 40 мА;

— постійний струм для контакту 3,3В: 50 мА;

— флеш-пам'ять: 256 КБ (8 КБ використовується завантажувачем);

— SRAM: 8 КБ;

— тактова частота: 16 МГц.

Щодо Ethernet Shield W5100 (див. рисунок 2.4) він дозволяє здійснювати провідне підключення Arduino до мережі Ethernet і сумісний з платформами Uno та Mega [20].

Рисунок 2.4 — Плата Ethernet Shield W5100

Також, Ethernet Shield W5100 має можливість швидкого підключення за допомогою стандартної бібліотеки Arduino Ethernet. Цей модуль також вбудований з пристроєм для читання SD-карт, що дозволяє зручно зберігати отримані дані (див. рисунок 2.5).

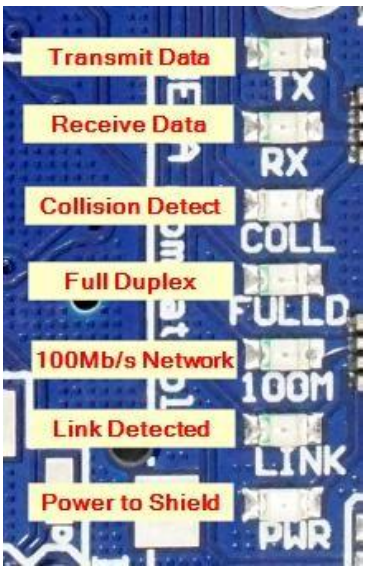

Рисунок 2.5 — Світлодіоди на платі Ethernet Shield W5100

Ця плата забезпечує можливість підключення Arduino до Інтернету або локальної мережі, використовуючи мікросхему Wiznet W5100 Ethernet. Чіп Wiznet W5100 підтримує протоколи TCP і UDP і може утримувати до 4 з'єднань, що можна тримати відкритими одночасно.

Модуль має наступні світлодіоди індикації:

— pwr: показує, що живлення увімкнено;

— link: вказує на успішне підключення до мережі;

— fulld: вказує на повне дуплексне підключення;

— 100m: увімкнено, коли виявлено мережу 100 Мбіт/с;

— rx: показує, що дані отримані;

— tx: світлодіод для індикації надсилання даних;

— coll: вказує на виявлення мережевого конфлікту.

RFID подібний до інших технологій бездротового зв'язку, таких як радіопередавачі і Bluetooth [21]. Системи RFID складаються з двох основних компонентів: тегів і зчитувачів (див. рисунок 2.6). Теги містять дані, а зчитувачі виявляють теги і обробляють інформацію, що міститься на них, коли вони знаходяться в діапазоні зчитування.

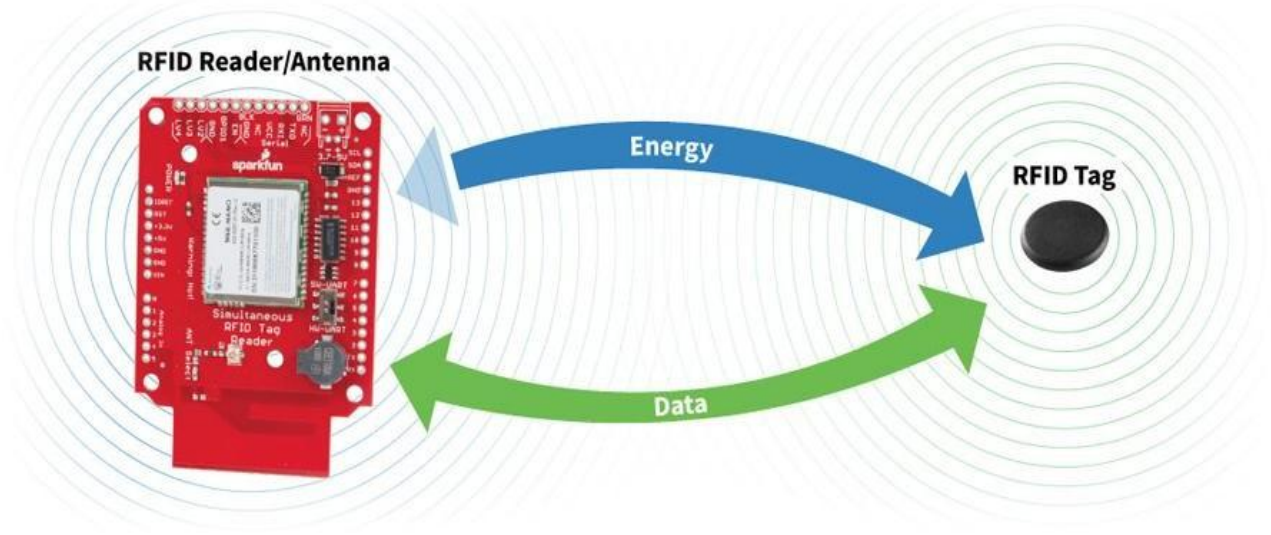

Рисунок 2.6 — Пояснення RFID технології

Теги мають обмежений обсяг пам'яті, де зберігається унікальний ідентифікатор тегу (TID), який не можна змінювати (див. рисунок 2.7). Малий обсяг пам'яті на тегах може бути доступний тільки для читання або запису.

Рисунок 2.7 — Бузер

Принциповою функцією пристроїв автосигналізації, таких як біпери або зумери, є перетворення електричного сигналу на звуковий. Ці пристрої можуть бути реалізовані у вигляді електромеханічних, п'єзоелектричних або механічних засобів.

Зазвичай вони живляться від постійного струму та використовуються у різноманітних пристроях: від таймерів до комп'ютерів і сигналізаційних систем. Залежно від конструкції, ці пристрої можуть генерувати різні звуки, такі як будильник, музика, дзвінок або сирена.

Вибір відповідного зумера варто проводити з урахуванням таких параметрів, як напруга живлення, сила струму, режим руху, розмір і метод підключення. Крім цього, важливими параметрами є звуковий тиск і частота звуку.

Наприклад, робоча напруга електромагнітного зумера може коливатися від 1,5 до 24 В, а п'єзоелектричного - від 3 В до 220 В. Для досягнення кращої якості звуку рекомендується вибирати п'єзоелектричні зумери з напругою більше 9 В.

На рисунку 2.8 показано фінальну компоновку станції, де можна побачити розміщення вибраного зумера та інших елементів системи автосигналізації.

Щодо використання бібліотек у вашому проекті Arduino, ось короткий опис кожної з них і їх призначення:

— ds3231: ця бібліотека призначена для роботи з модулем реального часу DS3231. Вона дозволяє зчитувати і встановлювати час і дату.

— spi: це базова бібліотека, яка потрібна для роботи з іншими бібліотеками, що використовують SPI (Serial Peripheral Interface), наприклад, Ethernet Shield.

— ethernet: бібліотека для роботи з Ethernet Shield, яка забезпечує можливість підключення Arduino до мережі Ethernet.

— simpletimer: ця бібліотека дозволяє вам створювати таймери та виконувати завдання в заданих інтервалах часу, що корисно для псевдопаралельного виконання завдань на Arduino.

— mfrc522: бібліотека для роботи з RFID-зчитувачем MFRC522. Вона надає функції для зчитування та запису даних на RFID-тегах.

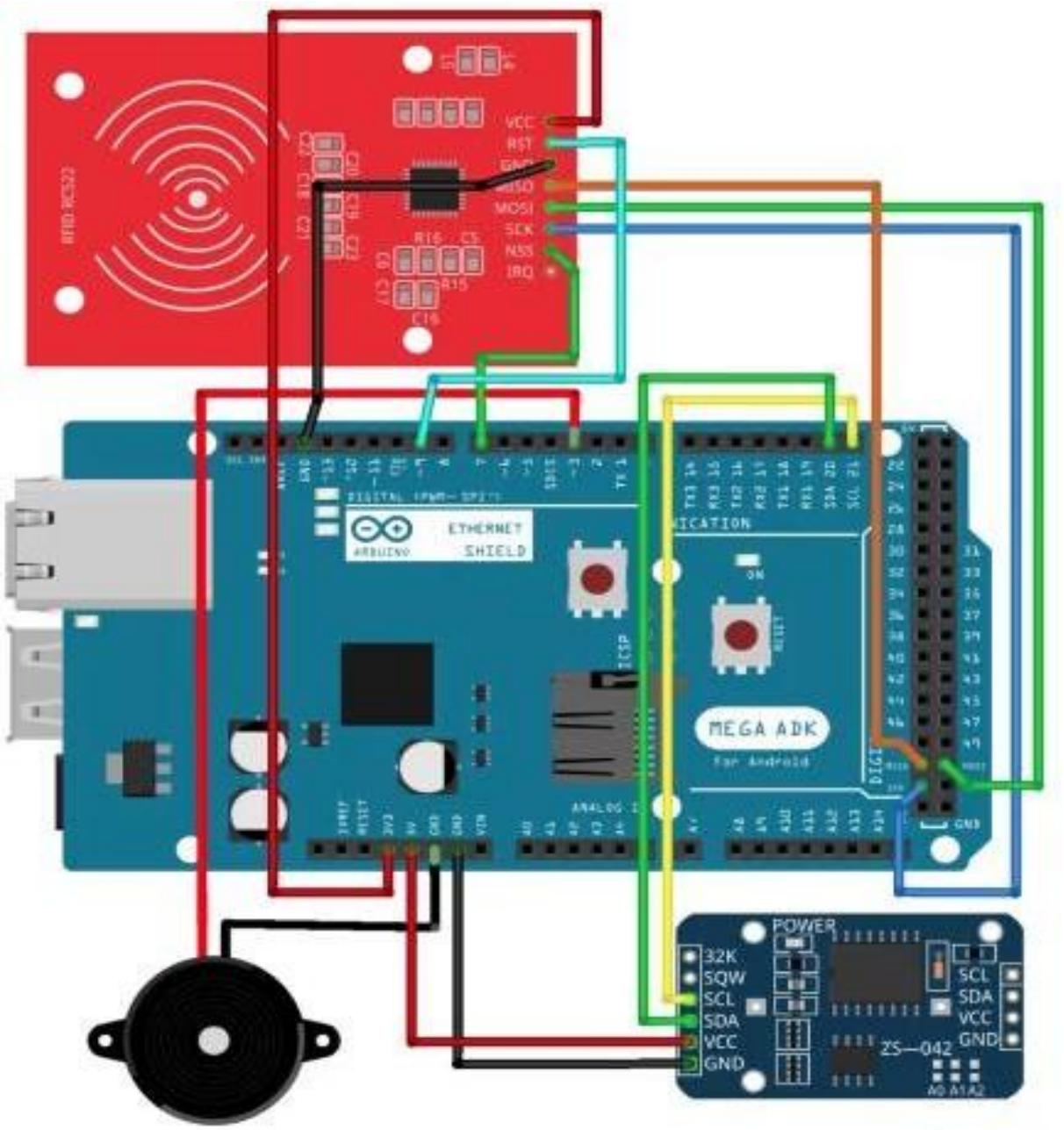

Рисунок 2.8 — Функціональна схема

— onewire: ця бібліотека використовується для роботи з пристроями, які підтримують протокол OneWire, але в вашому випадку вона не потрібна, якщо ви використовуєте інші способи для цілісності даних.

— memoryfree: ця бібліотека допомагає управляти оперативною пам'яттю Arduino, зокрема, вона може допомогти вам виявити і уникнути витоків пам'яті.

— pgmStrToRAM: ця бібліотека дозволяє зберігати текст, збережений в пам'яті FLASH, в оперативній пам'яті (RAM), щоб ви могли працювати з ними швидше і зручніше.

— eprom: бібліотека для роботи з EEPROM (Electrically Erasable Programmable Read-Only Memory), яка дозволяє зберігати дані, що мають залишитися навіть після вимкнення живлення Arduino.

Для використання цих бібліотек вам потрібно буде включити їх у вашу прошивку Arduino IDE і користуватися їхніми функціями для реалізації потрібних функцій у вашому проекті. Наступною кроком буде створення діаграми системи, яка може показати структуру вашої програми, включаючи взаємозв'язки між різними модулями та бібліотеками.

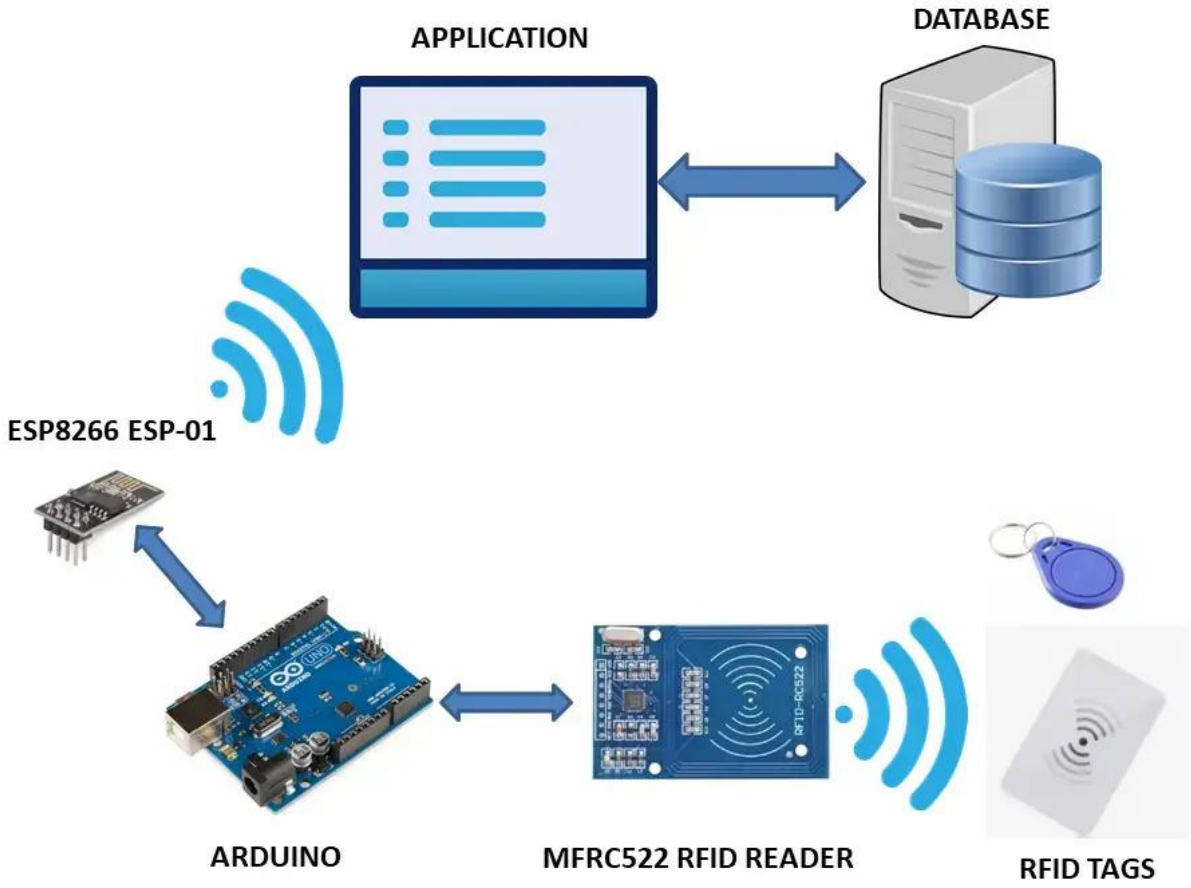

Рисунок 2.9 — Діаграма взаємодії модулів

Клієнт-серверна архітектура є основою для багатьох сучасних систем, включаючи RESTful API. Основні принципи цієї архітектури передбачають, що клієнт і сервер працюють незалежно один від одного, що забезпечує

відмовостійкість і високу масштабованість системи. Ось загальний алгоритм роботи клієнт-серверної архітектури:

— клієнт надсилає запит: клієнт ініціює запит до сервера через мережевий пристрій, як правило, за допомогою HTTP або іншого протоколу зв'язку;

— мережевий сервер приймає та обробляє запит: сервер отримує запит від клієнта через мережеве підключення, це може бути HTTP запит на конкретний URL або інший тип запиту в залежності від використовуваного протоколу;

— сервер обробляє запит: сервер виконує необхідні операції для обробки запиту, такі як доступ до бази даних, обробка логіки додатку або виконання певної операції;

— сервер генерує відповідь: після обробки запиту сервер генерує відповідь, яка може включати дані, які клієнт запитує, або статус операції, що відбулася;

— сервер відправляє відповідь клієнту: остаточна відповідь від сервера передається через мережеве з'єднання назад до клієнта.

У контексті RESTful API, цей процес використовує HTTP методи (GET, POST, PUT, DELETE) для здійснення операцій над ресурсами на сервері. Архітектура забезпечує розділення обов'язків між клієнтом, який ініціює запити, і сервером, який забезпечує реалізацію бізнес-логіки та обробку даних.

Ця модель дозволяє покращувати масштабованість, оскільки сервер може обробляти багато запитів від різних клієнтів паралельно. Вона також забезпечує відмовостійкість, оскільки відмова одного клієнта не впливає на інших, і навпаки.

Це стандартний підхід для розробки сучасних додатків, які використовують мережеві технології для обміну даними і послугами між клієнтами та серверами.

Сучасні технології стрімко розвиваються, що змушує сучасний бізнес активно використовувати IT для свого процвітання і збереження конкурентоспроможності. Організаціям важлива ефективна система, яка полегшує збір і обробку корпоративних даних, підвищуючи продуктивність

бізнес-процесів і забезпечуючи стійкість на ринку [26]. Модель «клієнт-сервер» забезпечує високий рівень обробки, що сприяє підвищенню продуктивності робочих станцій, розширенню можливостей робочих груп, дистанційному управлінню мережею, а також відповідності бізнес-потребам на ринку і збереженню інвестицій.

Ця архітектура надає сучасним організаціям потрібну структуру для вирішення швидкозмінюваних викликів IT-світу. Вона включає в себе використання різних апаратних і програмних ресурсів для клієнтських та серверних машин, можливості горизонтального та вертикального масштабування, а також пряму взаємодію програм на рівні транспортного протоколу для ефективної обміну інформацією між клієнтами і серверами.

Архітектура клієнт-сервер відіграє ключову роль у сучасному цифровому світі через свою широку застосовність. Ось три приклади, як вона використовується:

— сервери електронної пошти: ці сервери, використовуючи спеціальне програмне забезпечення, обмінюються електронними листами між користувачами, легкість та швидкість такого обміну зробили електронну пошту основною формою корпоративного спілкування, витіснивши традиційну пошту;

— файлові сервери: вони є централізованими місцями для зберігання файлів, доступних для багатьох клієнтів, наприклад, хмарні служби, такі як Google Docs або Microsoft Office, використовуються мільйонами користувачів як файлові сервери для спільної роботи над документами;

— веб-сервери: ці сервери господарюють багато різних веб-сайтів і надають доступ до них через Інтернет, процес доступу до веб-ресурсів включає кроки, такі як запит DNS для IP-адреси сервера та обмін даними через HTTP або HTTPS.

Переваги архітектури клієнт-сервер включають централізоване зберігання даних, високу масштабованість, ефективність та економічність. З іншого боку, існують певні недоліки, такі як вразливість до вірусів і атак типу DoS,

можливість підроблення даних під час передачі, а також ризики, пов'язані з виходом з ладу критично важливих серверів і атаками MITM.

Трирівнева архітектура клієнт-сервер (див. рисунок 2.10) складається з трьох рівнів: рівень презентації, відомий як інтерфейс користувача; рівень додатків, відомий як рівень обслуговування; та рівень бази даних, який включає сервер бази даних. Ця архітектура може бути розділена на три основні частини.

Рисунок 2.10 — Архітектура клієнт-сервер трьох рівнів

На рівні презентації (клієнтському рівні) зосереджується на інтерфейсі користувача. Прикладний рівень (бізнес-рівень) забезпечує обробку даних. Рівень бази даних (рівень даних) відповідає за зберігання інформації. У такій архітектурі клієнтська система керує рівнем презентації, сервер додатків відповідає за рівень додатків, а серверна система управляє рівнем бази даних.

Однорангові мережі, також відомі як P2P-мережі, складаються з групи комп'ютерів, які об'єднуються у мережу, де кожен пристрій може виступати як клієнт і як сервер одночасно. У такій мережі всі користувачі мають однакові обов'язки і права щодо обробки даних. Це значно відрізняється від клієнт-серверної моделі, де є чітка роль сервера, що надає послуги, і клієнта, що їх використовує.

Для кращого розуміння можна подати такий приклад: у клієнт-серверній моделі замовлення гамбургера у ресторані швидкого харчування відбувається через спілкування зі стійкою, де працює обслуговуючий персонал. У одноранговій мережі цей процес може відбуватися без прямого втручання персоналу, де інші користувачі можуть взяти на себе обробку вашого замовлення.

Основні відмінності між цими двома типами мереж включають:

— клієнт-серверні мережі вимагають централізованого сервера, що робить їх дорожчими у впровадженні, тоді як в однорангових мережах всі пристрої рівноправні;

— у клієнт-серверних мережах роль користувачів і постачальників послуг чітко визначена, тоді як в однорангових мережах всі користувачі мають однакові функції;

— клієнт-серверні мережі зазвичай пропонують вищий рівень безпеки, що робить їх більш надійними, в той час як у однорангових мережах безпека залежить від кожного користувача;

— в однорангових мережах зі зростанням кількості активних вузлів може знижуватись продуктивність, тоді як клієнт-серверні мережі забезпечують кращу стабільність і масштабованість;

— у однорангових мережах обмін файлами може бути швидшим і простішим, ніж у клієнт-серверних мережах;

— коли сервер у клієнт-серверній мережі виходить з ладу, доступ до послуг припиняється, в той час як у випадку виходу з ладу одного вузла у P2P-мережі інші вузли можуть продовжувати працювати незалежно.

Ці різниці визначають, яку модель мережі краще використовувати залежно від конкретних потреб і вимог проекту.

Протокол передачі даних (ПТД) визначає угоди на логічному рівні, які регулюють обмін даними між програмами, визначаючи, які дані передаватимуться, які частини системи будуть задіяні, і як інформація буде оброблятися. Протокол HTTP дозволяє передавати різноманітні гіпермедійні ресурси, такі як HTML документи, і був розроблений для зв'язку між браузерами та серверами, що становить основу обміну даними в Інтернеті. HTTPS є захищеною версією HTTP, де буква "S" означає шифрування, що ускладнює несанкціонований доступ до даних користувача, які передаються на сервер через HTTP.

Протокол FTP використовується для передачі файлів між комп'ютерами в мережі, тоді як POP3 є найпоширенішим протоколом для отримання

електронної пошти. За допомогою POP3 клієнт може завантажувати електронні листи на свій пристрій (наприклад, комп'ютер або смартфон) і видаляти їх з сервера. Клієнтська програма встановлює зв'язок з сервером, надсилаючи запити, на які сервер відповідає певними кодами стану.

Веб-протокол HTTP (Hypertext Transfer Protocol) використовує різні коди стану для передачі інформації про результати виконання запитів. Ось детальний огляд основних категорій кодів «Білий мішок»,стану HTTP:

1. Інформаційні коди (100-105):

—100 Continue — сервер задоволений деталями запиту, і клієнт може продовжувати надсилати дані;

—102 Processing — запит прийнятий, але обробка займає багато часу. Цей код використовується сервером для попередження клієнта про тривалу обробку.

2. Коди успіху (200-226):

— 200 OK — запит виконано успішно;

— 203 Non-Authoritative Information — запит виконано успішно, але отримана інформація походить з вторинного джерела через трансформуючий проксі-сервер;

— 204 No Content — запит виконано успішно, але відповідь не містить контенту для надсилання.

3. Повідомлення про переадресацію (300-307):

— 301 Moved Permanently — запитуваний документ переміщений на нову URL-адресу.

4. Помилки клієнта (400-499):

- 400 Bad Request — сталася синтаксична помилка під час отримання запиту клієнта.

— 401 Unauthorized — для доступу до ресурсу потрібна автентифікація;

— 403 Forbidden — сервер розуміє запит, але відмовляється його виконувати через обмеження доступу клієнта до ресурсу;

— 404 Not Found — запитуваний ресурс не знайдено.

5. Помилки сервера (500-510):

— 500 Internal Server Error — внутрішня помилка сервера;

— 502 Bad Gateway — сервер, який діє як шлюз або проксі-сервер, отримав недійсне повідомлення від вищестоящого сервера;

— 503 Service Unavailable — сервер тимчасово не може обробити запити з технічних причин;

— 504 Gateway Timeout — час очікування сервера на відповідь від шлюзу перевищив встановлений ліміт.

Ці коди стану допомагають регулювати взаємодію між клієнтськими програмами і серверами, повідомляючи про результати обробки запитів і вказуючи на наявні проблеми або успіх.

2.2 Обґрунтування вибору хостингу

Amazon Web Services (AWS) перевернули правила гри у світі веб-хостингу, ставши обраною платформою для багатьох великих і авторитетних компаній, включаючи Netflix, готелі Kempinski і навіть NASA! Це свідчить не лише про надійність AWS, але й про його потужність як віртуального рішення для веб-хостингу для організацій [28]. AWS, як офіційна дочірня компанія Amazon, заснована у 2002 році, пропонує хмарний підхід до веб-хостингу на вимогу. Ця модель платної підписки включає безкоштовний перший рік, під час якого користувачі мають доступ до повного набору віртуальних комп'ютерів через Інтернет. Вони мають всі стандартні характеристики, включаючи попередньо завантажене ПЗ та вибір операційних систем.

Користувачі отримують доступ до своєї AWS системи через веб-браузер, що дозволяє налаштовувати веб-сайти та керувати обліковим записом. AWS існує на серверних кластерах по всьому світу, підтримуваних дочірньою компанією Amazon.

Як найбільший у світі публічний сервіс хмарних обчислень, AWS займає домінуючу позицію в цьому секторі, пропонуючи користувачам економічність і масштабованість. Ви можете платити лише за ті ресурси, які фактично

використовуєте, що забезпечує величезну економію. Це рішення особливо привабливе для тих, хто шукає способи оптимізації витрат без необхідності укладення попередніх контрактів або витрат на інфраструктуру, яка не використовується повністю.

AWS дозволяє додавати ресурси у мережу в залежності від поточного попиту, уникнувши проблем традиційного хостингу з його обмеженими можливостями масштабування.

AWS пропонує вражаючі можливості масштабування, що дозволяє йому перевершувати стандартні варіанти веб-хостингу. Ви з легкістю можете збільшувати чи зменшувати пропускну здатність в залежності від трафіку на вашому сайті, просто керуючись натисканнями кнопок у консолі AWS. Це гарантує, що AWS завжди доступний для миттєвого реагування на потреби вашого веб-проекту. Одна з ключових переваг - це велика гнучкість: ви можете обирати операційну систему, мову програмування та платформу веб-додатків за власним вибором.

Налаштування AWS дозволяє вам вибирати конкретне програмне забезпечення та сервіси, що відповідають вашим потребам. Це спрощує процес міграції існуючих програм і підвищує ефективність вашого рішення. AWS також пропонує гнучкість для створення нових рішень, розроблену з урахуванням простоти використання. Консоль AWS є добре спланованою та інтуїтивно зрозумілою, що дозволяє легко управляти вашим веб-сайтом.

Хоча для ефективної роботи з AWS потрібні певні знання, порівняно з традиційними методами розміщення, використання AWS є значним кроком вперед. Його архітектура дозволяє розробникам безпечно та ефективно розміщувати програми, незалежно від їх типу - чи це існуюча програма, чи нова на основі SaaS. Все це разом гарантує надійний хостинг вашого сайту та захист під час його онлайн-присутності.

AWS надає можливість орендувати свої комп'ютери користувачам по всьому світу через захищені центри обробки даних на всіх континентах, з'єднаних кабелями. Ця гнучкість дозволяє використовувати віртуальні

комп'ютери зі всіма необхідними атрибутами, такими як RAM, жорсткий диск/SSD, вибір операційної системи, мережа і попередньо завантажене прикладне програмне забезпечення, наприклад, веб-сервери та бази даних.

Кожна система AWS віртуалізує консоль з клавіатурою, дисплеєм і мишею, що дозволяє абонентам підключатися до своєї системи через браузер. Цей браузер діє як вікно у віртуальний комп'ютер, дозволяючи передплатникам налаштовувати та управляти своїми віртуальними системами так само, як з фізичним комп'ютером.

AWS легко масштабується вгору чи вниз в залежності від потреб абонента, дозволяючи використовувати будь-яку кількість серверів, яка потрібна. Це робить Amazon готовим впоратися з будь-якими вимогами до даних, зберігаючи при цьому розумні витрати, незалежно від масштабування інсталяції з часом.

Наприклад, Netflix, онлайн-платформа для прокату фільмів, використовує три-чотири тисячі серверів у мережі Amazon. Навіть якщо ваш веб-додаток тільки починає свою роботу, оренда одного або двох серверів може бути достатньою, оскільки не завжди потрібні великі серверні ресурси. Середня вартість сервера може становити менше фунта за годину, а дуже великий сервер — всього три пенси на годину.

Корпорація Майкрософт розробила свою власну платформу під назвою Azure, яка відрізняється від Amazon. В Azure також доступні віртуальні сервери, які можна орендувати, проте їхній підхід має свої відмінності. AWS від Amazon надає сервери кінцевим користувачам, незалежно від того, чи вони використовують Linux, чи Windows. Це робить бізнес-модель Amazon спочатку більш зручною для користувачів. AWS дозволяє використовувати будь-який код на орендованій системі під час створення програми, незалежно від платформи, знаючи, що клієнт поступово перейде до використання більшої кількості послуг Amazon і стане прив'язаним до них.

При створенні комп'ютерної системи для запуску веб-програми важливо, щоб система була постійно онлайн. Однак абсолютна безвідмовна надійність є

важкодосяжною метою. Статистика показує, що будь-який веб-сервер може мати збій раз на кілька років. Навіть якщо політика хостингової компанії передбачає негайну заміну несправного сервера та копіювання на нього останнього стану даних, це може зайняти кілька годин. Для уникнення цього ризику може бути використане повне дублювання ваших серверів.

Гнучкість упаковки Spring Boot надає великий вибір при розгортанні програм [34]. Ви можете розгорнути програми Spring Boot на різних хмарних платформах, на віртуальних або фізичних машинах, або зробити їх повністю виконуваними для Unix-систем. AWS Elastic Beanstalk забезпечує простий спосіб налаштування та розгортання програм у сервісах AWS. Існують два способи розгортання програм у середовищі Elastic Beanstalk: через консоль AWS або за допомогою EB CLI (командний рядок).

AWS Elastic Beanstalk є простим у використанні сервісом для розгортання та масштабування веб-додатків і служб, розроблених з використанням Java, .NET, PHP, Node.js, Python, Ruby, Go та Docker на стандартних серверах, таких як Apache, Nginx, Passenger і IIS (див. рисунок 2.11).

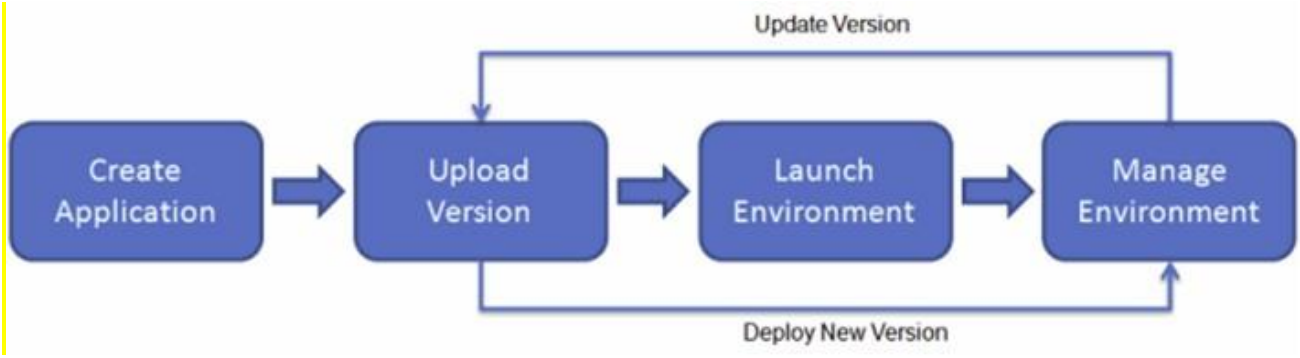

Рисунок 2.11 — Діаграма формування нової версії

Це найшвидший і найпростіший спосіб розгорнути вашу програму. За кілька хвилин програма буде готова до використання, без потреби в основній інфраструктурі або конфігурації ресурсів.

Для початку процесу хостингу вам потрібно згенерувати "fat jar" з усіма залежностями. Після успішної збірки файл JAR з'явиться у вашій папці. Важливо скопіювати цей файл JAR і зберегти його у зручному для вас місці.

Файл JAR є типом архіву, призначеним для пакування файлів класів Java і пов'язаних ресурсів для їх розповсюдження. Простими словами, файл JAR містить стислі версії файлів .class, аудіофайлів, зображень або каталогів. Його можна уявити як заархівований файл (.zip), який створюється за допомогою програмного забезпечення, наприклад, WinZip (див. рисунок 2.12). Навіть програмне забезпечення WinZip можна використовувати для роботи з вмістом файлів .jar, таких як стиснення даних без втрат, архівування та розпакування.

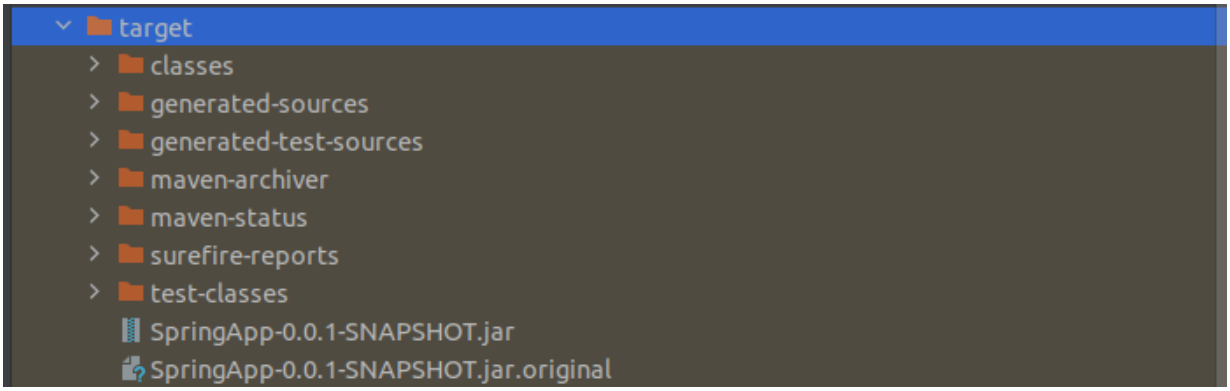

Рисунок 2.12 — Структура папки target

Наступним кроком буде розгорнути файл JAR в AWS Elastic Beanstalk. Для цього потрібно увійти в консоль керування AWS (див. рисунок 2.13) і перейти до служби Elastic Beanstalk.

Після входу в консоль керування AWS і переходу до служби Amazon Elastic Beanstalk, на домашній сторінці потрібно натиснути кнопку «Створити програму», щоб створити нову програму (див. рисунок 2.14). Потім вам буде запропоновано ввести назву програми в поле «Назва програми». Наприклад, ми можемо назвати додаток "my-spring-app".

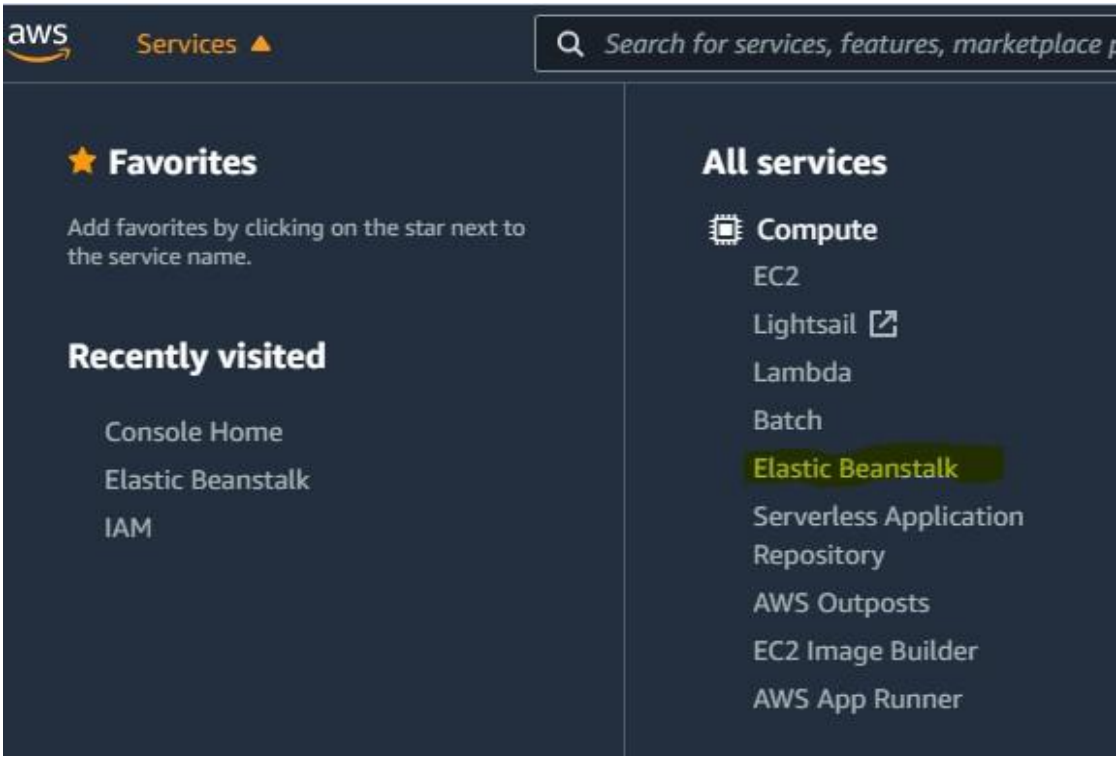

Рисунок 2.13 — Сервіси aws

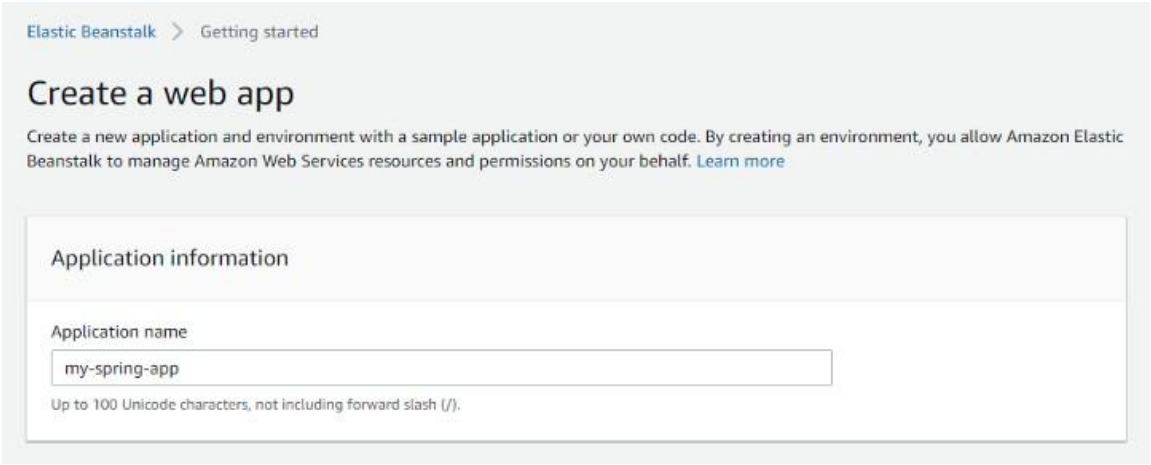

Рисунок 2.14 — Генерація додатку

Після створення нової програми у Amazon Elastic Beanstalk і введення назви програми «my-spring-app», наступним кроком є вибір деталей платформи в розділі «Платформа». Оскільки ми плануємо розгортати програму на Java, обираємо платформу Java (див. рисунок 2.15).

Далі, у розділі «Код програми», вибираємо опцію «Завантажити свій код».

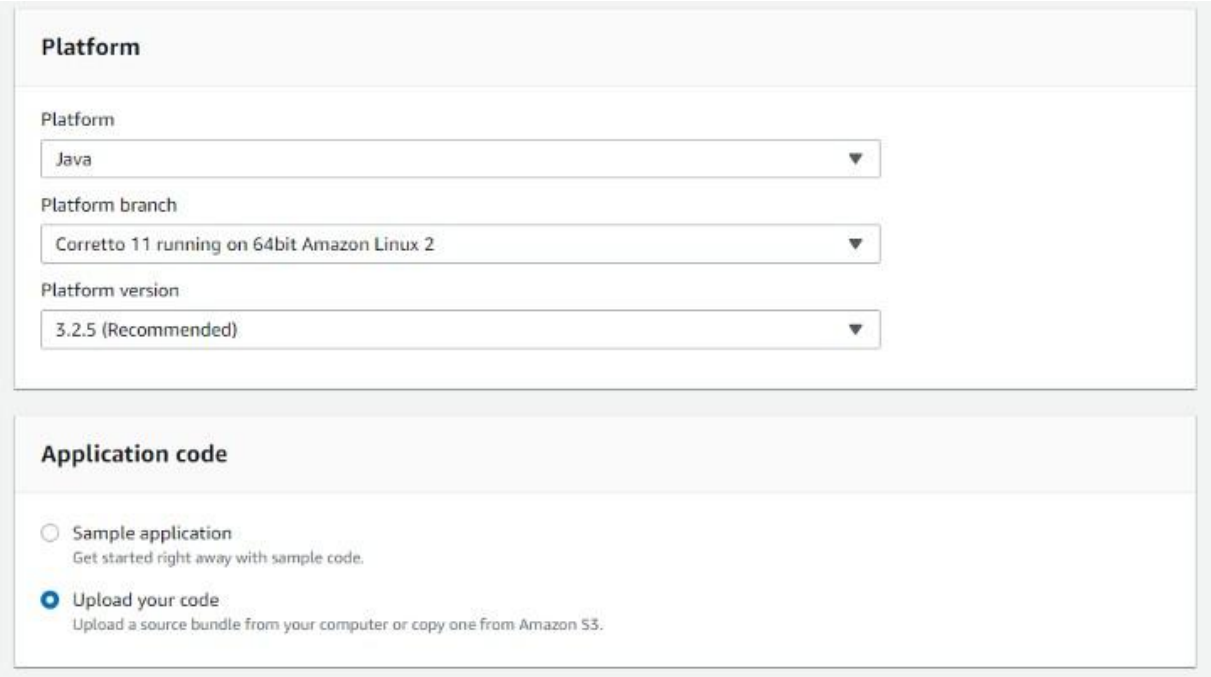

Рисунок 2.15 — Завантаження свого коду

Останнім етапом є вибір jar-файлу вашої Spring програми, який був створений під час збірки за допомогою Maven. Цей файл необхідно завантажити для подальшого розгортання в Amazon Elastic Beanstalk, як показано на екрані (рисунок 2.16).

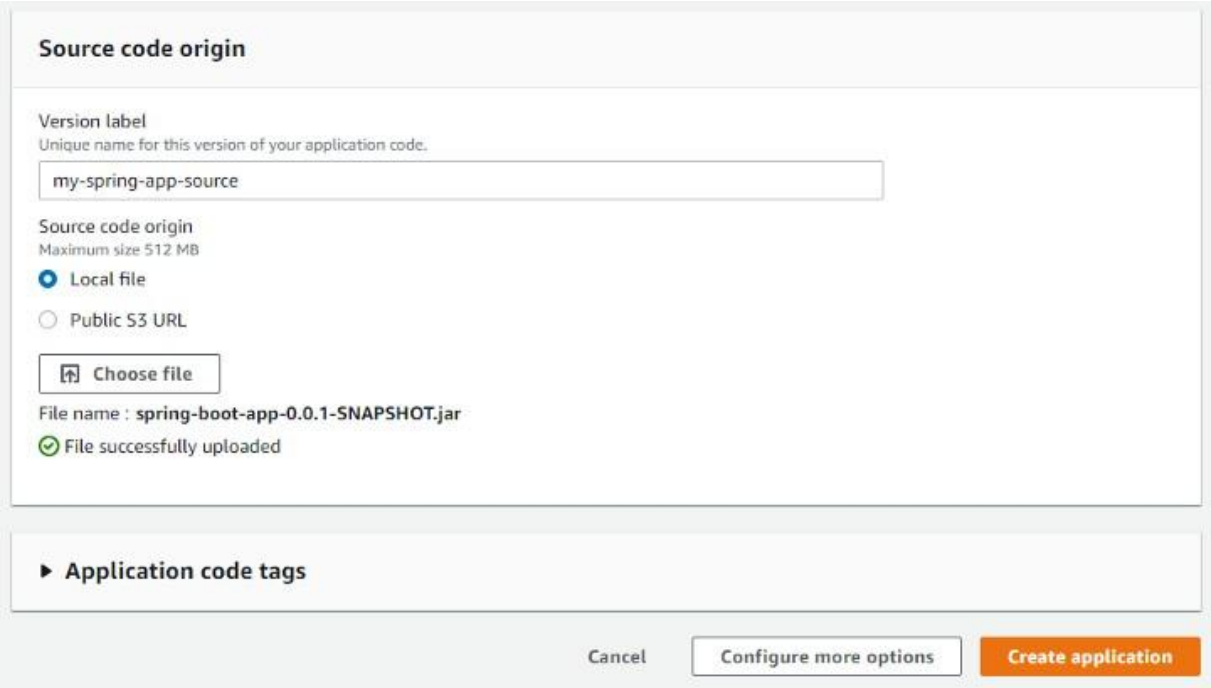

Рисунок 2.16 — Завантаження свого jar

Далі треба натиснути кнопку "Створити програму", і розгортання програми в AWS Elastic Beanstalk розпочнеться (рисунок 2.17).

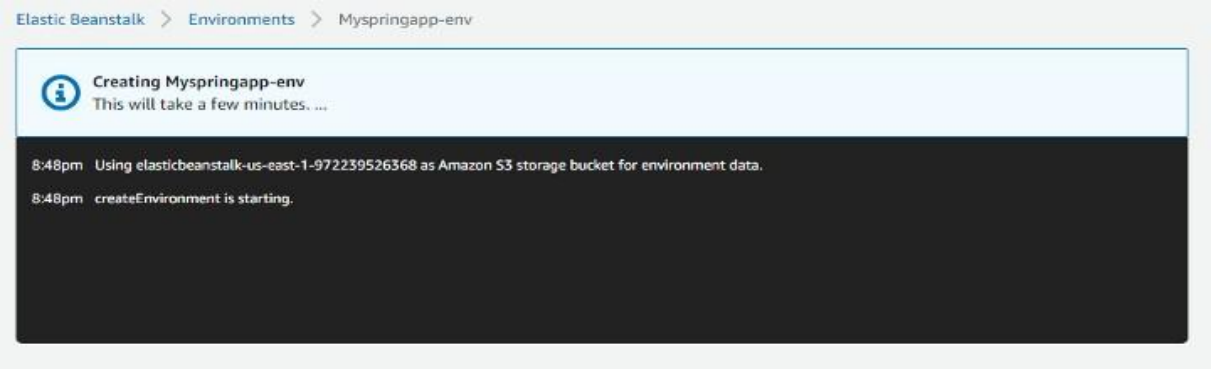

Рисунок 2.17 — Вебконсоль

Після успішного завершення процесу розгортання програми можна побачити відповідний запис у розділі "Середовища", зображеному на рисунку 2.18.

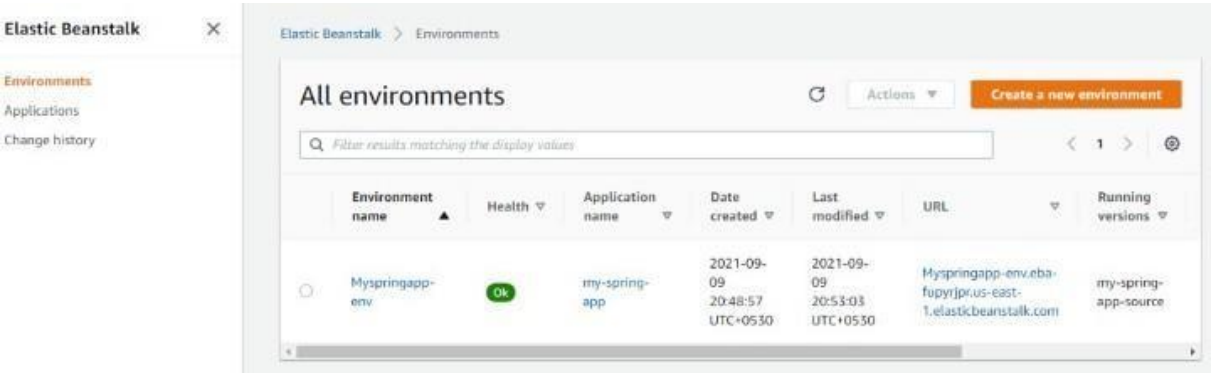

Рисунок 2.18 — Генерація додатку

Також видно, що програма працює нормально, і на екрані відображаються останні події (рисунок 2.19).

Рисунок 2.19 — Поточний стан додатку

Ви можете видалити екземпляр програми, просто обравши опцію "Видалити програму" у меню "Дії" (рисунок 2.20). Після цього AWS може потребувати певного часу на завершення операції видалення.

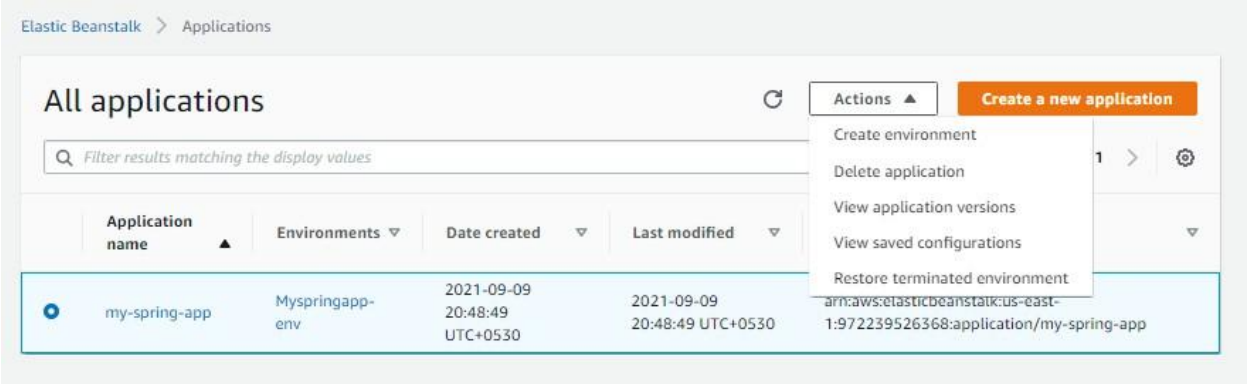

Рисунок 2.20 — Набір операцій для додатку

Основні переваги Elastic Beanstalk полягають у автоматизації конфігурації сервера, що значно економить час. Ця автоматизація охоплює всі необхідні завдання для робочої програми, такі як налаштування ротації файлів журналу, конфігураційних файлів nginx, налаштування служби Puma, встановлення пакетів Linux, Ruby та налаштування балансувальника навантаження, а також налаштування баз даних.

Хоча існують інші сервіси, такі як Heroku, Engine Yard та інші, які також надають схожі можливості, Elastic Beanstalk вирізняється своєю відмінною автоматизацією та контролем. Проте важливо враховувати деякі недоліки Elastic Beanstalk, такі як нестабільність розгортання, відсутність документації щодо оновлення стеку та додатків, а також загальна відсутність чіткої документації.

Покращення процесу розгортання за допомогою контейнерів, зокрема Docker, може забезпечити ще більшу універсальність та контроль над технологіями. Завдяки цим можливостям, Elastic Beanstalk дозволяє легко розгортати та оновлювати програми, що робить його надійним вибором для тих, хто прагне зменшити системні операції і зосередитися на розробці.

3 ПОКРАЩЕННЯ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ТА ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ SMART-ГОДИННИКА

3.1 Реалізація методу обробки сигналів

RFID (радіочастотна ідентифікація) — це швидко розвиваюча технологія, яка дозволяє бездротову ідентифікацію та відстеження за допомогою тегів і читачів. Вона є однією з найшвидше зростаючих галузей автоматичної ідентифікації та збору даних (AIDC). Сьогодні RFID використовується у сотнях, як не у тисячах застосувань і стає ефективною технологією управління ланцюгом поставок, замінюючи традиційні штрих-коди.

Зниження собівартості важливе для впровадження мікрохвильових систем RFID, особливо активних тегів, щоб мінімізувати їхнє енергоспоживання.

Пристрої RFID ідентифікуються за аналогією з штрих-кодами чи магнітними смужками на картках. Вони зберігають унікальний ідентифікатор для об'єктів і можуть бути скановані для отримання інформації про них.

Одним з ключових параметрів міток RFID є їх діапазон читання — максимальна відстань, на якій читач може зчитати інформацію з тега. Цей діапазон залежить від чутливості читача, орієнтації тега, матеріалу, до якого тег прикріплений, і середовища. Діапазон читання r можна розрахувати за допомогою формули вільного простору Фріїса:

$$r = \frac{\lambda}{4\pi}\sqrt{\frac{EIRP \cdot G_r \cdot \tau}{P_{chip}}},$$

де λ — довжина хвилі, а EIRP — еквівалент ізотропного випромінювання потужності, який визначається місцем та правилами країни знаходження (EIRP = 3,3 Вт в Європі);

P — мінімальна порогова потужність, яка потрібна, щоб активувати RFID-чіп;

G — коефіцієнт посилення антени-мітки для приймання;

τ — коефіцієнт передачі потужності між антеною і чіпом.

RFID відрізняється своєю ефективністю і можливістю використання в різних умовах, що робить його важливим інструментом у сучасних технологіях індустрії та логістики.

RFID система включає в себе три основні компоненти (див. рисунок 3.1): зчитувач, тег і антенну. Взаємодія між антеною і зчитувачем називається downlink, а взаємодія між тегом і зчитувачем — uplink.

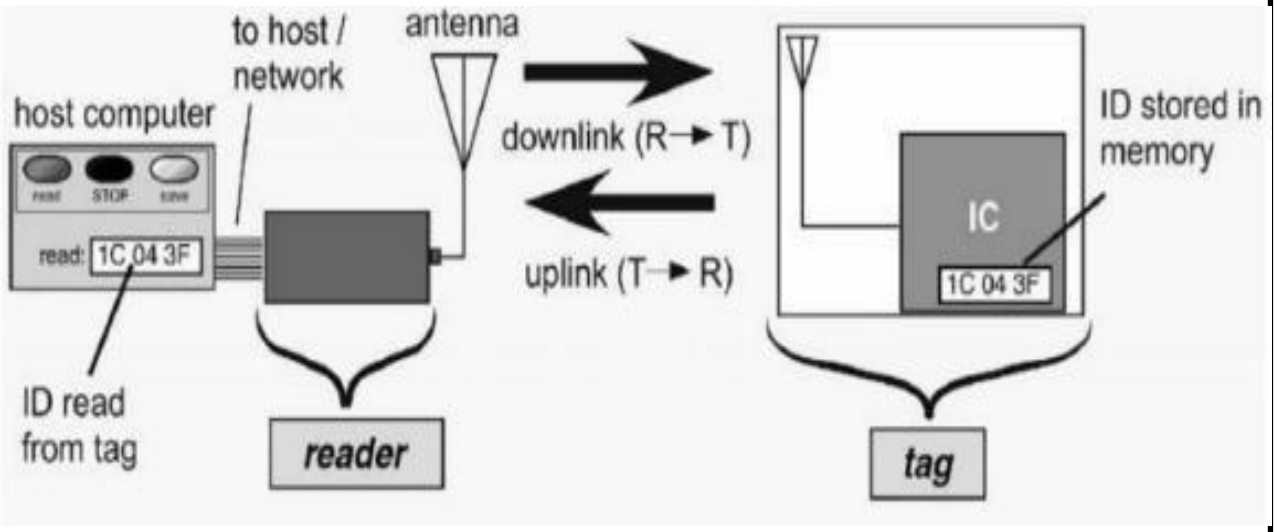

Рисунок 3.1 — Будова та принцип роботи RFID

Прикладаючи RFID тег до терміналу, працівник ідентифікується зчитувачем, який отримує інформацію про тег, включаючи його ідентифікатор, місцезнаходження та час. Оскільки працівники можуть проходити через кожен контрольно-пропускний пункт двічі, система може ненавмисно збирати дубльовані дані, що призводить до збільшення обсягу інформації.

$$(\text{ТЕГ1}, A, 3), (\text{ТЕГ2}, A, 2), (\text{ТЕГ3}, A, 4),$$
$$(\text{ТЕГ1}, A, 4), (\text{ТЕГ2}, A, 4), (\text{ТЕГ3}, A, 5),$$
$$(\text{ТЕГ1}, A, 5), (\text{ТЕГ2}, B, 5), (\text{ТЕГ3}, A, 7),$$
$$(\text{ТЕГ1}, B, 6), (\text{ТЕГ2}, B, 7), (\text{ТЕГ3}, D, 13),$$
$$(\text{ТЕГ1}, B, 9), (\text{ТЕГ2}, B, 8), (\text{ТЕГ3}, D, 16),$$
$$(\text{ТЕГ1}, C, 13), (\text{ТЕГ2}, D, 14), (\text{ТЕГ3}, E, 4),$$
$$(\text{ТЕГ1}, C, 9), (\text{ТЕГ2}, D, 8), (\text{ТЕГ3}, E, 19),$$
$$(\text{ТЕГ1}, C, 19), (\text{ТЕГ2}, D, 14), (\text{ТЕГ3}, E, 21).$$

Ідентифікатор тегу, знаходження та час виходу. Таким чином, кожен працівник ідентифікується лише один раз при вході і виході, що дозволяє уникнути дублювання інформації.

$$(\text{ТЕГ1}, A, 3, 5), (\text{ТЕГ2}, A, 2, 4), (\text{ТЕГ3}, A, 4, 7),$$
$$(\text{ТЕГ1}, B, 6, 9), (\text{ТЕГ2}, B, 5, 8), (\text{ТЕГ3}, D, 13, 16),$$
$$(\text{ТЕГ1}, C, 13, 19), (\text{ТЕГ2}, D, 14, 18), (\text{ТЕГ3}, E, 17, 19),$$
$$(\text{ТЕГ1}, K, 21, 32), (\text{ТЕГ2}, K, 19, 21), (\text{ТЕГ3}, K, 23, 25).$$

Оптимальним рішенням для зручності і читабельності буде трансформація попередніх записів у наступний формат, який відображає переміщення працівника по підприємству:

$$\text{ТЕГ1}: A[4,5] \rightarrow B[6,11] \rightarrow C[14,15],$$
$$\text{ТЕГ2}: A[7,11] \rightarrow B[12,13] \rightarrow C[16,18],$$
$$\text{ТЕГ3}: A[22,5] \rightarrow B[6,11] \rightarrow C[17,19],$$
$$\text{ТЕГ4}: A[3,5] \rightarrow B[7,11] \rightarrow C[12,15].$$

Використання трасуючих кодів дозволяє отримувати інформацію за допомогою двох типів запитів: запитів відстеження та запитів, орієнтованих на шлях. Запити відстеження використовуються для отримання історії переміщень тегу. Запити, орієнтовані на шлях, використовуються для вибору набору тегів і задоволення певної умови щодо переміщень.

Записи трасування зберігають інформацію про переміщення за допомогою двох чисел: номера кодування елемента (НКЕ) та порядкового номера кодування (НКП).

Припустимо, що локаціям Q, W, E, R відповідають прості числа 3, 5, 7, 11 відповідно. Також припустимо, що для тега згенерувався наступний трасуючий код TAG1: Q[3, 5] → W[6, 9] → E[13, 19]. Відповідно до функції декодування,

$$НКЕ = 3 \cdot 5 \cdot 7 = 105,$$
$$НКР \bmod 3 = 1,$$
$$НКР \bmod 5 = 2,$$
$$НКР \bmod 7 = 3,$$
$$НКР < 105,$$
$$НКР = 17.$$

Щоб отримати порядковий номер локації, яку відвідував працівник, використовуйте наступну формулу:

$$N = \text{НКП} \bmod L,$$

де N — порядковий номер,

НКП — номер кодування порядку,

L — код локації.

Розглядаючи вираз 17 mod 7 = 3, це означає, що при діленні числа 17 на 7 залишок дорівнює 3. Це можна інтерпретувати як те, що працівник вже втретє відвідав локацію з кодом 7 (E).

Також, якщо обчислити відповідно 17 mod 3, 5, 7, отримаємо коди локацій Q, W, E відповідно. Ця методика базується на Китайській теоремі про залишки, яка стверджує, що для системи рівнянь з взаємно простих модулів \(m_1, m_2, \ldots, m_k \) існує єдине рішення, що відповідає умовам \(N \equiv a_i \mod m_i \) для кожного \(i \). Ця теорія дозволяє ефективно розв'язувати такі системи

рівнянь, що зустрічаються в криптографії та інших областях математики та комп'ютерних наук:

$$X \bmod P_1 = A_1,$$
$$X \bmod P_2 = A_2,$$
$$X \bmod P_3 = A_3,$$
$$X \bmod P_4 = A_4,$$
$$\ldots\ldots\ldots$$
$$X \bmod P_n = A_n.$$

Сигнали, що використовуються в RFID, зазвичай мають цифрову модуляцію. Цифрово модульований сигнал (рисунок 3.2) представляє собою послідовність різних символів. Одним із широко відомих методів кодування в RFID є імпульсно-інтервальне кодування. Двійковий символ «1» кодується як короткий імпульс вимикання після тривалого інтервалу повної потужності, тоді як двійковий символ «0» кодується як коротший інтервал повної потужності з тим же імпульсом вимикання живлення.

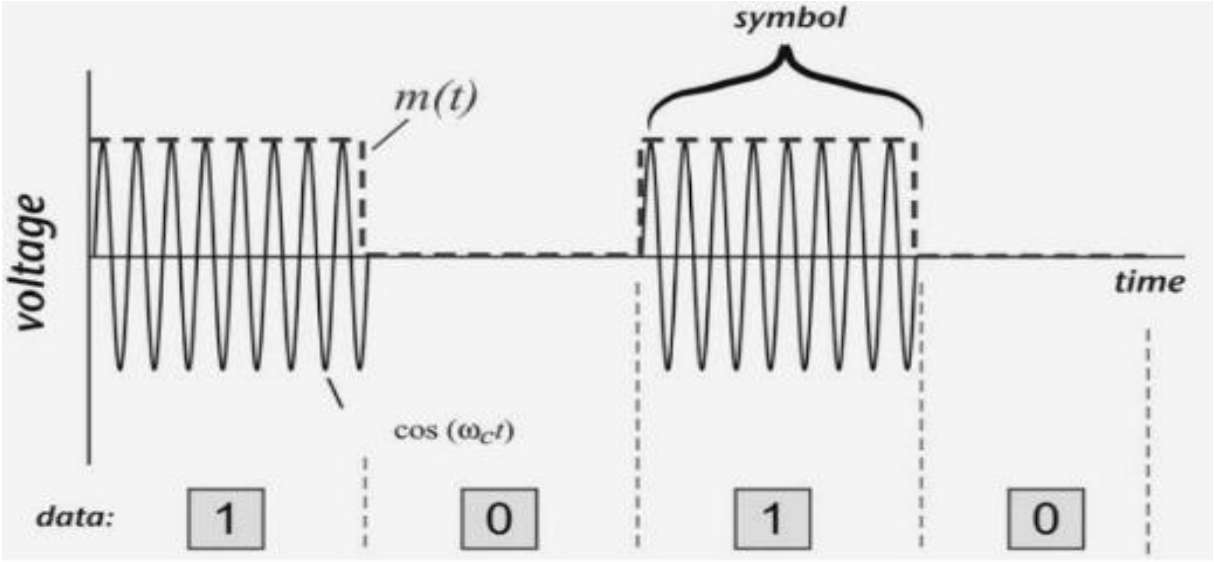

Рисунок 3.2 — Модуляція сигналу

У пасивній системі RFID передавач не вимикається, а замість цього тримається в стані передачі протягом періоду, коли мітка передає свій сигнал. Радіо RFID використовує спеціалізовані компоненти, такі як циркулятори або комутатори, щоб передавати до приймача лише відбиті сигнали, які можуть бути затемненими сильним передаваним сигналом.

3.2 Моделювання системи радіочастотної ідентифікації для Smart-годинника

Було створено модель бездротового зв'язку системи радіочастотної ідентифікації (RFID) з використанням фазової маніпуляції. Для проведення моделювання використовувалися інструменти програмування Matlab версії R2015a.

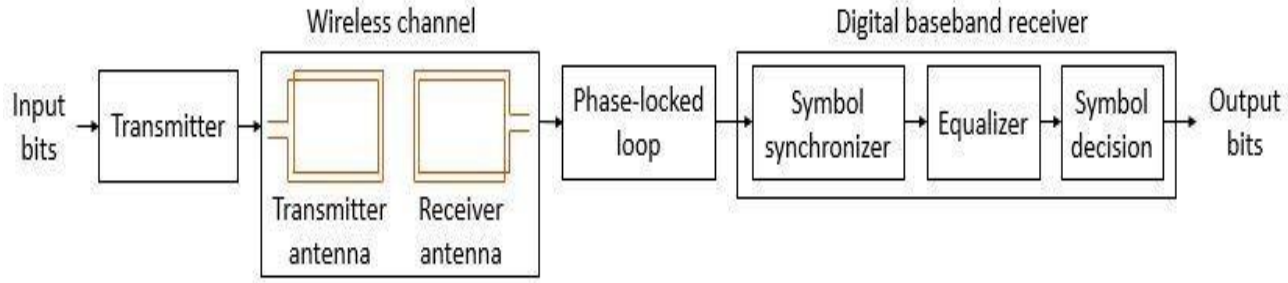

Рисунок 3.3 — Ланки безпровідного зв’язку

В цьому проекті моделюється ланцюг бездротового зв'язку, який включає наступні компоненти:

— передавач: генерує M сигналів фазового зміщення на несучій частоті \(f_c \) з швидкістю передачі даних \(\frac{f_c}{4} \);

— бездротовий канал і модель антени: передає сигнал через канал з урахуванням характеристик антени та бездротового середовища;

— фазове автопідстроювання частоти: забезпечує автоматичне налаштування частоти для зменшення міжсимвольного перешкоджання;

— цифровий приймач базової смуги: складається з блоків синхронізації символів, еквалайзера та блоків визначення символів для отримання і декодування переданих даних;

— файл `constellation.m`: визначає карти сузір'я PSK на комплексній площині та виконує бітове кодування символів, використовується диференціальне кодування для мінімізації впливу ковзання фази.

Цей ланцюг забезпечує передачу та приймання даних з високою ефективністю та мінімізацією помилок у бездротовому середовищі.

Передавач у даній моделі генерує сигнал із наступною структурою:

— початковий період мовчання: це не модульований синусоїдальний сигнал, який служить для підготовки каналу передачі даних;

— преамбула або заголовок: вказує на початок пакета даних і допомагає приймачу синхронізуватися з передачею;

— N — диференційно закодовані символи: це корисні дані, які передаються через канал зв'язку;

— послідовність символів кінця пакета: вказує на завершення пакета даних і допомагає приймачу визначити кінець інформаці;.

— остаточний період тиші: ще один не модульований синусоїдальний сигнал, який завершує передачу і дозволяє каналу відновитися до початкового стану.

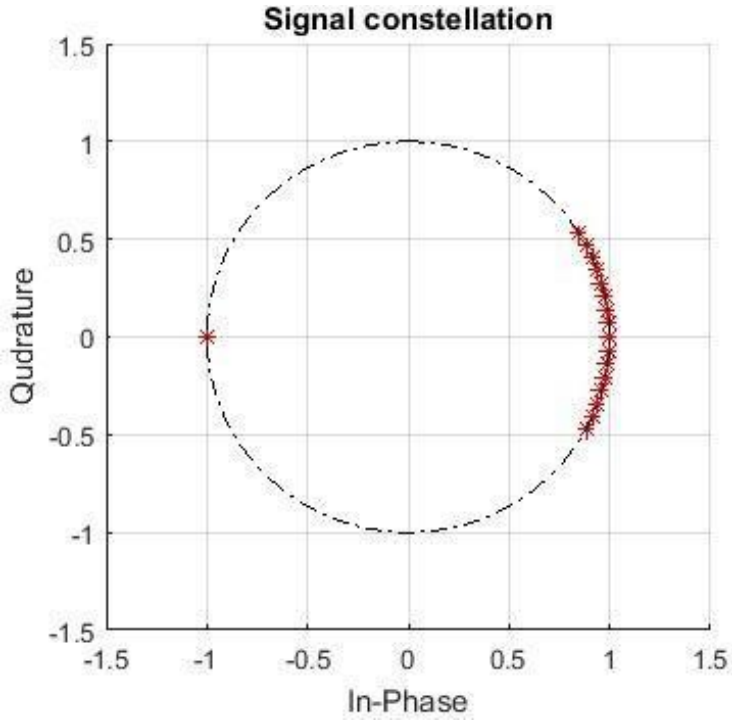

Рисунок 3.4 — Діаграма сигналу передавача

Параметри моделювання за замовчуванням для передавача зображені на рисунку 3.5.

```
N = 800;
M = 16;
fc = 13.56e6;
dataRate = fc/4;
nSampSim = 32;
nSamp = 2;
TsSim = 1/(nSampSim*dataRate);
Ts = 1/(nSamp*dataRate);
```

Рисунок 3.5 — Параметричні налаштування передавача

Важливо відзначити, що симуляція включає швидкісну вибірку: передавач і бездротовий канал моделюються з частотою 16 разів вище, ніж цифровий приймач базової смуги, що дозволяє зберігати синхронізацію і ефективність передачі даних. Бездротовий канал у системі RFID складається з антени передавача, середовища бездротової передачі (повітря) і антени приймача. Моделювання антенної схеми передавача та приймача виконується за допомогою послідовної та паралельної схем RLC відповідно.

Для практичних цілей, з урахуванням того, що відстань між передавачем і приймачем зазвичай менше 10 см у застосуваннях RFID, бездротовий канал можна змоделювати як взаємно зв'язану пару котушок антени передавача і приймача. Це призводить до еквівалентної схеми, де котушки індуктивності з'єднані взаємною індуктивністю.

$$M = K\sqrt{Л_t \cdot Л_к},$$

де коефіцієнт зв'язку k визначає близькість між первинною та вторинною котушками і зазвичай приймає значення від 0,03 до 0,3. Це значення k є точною одиницею, коли дві котушки з'єднані в одну котушку, як показано на рисунку 3.6.

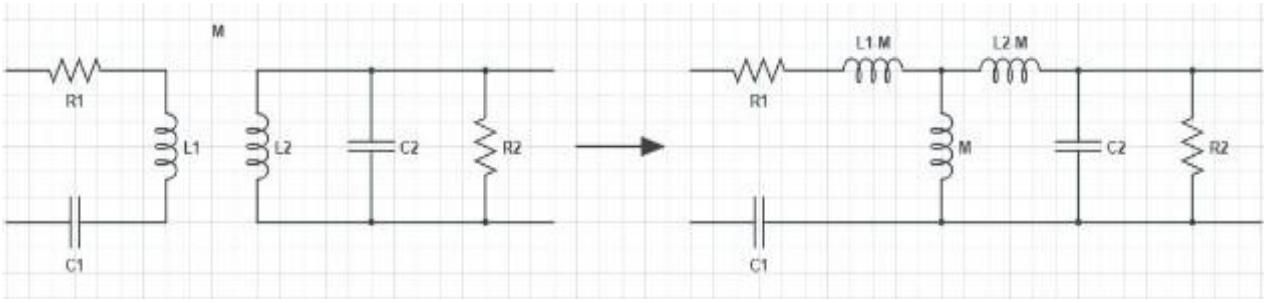

Рисунок 3.6 — Еквівалентні моделі антен

Функція передачі аналогового смугового каналу перетворюється на цифровий еквівалент низьких частот, який відображений на рисунку 3.7.

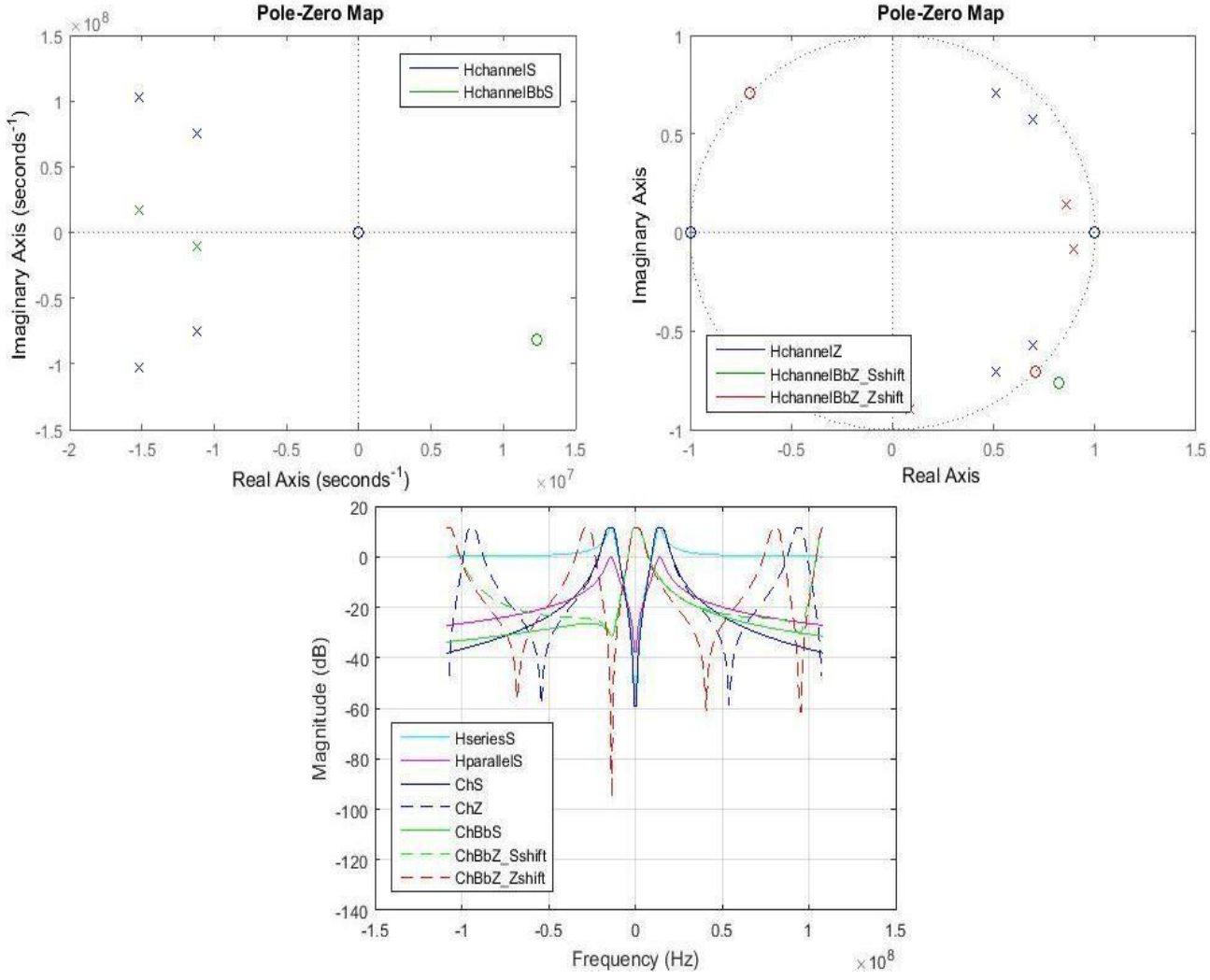

Рисунок 3.7 — Передаточні функції

Загальна передаточна функція комбінованого резонансного каналу визначається як:

$$\frac{C_1 M P_2 c^2}{\left[\begin{array}{c} P_2 + Л_2 s - C_1 M^2 c^3 + C_1 Л_1 Л_2 c^3 + C_1 Л_1 P_2 c^2 + C_1 Л_2 P_1 c^2 + C_2 Л_2 P_2 c^2 + \\ C_1 P_1 P_2 s - C_1 C_2 M^2 P_2 c^4 + C_1 C_2 Л_1 Л_2 P_2 c^4 + C_1 C_2 Л_2 P_1 P_2 c^3 \end{array}\right]}.$$

На рисунку 3.8 показана форма хвилі модульованої смуги пропускання, отримана на приймачі після проходження через передавач і бездротовий канал.

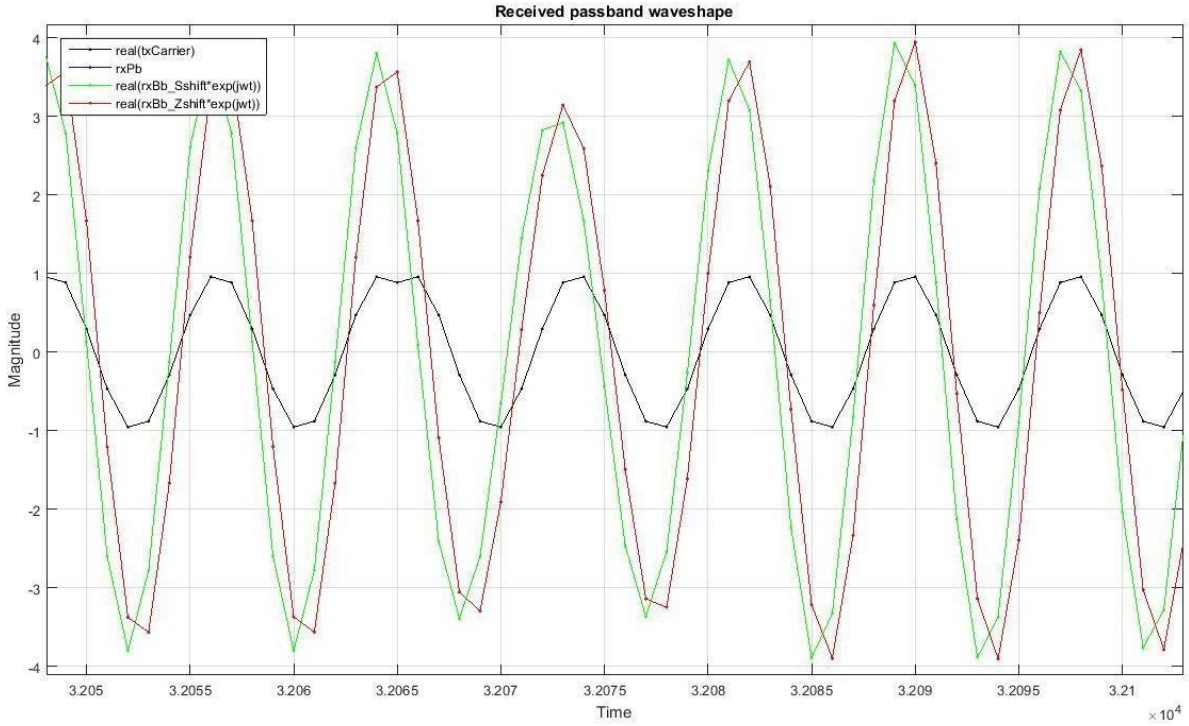

Рисунок 3.8 — Сигнал передавача

На рисунку 3.9 представлені параметри моделювання за замовчуванням для моделі бездротового каналу та антени.

```
k = 0.3;
Qr = 4;
Qc = 3;
fresr = 13.56e6;
fresc = 14e6;
```

Рисунок 3.9 — Параметри даних для антени

Вихідний сигнал надсилається до цифрового приймача основної смуги частот. Перш за все, для виявлення початку пакета даних використовується синхронізатор символів у вигляді псевдоузгодженого фільтра. Після виявлення пакету сигнал проходить через еквалайзер, за яким слідує блок визначення символу. Рішення приймаються на основі мінімальної евклідової відстані. Еквалайзер налаштований з дробовим інтервалом та зворотнім зв'язком, що забезпечує оптимальну узгоджену фільтрацію та синхронізацію часу символу. Еквалайзер з дробовим інтервалом гарантує однозначне визначення для кожного символу, який відбирається зі швидкістю, перевищуєю швидкість символу.

Це відповідає параметрам моделювання за замовчуванням для цифрової моделі приймача основної смуги частот, зображеним на рисунку 3.10.

```
kFwd_pos = 0:-1:-2;
kBwd = 2;
mu = 0.25;
L = 2;
gamma = 1;
bias = 0;
delay = 1;
fracEn = 1;
```

Рисунок 3.10 — Параметри приймача

Алгоритм проекції використовується як оптимізатор для оновлення вагових векторів. Вибір цього алгоритму обумовлений кількома перевагами:

— швидкість збіжності досягається завдяки компромісу між алгоритмом найменших квадратів та рекурсивним алгоритмом найменших квадратів, забезпечуючи ефективність оновлення вагових векторів;

— контроль адаптації: вагових коефіцієнтів, це важливо для підтримки стабільної роботи системи та попередження неправильного налаштування параметрів;

— управління величиною коефіцієнтів: алгоритм дозволяє налаштовувати величину коефіцієнтів у процесі оптимізації, що впливає на точність та стабільність оновлення вагових векторів.

Ці переваги роблять алгоритм проекції привабливим вибором для задач оновлення параметрів систем, де важливі якість результату та швидкість адаптації.

ВИСНОВКИ

В даному проекті була розроблена мікропроцесорна система Smart-годинника для обліку робочих годин працівників. У рамках проекту було створено серверну частину та функціональну схему терміналу, яка детально описує його особливості. Проведено аналіз існуючих систем обліку робочих годин та їх класифікацію, що дозволило вибрати оптимальні компоненти для проекту.

Основні компоненти серверної частини були обрані з урахуванням мови програмування, вебфреймворку та бази даних, що надійно підтримують вимоги проекту.

Для апаратної частини терміналу були детально обґрунтовані технічні характеристики, враховуючи фактори, такі як розмір, потужність, умови експлуатації, шум, вібрація та енергоспоживання.

Процес хостингу був організований з використанням AWS, що забезпечило надійність та безперебійну роботу системи.

Для зберігання та обробки даних був розроблений метод з використанням RFID тегів, що дозволив ефективно використовувати пам'ять.

Аналіз технологічного процесу та інформаційних алгоритмів показав необхідність вдосконалення системи обліку, що було враховано у проекті. Застосування математичного комплексу програмного забезпечення для обробки сигналів RFID дозволило знизити обсяг пам'яті та підвищити ефективність системи.

Розроблена система забезпечує надійну та якісну роботу, зокрема шляхом оперативного реагування на різні сценарії роботи та мінімізації навантаження на мережу. Це сприяє підвищенню надійності та забезпечує ефективне управління робочими процесами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іванов А. О. Теорія автоматичного керування: Підручник. — Дніпропетровськ: Національний гірничий університет. — 2003. — 250 с.

2. Louis C. Westphal. Handbook of Control Systems Engineering. — 2nd edition; The Springer International Series in Engineering and Computer Science. — Springer, 2001. Т. 635. — 1063 с. — ISBN 978-0792374947. англ.)

3. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - ІваноФранківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с.

4. Алгоритми та структури даних. Навчальний посібник / Т. О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. - 280 с.

5. Глоба Л. С. Розробка інформаційних ресурсів та систем [Електронний ресурс] : конспект лекцій / Л. С. Глоба, Т. М. Кот. - Київ : НТУУ "КПІ", 2014. - 318 с.

6. Грязнова В. О., Єфіменко С. В. Основи методології програмування. - К.: ВПЦ "Київський університет", 2010.

7. Інженерія якості програмного забезпечення: навч. посібник / Г.В Табунщик, Р.К. Кудерметов, Т.І. Брагіна. - Запоріжжя: ЗНТУ, 2013. - 180 с.

8. Технології створення програмних продуктів та інформаційних систем : навч. посібник / М. Ю. Карпенко, Н. О. Манакова, І. О. Гавриленко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. - Харків : ХНУМГ ім. О. М. Бекетова, 2017. - 93 с.

9. Шевчук І. Б. Інформаційні технології в регіональній економіці: теорія і практика впровадження та використання : монографія. Львів : Видавництво ННВК "АТБ", 2018. 448 с.

10. Головний сайт платформи Arduino, електронний ресурс. Режим доступу: http://arduino.ua

11. Головний сайт модуля розширення Cnc shield v3, електронний ресурс. Режим доступу: http://blog.protoneer.co.nz/arduino-cnc-shield/

12. Стаття, присвячена мові програмування g-code, електронний ресурс. Режим доступу: https:// wikipedia.org/wiki/G-code

13. Головна сторінка платформи GRBL, електронний ресурс. Режим доступу: https://github.com/grbl/grbl

14. Довідник за типами файлів, електронний ресурс. Режим доступу: http:// open- file.com.ua/types/nc

15. Довідникова сторінка онлайн-програми MakerCam, електронний ресурс. Режим доступу: http://www.makercam.com/help.html

16. Головна сторінка платформи UGS, електронний ресурс. Режим доступу: https://winder.github.io/ugs_website/

ДОДАТОК А

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф.
_____О. Д. Азаров
"06" травня 2024р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврського дипломного проекту
«Мікропроцесорна система Smart-годинника»
08-54.БДП.007.00.000 ТЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Кадук О.В.

Студента групи 1СП-20б

_____ Івахов П.В.

Вінниця 2024

1. Найменування та область застосування

Мікропроцесорна система Smart-годинника на базі платформи Arduino Mega 2560П з використанням RFID технології.

2. Основи для розробки

Необхідність реалізації мікропроцесорної системи Smart-годинника

3. Мета та призначення розробки

Мета створення системи полягає в забезпеченні реєстрації і аналізу часу входу і виходу працівників, обліку понаднормових, недопрацьованих та вихідних годин, що дозволить правильно нараховувати заробітну плату.

4. Етапи БДП та очікувані результати

Робота виконується у вісім етапів, що наведені в таблиці 4.1.

Таблиця 4.1 — Етапи виконання роботи

з/п	Назва етапів виконання бакалаврського проекту	Строк виконання етапів роботи	Примітка
1	Постановка задач проекту	27.02.24	
2	Огляд інформаційних джерел	06.03−20.03.24	
3	Аналіз системи системи Smart-годинника як об’єкта керування	27.03−7.04.24	
4	Вибір елементної бази для реалізації системи Smart-годинника	10.04−21.04.24	
5	Покращення функціональних можливостей та підвищення ефективності системи Smart-годинника	24.04−12.05.24	
6	Оформлення пояснювальної записки та ілюстративного матеріалу	15.05−26.05.24	
7	Аналіз виконання проекту. Висновки. Додатки	29.05−09.06.24	
8	Перевірка якості виконання бакалаврського проекту та усунення недоліків	06.06.24	

5. Матеріали, що подаються до захисту БДП

Пояснювальна записка БДП, графічні і ілюстративні матеріали, протокол попереднього захисту БДП на кафедрі, відзив наукового керівника, рецензія опонента, протоколи складання державних екзаменів, анотації до БДП українською та іноземною мовами, довідка про відповідність оформлення БДП діючим вимогам.

6. Порядок контролю виконання та захисту БДП

Виконання етапів графічної та розрахункової документації БДП контролюється науковим керівником згідно зі встановленими термінами. Захист БДП відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

7. Вимоги до оформлення БДП

При оформлюванні КБДР використовуються:
— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;
— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;
— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;
— документами на які посилаються у вище вказаних.

Технічне завдання до виконання отримав _____ Візінчук Д.С.

ДОДАТОК Б

Системи фіксації часу перебування

Рисунок Б1 — Системи фіксації часу перебування

ДОДАТОК В

Модульність структури Spring

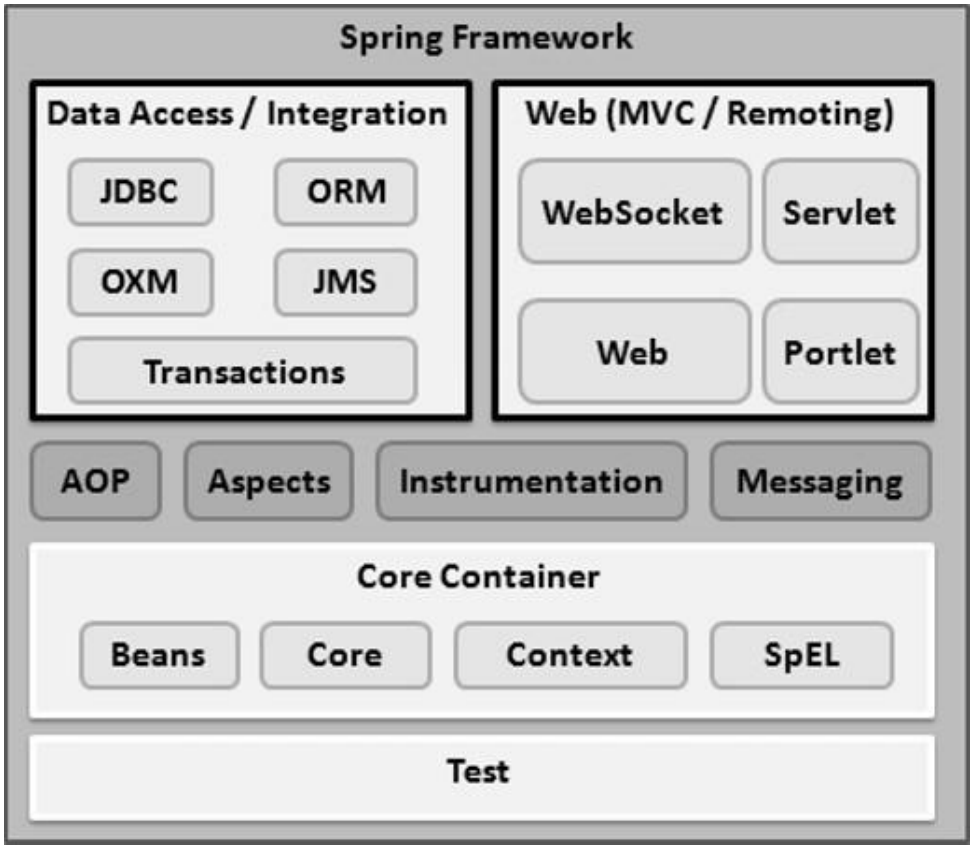

Рисунок В1 — Модульність структури Spring

ДОДАТОК Г

Функціональна схема

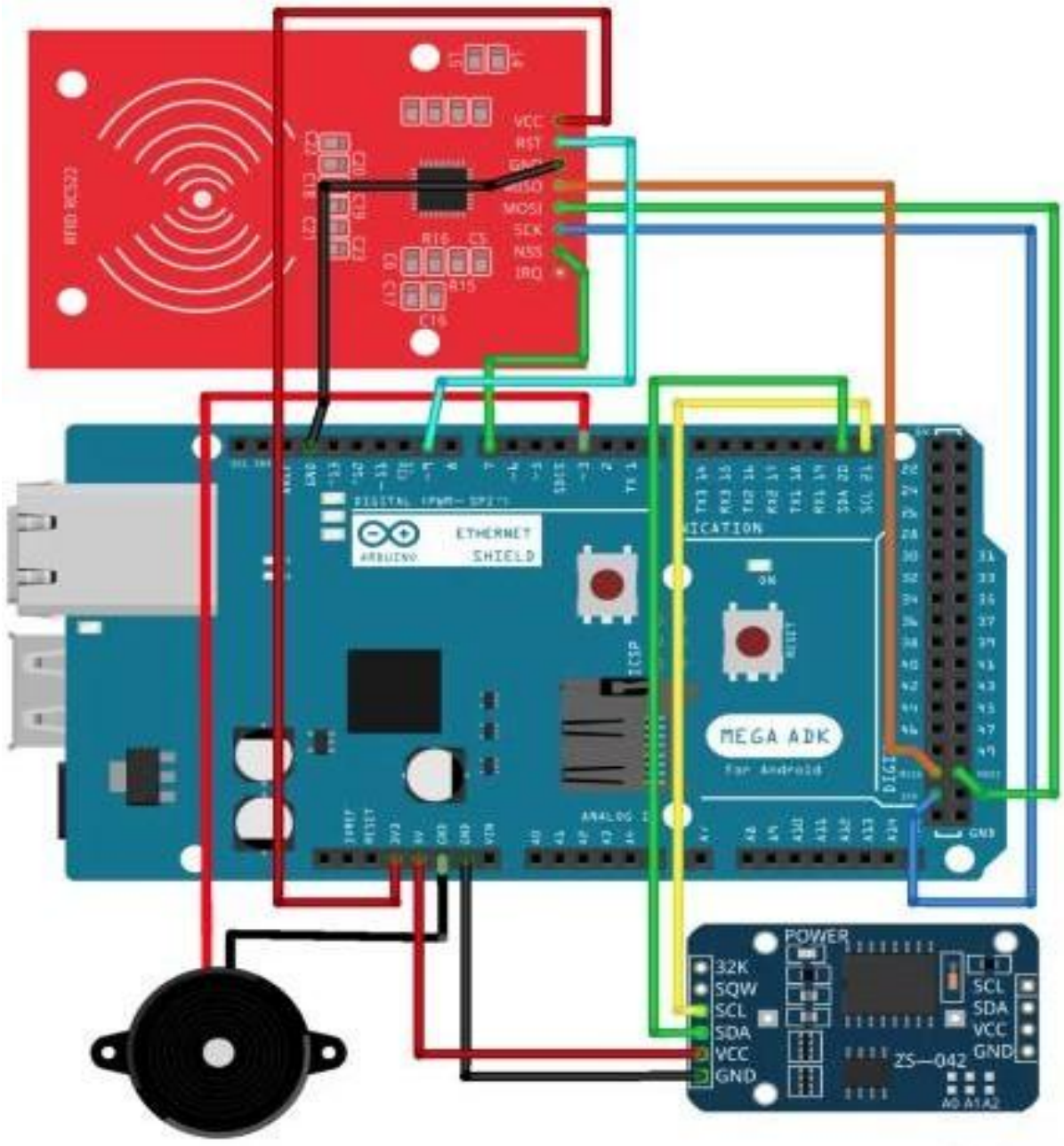

Рисунок Г1 — Функціональна схема

ДОДАТОК Д

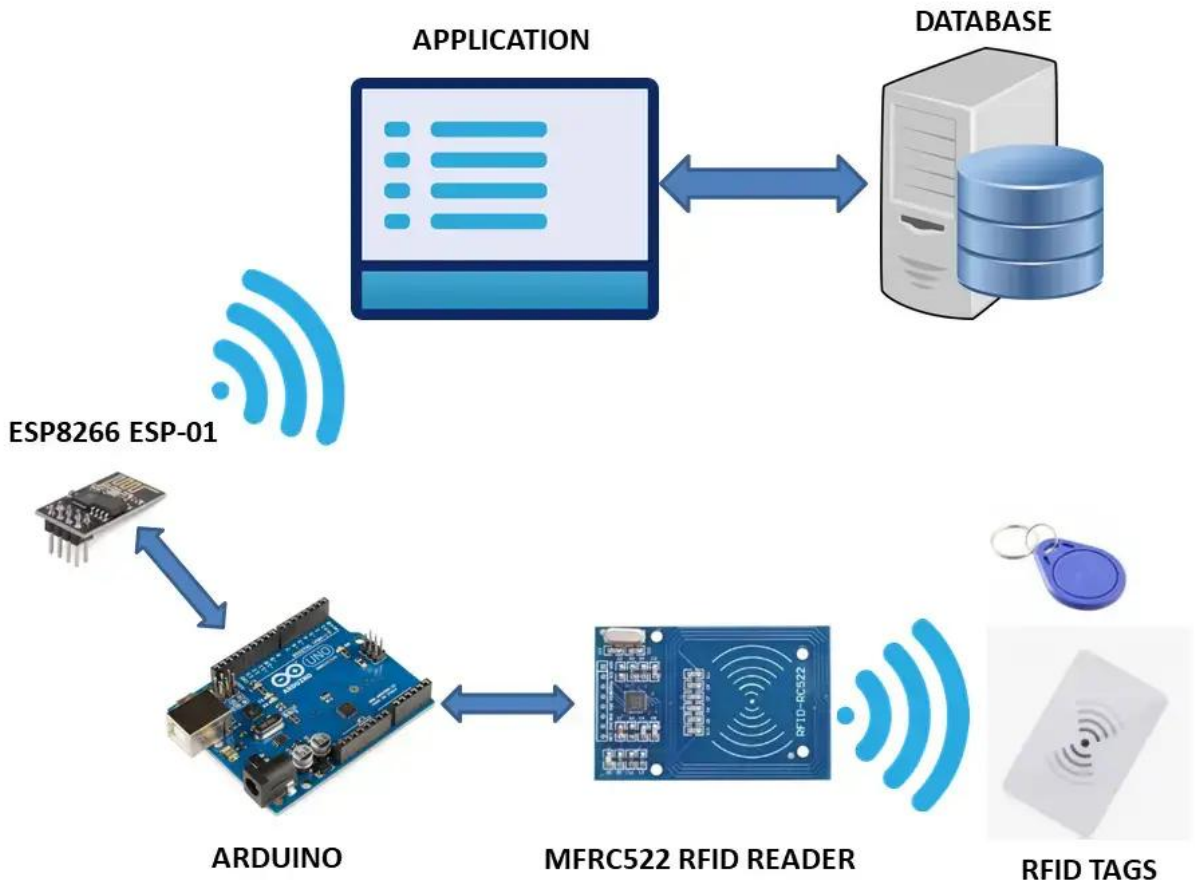

Рисунок Г1 — Діаграма взаємодії модулів

ДОДАТОК Е

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Мікропроцесорна система Smart-годинника

Тип роботи: Бакалаврський дипломний проект

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 91,2% Схожість 9,8%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Івахов П.В.
(підпис) (прізвище, ініціали)

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