Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет ____Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра ___Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

_____Розподілена система управління вирощуванням рослин_____
(тема)

Виконав: студент 2 курсу, групи СКСм-20-1

_____Дегтяр О.М._____
(прізвище, ініціали)

Спеціальність *123 – Комп'ютерна інженерія*

Тип програми ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

_____Освітня програма_____

_*Спеціалізовані комп'ютерні системи*_____
(повна назва освітньої програми)

Керівник роботи проф. Чумаченко С.В._
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерної інженерії та управління _____

Кафедра _____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти ___другий (магістерський)_____

Спеціальність_____123_ Комп'ютерна інженерія_____
(шифр і назва)

Тип програми _____Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Спеціалізовані комп'ютерні системи_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)

«__4__»___11____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____Дегтяру Олександру Миколайовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) _____
_____Розподілена система управління вирощуванням рослин_____
_____Distributed plants growing management system_____

затверджена наказом по університету від «04» _11_ 2021 р. № 1635 Ст

2. Термін подання студентом роботи до екзаменаційної комісії __20.12.2021_

3. Вихідні дані до роботи (проекту)_____
Мікрокомп'ютер Raspberry Pi 3B+_____
Мова програмування C++17_____
Операційна система Android_____
Мова програмування Kotlin_____

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз умов вирощування рослин_____
Розробка апаратного забезпечення пристрою_____
Аналіз підходу зменшення зв'язності програмних сутностей_____
Розподілення функціональності системи між сервісами_____
Реалізація програмного забезпечення пристрою_____
Реалізація мобільного застосунку_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

___25 слайдів_____

_____

_____

_____

_____

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання_____02.09.2021_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Отримання завдання	02.09.2021 - 08.09.2021	
2	Аналіз предметної області	09.09.2021 - 15.09.2021	
3	Аналіз джерел з проблемної галузі	30.09.2021 - 13.10.2021	
4	Розробка апаратного забезпечення	14.10.2021 – 31.10.2021	
5	Розробка програмної архітектури	01.11.2021 -15.11.2021	
6	Реалізація програмного забезпечення	16.11.2021 - 01.11.2021	
7	Оформлення пояснювальної записки	02.12.2021 - 10.12.2021	
8	Оформлення графічного матеріалу	11.12.2021 - 19.12.2021	
9	Захист проекту	20.12.2021 - 25.12.2021	

Студент _____
(підпис)

Керівник роботи (проекту) _____ проф. Чумаченко С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 127 сторінок, 116 рисунків, та 33 джерела за переліком посилання.

РОСЛИНИ, РОЗПОДІЛЕНІСТЬ, C++, KOTLIN, АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, CONTINUOUS INTEGRATION, ІРИГАЦІЯ, ДАТЧИКИ, ВОЛОГІСТЬ ҐРУНТУ, ОСВІТЛЕННЯ, ВЕНТИЛЯЦІЯ.

Метою роботи є розробка розподіленої системи управління вирощуванням рослин, що складається з пристрою управління та мобільного застосунку та забезпечує необхідний фізичний стан середовища за допомогою фітоосвітлення, вентиляції і іригації.

Проведено аналіз основних параметрів навколишнього середовища, необхідних для нормального розвитку рослин. Описано метод зменшення зв’язності програмних сутностей для полегшення автоматизованого тестування. Розроблено пристрій управління вирощуванням рослин та програмне забезпечення до нього. Розроблено мобільний застосунок для дистанційної взаємодії користувача з пристроєм. Розроблено автоматичні тести до програмного забезпечення системи. Налаштовано сервіс автоматичної перевірки коректності оновлень вихідного коду. Побудовано експериментальне середовище для тестування розробленої системи.

Пристрій управління вирощуванням рослин базується на мікрокомп’ютері Raspberry Pi 3B+, забезпечує і керує живленням фітоосвітлення, вентиляції, перистальтичного насосу та рідинних клапанів. Програмне забезпечення пристрою розроблене з використанням мови програмування C++. Мобільний застосунок розроблено мовою програмування Kotlin для операційної системи Android та дозволяє користувачу налаштовувати пристрій.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АЦП – аналого-цифровий перетворювач;

БД – база даних;

БЖ – блок живлення;

ЗС – змінний струм;

ООП – об'єктно-орієнтоване програмування;

ПЗ – програмне забезпечення;

ПС – постійний струм;

СУБД – система управління базами даних;

CD – Continuous Deployment (неперервне розгортання);

CI – Continuous Integration (неперервна інтеграція);

LED – Light-Emitting Diode (світловипромінюючий діод);

PR – Pull Request (запит на забирання змін);

RPC – Remote Procedure Call (виклик віддалених процедур);

VNC – Virtual Network Computing (протокол надання доступу до віддаленого комп'ютера).

ЗМІСТ

ВСТУП

Розповсюдження автоматизації та поширення сценаріїв використання програмного забезпечення, як у професійних сферах, так і у побуті, призводить до збільшення різноманітності типів пристроїв, що мають у своїй основі обчислювальні апаратні елементи. За концепцією “інтернету речей”, множина обчислювальних пристроїв, що оснащені сенсорами та обмінюються між собою інформацією, має давати змогу людині зменшити кількість необхідних для виконання рутинних задач.

Важливим компромісом, з яким стикається кожен виробник подібних пристроїв, є відношення обчислювальної потужності до вартості апаратного забезпечення. Зазвичай, саме вартість пристрою є одним з головних аспектів вибору для кінцевого користувача. У такому випадку, необхідно використати більш дешеву апаратну платформу користувацького девайсу, при цьому, не порушуючи функціональність системи, або навіть покращуючи її.

Можливим шляхом вирішення може стати розподілення системи. Перенесення користувацького інтерфейсу з самого пристрою управління вирощуванням рослин до смартфону не тільки дозволить виключити дисплей з апаратного забезпечення, яке має бути придбане користувачем, зменшуючи собівартість пристрою, а ще й поліпшує досвід використання, дозволяючи налаштовувати рослинне середовище дистанційно.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз умов вирощування рослин

Рослини, як будь-які інші живі організми, для свого розвитку потребують забезпечення низки умов. Стабільне забезпечення усього спектру потреб дозволяє вирощувати здорові та сильні рослини. В залежності від обраного типу культури, людина віддає пріоритет різним характеристикам: зовнішній вигляд декоративної рослини або смак та поживність врожаю. Тим не менш, незалежно від біологічного виду чи місця вирощування, основні потреби рослин спільні – вода, світло, повітря та поживні речовини. Більш того, для нормального розвитку кожна складова обов’язково має надаватись у певному обсязі. Забезпечення занадто великої кількості кожної компоненти, як і нестача, може пошкодити рослину.

Вода є головною речовиною для життя. Тканини рослин, як і багатьох інших живих істот, здебільшого складаються саме з води. Основну частину води рослина отримує шляхом поглинання кореневою системою з ґрунту. Забезпечення водою є критичним для життя рослини, оскільки сухий ґрунт призводить до висихання. Однак, надмірне зволоження призвести до гниття коренів та появи грибків. Саме тому іригація, – полив ґрунту, – має виконуватись з перервами на висихання, але й не допускаючи тривалої дегідратації.

Без світла неможливий фотосинтез – хімічний процес, в ході якого рослини виробляють глюкозу та кисень з вуглекислого газу та води, використовуючи при цьому енергію світла. Таким чином, світло необхідне рослинам для отримання їжі. У відкритому ґрунті, як і кімнатні рослини на підвіконнях, отримують світло природнім шляхом. Але вирощуючи рослини вдома або в інших закритих приміщеннях, зазвичай, світла виявляється недостатньо. Вирішити це можна за допомогою додаткового фітоосвітлення –

джерел, що випромінюють необхідні для розвитку рослини спектри світла.

Повітря є невід'ємною частиною росту та навіть виживання рослин. Воно забезпечує приток вуглекислого газу. Саме цей газ приймає участь у фотосинтезі і подальшому отриманні рослиною глюкози. Також невелика кількість кисню має бути присутня у ґрунті, оскільки рихлий, добре аерований ґрунт для здоров'я та розвитку кореневої системи. В умовах закритого ґрунту активна вентиляція потребує додаткової уваги через відсутність природнього вітру. Брак вентиляції у теплому вологому середовищі швидко призводить до розповсюдження грибків (рис. 1.1) [1] та шкідників, які негативно впливають на здоров’я рослин.

Рисунок 1.1 – Пліснява у ґрунті може бути викликана недостатньою вентиляцією середовища

Як люди та усі інші тварини, рослини для своєї життєдіяльності потребують поживних речовин – макро- та мікронутрієнтів. Необхідна кількість конкретної речовини відрізняється в залежності від біологічного виду рослини, стадії розвитку та багатьох інших параметрів. Тим не менш, усі рослини залежать від наявності у ґрунті таких хімічних елементів як азот, калій та фосфор. Азот необхідний рослинам для подовження стовбура та листя. Фосфор у зростанні коріння. А для розвитку судинної системи, що розповсюджує поживні речовини та воду по тілу рослини, необхідний калій.

1.2 Аналіз існуючих рішень

Проект PiGrow [2] (рис. 1.2) , найпопулярніший інструмент автоматизації вирощування рослин на платформі GitHub розвивається Кельвіном Роулінгом з 2016. Розробка являє собою множину утиліт,

імлементованих мовою Python для мікрокомп’ютеру Raspberry Pi, які користувач може налаштувати для свого середовища (рис. 1.3) у випадку, якщо необхідний інтерфейс взаємодії з обладнанням реалізовано у проекті. Система дозволяє керувати живленням світла та вентилятора за таймером, зчитувати та візуалізувати дані з датчиків температури і вологості повітря, налаштовувати перелічене за допомогою графічного інтерфейсу, передаючи дані через вбудований у Raspberry Pi Wi-Fi модуль.

Рисунок 1.2 – Сторінка збору пожертвувань на розвиток проекту PiGrow

Оскільки утиліти імплементовано інтерпретованою мовою програмування Python, це накладає додаткові обчислювальні витрати для роботи системи у порівнянні з реалізаціями, що можуть бути компільовано у машинний код кінцевої платформи. Залежність від інтерпретатора для виконання Python-програм обмежує спектр придатних до використання апаратних платформ, що знижує переносимість системи у контексті розробки embedded-систем.

Графічний інтерфейс системи розроблено з використанням модулю wx [3]. Оскільки обраний набір інструментів не підтримує популярні мобільні операційні системи такі як Android чи iOS, можливими платформами для використання користувацького інтерфейсу можуть бути персональний комп’ютер чи безпосередньо мікрокомп’ютер з подальшим підключенням за допомогою протоколу VNC.

Динамічна типізація мови Python [4] та відсутність використання у вихідному коді вбудованих у мову методів типізації є негативним фактором, оскільки унеможливлює читачем розуміння наявних атрибутів об’єкту. В тому числі, це знижує інформативність та точність результатів використання засобів статичного аналізу вихідного коду.

Відсутність практики автоматизованого тестування реалізації, та інших способів підвищення надійності, у системі з актуаторами, що безпосередньо впливають на навколишнє середовище, нажаль, може стати головною причиною користувача утриматися від використання даної системи.

Рисунок 1.3 – Експериментальне середовище автора проекту PiGrow

Естонський стартап Click&Grow [5], заснований у 2009 році, шляхом платформи Kickstarter зібрав більш ніж $1 млн доларів США інвестицій для розвитку проекту та початку виробництва. Компанія виробляє розумні горщики, що мають вирощувати рослини при мінімальній участі користувача. Конструкція горщику, який продемонстровано на рисунку 1.4, складається з баку для води в нижній частині пристрою, площини з отворами для висаджування рослин та LED-фітоосвітлення у верхній частині.

Рисунок 1.4 – Розумний горщик Click&Grow Smart Garden 3

Виробником заявляється, що такий пристрій моє вирощувати рослини без мануальної іригації та догляду з боку користувача. Система автоматично надасть необхідну кількість води, поживних речовин та світла для нормального розвитку рослини. Конструкція пристрою передбачає можливість зміни висоти освітлення над рослиною. Посадити рослину можна використовуючи спеціальні капсули, що мають в собі насіння та спеціальний субстрат.

Недоліками пристрою, з якими може зіткнутися користувач,

здебільшого викликані мінімалістичністю конструкції. Невелика максимальна висота стійки освітлення обмежує користувача у видах рослин, які можна вирощувати. Компактний розмір пристрою призводить до недостатнього об’єму, до якого може розвиватися рослина, не стикаючись з сусідніми екземплярами. Наприклад, фотографія горщику Smart Garden 9 (рис 1.5), який має 9 отворів для рослин, демонструє, що на пізніх стадіях розвитку рослин об’єму достатньою лише для трьох.

Рисунок 1.5 – Розумний горщик Click&Grow Smart Garden 9

Головною причиною, через яку користувач може утриматися від використання пристрою є його вартість та ціна спеціальних капсул з насінням рослини. Наприклад, за даними офіційного сайту українських дистриб'юторів компанії Click&Grow на момент кінця 2021 року, вартість горщику може сягати $330 доларів США, а комплект з трьох капсул з насінням зелені – $20 доларів США.

1.3 Формування вимог до системи

Користувацькі сценарії використання розподіленої системи управління вирощуванням рослин:

автоматична іригація чотирьох окремих зон;

а) періодична іригація;

б) іригація за рівнем вологості ґрунту;

в) визначення підключеності датчиків вологості ґрунту;

управління станом фітоосвітленням;

а) ручне управління користувачем;

г) автоматичне управління за часом відповідно до користувацьких

налаштувань;

управління вентиляцією;

а) ручне управління користувачем;

б) автоматичне управління за часом відповідно до користувацьких налаштувань;

налаштування потужності вентиляції за допомогою ручки регулювання;

віддалене налаштування системи за допомогою мобільного застосунку.

Конструкційні вимоги до апаратного забезпечення системи:

фізична ізольованість сантехнічних пристроїв (насосу та клапанів) від джерел живлення та управляючого обладнання;

підключення вузлів системи та зовнішнього обладнання за допомогою роз’ємів;

вбудованість схеми живлення апаратного забезпечення системи у корпус пристрою;

живлення всієї системи за допомогою підключення до мережі лише одного кабелю;

захист компонентів датчиків від впливу вологого середовища.

1.4 Задачі проекту

Для розробки розподіленої системи управління вирощуванням рослин та задовільнення поставлених вимог необхідно вирішити наступні задачі:

розробка апаратного забезпечення пристрою управління вирощуванням рослин;

модифікація ємнісних датчиків вологості ґрунту;

розробка програмного забезпечення пристрою;

розробка мобільного застосунку;

налаштування сервісу неперервної інтеграції – автоматична перевірка коректності компілювання та виконання автоматичних тестів для кожного оновлення вихідного коду програмного забезпечення пристрою та мобільного

застосунку;

створення та використання експериментального середовища.

2 ОГЛЯД ТЕХНОЛОГІЧНИХ ЗАСОБІВ

2.1 Апаратне забезпечення

2.1.1 Мікрокомп’ютер Raspberry Pi 3B+

Raspberry Pi [6] це популярний одноплатний комп’ютер розміром з кредитну картку. Спочатку цей мікрокомп’ютер розроблявся для цілей освіти – обладнання дешевих робочих місць у класах вивчення програмування. Однак далі, завдяки своїй компактності та універсальності набув популярності в контексті багатьох освітніх проектів вбудованих систем.

Мікрокомп’ютер має 40 контактів введення/виведення загального призначення (GPIO), до яких можна підключати різноманітну периферію та модулі. Мала споживана потужність, близько одного Ватта, дозволяє живити пристрій навіть за допомогою портативного акумулятору. Порт HDMI дозволяє підключати будь які сумісні дисплеї. Ethernet порт та Wi-Fi надає змогу використовувати комп’ютерні мережі і взаємодіяти з іншими пристроями.

Рисунок 2.1 – Зовнішній вигляд мікрокомп’ютеру Raspberry Pi 3B+

Модель 3B+ [7], що зображена на рисунку 2.1, має наступні характеристики:

процесор Broadcom BCM2837B0 на базі Cortex-A53 (ARMv8) 64-bit SoC з тактовою частотою 1.4 ГГц;

1 Гб LPDDR2 оперативної пам’яті;

2.4 ГГц та 5 ГГц IEEE 802.11.b/g/n/ac Wi-Fi;

Bluetooth 4.2 з підтримкою BLE;

Gigabit Ethernet порт, що працює на базі USB 2.0 та має пропускну здатність близько 300 Мбіт/сек;

40 GPIO контактів;

HDMI та DSI порти для підключення дисплею;

4 USB 2.0 порти;

CSI порт для підключення спеціальної камери;

4-хконтактний 3.5 мм аудіо-порт;

Слот для Micro SD карт пам’яті для зберігання операційної системи та будь яких інших файлів у постійній пам’яті;

Micro USB порт для живлення напругою 5В;

підтримка Power-over-Ethernet (PoE).

Наявність виходів GPIO, Wi-Fi модулю, постійної пам’яті для збереження даних про вологість ґрунту і налаштувань системи, підтримка повноцінного Linux дистрибутиву на базі Debian, та низька споживана потужність робить Raspberry Pi 3B+ придатною обчислювальною основою для виконання поставлених задач проекту.

2.1.2 АЦП ADS1115

ADS1115 (рис 2.2) [8]– це 16-бітний чотирьохканальний аналогово-цифровий перетворювач (АЦП). Даний АЦП може працювати у режимі вимірювання чотирьох окремих каналів або двох диференціальних. Використання всіх доступних каналів дозволить підключити до розроблюваного пристрою до 4-х датчиків та забезпечити іригацію в режимі за вологістю ґрунту чотирьох окремих зон. За допомогою цього перетворювача можна отримувати до 860 вимірів за секунду, а можна виконувати вимірювання лише за запитом з ведучого пристрою. Для такої фізичної величини як вологість ґрунту, режим роботи за запитом є доцільнішим через менші енерговитрати.

Рисунок 2.2 – Порівняння розмірів модулю на базі ADS1115 із PLS планкою

Взаємодія з АЦП (управління та запит даних) здійснюється за допомогою шини I2C. I2C (або IIC, I^2C) – це послідовна асиметрична шина зв'язку між інтегральними схемами, що для фізичної комунікації використовує лише дві двонаправлені лінії зв'язку: SDA (Serial Data) лінія послідовної передачі даних та SCL (Serial Clock) лінія тактування. Підключення декількох таких АЦП до однієї шини I2C можливе за допомогою апаратного вибору адресу. Для цього контакт ADDR необхідно під’єднати до однієї з 4-х інших вхідних ліній як показано на рисунку 2.3.

Рисунок 2.3 – Вибір адресу АЦП ADS1115

2.1.3 Фітолампа

Для зростання рослин потрібно кілька умов, однією з яких є достатнє освітлення. Простіше за всього освітлювати рослини природнім шляхом, – сонячним світлом, – висаджуючи культури у відкритому ґрунті або виставляючи горщики на підвіконня. Нажаль, інколи невеликої кількості періодичного освітлення з вікна може стати недостатньою. Наприклад, якщо рослина потребує більше світла, ніж зазвичай є у наданому кліматі, або якщо рослина знаходиться далеко від джерела сонячного світла. В таких випадках рішенням може використання фітоламп – спеціальних освітлювальних приладів, що випромінюють необхідні рослинам спектри світла.

Існує декілька технологій виготовлення фітоламп для освітлення

рослин: люмінесцентні, газорозрядні та світлодіодні. Люмінесцентні лампи (рис. 2.4), які також можуть бути знайомі користувачам як енергозберігаючі, у ролі фітоосвітлення використовують вже давно. До переваг такого типу ламп відносять гарну світлопередачу, низькою вартістю, невеликим нагріванням. Але, головним недоліком, через який користувач може віддати перевагу іншим технологіям, є негативний вплив на зір людини мерехтіння.

Рисунок 2.4 – Фітолампа люмінесцентного типу

Газорозрядні фітолампи краще застосовувати виключно в теплицях, оскільки характеризуються сильним нагріванням, навіть вищим, ніж у ламп розжарювання. При цьому, навіть у теплицях є висока ймовірність підвищеної вологості, а попадання вологи на ці лампи може призвести до її вибуху. При використанні даного типу освітлення у середовищах з невеликим об’ємом повітря, висока температура поверхні лампи потребує окремо виділеного контуру активного охолодження, як продемонстровано на рисунку 2.5. Незважаючи на описані недоліки, невисока ціна світла, велика потужність джерела освітлення та оптимальні для рослин спектри випромінювання є причинами, через які користувачі доволі часто використовують газорозрядні лампи навіть у невеликих за об’ємом приміщеннях.

Рисунок 2.5 – Контур активного охолодження газорозрядної лампи

Найбільш ефективним джерелом освітлення для вирощування рослин є світлодіодні фітолампи (рис. 2.6) через відносно низьку температуру

поверхні, високу енергоефективність, а також можливість використання та комбінування світлодіодів з точним спектром випромінювання [9].

Рисунок 2.6 – Експеримент з вирощування картоплі використовуючи LED-фітолампу, проведений NASA у 2001 році

Для побудування експериментального середовища обрано LED-фітолампу на базі світлодіодів Samsung LM301B [10], що мають універсальний для рослин спектр випромінювання, зображений на рисунку 2.7.

Рисунок 2.7 – Спектрограма випромінювання світлодіоду Samsung LM301B

Обрана фітолампа (рис. 2.8) має споживану потужність 40 Ватт. Для пасивного охолодження світлодіодної панелі використовується алюмінієвий радіатор. Джерело живлення підключається за допомогою роз’єму GX16-2P.

Рисунок 2.8 – Світлодіодна панель на базі Samsung LM301B

2.1.4 Канальний вентилятор

Вентиляція у середовищі вирощування рослин виконує одночасно декілька критичних функцій. Вуглекислий газ приймає участь у фотосинтезі і подальшому отримані рослиною глюкози. Відсутність притоку свіжого

повітря дуже скоро може призвести до появи і розповсюдження грибків та шкідників. Не менш важливі для рослин джерела штучного світла, в залежності від об’єму приміщення та типу лампи, можуть виділяти занадто велику кількість тепла, яке треба відводити за допомогою вентиляції.

При вирощуванні рослин у закритому ґрунті можливе використання декількох схем вентиляції [11], що зображені на рисунку 2.9:

пасивна – для просторих приміщень з великим об’ємом повітря та частим провітрюванням та відсутніми джерелами сильного виділення тепла;

витяжна – вентилятор у верхній частині приміщення виконує витяжку повітря назовні, приток повітря виконується пасивно через отвір у нижній частині приміщення;

приточно-витяжна – перший вентилятор у верхній частині приміщення виконує витяжку повітря назовні, другий вентилятор у нижній частині приміщення виконує приток повітря.

Рисунок 2.9 – Схеми вентиляції закритого середовища вирощування рослин. Пасивна, витяжна і приточно-витяжна відповідно

В умовах закритого приміщення вирощування, забезпечити задовільно часте оновлення повітря можна за допомогою канального вентилятору. При необхідності додаткового охолодження джерела світла з сильним тепловиділенням, вводиться додатковий контур активної вентиляції.

Для експерементального середовища проекту обрано канальний вентилятор, зображений на рисунках 2.10 та 2.11. Додатковий контур вентиляції для охолодження фітолампи не потрібен через низьке тепловиділення світлодіодів.

Рисунок 2.10 – Зовнішній вигляд фронтальної сторони канального вентилятору

Рисунок 2.11 – Зовнішній вигляд задньої сторони канального вентилятору

2.1.5 Перистальтичний насос

Перистальтичний насос – це пристрій, що дозволяє перекачувати рідину за допомогою гучної перистальтичної трубки, що встановлена у круглому корпусі та стискається роликами, встановленими на роторі. Під час обертання ротору, ролики видавлюють вміст трубки, рухаючи рідину. Створений після оберту ротору понижений тиск у трубці змушує наступну порцію рідини рухатися всередину насосу. Принцип роботи [12] графічно показаний на рисунку 2.12.

Рисунок 2.12 – Принцип роботи перистальтичного насосу

Завдяки можливості виконувати часткові оберти, перистальтичний тип насосів є дуже точним. Саме тому він використовується у медицині, харчовій, хімічній та фармацевтичній промисловості. Причиною для використання перистальтичних насосів у вимогливих сферах також є те, що лише гнучка силіконова трубка контактує з перекачуваною рідиною. Перистальтичну трубку легко обслуговувати, чистити, стерилізувати або замінювати.

Для реалізації блоку іригації обрано перистальтичний насос, зображений на рисунку 2.13, з живленням напругою 12 В, який перекачує 100

мл рідини за хвилину.

Рисунок 2.13 – Перистальтичний насос

2.1.6 Електромагнітний рідинний клапан

Електромагнітний (соленоїдний) рідинний клапан дозволяє керувати потоком води, який знаходиться під тиском. Функціональність полягає у відкриванні та закриванні отвору за допомогою плунжера, на який діє магнітне поле за допомогою електромагнітної котушки. Конструкція клапана [13], що проілюстрована на рисунку 2.14, складається з:
корпусу, який містить сідло для плунжера, впускний та випускний отвори;
арматурної трубки із сердечника, на яку встановлюється соленоїдна котушка;
плунжера, що ковзає у арматурній трубці;
соленоїдна котушка, що створює магнітне поле для управління положенням плунжера.

Рисунок 2.14 – Конструкція електромагнітного клапану: 1 – корпус, 2 – арматурна трубка, 3 – плунжер, 4 – соленоїдна котушка

Управляючи живленням такого клапану можна перекривати або пропускати рідину. Рівномірно розподіливши вихідний потік насосу між клапанами, можна реалізувати демультиплексування потоку води. У випадках, коли клапан має нижчу вартість ніж насос, або у системі може бути використаний лише один насос, демультиплексування потоку дозволяє знизити собівартість обладнання системи. Для імплементації блоку іригації

обрано соленоїдний клапан з живленням напругою 12 В та наріззю для підключення трубопроводу діаметром 1/2", зображений на рисунку 2.15.

Рисунок 2.15 – Соленоїдний рідинний клапан

2.1.7 Ємнісні датчики вологості ґрунту

Датчики вологості ґрунту необхідні для визначення ступеня висихання ґрунту, в якому розвиваються рослини. За принципом роботи поділяються на резистивні та ємнісні (рис. 2.16). За досвідом попередніх досліджень [14], с. 27-31], резистивні датчики вологості ґрунту швидше за ємнісні виходять з ладу через окислення електродів та подальшу втрату провідного покриття як показано на рисунку 2.17.

Рисунок 2.16 – Ємнісний датчик вологості ґрунту

Рисунок 2.17 – Датчики вологості ґрунту в результаті двомісячного використання. 1-3 – резистивні датчики, 4 – ємнісний датчик

Обрана модель ємнісних датчиків хоча і є більш стійкою до виходу з ладу через окислення ніж резистивні екземпляри, має конструктивний недолік. Як показано на рисунку 2.18, електронні компоненти датчику знаходяться безпосередньо на поверхні печатної плати. Такі компоненти схильні до виходу з ладу через можливі бризки води та поступового окислення контактів (рис. 2.19) у вологому середовищі вирощування рослин.

Рисунок 2.18 – Компоненти датчику, схильні до негативного впливу вологого середовища

Рисунок 2.19 – Сліди окислення на компонентах ємнісного датчику

Для збільшення терміну служби, розглянуті датчики необхідно модифікувати. Схильні до негативного впливу компоненти мають бути перенесені з поверхні печатної плати, яка занурюється у вологий ґрунт, або ізольовані від середовища за допомогою діелектричного матеріалу.

2.1.8 Реле

Реле виконує функцію перемикача (розірвання та відновлення електричного кола) з управлінням за допомогою електричного сигналу. Електромагнітні реле (Electromechanical relay, EMR) реалізовані за допомогою соленоїду (електромагніту) та феромагнітного провідника. Коли на обмотку електромагніту подається електричний струм, магнітне поле впливає на рухомий провідник, змінюючи його положення, що змінює цілісність електричного кола.

Транзисторні реле (Solid State Relay, SSR) є більш сучасною технологію реалізації та мають ряд переваг над електромагнітними: швидкість перемикання, відсутність характерного звуку клацання в момент переміщення провідника, більший термін працездатності, оскільки не мають рухомих частин. В свою чергу, електромагнітні реле мають порівняно нижчу вартість, є більш розповсюдженими та доступними для придбання, а головне,

через фізичну простоту реалізації, є універсальними, оскільки можуть комутувати як постійний, так і змінний, струм.

Оскільки розроблюваний пристрій, згідно до предметної області та поставлених вимог, не потребує частої зміни цілісності кола, для реалізації блоку управління, а саме управління живленням фітоосвітлення, вентиляції, насосу та клапанів, обрано модуль електромагнітних реле. Даний модуль, продемонстрований на рисунку 2.20, складається з восьми електромагнітних реле. Кожне з них відноситься до типу Single Pole Double Throw (SPDT) (рис. 2.21) [15], тобто один провідник перемикається між нормально-відкритим та нормально-закритим контактами в момент подання активного рівню управляючого сигналу. Живлення обраного модулю та комутація реле можуть бути здійснені напругою 3.3 В.

Рисунок 2.20 – Зовнішній вигляд модулю електромагнітних SPDT реле

Рисунок 2.21 – Схематичне зображення принципу роботи SPDT реле

2.1.9 Джерела живлення

Розглянуте апаратне забезпечення має наступні вимоги до живлення:
мікрокомп’ютер Raspberry Pi – ПС 5 В;
АЦП ADS1115 – ПС 3.3 В;
фітолампа – ПС 34 В;
канальний вентилятор – ЗС 220 В;
перистальтичний насос – ПС 12 В;
електромагнітні клапани – ПС 12 В;

датчики вологості ґрунту – ПС 3.3 В;

ЕМ-реле – 3.3 В.

Для поліпшення користувацького досвіду, система має живитись за допомогою підключення у мережу всього одного кабелю. Для реалізації цього, блоки живлення та інші перетворювачі електричного струму мають бути розташовані у корпусі пристрою.

Множина необхідних напруг для живлення усіх апаратних складових системи: 3.3 В, 5 В, 12 В, 34 В постійного струму та 220В змінного струму. Для задовільнення усіх перерахованих вимог до живлення, пропонується наступна схема перетворювань напруги (рис. 2.22).

Рисунок 2.22 – Перетворення напруги для живлення системи

Для реалізації розглянутих перетворень обрано:

блок живлення 12 В 10 А, що живиться змінним струмом напругою 220В (рис. 2.23);

блок живлення 34 В 1750 мА, що живиться змінним струмом напругою 220 В (рис. 2.24);

понижуючий DC-DC перетворювач потужністю 25 Ватт, з діапазоном вхідної напруги 9-35 В та вихідною напругою 5 В (рис. 2.25);

мікрокомп’ютер Raspberry Pi 3B+, що живиться напругою 5 В та має GPIO виходи з напругою 3.3 В [16] (рис. 2.26).

Рисунок 2.23 – Блок живлення 12 В 10 А

Рисунок 2.24 – Блок живлення 34 В 1750 мА

Рисунок 2.25 – DC-DC перетворювач 5 В 5 А

Рисунок 2.26 – Схема 3.3 В виходів мікрокомп’ютеру Raspberry Pi

Додатково, для налаштування частоти обертання канального вентилятору обрано симісторний регулятор потужності BTA-16 220В 2КВт, що зображений на рисунку 2.27.

Рисунок 2.27 – Симісторний регулятор потужності BTA-16 220В 2КВт

2.1.10 Роз’єми

Для поліпшення користувацького досвіду, а також зменшення негативного впливу середовища на апаратне забезпечення пристрою, підключення обладнання до системи має бути реалізоване через роз’єми.

Необхідні підключення, що мають підтримуватись пристроєм:
живлення пристрою напругою 220В;
живлення фітолампи постійним струмом 34 В;
живлення канального вентилятору змінним струмом 220 В;
живлення блоку іригації блоком управління;
підключення чотирьох датчиків вологості ґрунту;

приєднання п’яти шлангів до блоку іригації.

Для підключення пристрою до мережі та живлення змінним струмом 220 В обрано класичний роз’єм IEC 60320, зображений на рисунку 2.28. Цей стандарт роз’ємів є найбільш розповсюдженням способом підключення обчислювальних пристроїв до мережі. Через це користувач може дуже легко придбати кабель живлення необхідної довжини або використовувати наявний.

Рисунок 2.28 – Роз’єм та штекер IEC 60320

Для живлення фітолампи постійним струмом напругою 34 В необхідно дві лінії, тому обрано двоштирковий роз’єм GX16-2P (рис. 2.29), що має ступінь захисту IP55, тобто пилезахищений і допускає попадання бризок води. Через наявність ключа, тому не може бути підключений користувачем неправильно. Наявність нарізі на роз’ємі виключає можливість випадкового відключення під час роботи з рослинами.

Рисунок 2.29 – Роз’єм GX16-2P

Живлення канального вентилятору змінним струмом потребує наявності трьох ліній, через що обрано трьохштирковий роз’єм GX12-2P, зображений на рисунку 2.30. Конструкційно, даний роз’єм ідентичний з GX16 та відрізняється лише меншим діаметром.

Рисунок 2.30 – Роз'єм GX12-3P

Підключення блоку іригації до блоку управління має забезпечувати живлення постійним струмом напругою 12 В одного насосу та чотирьох електромагнітних клапанів. Замість виділення на живлення кожного елементу окремої пари ліній, загальну кількість контактів можна зменшити, використовуючи один спільний катод та п’яти окремих анодів. Завдяки цьому, необхідну кількість ліній можна зменшити до шости. Таким вимогам відповідає роз’єм GX16-6P, зображений на рисунку 2.31.

Рисунок 2.31 – Роз’єм GX16-6P

Аналогові датчики вологості ґрунту мають три контакти: два для живлення і один для результуючого показника вимірювання. Ще один додатковий контакт необхідний для реалізації визначення стану підключеності датчику. Оскільки датчики мають невеликі габарити та використовують для живлення лише напругу 3.3 В, бажано надати перевагу невеликим роз’ємам.

Прикладом роз’єму, що відповідає поставленим вимогам є розповсюджений роз’єм TRS 3,5 мм (mini-jack) (рис. 2.32), що зазвичай використовується для підключення аудіо-приладів. Існують реалізації роз’єму з різною кількістю контактів, однак для підключення датчиків необхідним є саме 4-штирковий варіант. Висока популярність роз’єму дозволить користувачу легко придбати аудіо-кабелі необхідної довжини для підключення датчиків до пристрою.

Рисунок 2.32 – Аудіокабель з 4-штирковим mini-jack штекером

2.2 Програмне забезпечення

2.2.1 Операційна система Raspberry Pi OS

Raspberry Pi OS [17] (раніше Raspbian) – це Linux-based операційна система, що є рекомендованою Raspberry Pi Foundation для використання на мікрокомп'ютері Raspberry Pi 3B+.

Дана операційна система спрощує взаємодію з зовнішнім апаратним забезпечення, оскільки Linux надає стандартизований програмний інтерфейс роботи з багатьма популярними шинами передачі даних. Узагальнений приклад роботи з шиною I2C засобами стандартної бібліотеки Linux та мови програмування C++ наведено у лістингу 2.1.

Лістинг 2.1 – Приклад роботи з шиною I2C засобами Linux

```
i2c_dev = open("/dev/i2c-1", O_RDWR);
if (i2c_dev < 0)
      // Failed to open the bus
if (ioctl(i2c_dev, I2C_SLAVE, ADDRESS) < 0)
      // Failed to acquire bus access
// Write data
if (write(i2c_dev, data, size) != size)
      // Failed to write
// Read data
uint8_t buffer [read_size] {};
if(read(i2c_dev, &buffer, read_size) != read_size)
      // Failed to read
```

Стандартний для Raspberry Pi OS пакетний менеджер apt дозволяє легко встановлювати у систему компілятор, систему збірки, бібліотеки та інші залежності, необхідні для розробки проекту та вирішення поставлених задач. Приклад встановлення залежностей за допомогою apt наведено у лістингу 2.2.

Лістинг 2.2 – Встановлення залежностей проекту пакетним менеджером apt

```
apt-get update
```

```
apt-get install git cmake build-essential qt5-default autoconf libtool
pkg-config
```

Операційні системи на базі Linux використовують менеджер служб systemd. За допомогою цього менеджеру легко налаштовувати запуск системних сервісів (демонів), обслуговувати монтування накопичувальних пристроїв, підтримка знімків (збережений стан процесу), відстеження та управління процесами, ведення журналів та багато інших засобів системного адміністрування. Саме цей засіб обраний для управління сервісами розроблюваної системи. Приклади створення та управління станом сервісу наведено у лістингах 2.3 та 2.4 відповідно.

Лістнинг 2.3 – Створення сервісу systemd

```
[Unit]
Description=Опис сервісу
[Service]
ExecStart=/usr/bin/sample_executable
[Install]
WantedBy=multi-user.target
```

Лістнинг 2.4 – Управління сервісом systemd

```
# systemctl start sample-service
# systemctl stop sample-service
# systemctl restart sample-service
$ systemctl status sample-service
```

2.2.2 Мова програмування C++

C++ – це статично типізована мова програмування, що компілюється безпосередньо для цільової платформи. Кросплатформеність програмного забезпечення, розробленого даною мовою, реалізується шляхом кроскомпіляції, тобто налаштування засобів та середовища збірки проекту безпосередньо для конкретної цільової платформи [18]. Платформа, на якій збірка виконується, може відрізнятись архітектурою, операційною системою або іншими характеристиками від цільової.

Обрана операційна система Raspberry Pi OS, на момент виконання

дослідження, для збірки C++ проектів надає компілятор GCC версії 8.3, що підтримує стандарт мови, затверджений приблизно у 2017 році. C++17, у порівнянні з попередніми стандартами, надає наступні нові можливості [19]:
декомпозицію при оголошенні змінних (structural bindings);
автоматичне виведення параметрів шаблонів;
оголошення вкладених просторів імен;
атрибути maybe_unused, fallthrough, nodiscard;
клас string_view, variant та optional;
бібліотеку для роботи с файловими системами.

Наявність великої стандартної бібліотеки, підтримка багатьох існуючих сторонніх бібліотек, розроблених мовами C та C++, можливість одночасного використання декількох парадигм програмування, кроскомпіляція, швидкість, а також активний розвиток мови у відповідності до сучасних підходів розробки, робить мову C++17 ефективним засобом для розробки системи і реалізації поставлених задач.

2.2.3 Система збірки CMake

CMake [20] – це популярна кросплатформенна утиліта для збірки проектів з вихідного коду, що зазвичай використовується для мов C та C++. Кросплатформеність зробила CMake де-факто основним засобом збірки застосунків та бібліотек у екосистемі мови C++. Це досягається за рахунок того, що дана утиліта не виконує безпосередньо компіляцію вихідного коду та інші етапи збірки, а лише є абстракцією над іншими популярними, але платформозалежними, системами збірки: make, Ninja чи MS Build.

Для впровадження системи збірки CMake у розроблюваний проект, необхідно імперативно описати процес збірки у текстовому файлі CMakeLists.txt, що має знаходитись у кореневій директорії вихідного коду. Приклад збірки застосунку, що складається з одного файлу вихідного коду та використовує одну бібліотеку, наведено у лістингу 2.5.

Лістинг 2.5 – Приклад збірки C++ застосунку за допомогою CMake

```
project(sample)
add_executable(sample source.cpp)
target_link_libraries(sample dependency_lib)
```

Можливість використання дерева таких файлів опису дозволяє декомпозувати комплексні проекти. Зазвичай великі проекти потребують використання сторонніх бібліотек або фреймворків. Приклад пошуку у системі та використання бібліотек популярного C++ фреймворку Qt наведено у лістингу 2.6.

Лістинг 2.6 – Пошук та використання сторонніх бібліотек за допомогою CMake

```
project(complex)
add_library(${PROJECT_NAME} complex_source.cpp)
find_package(Qt5 REQUIRED COMPONENTS Core)
target_link_libraries(${PROJECT_NAME} Qt5::Core)
```

2.2.4 Виклик віддалених процедур gRPC

Виклик віддалених процедур – це один із шляхів реалізації мережевої комунікації між застосунками [21]. Підхід грунтується на реалізації процедур (функцій чи методів), виклик яких синтаксично виглядає як використання звичайної, локальної, процедури. При цьому, смислове навантаження процедури виконується у іншому застосунку, повертаючи результат виконання в місце виклику локальної процедури. Діаграма послідовності виконання віддаленої процедури зображена на рисунку 2.33.

Рисунок 2.33 – Діаграма послідовності виконання віддаленої процедури

В залежності від імплементації та налаштувань, шляхом протоколів

передачі даних комп’ютерних мереж або засобів ОС міжпроцесної комунікації, віддалена процедура може виконуватись як на тому ж комп’ютері у окремому процесі, так і на віддаленій географічно машині.

Синтаксична простота використання, абстрагування предметної логіки процедури від мережевих комунікацій та простота зміни місця виконання функції, робить виклик віддалених процедур корисним засобом для реалізації розподілених систем [22].

gRPC [23] – це фреймворк віддаленого виклику процедур з відкритим вихідним кодом, що розповсюджується за ліцензією Apache License 2.0, тобто є безкоштовним для використання у комерційних і некомерційних продуктах та допускає модифікацію, якщо це необхідно. Головними позитивними особливостями фреймворку gRPC є:

розповсюдженість та популярність – детальна документація та велика кількість прикладів;

простота створення сервісу та сторін комунікації;

підтримка двосторонньої комунікації – клієнт та сервер є рівнозначними учасниками зв’язку та можуть посилати запити у обох напрямках;

кросплатформеність та кросмовність – сторони комунікації можуть бути імплементовані різними мовами програмування та виконуватись на різних платформах.

Можливість використання даного засобу комунікації на різних платформах та розробки застосунків різними мовами програмування є дуже важливою для імплементації розподіленої системи. Частини системи, що беруть участь у комунікаціях, можуть розроблятись засобами, які є оптимальними для даного типу задач та платформи. За допомогою лише фреймворку gRPC можна реалізувати взаємодію частин системи як між сервісами на пристрої управління вирощуванням рослин, так і їх налаштування користувачем з мобільного застосунку.

2.2.5 СУБД SQLite 3

SQLite [24] – це бібліотека, розроблена мовою C, що виконує функції системи управління базами даних. На відміну від багатьох інших СУБД, SQLite для роботи з даними не потребує активного серверного процесу, що буде обробляти запити до баз даних. База даних SQLite – це лише файл у файловій системи, дані у якому упорядковані для ефективної взаємодії з ними. Запити на роботу з даними описуються декларативною мовою SQL.

Через свою швидкість, легкість та надійність, СУБД SQLite набула високої популярності та використовується для збереження та упорядкування даних при розробці мобільних та вбудованих систем. Основною перевагою SQLite є простота. Оскільки БД SQLite це лише файл на диску, робота з ним не потребує вимогливого до обчислювальних ресурсів серверного процесу. Однак, через цю простоту, SQLite має очевидні недоліки у порівнянні з комплексними аналогами. Через обмеження файлових систем, бази даних SQLite не підтримують одночасну роботу з декількома потоками чи процесами. Також, через локальний характер збереження та роботи з даними, SQLite не підтримує доступ через мережу та аутентифікацію користувачів.

СУБД SQLite 3 це легкий ефективний інструмент [25], коли дані необхідно зберігати локально та тільки один потік одночасно оновлює базу даних. Оскільки кожен сервіс розроблюваної системи має свій окремий набір налаштувань для збереження, обрана бібліотека є оптимальним засобом роботи з даними. У випадках, коли доступ к даним може стати необхідним для інших пристроїв, запит може бути оброблений окремо виділеним сервісом та віддаленим викликом процедур.

2.2.6 Автоматизоване тестування GoogleTest

Автоматизація тестування є важливим фактором підвищення надійності розроблюваного програмного забезпечення [26]. На відміну від мануального тестування, коли відповідність продукту вимогам перевіряє людина та

складає звіт, автоматизоване тестування виконується за допомогою спеціального ПЗ. Участь людини зменшується, мінімізуючи можливі помилки через неуважність. Швидкість виконання сильно підвищується, дозволяючи раніше виявляти факт несправності, визначати її місце та характер, що в результаті робить процес виправлення простішим та дешевшим.

GoogleTest [27] – це фреймворк для реалізації автоматичних тестів мовою C++, що має відкритий вихідний код та розроблений компанією Google. GoogleTest дозволяє створювати швидкі, ізольовані один від одного тести, які у випадку виявлення несправності згенерують детальний звіт для подальшого аналізу розробником. Приклад реалізації автоматичного тесту для тестування функції обчислення факторіалу наведено у лістингу 2.7.

Лістинг 2.7 – Тестування функції обчислення факторіалу

```
TEST(calcFactorialTest, CheckPositiveInput) {
  EXPECT_EQ(calcFactorial(1), 1);
  EXPECT_EQ(calcFactorial(2), 2);
  EXPECT_EQ(calcFactorial(3), 6);
  EXPECT_EQ(calcFactorial(8), 40320);
}
```

При тестуванні комплексних сутностей, що отримують дані чи сигнали управління від інших сутностей, постає задача точного відтворення стану та сценаріїв поведінки об’єктів взаємодії. Для рішення цієї задачі необхідно мінімізувати зв’язність частин системи під час побудови архітектури програмного забезпечення. Це надасть можливість створювати фіктивні сутності (mock objects), що надають вихідні дані до предметної логіки системи або аналізують результуючу поведінку чи дані. Вихідні дані фіктивного об’єкту доступу можуть бути заздалегідь визначені тестовою вибіркою, а можуть базуватися на взаємодії з іншою сутністю. Приклад реалізації та використання фіктивної сутності за допомогою GoogleTest наведено у лістингу 2.8.

Лістинг 2.8 – Реалізація та використання фіктивної сутності

```
class RelayMock : public IRelay
{
public:
    MOCK_METHOD(bool, enabled, (), (const, override));
    MOCK_METHOD(void, enable, (), (override));
    MOCK_METHOD(void, disable, (), (override));

}

// ...

TEST_F(RelayUserTest,Enable)
{
    EXPECT_CALL(relay, enable())
            .Times(1);
    user.emulate_enable();
    EXPECT_TRUE(relay.enabled());
}
```

2.2.7 Операційна система Android

Android [28] – це популярна мобільна операційна система, що розробляється компанією Google та має відкритий вихідний код. На її основі працюють смартфони, планшети, ігрові приставки, розумні годинники, телевізори, мультимедійні програвачі та електронні книги різноманітних виробників обчислювальної техніки. У травні 2017 року на конференції для розробників компанія Google оголосила, що за всю історію ОС Android було активовано на понад 2 млрд. мобільних пристроїв (рис. 2.34).

Через високу популярність операційної системи та орієнтацію саме на портативні пристрої, для поліпшення користувацького досвіду використання системи управління вирощуванням рослин, ОС Android є оптимальною цільовою платформою для розробки мобільного застосунку.

Рисунок 2.34 – Смартфон на базі ОС Android

2.2.8 Мова програмування Kotlin

Kotlin – це статично типізована мова програмування, що розробляється

компанією JetBrains. У 2017 на конференції Google IO [29] команда розробників ОС Android повідомила про отримання мовою програмування Kotlin офіційної підтримки для розробки Android-застосунків.

Мова програмування швидко набула популярність серед розробників програмного забезпечення через підтримку компіляції у байткод JVM, завдяки чому можливе використання усієї наявної кодової бази мови програмування Java, усіх бібліотек та фреймворків, безпосередньо з вихідного коду на Kotlin. Ця особливість є дуже важливою у контексті мобільної розробки, оскільки до початку підтримки програмування на Kotlin, застосунки для Android, як і сама операційна система, розроблялись мовою програмування Java. Порівняння вихідного коду на Java та Kotlin для реалізації ідентичних функцій наведено у лістингах 2.9 та 2.10 відповідно.

Лістинг 2.9 – Приклад вихідного коду мовою програмування Java

```
public void progressBar(int progress){
  String text;
  Color color;
  if (progress < 10) {
    text = “start transmition”;
    color = RED;
  } else if (progress < 95) {
    text = “transmition”;
    color = YELLOW;
  } else {
    text = “transmition ending”;
    color = GREEN;
  }
  // ...
}
```

Лістинг 2.10 – Приклад вихідного коду мовою програмування Kotlin

```
fun progressBar(progress: Int) {
  val (text, color) = when {
  progress < 10 -> “start transmition” to RED
  progress < 95 -> “transmition” to YELLOW
  else -> “transmition ending” to GREEN
  }
  // ...
}
```

2.2.9 Система збірки Gradle

Gradle [30] – це система збірки, що зазвичай використовується у Java та Kotlin проектах. Процес збірки проекту за допомогою Gradle описується за допомогою графу задач (task), які надаються плагінами. Кожна задача може залежити від іншої. Саме у вигляді задач виконується робота процесу збірки. Приклад додання фреймворку gRPC до Kotlin проекту за допомогою Gradle наведено у лістингу 2.11.

Лістинг 2.11 – Додання фреймворку gRPC до проекту

```
plugins {
  id 'com.android.application'
  id 'kotlin-android'
  id 'com.google.protobuf'}
def  CURRENT_GRPC_VERSION='1.35.0'
def  CURRENT_GRPC_KOTLIN_VERSION='1.1.0'
dependencies {
  implementation "io.grpc:grpc-okhttp:$CURRENT_GRPC_VERSION"
  implementation "io.grpc:grpc-kotlin-stub:$CURRENT_GRPC_KOTLIN_VERSION"
  implementation "io.grpc:grpc-protobuf-lite:$CURRENT_GRPC_VERSION"
}
```

Одразу після ініціалізації Gradle проекту генерується «обгоротка» (wrapper), що містить інформацію про використовувану версіє системи збірки. Для використання обгортки також згенеровано два скрипти: gradlew для Linux та gradlew.bat для Windows. Ці скрипти є дуже корисними для налаштування сервісів неперервної інтеграції та збірки Kotlin проектів.

2.2.10 Неперервна інтеграція

Неперервна інтеграція (continuous integration, CI) це набір практик, що мають зменшити проміжки між розробкою системи, її експлуатацією та командами-учасниками. Досягнення цього можливе за допомогою впровадження автоматизації збірки, тестування, та інших етапів підготовки застосунків до подальшого розгортання на цільових середовищах. Зазвичай практики безперервної інтеграції поєднуються з безперервним розгортанням системи (continuous deployment, CD).

Шлях системи від вихідного коду програмного забезпечення до

цільового середовища у командах без практики CI/CD проілюстровано на рисунку 2.35.

Рисунок 2.35 – Розробка та розгортання системи без CI/CD

1. Інженер оновлює вихідний код ПЗ та додає зміни до репозиторію системи контролю версій;
2. Відповідальна за збірку людина підготовлює оновлений застосунок до подальшого тестування. Можливі на цьому етапі помилки: невідповідність середовищ збірки, версій залежностей та налаштувань системи. Збірка застосунку може бути невдалою, якщо перший інженер допустив помилку;
3. Виконується автоматизоване або мануальне тестування системи. Можливі на цьому етапі помилки: тестування неправильної версії системи, невідповідність середовища тестування;
4. Ручне розгортання системи на цільовому середовище. Виконання цього етапу мануально є довгим процесом, в ході якого можливі помилки через неуважність.

Впроваджуючи практики та засоби CI/CD, шлях системи від вихідного коду програмного забезпечення до цільового середовища змінюється відповідно до схеми на рисунку 2.36.

Рисунок 2.36 – Розробка та розгортання системи з CI/CD

1. Інженер оновлює вихідний код ПЗ та додає зміни до репозиторію системи контролю версій.

5. CI сервіс виявляє подію оновлення вихідного коду та ініціює процес збірки та автоматизованого тестування у ізольованих середовищах із затвердженими налаштуваннями.

6. У випадку невдалої збірки застосунку через некоректність оновлення вихідного коду, сервіс повідомляє про це інженеру.

7. Вдало зібраний застосунок упаковується у артефакт. Відповідальний за подальше мануальне тестування застосунку інженер отримує повідомлення.

8. Інженер проводить мануальне тестування.

а) У випадку виявлення несправностей системи, інженер-тестувальник відзначає артефакт як недопустимий. Інженер-розробник отримує повідомлення.

б) Інакше, інженер-тестувальник відзначає артефакт як допустимий. Починається процес розгортання;

9. CD сервіс ініціює процес автоматизованого розгортання.

Прикладом CI/CD сервісу є GitHub Actions. Цей інструмент не потребує розгортання на власному сервері, а є вбудованим безпосередньо у популярний сервіс збереження проектів GitHub. Корисною перевагою є наявність великої кількості шаблонів для автоматизації CI/CD задач для різних мов програмування та технологій. Приклад встановлення залежностей, збірки та подальшого автоматизованого тестування функції обчислення факторіалу засобами GitHub Actions наведено у лістингу 2.12.

Лістинг 2.12 – Неперервна інтеграція функції обчислення факторіалу засобами GitHub Actions

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2

    - name: Install GoogleTest framework
        run: sudo apt-get install libgtest-dev; cd /usr/src/gtest; sudo
cmake CMakeLists.txt; sudo make; sudo cp ./lib/lib*.a /usr/lib; sudo ln -s
/usr/lib/libgtest_main.a /usr/local/lib/libgtest_main.a; sudo ln -s
/usr/lib/libgtest.a /usr/local/lib/libgtest.a;

    - name: Configure CMake project
```

```
              run: cmake -B ${{github.workspace}}/build -DCMAKE_BUILD_TYPE=$
{{env.BUILD_TYPE}}

          - name: Build project
                run: cmake --build ${{github.workspace}}/build --config $
{{env.BUILD_TYPE}}

          - name: Test function
            working-directory: ${{github.workspace}}/build
            run: ./calcFactorialTest
```

2.3 Висновки по розділу 2

На основі проведеного аналітичного огляду апаратного забезпечення, для розробки пристрою управління вирощуванням рослин, оптимальними засобами є: мікрокомп’ютер Raspberry Pi 3B+, АЦП ADS1115, LED-фітолампа, канальний вентилятор, перистальтичний насос, електромагнітні клапани, ємнісні датчики вологості ґрунту, електромагнітні реле, множина перетворювачів електричного струму та роз’ємів для підключення обладнання. Датчики вологості ґрунту потребують модифікації для збільшення терміну служби та реалізації можливості визначення підключеності до пристрою.

Для розробки програмного забезпечення пристрою, оптимальними є наступні засоби: мова програмування C++17, система збірки проектів CMake, СУБД SQLite3. Мобільний застосунок для ОС Android має розроблятись мовою програмування Kotlin з використанням системи збірки Gradle. Міжсервісна комунікація програмного забезпечення пристрою, а також комунікація мобільного застосунку з пристроєм, може бути імплементована використовуючи фреймворк виклику віддалених процедур gRPC.

Впровадження практик автоматизованого тестування та неперервної інтеграції має підвищити надійність системи.

3 РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОЇ СИСТЕМИ

3.1 Розробка апаратного забезпечення

У процесі реалізації розподіленої системи управління вирощуванням рослин розробка апаратного забезпечення є першим етапом. Для реалізації пристрою необхідно вирішити наступні задачі:
розробити схему підключення апаратного забезпечення системи;
ізолювати сантехнічне обладнання від управляючого та джерел живлення;
розробити корпус блоку управління;
розробити плати розширення для спрощення компонування обладнання у корпусі;
зібрати блок управління відповідно до схеми підключення;
розробити корпус блоку іригації;
підготувати сантехнічне обладнання до використання;
зібрати блок іригації відповідно до схеми підключення;
провести мануальне тестування демультиплексування потоку води;
розробити спосіб визначення підключеності датчиків;
модифікувати датчики вологості ґрунту для підвищення терміну служби.

3.1.1 Схема підключення

Схема підключення всього апаратного забезпечення системи має враховувати наступні вимоги:
забезпечення можливості живлення обладнання напругами 3.3 В, 5 В, 12 В, 34 В постійного струму та 220 В змінного струму;
живлення та подальше автоматичне управління насосом та рідинними клапанами;
живлення та подальше автоматичне управління фітолампою;

регулювання швидкості обертання, живлення подальше автоматичне управління канальним вентилятором;

наявність вимикачів для ручного відключення живлення сантехнічного обладнання, фітолампи та канального вентилятору;

живлення мікрокомп’ютеру;

підключення аналогових датчиків вологості ґрунту до АЦП та подальшу передачу даних до мікрокомп’ютеру;

використання підтягуючих резисторів для реалізації визначення підключеності датчиків.

Для реалізації поставлених вимог до системи, необхідно використовувати інтерфейс введення/виведення загального призначення (GPIO) мікрокомп’ютеру Raspberry Pi. Забезпечивши живлення напругою 5 В, мікрокомп’ютер може живити підключені до нього модуль реле, АЦП та датчики напругою 3.3 В. Виходи SCL та SDA необхідні для комунікації з АЦП за допомогою шини I2C. Сім виходів для сигналів управління реле, що розривають кола живлення освітлення, вентиляції та сантехнічного обладнання. Ще чотири виходи для зчитування стану підключеності датчиків. Схематичне зображення мікрокомп’ютеру Raspberry Pi 3B+ та використовуваних виходів зображено на рисунку 3.1.

Рисунок 3.1 – Схематичне зображення мікрокомп’ютеру та GPIO виходів

Датчики вологості ґрунту передають результати вимірювань у вигляді аналогового сигналу – напруги, що змінюється в залежності від фізичного стану вимірюваного середовища. Аналоговий сигнал перетворюється на цифровий за допомогою АЦП та далі може бути зчитаний мікрокомп’ютером шляхом шини I2C. Схема підключення датчиків та АЦП зображено на рисунку 3.2.

Рисунок 3.2 – Схема підключення АЦП та датчиків вологості ґрунту до мікрокомп’ютеру

За допомогою підтягуючих резисторів можна реалізувати визначення, чи підключений датчик до пристрою. Для цього вводиться додаткова четверта лінія підключення датчиків, що на схемі позначена як SC (sensor connected, датчик підключений), та резистори номіналом 10 КОм. Схема реалізації визначення підключеності зображено на рисунку 3.3.

Рисунок 3.3 – Схема підключення лінії визначення підключеності датчику

Для забезпечення усієї множини напруг, необхідних для живлення елементів системи, використовується підключення до мережі 220 В змінного струму, блоки живлення 12 і 24 В, понижуючий перетворювач постійного струму з вихідною напругою 5В та мікрокомп’ютер, що має виходи живлення 3.3 В. Схема кола живлення системи продемонстрована на рисунку 3.4.

Рисунок 3.4 – Схема підключення джерел живлення

Живлення LED-фітолампи забезпечується блоком живлення 34 В. Автоматичне управління реалізується шляхом розірвання та відновлення цілісності електричного кола живлення за допомогою реле, що отримує логічний сигнал від GPIO виходу мікрокомп’ютеру. Додатково користувачу

надається фізичний вимикач, підключений шляхом монтажного «І». Схема підключення живлення та автоматичного управління фітоосвітленням наведено на рисунку 3.5.

Рисунок 3.5 – Схема живлення та автоматичного управління фітоосвітленням

Для роботи канальний вентилятор потребує живлення змінним струмом 220 В та може бути підключений безпосередньо до мережі. Однак для реалізації автоматичного управління вентиляцією використовується електромагнітне реле подібно до фітолампи. Додатково користувачу надається регулятор потужності для встановлення бажаної частоти обертання вентилятору та вимикач. Схема підключення вентиляції продемонстрована на рисунку 3.6.

Рисунок 3.6 – Схема живлення та автоматичного управління канальним вентилятором з можливістю регулювання частоти обертів

Перистальтичний насос, як і рідинні електромагнітні клапани потребують живлення напругою 12 В, що забезпечений з відповідного блоку живлення. Для зменшення фізичних ліній живлення, усі прилади управління потоком води використовують спільний катод, що може бути розірваний вимикачем, якщо знадобиться одночасове відключення усього сантехнічного обладнання системи. Для управління активністю окремих приладів використовуються реле, які розривають відповідні аноди як показано на рисунку 3.7.

Рисунок 3.7 – Схема живлення та автоматичного управління сантехнічним обладнанням

Розглянуті схеми підключення враховують усі поставлені вимоги. Загальну схема підключення усього апаратного забезпечення системи наведено на рисунку 3.8.

Рисунок 3.8 – Загальна схема підключення апаратного забезпечення системи

3.1.2 Ізоляція сантехнічного обладнання

Розподілена система управління вирощуванням рослин є комплексною, оскільки базується на аграрній предметній області, використовує програмне забезпечення для імплементації логіки системи, апаратне забезпечення для аналізу та впливу на навколишнє середовище, в тому числі сантехнічне обладнання для створення потоку води та зволоження ґрунту рослин.

Через провідникові фізичні властивості води, несправності сантехнічного обладнання можуть призвести до виведення з ладу коштовного та критичного для роботи системи обчислювального обладнання.

Для рішення даної проблеми сантехнічне обладнання потрібно фізично ізолювати від обчислювального обладнання та джерел живлення. Вводяться поняття «блоку управління» та «блоку іригації». Блок управління інкапсулює забезпечення живлення та управління системою. Блок іригації містить лише сантехнічне обладнання, необхідне для реалізації демультиплексування потоку води. Фізичне розділення апаратного забезпечення між блоком управління, блоком іригації та зовнішнім середовищем наведено на рисунку

3.9. Взаємодія частин системи має відбуватись шляхом розглянутих раніше роз’ємів для спрощення процесу підключення.

Рисунок 3.9 – Фізична декомпозиція пристрою на блок управління та блок іригації

3.1.3 Корпус блоку управління

Для розробки корпусу блоку управління обрано пластиковий паралелепіпед з розмірами 25x16x8 см, зображений на рисунку 3.10. Звичайний харчовий контейнер є хорошою заготовкою для прототипування корпусу пристрою, оскільки виготовлений із діелектричного матеріалу, що легко обробляється без спеціальних інструментів чи верстатів. Такі контейнери мають невисоку вартість, виготовляються різних розмірів, прості для вибору та придбання.

Рисунок 3.10 – Заготовка корпусу блоку управління

Відповідно до схеми підключення апаратного забезпечення та схеми декомпозиції пристрою, блок управління має підтримувати наступні підключення за допомогою розглянутих роз’ємів:
живлення системи, під’єднавши кабелем до мережі – роз’єм IEC 60320;
живлення фітолампи постійним струмом – роз’єм GX-16 2-штирковий;
живлення канального вентилятору – роз’єм GX-12 3-штирковий;
живлення та управління сантехнічним обладнанням (блок іригації) – роз’єм GX-16 6-штирковий;

підключення чотирьох датчиків вологості ґрунту – роз’єм 3,5 мм (mini jack).

Додатково необхідно розташувати на корпусі блоку управління ручку регулювання обертів вентилятору та вимикачі освітлення та вентиляції. Зібраний корпус блоку управління з встановленими роз’ємами, ручкою вентиляції та двома вимикачами зображено на рисунках 3.11 та 3.12.

Рисунок 3.11 – Роз’єми на задньому боці блоку управління

Рисунок 3.12 – Вимикачі на передньому боці блоку управління

3.1.4 Плати розширення

Через велику кількість провідників та з’єднань у блоці управління, для спрощення підключення апаратного забезпечення, розроблено плати розширення.

Перша під’єднується безпосередньо до GPIO мікрокомп’ютеру Raspberry Pi за допомогою 40-контактного PLS-гнізда (рис. 3.13), та на своїй поверхні має 10-штиркову та 16-штиркову PLS-планки, АЦП ADS1115 і чотири резистори номіналом 10 КОм (рис. 3.14). Схема підключення компонентів першої плати розширення зображена на рисунку 3.15.

Рисунок 3.13 – Нижня площина першої плати розширення

Рисунок 3.14 – Верхня площина першої плати розширення

Рисунок 3.15 – Схема підключення компонентів першої плати розширення

Друга плата розширення необхідна дня спрощення підключення датчиків вологості ґрунту. Плата має п’ять трьохштиркових роз’ємів для підключення та живлення датчиків, отвори для кріплення на поверхню за допомогою стійок з наріззю М3 і шлейфу і 16-контактного PLS-гнізда для підключення до першої плати розширення (рис. 3.16). Схема другої плати розширення продемонстрована на рисунку 3.17.

Рисунок 3.16 – Роз’єми для підключення датчиків вологості ґрунту на другій платі розширення

Рисунок 3.17 – Схема другої плати розширення

3.1.5 Компонування блоку управління

Для зменшення площі корпусу, необхідної для закріплення усіх складових пристрою, можна використовувати різьбові стійки та пошарове розташування невеликих за об’ємом елементів. Саме таким чином закріплено деякі частини блоку управління, а саме мікрокомп’ютер Raspberry Pi, модуль електромагнітних реле, понижуючий перетворювач постійного струму, необхідний для живлення мікрокомп’ютеру, а також плата розширення із

роз’ємами підключення датчиків (рис. 3.18). Закріплення модулю реле та роз’ємів для датчиків вологості ґрунту на верхньому шарі сильно спрощує процес монтажу, оскільки робить доступними необхідні контакти та клеми. Між собою частини пристрою з’єднані за допомогою шлейфів, укритих спіральним обплетенням з декоративною метою.

Рисунок 3.18 – Пошарове закріплення складових пристрою за допомогою різьбових стійок

Площина верхнього шару та апаратне забезпечення на ньому закріплені за допомогою стійок та гвинтів, що мають нарізь М3. Натомість мікрокомп’ютер Raspberry Pi, через високу щільність розташування компонентів на печатній платі, має отвори під кріплення із наріззю М1, що продемонстровано на рисунку 3.19.

Рисунок 3.19 – Стійки з наріззю М1 для закріплення мікрокомп’ютеру Raspberry Pi у корпусі пристрою

3.1.6 Корпус блоку іригації

Функцією блоку іригації є демультиплексування потоку води. Оскільки електромагнітні клапани мають меншу вартість ніж окремі насоси для кожної зони іригації, за допомогою цього методу можна зменшити загальну собівартість апаратного забезпечення системи.

Розроблюваний блок іригації містить один насос та чотири електромагнітних рідинних клапани. Корпус блоку містить шостиконтактний

роз’єм для підключення живлення сантехнічного обладнання та один вимикач. Використовуючи спільний катод при живленні усіх перерахованих елементів можна не тільки зменшити кількість провідникових ліній у кабелі з’єднання з блоком управління, а також реалізувати можливість екстреного відключення всього блоку та його складових за допомогою одного вимикача. Схема підключення живлення та рідинних шлангів до насосу та клапанів зображена на рисунку 3.20.

Рисунок 3.20 – Схема підключення блоку іригації

Згідно зі схемою підключення сантехнічного обладнання, корпус блоку іригації містить вимикач, роз’єм GX16-6P та п’ять з'єднань для рідинних шлангів, що продемонстровано на рисунках 3.21, 3.22 та 3.23 відповідно.

Рисунок 3.21 – Вимикач екстреного відключення блоку іригації

Рисунок 3.22 – Блок іригації, підключений до блоку управління за допомогою роз’єму GX16-6P

Рисунок 3.23 – З’єднання з рідинними шлангами на корпусі блоку іригації

3.1.7 Підготовка сантехнічного обладнання

Для приєднання трубопроводу до електромагнітних рідинних клапанів, обраних для реалізації блоку іригації, використовується нарізь діаметром ½” (рис. 3.24). Натомість, внутрішній діаметр трубок насосу (рис. 3.25) дорівнює лише 5 мм.

Рисунок 3.24 – Нарізь ½” на корпусі рідинного клапана

Рисунок 3.25 – Трубки перистальтичного насосу

Для підключення трубок перистальтичного насосу до соленоїдних клапанів і, далі, до іригаційних шлангів, використовуються різьбові з’єднання та штуцери, які зображені на рисунку 3.26.

Рисунок 3.26 – Сантехнічні з’єднання

Для забезпечення справного потоку води по трубопроводу, безпосереднього з’єднання сантехнічних засобів за допомогою різьби замало. Як показано на рисунку 3.27, через недостатньо щільне різьбове з’єднання елементів, при активації насосу рідина протікає у місцях приєднання.

Рисунок 3.27 – Протікання рідини через неправильне з’єднання сантехнічних

засобів

Для герметизації різьбових з’єднань трубопроводу використовується фум-стрічка та набір гайкових ключів. Результат ущільнення різьбових з’єднань трубопроводу за допомогою фум-стрічки показано на рисунку 3.28.

Рисунок 3.28 – Іригаційні шланги, герметично під’єднані до електромагнітного клапану

3.1.8 Компонування блоку іригації

Після підготовки герметичного з’єднання сантехнічних компонентів та реалізації корпусу, стає можливим остаточне компонування блоку іригації. Підключення живлення сантехнічного обладнання та провідників рідини виконується відповідно до схеми на рисунку 3.20. Готовий до тестування блок іригації зображено на рисунку 3.29.

Електричні провідники закріплюються у верхній частині, контакти підключення ізолюються діелектричним матеріалом для мінімізації можливості замикання у випадку виникнення сантехнічних несправностей.

Рисунок 3.29 – Готовий до тестування блок іригації

Зовнішній вигляд розробленого пристрою управління, що складається з двох блоків, наведено на рисунках 3.30 і 3.31.

Рисунок 3.30 – Задня сторона пристрою управління вирощуванням рослин

Рисунок 3.31 – Передня сторона пристрою управління вирощуванням рослин

3.1.9 Мануальне тестування демультиплексування потоку води

Для виконання тестування, блок іригації підключається до блоку управління за допомогою відповідного роз’єму. Демультиплексування потоку води, – основна функція блоку іригації, – активується шляхом відкриття одного з клапанів та подальшої активації насосу. При проведенні перевірки, у воду додається харчовий барвник для простішого спостереження за потоком рідини. Процес функціонування блоку іригації зображено на рисунку 3.32.

Рисунок 3.32 – Процес демультиплексування потоку води

3.1.10 Визначення підключеності датчиків

Дані про вологість ґрунту отримуються системою шляхом дискретизацією аналого-цифровим перетворювачем аналогового сигналу з датчиків. Як можна побачити на рисунку 3.33, датчик має лише три виходи: два для живлення компонентів та вихід для результуючого аналогового сигналу.

Рисунок 3.33 – Ємнісний датчик вологості ґрунту

У випадку, коли вихід аналогового сигналу не підключений до АЦП, а зчитування виконується, перетворювач поверне випадкове значення, що здебільшого залежить від електромагнітного стану середовища, та яке неможливо визначити заздалегідь. Для коректної обробки системою даного сценарію, необхідно реалізувати визначення, чи підключений датчик до пристрою, і тільки в такому стані виконувати зчитування даних про вологість.

Схемотехнічно поставлена задача може бути вирішена за допомогою підтягуючого резистору [31], загальна схема використання якого наведена на рисунку 3.34. Зображена система може перебувати у двох станах: вимикач замкнено та розімкнуто. Коли вимикач розімкнуто, на контакт мікроконтролеру подається високий логічний рівень. А коли вимикач замикається, через значно менших опір, контакт мікроконтролеру отримує логічний нуль від заземлення.

Рисунок 3.34 – Схема використання підтягуючого резистору

Простіше всього реалізувати обробку відключеного датчику шляхом під’єднання лінії аналогового сигналу до понижуючого (підтягуючого до заземлення) резистору як показано на схемі на рисунку 3.35. У випадку підключення датчику до роз’єму, лінія, що під’єднана до АЦП отримує напругу аналогового сигналу датчику. Якщо датчик відключено від роз’єму пристрою, завдяки понижуючому резистору, лінія отримає логічній нуль. Перевагою такого рішення є простота реалізації. Однак головним недоліком є те, що нульове цифрове значення, отримане з АЦП, може означати дві причини: роз’єм розімкнуто і нуль отримано за допомогою понижуючого резистору, та роз’єм зімкнуто, але нульова напруга аналогового сигналу є

поточним значенням з датчику відповідно до стану ґрунту.

Рисунок 3.35 – Схема підтягування лінії аналогового сигналу до логічного нуля

Іншим рішенням визначення підключеності датчику, що не має описаного недоліку двоякості нулю, є введення додаткової, четвертої, лінії. Лінія підключеності датчику (Sensor Connected, SC) підтягується за допомогою резистору до логічного нуля. Безпосередньо на датчику лінії логічної одиниці та підключеності датчику поєднані. Схему підключення зображено на рисунку 3.36. Таким чином, в момент підключення датчику до пристрою, лінія визначення підключеності змінює стан від логічного нуля на одиницю, що може бути зчитано мікрокомп’ютером. Якщо датчик виявляється не підключеним, програмне забезпечення, що виконується на мікрокомп’ютері, не виконує запит до АЦП про дані вологості ґрунту.

Рисунок 3.36 – Схема використання додаткової лінії підключеності датчику

Для перевірки запропонованого методу визначення підключеності датчику підготовлено тестовий макет відповідно до схеми, що зображена на рисунку 3.37. Схема містить підтягуючий резистор та чотири лінії підключення датчику відповідно до запропонованого методу. Додатково введено два світлодіоди. Перший з яких демонструє наявність живлення. Другий – логічний стан лінії підключеності. Демонстрація макету у стані підключеного та відключеного датчику зображено на рисунках 3.38 та 3.39 відповідно. Відповідно до результатів перевірки можна стверджувати, що

метод справно реалізує поставлену задачу.

Рисунок 3.37 – Схема макету перевірки методу визначення підключеності датчику

Рисунок 3.38 – Стан макету, коли датчик підключений

Рисунок 3.39 – Стан макету, коли датчик відключений

3.1.11 Модифікація датчиків вологості ґрунту

Розглянуті датчики вологості ґрунту схильні до виходу з ладу через, близько, пів року використання через окислення компонентів (рис. 2.19), що розташовані безпосередньо на печатній платі датчику, який занурюється у вологий ґрунт. Для підвищення терміну служби, конструкція датчику потребує модифікації.

Стандартний роз’єм підключення датчику розташований на поверхні плати та має відкриті контакти (рис. 3.40), на які може попасти волога, що призведе до замикання. Місце підключення кабелю до датчику має бути винесене на відстань 15-20 сантиметрів від плати. Для спрощення підключення та недопущення його неправильного виконання користувачем, а також реалізації описаного методу визначення підключеності датчику, симетрична трьохштиркова PLS-планка замінена на чотирьохконтактне гніздо роз’єму 3.5 мм (mini jack).

Рисунок 3.40 – Стандартний роз’єм та кабель підключення

Процес тестування функції визначення підключеності до пристрою, а також передачі даних про вологість ґрунту з датчику, що має модифікований роз’єм, зображено на рисунку 3.41.

Рисунок 3.41 – Тестування датчику з модифікованим роз’ємом підключення

Мінімізації негативний вплив вологого середовища на відкриті компоненти датчику, а також не допустити замикання через бризки води, поверхню плати, окрім вимірювальної частини, було вкрито діелектричним матеріалом.

Для виконання ізоляції було обрано епоксидну смолу через її діелектричні властивості, стійкості до вологи і сонячного випромінювання, та насамперед можливості лиття, що зменшує до мінімуму ймовірність подальшого окислення компонентів у вологому середовищі. Форма для лиття епоксидної смоли на необхідну частину поверхні датчиків зображена на рисунках 3.42 та 3.43.

Рисунок 3.42 – Нижня частина форми для лиття епоксидної смоли

Рисунок 3.43 – Верхня частина форми у процесі затвердіння смоли

Остаточний зовнішній вигляд модифікованих датчиків вологості ґрунту продемонстровано на рисунку 3.44.

Рисунок 3.44 – Модифіковані датчики вологості ґрунту

3.2 Розробка програмного забезпечення

Для реалізації розподіленої системи управління вирощуванням, на базі розробленої апаратної складової пристрою необхідно розробити програмне забезпечення. Відповідно до функціональних вимог, потрібно розв'язати наступні задачі:

обрати підходи для розробки надійного програмного забезпечення, що придатне для автоматизованого тестування та масштабування;

декомпозувати функціональність на окремі сервіси з мінімальною відповідальністю;

визначити протоколи комунікації сервісів;

розробити бібліотеку, що інкапсулює імплементацію засобів міжсервісної комунікації;

реалізувати сервіс збору даних про вологість ґрунту;

реалізувати сервіс іригації з режимом поливу за даними про вологість ґрунту та режимом періодичної іригації;

реалізувати сервіс управління фітоосвітленням, що буде надавати змогу ручного та автоматичного за часом дня управління живленням фітолампи;

реалізувати сервіс управління вентиляцією для ручного та автоматичного управління живленням канального вентилятору;

реалізувати збереження налаштувань та стану системи для подальшого

відновлення при перезавантаженні;
реалізувати фіктивні сутності для залежностей розроблюваного програмного забезпечення;
реалізувати автоматичні тести для підвищення надійності системи, використовуючи розроблені фіктивні сутності;
розробити макети графічного інтерфейсу користувача мобільного застосунку;
реалізувати сторінки налаштувань сервісів системи мобільного застосунку;
реалізувати комунікацію мобільного застосунку з пристроєм управління вирощуванням рослин;
налаштувати сервіс неперервної інтеграції на виконання збірки та автоматичного тестування оновлень вихідного коду для підвищення надійності системи.

3.2.1 Загальні підходи

Часто, при розробці програмних систем постає питання, яким чином імплементувати комплексну сутність, щоб гарантувати її надійність та не допустити змін поведінки у разі подальшого розвитку системи. Рішенням може стати реалізація автоматичних тестів для сутностей системи. Однак якщо імплементація сутності є комплексною, тобто може виконувати декілька задач чи складається з великої кількості логічних частин, важко визначити спосіб впливу для ініціації необхідного стану або події, та як перевірити коректність поведінки (рис. 3.45).

Рисунок 3.45 – Узагальнена комплексна програмна сутність та проблема її автоматизованого тестування

Вирішити розглянуту проблему можна шляхом розділення комплексної

сутності на частини відповідно до груп задач, що виконуються у системі, та подальшої взаємодії сутності з залежностями шляхом виділених абстрактних інтерфейсів. Такий підхід складається з принципів проектування об’єктно-орієнтованих систем: єдиного обов’язку (single responsibility), розділення інтерфейсу (interface segregation) та інверсії залежностей (dependency inversion).

Розглянемо приклад використання запропонованого підходу. Нехай необхідно розробити контролер, що в залежності від налаштувань, часу, або запиту від користувача буде змінювати стан реле та повідомляти про це. Зовнішніми залежностями такого контролеру є сутності, що зможуть надати дані про час (Time Provider) та налаштування (Config Provider), події запиту управління (Manual Control Provider), а також реле (Relay) і отримувач оновлень стану (Status Consumer). Визначаються відповідні абстрактні інтерфейси (рис. 3.46).

Важливо, що інтерфейси не є остаточними об’єктами. Це лише опис можливих взаємодій з об’єктом. Наприклад, абстрактне реле можна включити та виключити. Будь які інші можливості чи особливості імплементації реле контролером не використовуються, тому не включаються до інтерфейсу.

Рисунок 3.46 – Інтерфейси залежностей контролеру

Шляхом описаних абстрактних інтерфейсів контролеру надаються об’єкти реалізації. Імплементація самого контролеру тепер є доволі простою, оскільки базується лише на поставлених функціональних вимогах та використанні залежностей через лаконічні інтерфейси. Діаграма реалізованого контролеру наведена на рисунку 3.47.

Рисунок 3.47 – Реалізація сутності, що використовує залежності через абстрактні інтерфейси

Для автоматизованого тестування розробленого контролеру вводяться фіктивні сутності (mock object) – об’єкти реалізації описаних інтерфейсів, поведінка яких може бути налаштована відповідно до тестового сценарію. Надані контролеру фіктивні сутності (рис. 3.48) використовуються ним без змін реалізації. Це гарантує, що сутність, яка тестується, має ідентичну поведінку впродовж тестування та використання. Залежності-постачальники налаштовуються на надання даних із тестової вибірки, а результати взаємодії контролеру із залежностями-споживачами перевіряється тестом.

Рисунок 3.48 – Тестування сутності, надаючи фіктивні сутності

3.2.2 Побудова програмної архітектури

Декомпозиція комплексної системи на окремі частини відповідно до задач, які вони вирішують, є важливим етапом проектування програмного забезпечення. Розділення можливе на окремі процеси, бібліотеки, класи, файли чи просто функції. При декомпозиції програмного забезпечення можна виділити декілька рівнів та переваги, що вони надають:

Програмні сутності. Окремі сутності, що мають мінімальну відповідальність та інкапсулюють у собі логічно зв’язані атрибути, дозволяють абстрагуватись від деталей реалізації, в результаті чого полегшується процес відладки, розробки та модульне тестування.

Бібліотеки. Об’єднання множини пов’язаних між собою програмних сутностей у бібліотеку може зменшити час збірки проекту в деяких частих

сценаріях, а також полегшити процес розгортання оновлень окремих частин.
Сервіси. Відокремленні процеси, повноцінні програми, що виконують одну задачу у системі. Обмін даними між сервісами відбувається за допомогою мережевих з’єднань. Такий універсальний шлях передачі даних дає змогу швидко переміщувати частини системи між апаратними вузлами, або навіть легко змінювати технологію, на базі якою розробляється один сервіс, не пошкоджуючи інші.
Апаратні вузли. Комп’ютери, на яких виконується програмне забезпечення. У поєднанні з розподіленою архітектурою, дає можливість гнучко адаптувати систему під зміни потреб. Наприклад, множина апаратних вузлів з продубльованими сервісами і балансуванням запитів між ними утворюють кластер, що може обробляти великі навантаження. Розміщення вузлів у географічно відокремлених місцях дозволить підтримувати стабільність системи при пошкодженні апаратури, відсутності мережевого з’єднання або навіть електроенергії. Для міжнародних систем кластери можуть бути розміщені одночасно у різних частинах світу для мінімізації затримок при взаємодії з користувачем.

Взаємодія користувача із системою можлива за допомогою графічного інтерфейсу. В залежності від складності даних для візуалізації та кількості елементів управління, дисплей може вимагати наявності сенсорного екрану для зручної взаємодії з системою. В результаті, лише розташування графічного інтерфейсу безпосередньо на пристрої користувача потребує високовартісного дисплею, що сильно збільшує загальну ціну апаратної частини системи.

Відповідно до поставлених функціональних вимог, для реалізації розподіленої системи управління вирощуванням рослин, виділено наступні застосунки, що взаємодіють відповідно до схеми на рисунку 3.49:
сервіс вологості ґрунту;
сервіс іригації;

сервіс вентиляції;

сервіс освітлення;

мобільний застосунок.

Рисунок 3.49 – Програмна архітектура системи

Графічний інтерфейс користувача не тільки перенесено за рамки пристрою управління вирощуванням рослин, зменшуючи вартість (порівняно із попередніми дослідженнями [14]), а і розширено сценарії взаємодії. Оскільки у централізованій системі дисплей та графічний інтерфейс було вбудовано у пристрій, користувач міг взаємодіяти з системою тільки у випадку безпосередньому фізичному доступі. Реалізуючи мобільний застосунок, графічний інтерфейс користувача може бути перенесений на будь який, вже наявний, смартфон користувача, що має доступ до мережі.

3.2.3 Визначення протоколів комунікації

Обмін даними або подіями між сервісами можливий за допомогою комп’ютерних мереж або засобів операційної системи. Для реалізації взаємодії сервісів обрано фреймворк gRPC, через його кросмовні та кросплатформені можливості. Мережева комунікація реалізована на базі протоколу рівня застосунку моделі OSI – HTTP2.

Поведінка застосунків, що використовують виклик віддалених процедур, описується шляхом оголошення множини методів, що обробляє сервер та можуть бути викликані клієнтом, а також структур даних, якими виконується обмін. Приклад декларації комунікації з сервісом управління фітоосвітленням за допомогою виклику віддалених процедур наведено у лістингу 2.13. За допомогою компілятора gRPC може бути згенерований

допоміжний вихідний код як для мови програмування C++, так і для Kotlin.

Лістинг 2.13 – Декларація сервісу віддаленого виклику процедур

```
syntax = "proto3";
import "google/protobuf/empty.proto";

package open_greenery.rpc.relay;
option java_package = "com.open_greenery.mobile";
option java_outer_classname = "RelayProto";

service Relay {
  rpc SetConfig(Config) returns (google.protobuf.Empty) {}
  rpc ManualControl(ManualControlRequest) returns (google.protobuf.Empty) {}
  rpc SetMode(ModeSetting) returns (google.protobuf.Empty) {}
  rpc GetRelayStatus(google.protobuf.Empty) returns (RelayStatus) {}
  rpc GetServiceStatus(google.protobuf.Empty) returns (ServiceStatus) {}
}
message Config {
  // msecs since start of day
  int32 day_start = 1;
  int32 day_end = 2;
}
message ManualControlRequest {
  enum Control {
    CONTROL_ENABLE = 0;
    CONTROL_DISABLE = 1;
    CONTROL_TOGGLE = 2;
  }
  Control control = 1;
}
message ModeSetting {
  enum Mode {
    MODE_MANUAL = 0;
    MODE_AUTO = 1;
  }
  Mode mode = 1;
}
message RelayStatus {
  bool is_enabled = 1;
}

message ServiceStatus {
  ModeSetting mode_settings = 1;
  RelayStatus relay_status = 2;
  Config config = 3;
}
```

3.2.4 Програмний інтерфейс міжсервісної комунікації

Компілятор фреймворку gRPC на основі наведеної декларації сервісу генерує допоміжний вихідний код – заготовки для реалізації серверу та клієнту. Реалізація серверу має наслідуватись від заголовки та надати

імплементацію усіх обробників запитів. Приклад імплементації такого методу наведено у лістингу 3.1.

Лістинг 3.1 – Реалізація обробника запиту на оновлення конфігурації сервісу

```
grpc::Status Server::Service::SetConfig(
        grpc::ServerContext * context,
        const open_greenery::rpc::relay::Config * request,
        google::protobuf::Empty * response)
{
    std::ignore = context;
    std::ignore = response;
    open_greenery::dataflow::relay::Config received_config{
            QTime::fromMSecsSinceStartOfDay(request->day_start()),
            QTime::fromMSecsSinceStartOfDay(request->day_end())
    };

             auto TimeStr = [](const QTime & t) { return
t.toString("hh:mm:ss").toStdString(); };
    spdlog::debug("Relay config received: day start={}, end={}",
                  TimeStr(received_config.day_start),
                  TimeStr(received_config.day_end));

    m_config_update_handler(received_config);
    return {}; // OK
}
```

Заготовка реалізації клієнту має бути використана для перетворення типів даних проекту у відповідні аналоги, надані бібліотекою серіалізації даних protobuf, на використані якої базується gRPC. У лістингу 3.2 наведено приклад імплементації клієнтського запиту.

Лістинг 3.2 – Реалізація клієнтського запиту на оновлення конфігурації сервісу

```
void Client::set(open_greenery::dataflow::relay::Config record)
{
    grpc::ClientContext context;
    Config request;
    google::protobuf::Empty response;

    const auto day_start = record.day_start.msecsSinceStartOfDay();
    request.set_day_start(day_start);
    const auto day_end = record.day_end.msecsSinceStartOfDay();
    request.set_day_end(day_end);

               auto TimeStr = [](const QTime & t){return
t.toString("hh:mm:ss").toStdString();};
    spdlog::debug("Send relay config: start={}, end={}",
                  TimeStr(record.day_start),
                  TimeStr(record.day_end));
```

```
    auto status = m_stub->SetConfig(&context, request, &response);
    if (!status.ok())
    {
        spdlog::warn("Unsuccessful Relay::SetConfig request: {} {}",
                     status.error_code(),
                     status.error_message());
    }
}
```

Імплементації класів серверу та клієнту є реалізаціями абстрактних інтерфейсів – постачальників даних та подій, і споживачів. Відповідно до описаного підходу зменшення зв’язності сутностей, такі реалізації можуть бути надані для використання шляхом абстрактних інтерфейсів. Завдяки цьому, сутності можуть бути замінені на фіктивні для проведення тестування, не змінюючи клас-користувач. Розроблені класи коммунікацій сервісів інкасульовано у статичну бібліотеку засобами CMake, як показано у лістингу 3.3, для спрощення повторного використання у проекті.

Лістинг 3.3 – Створення бібліотеки міжсервісної комунікації за допомогою CMake

```
set(SRCS
        src/Client.cpp
        src/Server.cpp)
add_library(${PROJECT_NAME}
        ${grpc_srcs}
        ${grpc_hdrs}
        ${proto_srcs}
        ${proto_hdrs}
        ${SRCS})
target_include_directories(${PROJECT_NAME} PUBLIC inc ${PROTOC_GEN})
target_link_libraries(${PROJECT_NAME}
        ${_REFLECTION}
        ${_GRPC_GRPCPP}
        ${_PROTOBUF_LIBPROTOBUF}
        spdlog
        dataflow)
```

3.2.5 Сервіс збору даних про вологість ґрунту

Завдяки модифікації, датчики вологості ґрунту дозволяють програмному забезпеченню системи визначити стан підключеності за допомогою перевірки логічного рівня відповідної лінії. У випадку, якщо датчик не підключений, виконувати запит показань АЦП не потрібно.

Опціональність можливих значень може бути зручно оброблена за допомогою шаблонного класу стандартної бібліотеки C++ std::optional, а також модифікатора optional описі серіалізації даних бібліотекою protobuf. Якщо датчик підключений, об’єкт класу std::optional буде містити значення вологості ґрунту. Отримане значення надається сервісу іригації для обробки. Реалізація класу публікації даних модифікованого датчику наведена у лістингу 3.4.

Лістинг 3.4 – Публікація даних про вологість ґрунту з модифікованого датчику

```
      ConnectableAnalogSensorPublisher::ConnectableAnalogSensorPublisher(
      std::shared_ptr<open_greenery::dataflow::irrigation::ISensorReadProvider
> _provider,
      std::shared_ptr<open_greenery::gpio::IInputPin> _is_connected,
      std::chrono::milliseconds _period)
          : AnalogSensorPublisher(_provider, _period),
          m_is_connected(_is_connected)
      {
              if (!m_is_connected)
              {
                          throw  std::logic_error("Is-sensor-connected  pin  is
nullptr");

              }
      }

      void ConnectableAnalogSensorPublisher::notify(std::int16_t _val) const
      {
          std::lock_guard<std::mutex> l(m_notificators_mutex);
          for (const Notificator & ntf : m_notificators)
          {
                                          if     (m_is_connected->read()     ==
open_greenery::gpio::LogicLevel::HIGH)
              {
                  ntf(_val);
              }
          }
      }
```

3.2.6 Сервіс іригації

Відповідно до функціональних вимог системи, сервіс іригації має підтримувати полив чотирьох незалежних зон з рослинами. Кожна зона може бути окремо налаштована. Активація зволоження може виконуватись періодично, або за ступенем вологості ґрунту.

Режим періодичної іригації може використовуватися для початкових стадій розвитку рослини, коли об'єму ґрунту ще недостатньо для занурення датчику. Або коли необхідно періодичне надання константного об'єму рідини, незалежно від швидкості висихання ґрунту. Наприклад, удобрення рослини рідкими засобами. Залежностями обробника режиму періодичної іригації є користувацькі налаштування, поточна дата та час, і об'єкт виконання поливу, що надаються через абстрактні інтерфейси. Реалізація обробнику даного режиму наведена у лістингу 3.5.

Лістинг 3.5 – Реалізація обробнику періодичного режиму іригації

```
PeriodicModeHandler::PeriodicModeHandler(
        std::unique_ptr<open_greenery::dataflow::irrigation::IPeriodicMo
deConfigProvider> config_provider,
        std::shared_ptr<open_greenery::dataflow::time::ICurrentDateTimeP
rovider> datetime_provider,
        std::shared_ptr<open_greenery::pump::IPump> pump
        )
        :m_config_provider(std::move(config_provider)),
        m_datetime_provider(std::move(datetime_provider)),
        m_pump(std::move(pump))
{
    assert(m_config_provider);
    assert(m_datetime_provider);
    assert(m_pump);
}

void PeriodicModeHandler::handle()
{
    // Update config
    const auto cfg = m_config_provider->get();
    if (cfg.has_value())
    {
        m_current_config = cfg;
    }

    // Skip if config is empty
    if (!m_current_config.has_value())
    {
        return;
    }

    const auto now = m_datetime_provider->get();
    if (now > m_next_irrigation_tp)
    {
        m_pump->water(m_current_config->volume);
                                                m_next_irrigation_tp     =
m_next_irrigation_tp.addMSecs(m_current_config->duration.count());
    }
}
```

У режимі іригації за вологістю ґрунту полив має виконуватись тільки

після достатнього висихання. Завеликий константний рівень вологості може призвести до розповсюдження грибків та гниття коренів. Натомість низький константний рівень вологості призведе до поступового висихання рослини. Саме тому стан ґрунту має змінюватись від сухого до мокрого і назад.

Реалізація обробника режиму іригації за вологістю ґрунту базується на іригаційному автоматі, розробленому в ході попередніх досліджень [14]. Граф схема іригаційного автомату наведена на рисунку 3.50.

Рисунок 3.50 – Граф схема іригаційного автомату

3.2.7 Демультиплексування потоку води

Надання реалізацій залежностей шляхом абстрактних інтерфейсів не тільки поліпшує придатність до автоматизованого тестування, а також полегшує додання нової функціональності.

Впровадження у систему підтримки іригації окремих зон потребує використання демультиплексування потоку води за допомогою розробленого блоку іригації. З боку програмного забезпечення це потребує послідовної активації рідинного клапану та насосу.

Використання сервісом іригації об’єктів виконання поливу через абстрактний інтерфейс дозволяє легко впровадити нову функціональність без жодних змін у самій реалізації іригаційного контролеру. Замість звичайного насосу, контролеру надається сутність, що має той же програмний інтерфейс, але виконує спочатку відкриття клапану і тільки потім активує насос, як показано на рисунку 3.51.

Рисунок 3.51 – Впровадження функціональності демультиплексування потоку води у сервіс іригації

3.2.8 Сервіс управління фітоосвітленням

Сервіс управління фітоосвітленням має підтримувати включення та виключення фітолампи при запитах від користувача, а також автоматично в заданий час відповідно до налаштувань. Програмними залежностями контролеру фітоосвітлення є реле, налаштування режиму роботи (автоматично чи тільки за запитом), час автоматичного включення та виключення, поточний час, та сутність-споживач поточного стану контролеру.

Завдяки зниженню зв’язності між сутностями за допомогою обраного підходу, а також наданню залежностей через абстрактні інтерфейси, реалізація самого контролеру стає доволі простою. Реалізація обробки запиту користувача на зміну стану наведена у лістингу 3.6. Реалізація автоматичного управління фітоосвітленням за часом наведена у лістингу 3.7.

Лістинг 3.6 – Обробка запиту користувача на зміну стану фітолампи

```
void RelayController::handleAutomaticControl()
{
    spdlog::trace("Handle auto mode");

    auto is_event = [this](QTime some_time, QTime event_time) {
                return (some_time >= event_time) && (some_time < 
event_time.addMSecs(THREAD_PERIOD.count()));
    };

    spdlog::trace("Current time request");
    const auto current_time = m_current_time_provider->get();
    std::lock_guard<std::mutex> l (m_config_mutex);
     if (is_event(current_time, m_current_config.day_start) && !m_relay-
>enabled())
    {
        spdlog::info("Time to enable");
        m_relay->enable();
    }
      if (is_event(current_time, m_current_config.day_end) && m_relay-
>enabled())
    {
        spdlog::info("Time to disable");
        m_relay->disable();
    }
}
```

Лістинг 3.7 – Обробник автоматичного управління фітоосвітленням

```
void RelayController::handleAutomaticControl()
{
    spdlog::trace("Handle auto mode");

    auto is_event = [this](QTime some_time, QTime event_time) {
                return (some_time >= event_time) && (some_time < event_time.addMSecs(THREAD_PERIOD.count()));
    };

    spdlog::trace("Current time request");
    const auto current_time = m_current_time_provider->get();
    std::lock_guard<std::mutex> l (m_config_mutex);
     if (is_event(current_time, m_current_config.day_start) && !m_relay->enabled())
    {
        spdlog::info("Time to enable");
        m_relay->enable();
    }
      if (is_event(current_time, m_current_config.day_end) && m_relay->enabled())
    {
        spdlog::info("Time to disable");
        m_relay->disable();
    }
}
```

3.2.9 Сервіс управління вентиляцією

Відповідно до принципу розробки програмного забезпечення DRY, у системі, що уникає дублювання вихідного коду, модифікація будь-якого окремого елемента системи не вимагає зміни інших логічно не пов'язаних елементів. Крім того, елементи, які логічно пов’язані, змінюються передбачувано й рівномірно, і, таким чином, оновлюються синхронізовано.

Користувацькі сценарії управління вентиляцією, згідно з вимогами до системи, є ідентичними до управління фітоосвітленням. Оскільки управління живленням як фітоосвітлення, так і вентиляції, виконується за допомогою реле, архітектурно правильним рішенням є об’єднання розглянутої реалізації функціональності у окрему сутність, а відмінність параметризувати. Для реалізації сервісу управління вентиляцією достатньо надати імплементованому контролеру окремий канал комунікації з користувачем та реле. Створення описаного об’єкту контролеру у сервісі управління вентиляцією наведено у лістингу 3.8.

Лістинг 3.8 – Верхньорівнева реалізація сервісу управління вентиляцією

```
      constexpr std::uint8_t VENTILATION_PIN_NUMBER {7u};
                  constexpr   open_greenery::gpio::PinId   VENTILATION_PIN_ID
{open_greenery::gpio::PinNumber(VENTILATION_PIN_NUMBER),
                                                                        open_greene
ry::gpio::Pinout::WIRING_PI};
                                          auto        ventilation_pin        =
open_greenery::gpio::GPIOFactory::getInstance().getOutputGPIOctl(VENTILATION_
PIN_ID);
                                        auto        ventilation_relay        =
std::make_shared<open_greenery::relay::Relay>(
                  ventilation_pin,
                  open_greenery::gpio::LogicLevel::LOW);

          // time provider
                                           auto         time_provider         =
std::make_shared<open_greenery::relay::CurrentTimeProvider>();

          // rpc client
          constexpr char RPC_HOST [] {"localhost:8091"};
                                             auto         rpc_server         =
std::make_shared<open_greenery::rpc::relay::Server>(RPC_HOST);

          // ventilation controller
          s_controller.emplace(
                  ventilation_relay,
                  rpc_server,
                  time_provider,
                  rpc_server,
                  rpc_server,
                  rpc_server,
                  rpc_server
              );
          s_controller_finish = s_controller->start();
          if (s_controller_finish->valid())
          {
                    s_controller_finish->wait();    // Execute service until
termination signal
          }
```

3.2.10 Збереження налаштувань

Усі оновлення даних та події надаються контролерам за допомогою виділених сутностей, що інкапсулюють міжсервісну комунікацію. Завдяки цьому, для реалізації збереження останнього стану системи, а також подальшого надання цих даних, достатньо розробити клас, що буде мати ідентичний інтерфейс, але між отриманням оновлення та наданням його контролеру, буде виконувати збереження даних у БД. Діаграма реалізованої сутності-постачальника, що впроваджує у систему функціональність

збереження налаштувань у БД SQLite , зображена на рисунку 3.52.

Рисунок 3.52 – Впровадження функціональності збереження налаштувань системи

3.2.11 Фіктивні сутності залежностей

Фіктивні сутності реалізують той же абстрактний інтерфейс, що і оригінальна сутність, яку треба замінити для виконання автоматизованого тестування. Сутності-постачальники налаштовуються відповідно до тестової вибірку та конкретного сценарію. Натомість сутності-споживачі, тобто ті, на які впливає тестований об’єкт, налаштовуються на збереження звертань, аргументів, та їх послідовності. Приклад реалізації фіктивної сутності наведено у лістингу 3.9.

Лістинг 3.9 – Фіктивна сутність, що реалізує абстрактний інтерфейс GPIO

```
class OutputPinMock : public ogio::IOutputPin
{
public:

    void write(const ogio::LogicLevel _value) const override
    {
        m_write_calls.push_back(_value);
    }

    const std::vector<ogio::LogicLevel> & writeCalls() const
    {
        return m_write_calls;
    }

private:
    mutable std::vector<ogio::LogicLevel> m_write_calls;
};
```

3.2.12 Реалізація автоматичних тестів

Мета введення практики автоматизованого тестування полягає в тому,

щоб розробляти тести для кожної сутності та їх взаємодій. Саме імплементація автоматичних, легких для виконання тестів, дозволяє досить швидко перевірити, чи не призвела чергова зміна коду до регресії, тобто до появи помилок в уже перевірених раніше місцях програми, а також полегшує виявлення і усунення таких помилок.

Автоматичні тести дають високий рівень впевненості, що система працює як згідно з поставленими вимогами. Такі тести можна виконувати багато разів. Успішне виконання тестів покаже розробнику, що його зміни не призвели до появи несправностей. Невдале завершення тесту дозволяє виявити, що виконані зміни вихідного коду призвели до неочікуваних змін поведінки сутності. Дослідження помилки і порівняння очікуваного результату з отриманим, надасть можливість зрозуміти, де саме виникла помилка, не суттєво, у реалізації модуля, тесту чи навіть у описаних вимогах. У лістингу 3.10 наведена реалізація тестів класу реле, що використовує фіктивну сутність GPIO виходу.

Лістинг 3.10 – Реалізація тесту з використанням фіктивної сутності

```
TEST_F(RelayTest, DisabledByDefault)
{
    EXPECT_FALSE(m_relay.enabled());
    ASSERT_EQ(m_output_pin_mock->writeCalls().size(), 1u);
                        EXPECT_EQ(m_output_pin_mock->writeCalls().back(),
ogio::LogicLevel::LOW);
}

TEST_F(RelayTest, Enable)
{
    m_relay.enable();

    EXPECT_TRUE(m_relay.enabled());
     ASSERT_EQ(m_output_pin_mock->writeCalls().size(), 2u);  // Disable
on start, enable after call
                        EXPECT_EQ(m_output_pin_mock->writeCalls().back(),
ogio::LogicLevel::HIGH);
}

TEST_F(RelayTest, Disable)
{
    m_relay.disable();

    EXPECT_FALSE(m_relay.enabled());
     ASSERT_EQ(m_output_pin_mock->writeCalls().size(), 2u);  // Disable
on start, disable after call
                        EXPECT_EQ(m_output_pin_mock->writeCalls().back(),
ogio::LogicLevel::LOW);
```

```
}

TEST_F(RelayTest, Toggle)
{
    const auto first {m_relay.enabled()};
    m_relay.toggle();
    const auto second {m_relay.enabled()};

    EXPECT_NE(first, second);
}
```

3.2.14 Графічний інтерфейс мобільного застосунку

Перед початком розробки мобільного застосунку, необхідно визначити вимоги до графічного інтерфейсу, що буде передбачати отримання користувачем можливості взаємодії з усією функціональністю системи. Згідно з поставленими вимогами до системи управління вирощуванням рослин та користувацькими сценаріями, графічний інтерфейс має передбачати:

сторінки налаштувань сервісів фітоосвітлення, вентиляції, іригації;

ручне управління станом фітоосвітлення;

можливість вибору режиму роботи сервісу управління фітоосвітленням: ручний чи автоматичний;

налаштування часу включення та виключення фітолампи у автоматичному режимі;

невідкладне управління станом вентиляції;

можливість вибору режиму роботи сервісу управління вентиляції: ручний чи автоматичний;

налаштування часу включення та виключення канального вентилятору у автоматичному режимі;

вибір зони іригації для налаштування;

вибір способу іригації для конкретної зони: періодичний чи за вологістю ґрунту;

налаштування об’єму води одного зволоження та період (окремі для двох способів іригації).

Враховуючи всі поставлені вимоги до графічного інтерфейсу, було розроблено мобільний застосунок мовою Kotlin для ОС Android. Зовнішній вигляд сторінки налаштувань сервісу управління фітоосвітленням у режимі ручного управління та перемикач стану фітолампи наведено на рисунку 3.53. На рисунку 3.54 зображена сторінка налаштувань сервісу управління вентиляцією у режимі автоматичного управління, що має перемикач стану, якщо це терміново знадобиться користувачу, та поля введення часу автоматичного включення і виключення канального вентилятору. Для зручного вибору часу використовується графічний елемент, зображений на рисунку 3.55. Інтерфейс налаштування сервісу іригації наведено на рисунку 3.56.

Рисунок 3.53 – Інтерфейс налаштувань сервісу фітоосвітлення у режимі ручного управління

Рисунок 3.54 –Інтерфейс налаштувань сервісу фітоосвітлення у режимі автоматичного управління

Рисунок 3.55 – Інтерфейс вибору часу

Рисунок 3.56 – Інтерфейс налаштувань сервісу іригації

3.2.15 Комунікація з пристроєм управління вирощуванням рослин

Взаємодія мобільного застосунку із сервісами, що виконуються на пристрої управління вирощуванням рослин, реалізована за допомогою віддаленого виклику процедур. Кросмовні можливості фреймворку gRPC дозволяють згенерувати заготовку реалізації клієнту для мови Kotlin із того ж файлу опису протоколу, що і використовується сервісом, який реалізований мовою C++. За принципом єдиного джерела (single source of truth, SSOT), це збільшує надійність системи, оскільки гарантується ідентичність способів комунікації клієнту та серверу. Реалізація клієнту віддаленого виклику процедур мовою програмування Kotlin наведена у лістингу 3.11.

Лістинг 3.11 – Клієнт віддаленого виклику процедур мобільного застосунку

```
        val service_status: ServiceStatus
            get() = getServiceStatus()

        fun getServiceStatus(): ServiceStatus {
            val service_status: RelayProto.ServiceStatus
                = stub.getServiceStatus(Empty.getDefaultInstance())
            val mode: Mode =
                                    if   (service_status.modeSettings.mode   ==
RelayProto.ModeSetting.Mode.MODE_AUTO)
                    Mode.AUTO
                else
                    Mode.MANUAL
            val relayEnabled = service_status.relayStatus.isEnabled

            // Milliseconds from day start to seconds
            val dayStart = LocalTime.ofSecondOfDay(
                service_status.config.dayStart
                    .div(1000).toLong())
            val dayEnd = LocalTime.ofSecondOfDay(
                service_status.config.dayEnd
                    .div(1000).toLong())
            return ServiceStatus(
                mode,
                relayEnabled,
                Config(dayStart, dayEnd)
            )
        }

        var config: Config
            get() = service_status.config
            set(value) = updateConfig(value)

        fun updateConfig(config: Config) {
            val proto_config = RelayProto.Config.newBuilder()
                .setDayStart(config.dayStart.toSecondOfDay() * 1000)
```

```
                .setDayEnd(config.dayEnd.toSecondOfDay() * 1000)
                .build()
            stub.setConfig(proto_config)
        }

        fun updateDayStart(dayStart: LocalTime) {
            config = Config(dayStart, config.dayEnd)
        }

        fun updateDayEnd(dayEnd: LocalTime) {
            config = Config(config.dayStart, dayEnd)
        }

        var mode: Mode
            get() = service_status.mode
            set(m) {
                val protoMode =
                    if (m == Mode.MANUAL)
                        RelayProto.ModeSetting.Mode.MODE_MANUAL
                    else
                        RelayProto.ModeSetting.Mode.MODE_AUTO
                stub.setMode(RelayProto.ModeSetting.newBuilder().setMode(pro
toMode).build())
            }
```

3.3 Налаштування неперервної інтеграції

Підвищити загальну надійність системи можна, гарантуючи, що кожне оновлення вихідного коду успішно компілюється та успішно проходить виконання автоматичних тестів. Безпосередньо у віддаленому репозиторії проекту налаштовано сервіс безперервної інтеграції засобами GitHub Actions , що виконує збірку та тестування проекту при кожному оновленні або створенні запиту на забирання змін (pull request).

3.3.1 Програмне забезпечення пристрою

Через необхідність збірки залежностей програмного забезпечення, розробленого мовою C++, із вихідного коду бібліотеки, повна збірка розробленої системи у чистому оточенні може займати багато часу. Для прискорення збірки, сторонні залежності проекту та середовище для подальшої збірки підготовлено у вигляді образу Docker контейнеру.

Docker – це інструмент управління контейнерами на базі ОС Linux (рідше Windows). Контейнери є ізольованими оточеннями виконання

процесів. Образи контейнерів можуть пошарово зберігати встановлені налаштування або зміни оточення. Натомість зміни оточення, набуті в процесі роботи контейнеру, анулюються після його завершення. При налаштуванні сервісу неперервної інтеграції, описана особливість дозволяє гарантувати, що кожна перевірка розроблюваної системи буде виконуватись у абсолютно ідентичному оточенні, не зміненому попередніми використанням. Опис образів Docker-контейнерів для підготовки залежностей, збірки та тестування проекту наведено у лістингах 3.12, 3.13 та 3.14 відповідно. Результат виконання автоматичної перевірки оновлення програмного забезпечення пристрою управління вирощуванням рослин наведено на рисунку 3.57.

Лістинг 3.12 – Опис контейнеру підготовки залежностей проекту

```
FROM ubuntu:20.04
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && \
    apt-get install -y \
    # OpenGreenery
    git \
    cmake \
    build-essential \
    qt5-default \
    libqt5charts5-dev \
    # gRPC
    build-essential autoconf libtool pkg-config
# Install gRPC
WORKDIR /tmp
RUN git clone --recurse-submodules -b v1.41.0
https://github.com/grpc/grpc
RUN cd grpc && mkdir -p cmake/build && cd cmake/build && \
    cmake -DgRPC_INSTALL=ON -DgRPC_BUILD_TESTS=OFF ../.. && \
    cmake --build . && make install
```

Лістинг 3.13 – Опис контейнеру збірки проекту

```
FROM alexdegtyar/og-dependencies:x86-v1
ENV BRANCH="develop"
ENV ARTIFACTS=""
ENV ARTIFACTS_FOLDER="/var/og-artifacts"
ENV PROJECT_ROOT="OpenGreenery/src/device"
ENV BUILD_DIRECTORY="${PROJECT_ROOT}/build"
CMD git clone --recursive -b ${BRANCH} --single-branch \
    https://github.com/OpenGreenery/OpenGreenery.git && \
    mkdir -p ${BUILD_DIRECTORY} && \
    cmake -S ${PROJECT_ROOT} -B ${BUILD_DIRECTORY} && \
    cmake --build ${BUILD_DIRECTORY} && \
    for f in ${ARTIFACTS}; do \
        echo Copy artifact ${f}; \
        cp ${BUILD_DIRECTORY}/${f} ${ARTIFACTS_FOLDER}; \
    done
```

Лістинг 3.14 – Опис контейнеру тестування проекту

```
FROM alexdegtyar/og-dependencies:x86-v1

ENV ARTIFACTS_FOLDER="/var/og-artifacts"

CMD ${ARTIFACTS_FOLDER}/tests
```

Рисунок 3.57 – Результат виконання автоматичної перевірки оновлення програмного забезпечення пристрою

3.3.2 Мобільний застосунок

Завдяки кросплатформеним особливостям мови програмування Java, встановлення залежностей для збірки проектів, розроблених мовою Kotlin для Android, є доволі простим. Реалізація збірки мобільного застосунку безпосередньо у оточенні ОС Ubuntu (дистрибутив Linux) за допомогою GitHub Actions наведена у лістингу 3.15. Результат виконання автоматичної перевірки оновлення мобільного застосунку наведено на рисунку 3.58.

Лістинг 3.15 – Перевірка оновлення мобільного застосунку засобами GitHub Actions

```
name: Android Application
on:
  push:
    paths:
      - 'src/android/**'

  pull_request:
    paths:
      - 'src/android/**'

  workflow_dispatch:

defaults:
  run:
    working-directory: src/android

jobs:
  build:
```

```
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v2

- name: Set up JDK 11
  uses: actions/setup-java@v2
  with:
    java-version: '11'
    distribution: 'adopt'
    cache: gradle

- name: Grant execute permission for gradlew
  run: chmod +x gradlew

- name: Build with Gradle
  run: ./gradlew build
```

Рисунок 3.58 – Результат виконання автоматичної перевірки оновлення мобільного застосунку

В результаті, головна сторінка репозиторію проекту, що зображена на рисунку 3.59, інформує про поточний стан розробленого програмного забезпечення системи: сервісів пристрою та мобільного застосунку.

Рисунок 3.59 – Стабільний стан програмного забезпечення на головній сторінці репозиторію проекту

4 ЕКСПЕРИМЕНТАЛЬНЕ СЕРЕДОВИЩЕ ТА ЗАСТОСУВАННЯ СИСТЕМИ

Обладнання експериментального середовища необхідне для перевірки, що розроблена система управління вирощуванням рослин забезпечує оптимальні умови для розвитку самих рослин. Основою для створення середовища вирощування рослин може стати простора мебель (рис. 4.1), або спеціалізований тент.

Підвищити ефективність освітлення можна вкривши стінки середовища світловідбивним матеріалом. Наприклад, використовуючи теплоізоляційний матеріал із фольгованим покриттям, як показано на рисунку 4.2.

Рисунок 4.1 – Середовище вирощування рослин, обладнане у шафі

Рисунок 4.2 – Теплоізоляційний матеріал із фольгованим покриттям

При нестачі об’єму шафи через кількість чи висоту вирощуваних рослин, конструкцію середовища можна створити самостійно із доступних будівельних матеріалів. Натомість, функціональним і, можливо, більш дешевим рішенням може стати спеціалізований тканинний тент, продемонстрований на рисунку 4.3, із вентиляційними отворами, вкритий міцним світловідбивним покриттям. Саме такому тенту надано перевагу.

Рисунок 4.3 – Тканинний тент для вирощування рослин

В нижній частині середовища висаджено рослини. Молоді паростки помідорів зображені на рисунку 4.4. А невеличкі грядки для пророщення м’яти та меліси на рисунку 4.5.

Рисунок 4.4 –Паростки томатів

Рисунок 4.5 – Ґрунт із висадженими м’ятою і мелісою

У верхній частині середовища закріплено пристрій управління вирощуванням рослин, що складається із блоку управління та, нижче, блоку іригації. На відстані близько 30 см над рослинами встановлено фітолампу. Хоча температура поверхні світлодіодної панелі відносно невисока, контакт листя із радіатором може призвести до незначних, але небажаних, опіків. У вентиляційному отворі у верхній частині середовища закріплено канальний вентилятор.

Кабель живлення, фітолампа, канальний вентилятор та датчики підключаються до пристрою за допомогою роз’ємів. До блоку іригації приєднано рідинні трубки. Обладнання підвішено до каркасу тенту за допомогою ременів, довжина яких регулюється. Вільні вентиляційні отвори використовуються для прокладання кабелів та іригаційних трубок. Встановлене та готове до використання обладнання експериментального середовище зображено на рисунку 4.6.

Рисунок 4.6 – Готове до використання обладнання експериментального середовища

Коли об’єму ґрунту недостатньо для занурення датчику, іригація виконується у періодичному режимі. На рисунку 4.7 зображено кущі чайних трав і іригаційну трубку. Налаштування періодичної іригації даної зони за допомогою мобільного застосунку зображено на рисунку 4.8.

Рисунок 4.7 – М’ята, меліса, та іригаційна трубка для їх зволоження

Рисунок 4.8 – Налаштування періодичної іригації м’яти

Більш дорослі рослини томатів пересаджено у горщики, глибиною достатньою для занурення датчиків. Модифіковані для довшого терміну служби ємнісні датчики вологості ґрунту, занурені у горщики помідорів, зображені на рисунку 4.9. Така зона налаштовується на іригації у режимі за вологістю ґрунту. Полив виконується тільки коли земля достатньо просохне, та поступово зволожується невеличкими порціями води.

Рисунок 4.9 – Модифіковані датчики, занурені у ґрунт для вимірювання вологості

В ході експериментального використання розробленої системи, успішно вирощувались горох (рис. 4.10), базилік (рис. 4.11), томати (4.12), та

авокадо (рис. 4.13). Зовнішній вигляд середовища впродовж експерименту продемонстровано на рисунку 4.15. Врожай помідорів, отриманий в ході розробки системи, зображений на рисунку 4.16.

Рисунок 4.10 – Плоди гороху

Рисунок 4.11 – Базилік

Рисунок 4.12 – Молоді томати

Рисунок 4.13 – Дерево авокадо

Рисунок 4.15 – Рослини в ході експериментального використання системи

Рисунок 4.16 – Збір врожаю помідорів

ВИСНОВКИ

Результатом виконання даного проекту є розроблена розподілена система управління вирощуванням рослин, що складається з пристрою управління вирощуванням та мобільного застосунку.

Розділення пристрою на блок управління та блок іригації ізолює сантехнічне обладнання від засобів живлення та управління. Наявність на корпусі пристрою роз’ємів спрощує підключення зовнішнього обладнання користувачем. Коло перетворювань напруги, розташоване у блоці управління, дозволяє живити весь пристрій та зовнішнє обладнання за допомогою підключення лише одного кабелю у мережу. Демультиплексування потоку води, що реалізується блоком іригації, дозволяє зменшити вартість необхідного сантехнічного обладнання.

Розподіленість системи і реалізація міжсервісних комунікацій шляхом віддаленого виклику процедур надає можливості для легкого масштабування та зменшення вартості апаратного забезпечення пристрою, що продемонстровано перенесенням графічного інтерфейсу користувача на смартфон.

Обраний спосіб зменшення зв’язності сутностей вихідного коду, запровадження практики автоматизованого тестування програмного забезпечення та неперервної інтеграції збільшує надійність системи.

Модифіковано датчики вологості ґрунту для подовження терміну служби, спрощення та визначення підключення до пристрою.

Можливими шляхами розвитку проекту є:

розробка печатної плати розширення;

веб застосунок;

виділення аналізу та іригації ґрунту у окремий пристрій;

виявлення несправностей сантехнічного обладнання;

користувацькі повідомлення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. How to Get Rid of White Mold on Plants [Електронний ресурс] – Режим доступу до ресурсу: https://leafyplace.com/white-mold-on-plants/

2. Збір пожертвувань для проекту PiGrow [Електронний ресурс] – Режим доступу до ресурсу: https://www.patreon.com/Pigrow

3. wxPython API Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.wxpython.org/

4. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/about/

5. Click&Grow [Електронний ресурс] – Режим доступу до ресурсу: https://clickgrow.com.ua/

6. Raspberry Pi [Електронний ресурс] – Режим доступу до ресурсу: https://www.raspberrypi.org/

7. Raspberry Pi 3B+ Pi [Електронний ресурс] – Режим доступу до ресурсу: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

8. ADS1115 datasheet [Електронний ресурс] – Режим доступу до ресурсу: http://www.bristolwatch.com/rpi/geany/ads1115.pdf

9. From spuds to space medicine [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20130227115749/http://www.msfc.nasa.gov/news/news/photos/2000/photos00-336.htm

10. Samsung LM301B datasheet [Електронний ресурс] – Режим доступу до ресурсу: https://www.samsung.com/led/lighting/mid-power-leds/3030-leds/lm301b/

11. Схемы вентиляции [Електронний ресурс] – Режим доступу до ресурсу: https://olvent.ua/stati/stati-o-ventiltjatsii/ventiljatsija-grouboksa

12. How do peristaltic pumps work? [Електронний ресурс] – Режим доступу до ресурсу: https://www.watson-marlow.com/gb-en/support/how-do-

peristaltic-pumps-work-industrial/

13. Принцип действия электромагнитного клапана [Електронний ресурс] – Режим доступу до ресурсу: https://www.italgaz.com.ua/wiki/solenoid-valve.html

14. Дегтяр О. М. Автоматизований іригаційний контролер : 123 / Дегтяр Олександр Миколайович – Харків, 2020. – 72 с.

15. What is a Relay and How it Works? [Електронний ресурс] – Режим доступу до ресурсу: https://www.codrey.com/electrical/what-is-a-relay-and-how-it-works/

16. Raspberry Pi Pinout [Електронний ресурс] – Режим доступу до ресурсу: https://pinout.xyz/

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

18. Stroustrup B. Programming: Principles and Practice Using C++ / Bjarne Stroustrup., 2016. – 1312 с. – (Addison-Wesley Professional).

19. C++17 standard [Електронний ресурс] – Режим доступу до ресурсу: https://isocpp.org/std/the-standard

20. CMake documentation [Електронний ресурс] – Режим доступу до ресурсу: https://cmake.org/documentation/

21. Newman S. Building Microservices: Designing Fine-Grained Systems / Sam Newman., 2015. – 280 с. – (O'Reilly Media).

22. Richardson C. Microservices Patterns: With examples in Java / Chris Richardson., 2018. – (Manning Publications).

23. gRPC framework [Електронний ресурс] – Режим доступу до ресурсу: https://grpc.io/

24. About SQLite [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlite.org/about.html

25. Kreibich J. A. Using SQLite: Small. Fast. Reliable. Choose Any Three / Jay A. Kreibich., 2010. – 528 с. – (O'Reilly Media).

26. Langr J. Modern C++ Programming with Test-Driven Development:

Code Better, Sleep Better / Jeff Langr., 2013. – 589 с.

27. Googletest Primer [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/google/googletest/blob/master/googletest/docs/primer.md.

28. Android [Електронний ресурс] – Режим доступу до ресурсу: https://www.android.com/

29. Google IO 2017 [Електронний ресурс] – Режим доступу до ресурсу: https://android-developers.googleblog.com/2017/05/google-io-2017-empowering-developers-to.html

30. Gradle Build Tool [Електронний ресурс] – Режим доступу до ресурсу: https://gradle.org/

31. Horowitz P. The Art of Electronics / P. Horowitz, W. Hill., 1980. – 1220 с.

32. GitHub Actions Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.github.com/en/actions

33. Docker Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/

34.