Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Навчально - діючий стенд "Екокліматична станція з вирощування біорізноманіття" INDOOR FOREST»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

_____ /Марина СТЕШЕНКО/
(підпис студента) (Ім`я ПРІЗВИЩЕ)

Керівник: _____ /доц. Олександр ІВАНОВ/
(підпис) (посада, Ім`я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім`я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Training and operating stand "Ecoclimatic station for growing biodiversity" INDOOR FOREST»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group П31811: /Maryna STESHENKO/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Стешенко Марині Артурівні
1. Тема роботи: «Навчально - діючий стенд "Екокліматична станція з вирощування біорізноманіття" INDOOR FOREST»
Керівник роботи: Іванов Олександр Петрович, доцент
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: 07.06.202_ р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Вступ
Збір вимог до програмного забезпечення
Зовнішнє і внутрішнє проектування
Розробка застосунку
Тестування та налагодження
Загальні висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Презентація
Відео роботи програми

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі, збір інформації, вибір та обґрунтування критеріїв розробки	31.01.22-18.02.22	
2	Попередній вибір методів вирішення задач		
3	Визначення вимог до технічних засобів		
4	Узгодження і затвердження технічного завдання		
5	Програмування та відлагодження програми	19.02.22-20.05.22	
6	Тестування програми	20.05.22-27.05.22	
7	Розробка, узгодження і затвердження програмної документації	27.05.22-12.06.22	
8	Подання кваліфікаційної роботи до кафедри	07.06.22	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.22	

Студент _____ Марина СТЕШЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Олександр ІВАНОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 8 розділів:

- вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 2 сторінок;
- збір та аналіз вимог до програмного забезпечення – у цьому розділі розглянуто аналоги, проаналізовано їх переваги та недоліки, представлено результати опитувань зацікавлених сторін. Складається з 4 сторінок;
- зовнішнє і внутрішнє проектування – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі, розробка фізичного проекту, приводиться опис об’єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, аналіз проекту, проектування динаміки системи, вибір мови програмування. Складається з 27 сторінок;
- розробка застосунку – у цьому розділі описано використання для розробки засоби, представлено структуру вихідного коду, скріншоти розроблених екранних форм та використані алгоритми. Складається з 11 сторінок;
- тестування та налагодження – включає в себе вибір стратегії тестування, опис тестів методами «чорного» та «білого» ящика. Також аналіз помилок їх вплив на систему та вирішення проблеми. Складається з 12 сторінок;
- загальні висновки – підсумки всієї роботи. Складається з 1 сторінки;
- список використаних джерел – включає в себе бібліографічний список використаної літератури. Складає 2 сторінки;
- додатки – містить технічне завдання і текст програми.

Кількість таблиць: 15 штук. Кількість рисунків: 29 штук. Кількість слів: 12813.

ЗМІСТ

Вступ 7
Розділ 1. Збір вимог до програмного забезпечення 9
1.1 Огляд аналогів 9
1.2 Опитування зацікавлених сторін 11
1.3 Постановка задачі 13
Висновки до розділу 1 13
Розділ 2. Зовнішнє і внутрішнє проектування 15
2.1 Функціональні вимоги 15
2.2 Вибір мови програмування 17
2.3 Опис об’єктно-орієнтованого програмування 21
2.4 Проектування архітектури системи 23
2.5 Проектування інтерфейсу користувача 28
2.6 Проектування динаміки системи 33
2.7 Проектування системи на фізичному рівні 39
Висновки до розділу 2 40
Розділ 3. Розробка застосунку 42
3.1 Опис використаних для розробки засобів 42
3.2 Структура вихідного коду 43
3.3 Розроблені графічні компоненти 46
3.4 Опис використаних алгоритмів 50
Висновки до розділу 3 51
Розділ 4. Тестування та налагодження 53
4.1 Огляд стратегій тестування 53
4.2 Опис розроблених тестів 54
4.3 Результат виконання тестування та аналіз виявлених помилок 65
Висновки до розділу 4 66
Загальні висновки 66
Список літератури 67
Додатки **70**

ВСТУП

Сучасний світ важко уявити без інновацій та технологій, яким знайшли застосування у багатьох сферах людської діяльності. Задля покращення рівня життя та забезпечення конкурентоспроможності нашої держави необхідно враховувати ці зміни та відповідно адаптуватися до них. Саме тому важливими факторами розвитку економіки є наукоємні та високотехнологічні галузі. На жаль, зараз спостерігається втрата популярності предметів природничої, технологічної та математичної освітніх галузей. Перед сферою освіти зараз постає завдання розвитку цих галузей та заохочення учнів та студентів до їх вивчення. Одним з пріоритетів розвитку сфери освіти зараз є природничо-математична освіта STEM.

Природничо-математична освіта (STEM-освіта) – цілісна система природничої і математичної освітніх галузей, метою якої є розвиток особистості через формування компетентностей, природничо-наукової картини світу, світоглядних позицій і життєвих цінностей з використанням трансдисциплінарного підходу до навчання, що базується на практичному застосуванні наукових, математичних, технічних та інженерних знань для розв’язання практичних проблем для подальшого використання цих знань і вмінь у професійній діяльності [2].

STEM-лабораторія – навчальний кабінет або приміщення закладу освіти, оснащене сучасними засобами навчання та обладнанням для залучення здобувачів освіти до навчально-дослідницької, дослідницько-експериментальної, конструкторської, винахідницької та пошукової діяльності відповідно до стандартів освіти, освітніх та навчальних програм з використанням проектних технологій в освітньому процесі [2].

STEM-центр – структурний підрозділ закладу освіти, утворений з метою забезпечення природничо-математичної освіти (STEM-освіти), організації та взаємодії заінтересованих осіб [2].

Для забезпечення науково-методичної підтримки природничо-математичної освіти (STEM-освіти) важливе значення має розроблення інтегрованих навчальних програм для всіх типів закладів освіти для викладання спеціальних курсів, факультативів, організації роботи гуртків з робототехніки, інженерії, природничих та

аграрних дисциплін, сучасних наукових напрямів, новітніх технологій з урахуванням кращого національного та міжнародного досвіду [2].

Метою проекту є створення навчально-діючого STEM-стенду для використання на уроках фізики, біології, хімії, екології та залучення здобувачів освіти до дослідницько-експериментальної, конструкторської діяльності.

Експлуатаційне призначення – стенд та додаток до нього виконують навчально-демонстраційну роль на дисциплінах фізики, хімії, біології та екології. Вони використовуються для вивчення рослин, їхнього життєвого циклу, будови, особливостей умов їх зберігання, застосування сучасних технологій для догляду за ними, принципу роботи датчиків, клапанів, сонячних батарей.

РОЗДІЛ 1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд аналогів

eWeLink (рис. 1.1) – це безкоштовний додаток для розумного дому, який може керувати сотнею розумних пристроїв 80 брендів. eWeLink підтримує декілька мов. Наразі він сумісний з більшістю популярних пристроїв домашньої автоматизації, таких як Google Nest, Amazon Echo. eWeLink може підключатися до широкого спектру розумних пристроїв, включаючи перемикачі, розетки, світлодіоди, датчики тощо [3].

Переваги:

- сумісний з багатьма різними пристроями;
- надає можливість створювати власні сценарії керування;
- існують версії різними мовами.

Недоліки:

- оповіщення можуть приходити із затримкою;
- деякі функції додатку є платними;
- працює лише за наявного підключення до мережі.

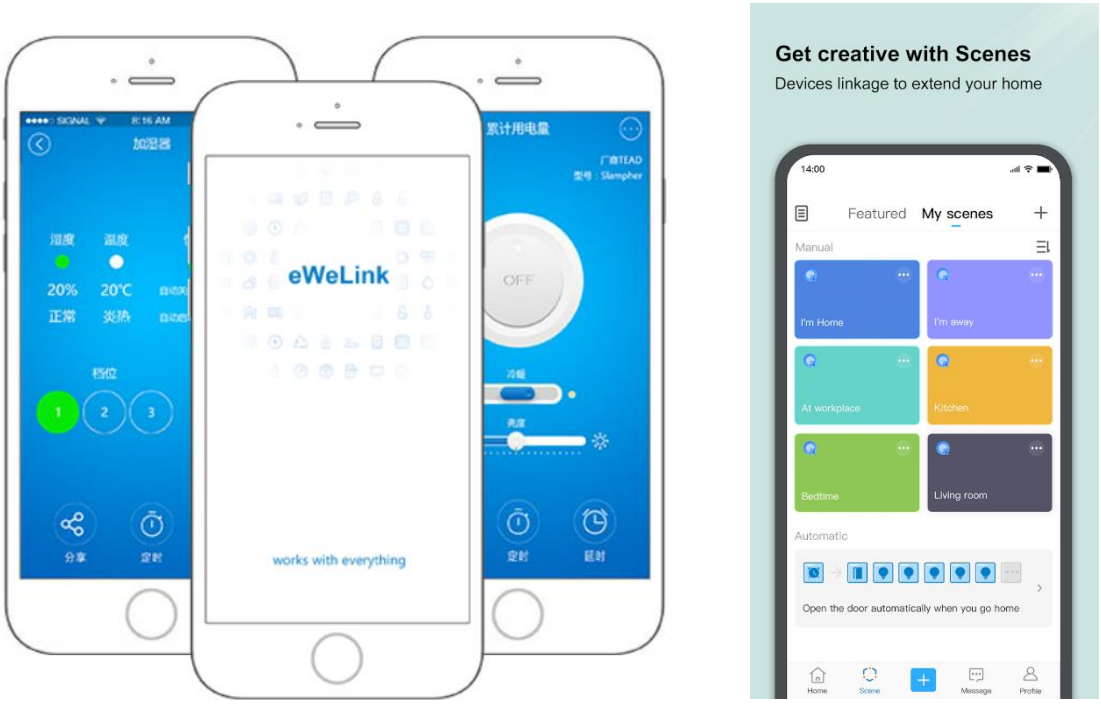


Рисунок 1.1 – GUI застосунку eWeLink

AquaPot E-mode 4 (рис. 1.2) – це гідропонна система, яка поєднує в собі принципи дії систем типу DWC/RDWC та системи крапельного поливу. APE 4 призначена для вирощування рослин у домашніх умовах, у теплицях, оранжереях та розрахована на 4 посадочні місця.

Рисунок 1.2 – Вигляд системи AquaPot E-mode 4

Для автоматизації процесів приготування та підтримання живильного розчину розчинний бак оснащений слотами для встановлення модулів E-mode:

- регулятор pH — модуль автоматичного керування діапазоном pH;
- регулятор ЄС – модуль автоматичного контролю концентрації добрив;
- регулятор рівня (LVL) – модуль контролю рівня рідини.

Встановлені модулі контролю з'єднуються із датчиками. Датчик pH та датчик ЄС встановлюються у пробовідбірник. Датчик рівня рідини встановлюється на місце

згідно з інструкцією. Пробовідбірник, інтегрований у кришку бака, разом із приладами, забезпечує безперервну циркуляцію розчину. Це забезпечує хімічну рівномірність середовища та своєчасне перемішування при додаванні поживних компонентів або регулюванні рівня pH [4].

Переваги:

- повністю автоматизована система, не залежить від наявності мережі;
- компактний розмір;
- надає можливість керувати рівень кислотності, концентрацію добрив та рівень рідини.

Недоліки:

- складне керування;
- відсутність мобільного застосунку.

1.2 Опитування зацікавлених сторін

Вузька направленість навчально-демонстраційного стенду не дозволяє використовувати існуючі аналоги з ним безпосередньо, але вони можуть слугувати прикладами проектування та розробки власного програмного забезпечення. Зокрема, серед перелічених застосунків у п. 1.1 можна виділити наступні переваги та недоліки:

- ПЗ має надавати користувачу можливість налаштування застосунку (встановлення часу оновлення, порогових значень показників);
- інтерфейс ПЗ має бути простим, не перевантаженим зайвою інформацією;
- у графічному інтерфейсі перевагу надавати синім та зеленим кольорам;
- сумісність застосунку з багатьма різними пристроями може погано вплинути на його продуктивність.

Іншим важливим аспектом проекту, що розробляється, є можливість його застосування у освітньому процесі, зокрема у демонстраційних цілях. Для визначення основних потреб освітян, їх зацікавленості у розроблюваному стенді та можливих слабких місць у розробці, проект було представлено на різних конференціях та

презентаціях, а відгуки слухачів та відвідувачів записано у фолоу-ап документах [5]. Загалом, інформацію про ці опитування можна представити у таблиці 1.1.

Таблиця 1.1 – Загальна інформація про опитування

Кількість проведених презентацій проекту	4
Кількість онлайн-конференцій, у яких було взято участь	3
Загальна кількість відвідувачів усіх презентацій та конференцій	~210
Цільова аудиторія	Викладачі шкіл та університетів, учні шкіл
З усіх присутніх виявило зацікавленість до проекту	87%

З зацікавленими у проекті буди проведені додаткові опитування. Їх результати наведено у таблиці 1.2.

Таблиця 1.2 – Результати опитування зацікавлених

Вважають, що STEM-освіта є перспективною та її необхідно запроваджувати	76%
Вважають, що розроблюваний стенд покращить якість навчання та приверне увагу учнів і студентів	84%
Позитивно ставляться до використання гаджетів під час навчання	73%
Використовують демонстраційні матеріали та сучасне обладнання під час проведення занять	71%

Серед побажань стосовно програмного забезпечення до стенду були висловлені наступні:

- програмне забезпечення має бути мобільним додатком для операційних систем Android та iOS;
- програмне забезпечення має надавати можливість перегляду поточних даних з датчиків;

- програмне забезпечення має надавати можливість виконувати налаштування датчиків, зокрема задавати порогові значення;
- програмне забезпечення має працювати незалежно від підключення до мережі.

Велика кількість опитаних виразила бажання мати у додатку доступ до лекційного та практичного матеріалу, а також можливість проводити дистанційне тестування. Ці вимоги були враховані, але для їх виконання необхідно більше часу та залучення спеціалістів з інших галузей, зокрема викладачів та методистів, тому вони не були включені в вимоги до поточної версії розроблюваного програмного забезпечення.

1.3 Постановка задачі

Програмний продукт, що розробляється, є мобільним додатком до стенду з системою поливу та освітлення для 10 рослин. Він має отримувати дані з датчиків та керувати елементами системи поливу та освітлення згідно отриманих даних. Також має надавати користувачу можливість переглядати поточні дані та встановлювати порогові значення.

Додаток має мати змогу працювати у двох режимах: автоматичному та ручному. У автоматичному режимі додаток самостійно отримує значення з датчиків із заданою періодичністю, у ручному – оновленням даних керує користувач.

Висновки до розділу 1

Після збору та аналізу вимог, зібрана інформація була проаналізована і на її основі було розроблено бізнес-процес системи. Він має наступний вигляд:

1. Користувач встановлює порогові значення вологи ґрунту та рівня освітлення.
2. Стенд у режимі реального часу отримує дані з датчиків вологості та освітлення.
3. Стенд приймає рішення щодо необхідності ввімкнення поливу та зміни рівня освітлення.
    a. Якщо фактична вологість ґрунту менша за необхідну вмикається полив.

   b. Якщо фактичний рівень освітленості менший за необхідний збільшується освітлення.
   c. Якщо фактичний рівень освітленості більший за необхідний – зменшується освітлення.
4. Користувач може отримати інформацію про дні з датчиків.

РОЗДІЛ 2. ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

2.1 Функціональні вимоги

Програмний продукт повинен забезпечувати можливість встановити та змінювати порогові значення даних з датчиків. До них відносяться:

- порогове значення вологості ґрунту – встановлюється в відсотках (%), діапазон значень від 0 до 100;
- порогове значення освітленості – встановлюється в люксах (лк), діапазон значень від 0 до 2000.

Програмний продукт має отримувати дані з датчиків та надавати можливість їх перегляду. До датчиків, з якими буде працювати додаток, відносяться:

- датчик вологості ґрунту (вимірюється в відсотках, %);
- датчик освітленості (вимірюється в люксах, лк);
- датчик температури (вимірюється в градусах Цельсія, °С), тиску (вимірюється в Паскалях, Ра) та вологості повітря (вимірюється в відсотках, %).

Згідно отриманих даних програмний продукт має виконувати керування елементами системи поливу та освітлення. Рішення щодо виконуваних команд продукт приймає базуючись на наступних правилах:

1. Якщо фактичне значення вологості ґрунту менше за встановлене порогове, то вмикається полив до збільшення значення до необхідного.
2. Якщо фактичне значення освітленості менше за встановлене порогове, то збільшується освітлення до необхідного.
3. Якщо фактичне значення освітленості більше за встановлене порогове, то зменшується освітлення до необхідного.

Має бути передбачена робота програмного продукту у двох режимах: автоматичному та ручному. В ручному режимі дані оновлюються по запиту користувача. В автоматичному режимі програмний продукт має отримувати дані з датчиків із заданою у налаштуваннях періодичністю. Час оновлення встановлюється у секундах (с), за замовченням він дорівнює 2с.

Програмний додаток через зазначені проміжки часу або по запиту користувача отримує наступні дані з датчиків:

- вологість ґрунту, %;
- освітленість, лк;
- температуру повітря, °С;
- повітряний тиск, Ра;
- вологість повітря, %.

Також вхідними даними є порогові значення:

- вологості ґрунту, %;
- освітленості, лк.

Вихідними даними є дані з датчиків, які оновлюються з заданою періодичністю:

- вологість ґрунту, %;
- освітленість, лк;
- температуру повітря, °С;
- повітряний тиск, Ра;
- вологість повітря, %.

Команди керування елементами стенду:

- ввімкнення поливу;
- вимкнення поливу;
- збільшення освітлення;
- зменшення освітлення.

Специфікація функціональних вимог представлена у вигляді діаграми прецедентів [6] (рис. 2.1).

2.2 Вибір мови програмування

Мова програмування – це набір формальних правил, за якими пишеться код для програмного забезпечення. Наразі існує безліч різних мов програмування, а вибір конкретної залежить від цілей і можливостей розробника, операційної системи, типу програми та вимог до неї [7].

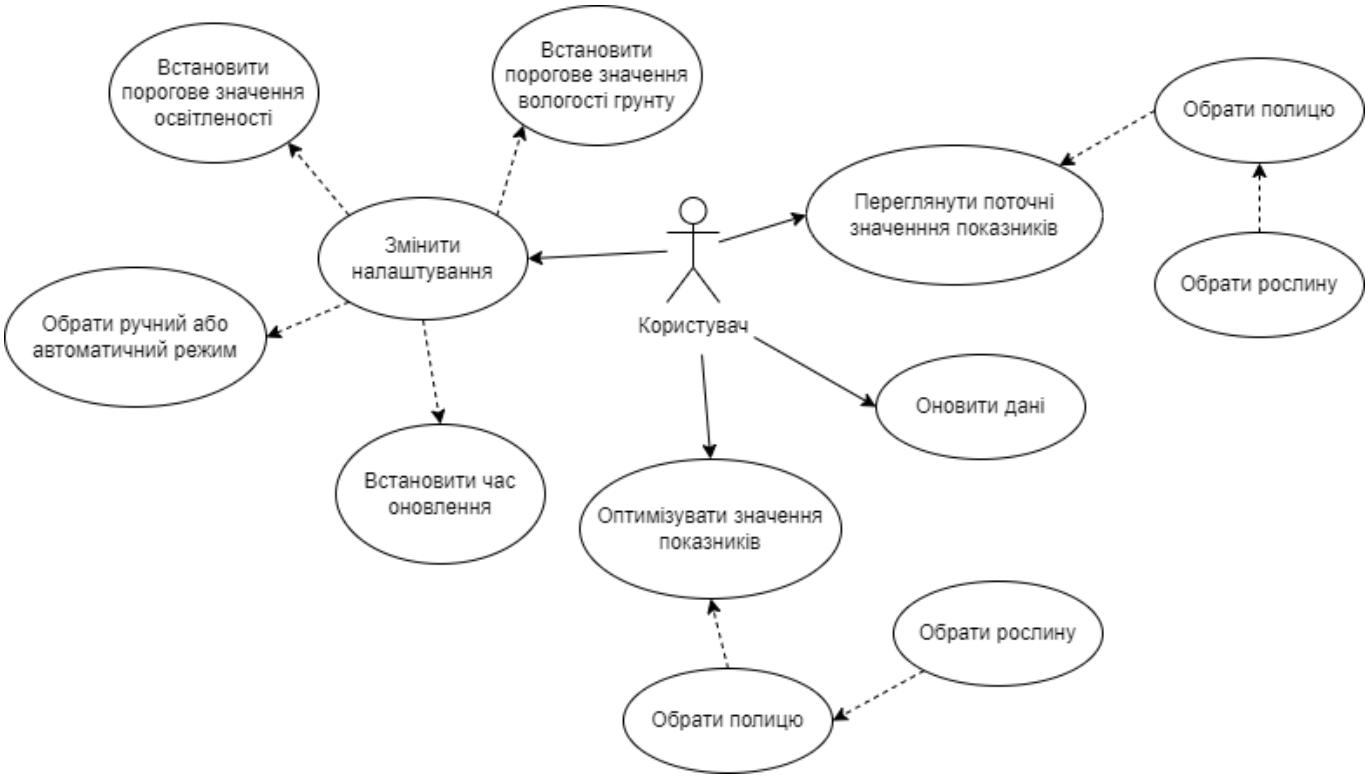


Рисунок 2.1 – Діаграма прецедентів

Kotlin – мова програмування для створення програм під Android, рекомендована Google. Вона була створена у 2010 році міжнародною компанією JetBrains для подолання недоліків Java та використовується для написання коду майже всіх нових програм на Android. Kotlin за кілька років завоював довіру програмістів та став галузевим стандартом в Android-розробці. У 2019 році компанія Google оголосила Kotlin кращою мовою для розробки програм під її мобільну операційну систему, що означає пріоритетну підтримку з боку компанії у всіх інструментах, компонентах та API порівняно з іншими мовами [8].

Kotlin знайшов застосування і в серверній розробці, де почав тіснити позиції Java. В останні роки також розвивається Kotlin Multiplatform Mobile (KMM) – кросплатформова версія мови Kotlin, що дозволяє створювати спільну бізнес-логіку iOS та Android-додатків [9].

Переваги:

- дозволяє обходитися меншою кількістю коду, ніж Java;
- Kotlin взаємозамінний з Java, тому різні частини інтерфейсу можуть бути написані різними мовами, але при цьому добре працювати;

- Безпека. Усі синтаксичні помилки та баги, пов'язані з неправильним зверненням до об'єктів, можна знайти та виправити під час збирання. Це спрощує тестування;
- програми Kotlin можуть використовувати фреймворки та бібліотеки, написані на Java.

Недоліки:

- швидкість складання програм на Kotlin трохи нижче, ніж у програм на Java;
- кросплатформова версія мови (KMM) поки не набула широкого поширення і поступається іншим кросплатформним рішенням.

Swift – мова програмування від Apple, на якій можна створювати програми для iOS, Apple Watch та Apple TV [10]. Swift не тільки перейняв усі можливості Objective C, але й був наділений новими функціями, які спрощують написання та реалізацію коду [9]:

Переваги:

- висока швидкість;
- простий для читання синтаксис та код;
- підвищена безпека, якщо порівнювати з Objective;
- спрощений спосіб виправлення помилок у коді;
- стабільність за рахунок бібліотек, які автоматично зв'язуються з оновленою версією та приєднуються до програми;
- забезпечує безпечне керування пам'яттю.

Недоліки:

- Swift використовується лише для розробки під пристрої Apple;
- погана сумісність із попередніми версіями мови;
- для розробки обов'язково потрібна техніка Apple.

Java до 2018 року була основною мовою для створення програм під Android, але і в 2022 її продовжують використовувати розробники для підтримки старих проектів [11]. Важливо відзначити, що сама по собі мова не втратила популярності в IT, але

набагато частіше зараз її використовують для Back-end розробки. Нові програми для Android цією мовою пишуть рідко [9].

Переваги:

- природний код для Android. Сама ОС частково теж написана на Java, а ядро становлять Linux та власна віртуальна машина Virtual Machine;
- дозволяє легко масштабувати та оновлювати проекти за рахунок об'єктно-орієнтованого коду;
- багато готових інструментів, які за умовчанням сумісні з Java, що також збільшує швидкість розробки;
- на мові Java можна розробляти не лише мобільні, а й серверні, десктопні, системні та інші програми.

Недоліки:

- програми на Java більше схильні до помилок, ніж на Kotlin;
- програми виходять досить багатослівними, що ускладнює їхнє читання;
- нові бібліотеки для Android-розробки насамперед орієнтовані на Kotlin, а не Java.

Objective C почали використовувати в 80-х роках 20 століття. Він був створений на основі C і Smalltalk, а в 2008 Apple випустив iPhone SDK 2.0, що дозволяє створювати програми для iOS [12]. Спочатку його вважали надбудовою до C, але коли його ліцензували NextStep і Apple, Objective C став офіційною мовою всіх інтерфейсів під iOS. У 2014 році вийшов потужніший Swift, який взяв собі все краще від Objective C, але був позбавлений його недоліків. Зараз більшість програмістів вибирають Swift, але Objective C все ще використовується для підтримки проектів Legacy [9].

Переваги:

- існує багато документації, яка спрощує роботу;
- сумісний зі Swift;

Недоліки:

- невисока продуктивність у порівнянні Swift;
- складний синтаксис.

Dart є мовою програмування загального призначення від компанії Google, що розробляється з 2011 року [13]. Спочатку він призначався для розробки веб-додатків. Але сфера його застосування значно змінилася коли вийшла перша версія Flutter – комплекту засобів розробки та фреймворку з відкритим вихідним кодом для створення мобільних додатків під Android та iOS, веб-додатків, а також десктопних додатків під Windows, macOS та Linux.

Flutter та лежача в його основі мова Dart активно розвиваються, переймаючи найкращі ідеї з Kotlin, Swift та інших мов програмування. Сьогодні Flutter є одним із найпопулярніших фреймворків у розробників кросплатформових додатків [9].

Переваги:

- проста в освоєнні мова програмування, однаково зручна для програмістів, які раніше писали нативний код для Android, iOS або веб-додатків;
- висока продуктивність щодо більшості інших кросплатформових фреймворків завдяки компіляції нативний код для цільової платформи.
- можливість писати єдиний код будь-якої платформи;
- незважаючи на свою молодість, Flutter та Dart вже завоювали велику популярність серед програмістів, тому для даного фреймворку адаптовано безліч бібліотек, інструментів, можна легко знайти документацію та приклади.

Недоліки:

- число фахівців, які знають Flutter, зростає, але поки що поступається кількістю нативних програмістів;
- мова Dart поки що поступається в гнучкості мови Kotlin, хоча і тут відставання скорочується.

JavaScript – одна з найпопулярніших мов програмування у світі. [14] Вона використовується для створення інтерактивних веб-сторінок, мобільних програм і навіть у серверній розробці. У контексті мобільних програм JavaScript застосовується в React Native: це кросплатформовий фреймворк з відкритим вихідним кодом для розробки мобільних і десктопних програм. React Native підтримує такі платформи як

Android, iOS, macOS, Web, Windows та UWP, дозволяючи розробникам використовувати можливості бібліотеки React поза браузером для створення програм, що мають повний доступ до системних API платформ [9].

Переваги:

- легкість освоєння React Native за рахунок мови JavaScript;
- велика поширеність: до половини кроссплатформенных додатків розробляються на React Native;
- React Native підтримує інтеграцію в вже існуючі програми – наприклад, частина інтерфейсу мобільного додатку може бути реалізована на React Native, а частина – за допомогою суто платформних засобів.

Недоліки:

- продуктивність програм на React Native поступається програмам на Flutter;
- простота мови розробки негативно позначається на захищеності від помилок, порівняно з більш строгими мовами.

Виходячи з розглянутих переваг та недоліків найбільш використовуваних для створення мобільних додатків мов програмування, було вирішено обрати для розробки проекту мову Dart та фреймворк Flutter.

2.3 Опис об’єктно-орієнтованого програмування

Обрана для розробки проекту мова програмування Dart є об’єктно-орієнтованою.

Об'єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило

переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП або ж є цілком об'єктно-орієнтованими [15].

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-х роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП-програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт – своєрідний незалежний автомат з окремим призначенням та відповідальністю.

На думку Алана Кея, розробника мови Smalltalk, якого вважають одним з «батьків-засновників» ООП, об'єктно-орієнтований підхід полягає в наступному наборі основних принципів:

- все є об'єктами;
- всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, під час якої один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи й отримуючи повідомлення. Повідомлення – це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися під час виконання дії;
- кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів;
- кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів;
- у класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії;

– класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Таким чином, програма є набором об'єктів, що мають стан та поведінку. Об'єкти взаємодіють використовуючи повідомлення. Будується ієрархія об'єктів: програма в цілому – це об'єкт, для виконання своїх функцій вона звертається до об'єктів що містяться у ньому, які своєю чергою виконують запит шляхом звернення до інших об'єктів програми. Звісно, щоб уникнути нескінченної рекурсії у зверненнях, на якомусь етапі об'єкт трансформує запит у повідомлення до стандартних системних об'єктів, що даються мовою та середовищем програмування. Стійкість та керованість системи забезпечуються шляхом чіткого розподілу відповідальності об'єктів (за кожну дію відповідає певний об'єкт), однозначного означення інтерфейсів міжоб'єктної взаємодії та повної ізольованості внутрішньої структури об'єкта від зовнішнього середовища (інкапсуляції).

2.4 Проектування архітектури системи

У додатку, що проектується, можна умовно виділити два модулі: модуль з основною логікою програми та модуль з графічним представленням. Під час розробки проекту файли окремих модулів доцільно зберігати у різних директоріях.

Основна логіка програми буде представлена наступними класами:

– CurrentView;
– RandomStatisticsGetter;
– SettingsData;
– Statistics;
– StatisticsData;
– StatisticsGetter;
– TimerFunctions.

Графічний інтерфейс додатку буде представлений наступними класами:

– CustomStyles;

- NavDrawer;
- SettnigsForm;
- _SettingsFormState;
- TreeWidget;
- _TreeWidgetState;
- MyApp;
- _MyAppState.

Statistics – основний клас логіки програми. Надає зовнішній інтерфейс для керування даними з датчиків. Серед його методів є наступні:

- update() – отримує та зберігає поточні дані з усіх датчиків;
- getTemperatue(), getHumidity(), getPressure(), getLuminosity(int), getSoilHumidity(int) – повертають останні збережені дані температури, вологості повітря, тиску, освітленості полиці, вологості ґрунту рослини відповідно;
- changeLuminosity(int), changeHumidity(int) – оптимізують значення освітленості для полиці або значення вологості ґрунту рослини відповідно;
- optimize() – оптимізують значення показників освітленості та вологості ґрунту для усіх полиць і рослин.

StatisticsData – клас для зберігання значень показників датчиків.

StatisticsGetter – абстрактний клас для визначення методів, необхідних для отримання даних з датчиків. Включає визначення таких методів:

- int getTemperature() – метод має повертати отримане з датчика значення температури повітря;
- int getHumidity() – метод має повертати отримане з датчика значення вологості повітря;
- int getPressure() – метод має повертати отримане з датчика значення тиску;

- List<int> getLuminosity() – метод має повертати отримані з датчиків значення освітленості з усіх полиць;
- List<int> getSoilHumidity() – метод має повертати отримані з датчиків значення вологості ґрунту усіх рослин.

RandomStatisticsGetter – тестова реалізація методів класу StatisticsGetter. Пропонує реалізацію зазначених методів за допомогою випадкового генерування значень.

TimerFunctions – клас, що містить функції для роботи з таймером. Включає наступні методи:

- getAutoUpdate() – повертає стан змінної, що вказує, чи ввімкнено режим автоматичної роботи застосунку;
- setAutoUpdate() – встановлює значення змінної, що вказує, чи ввімкнено режим автоматичної роботи застосунку;
- restart() – перезапускає виконання автоматичного оновлення даних з датчиків;
- startTimer() – переводить додаток у автоматичний режим та починає автоматично оновлювати дані з датчиків;
- cancelTimer() – зупиняє виконання автоматичного оновлення даних з датчиків;

CurrentView – перелік, що зберігає можливі значення поточного зображуваного виду стенду.

- GENERAL – відображення усього стенду.
- SHELF – відображення полиці з двома рослинами.
- PLANT – відображення однієї рослини.

MyApp – основний клас графічного представлення застосунку, наслідує стандартний клас StatefulWidget. Пов’язаний із класом стану _MyAppState. Відповідає за відображення головної форми застосунку.

SettingsForm – клас, що відповідає за відображення форми налаштувань, наслідує стандартний клас StatefulWidget. Пов’язаний з класом стану _SettingsFormState.

TreeWidget – клас, що відповідає за відображення зображення зі стендом на головній формі застосунку, наслідує стандартний клас StatefulWidget. Пов’язаний з класом стану _TreeWidgetState.

NavDrawer – клас, що відповідає за відображення бокової панелі з навігацією, наслідує стандартний клас StatelessWidget.

CustomStyles – клас, який зберігає налаштування стилів відображення тексту та кнопок у додатку.

Спроектовані класи представлені у вигляді діаграми класів (рис. 2.2) [6].

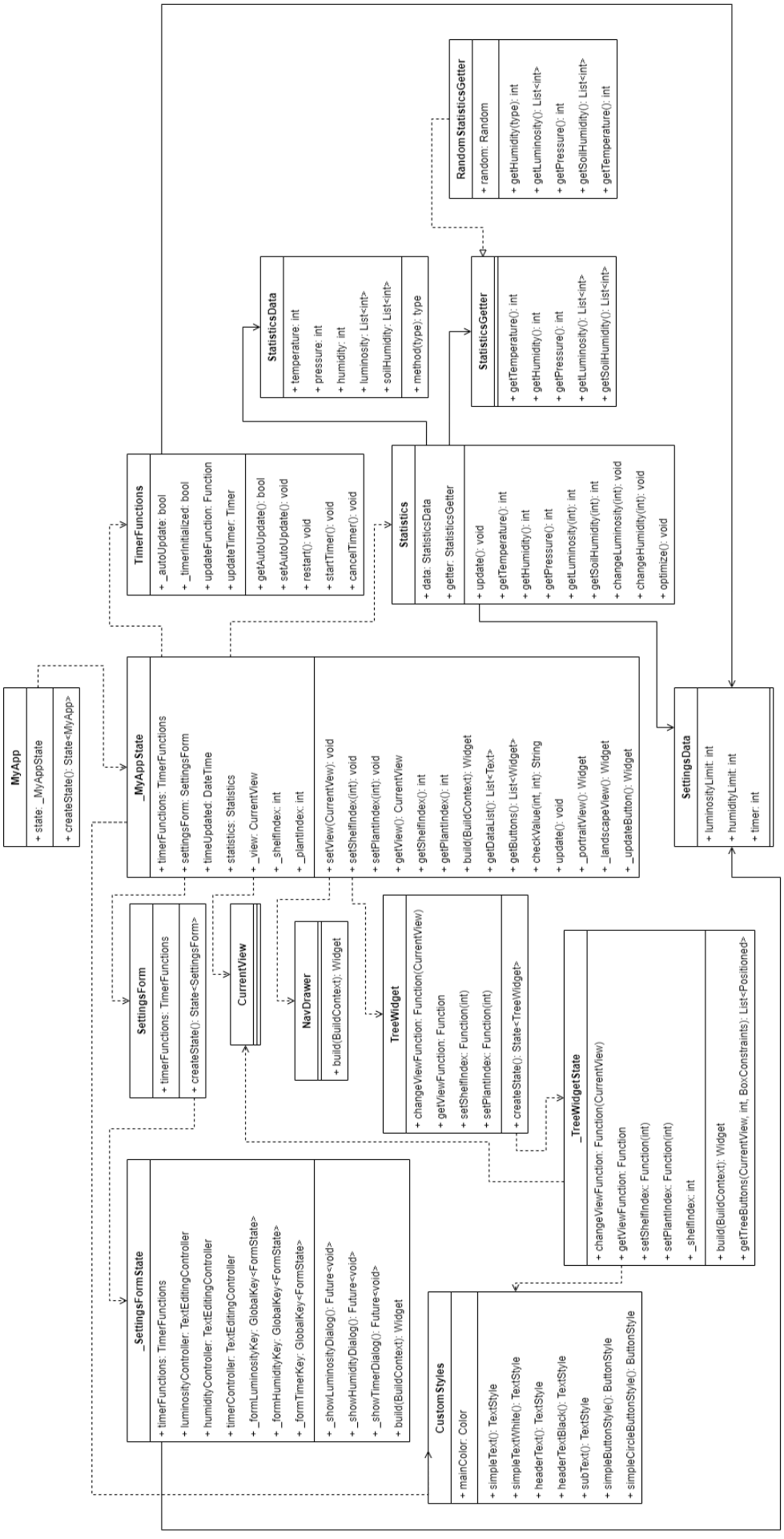


Рисунок 2.2 – Діаграма класів

2.5 Проектування інтерфейсу користувача

Згідно до функціональних вимог застосунку та спроектованих класів можна виділити наступні форми інтерфейсу:

– Головна форма – форма, де користувач може переглянути поточні дані з датчиків та відправити запити по оптимізації їхніх значень.
– Форма налаштувань – форма, де користувач може встановити порогові значення освітленості та вологості ґрунту, а також ввімкнути або вимкнути автоматичний режим та встановити час оновлення.

Спроектований сценарій діалогу програми наведено у діаграмі станів користувача програми (рис. 2.3) [6].

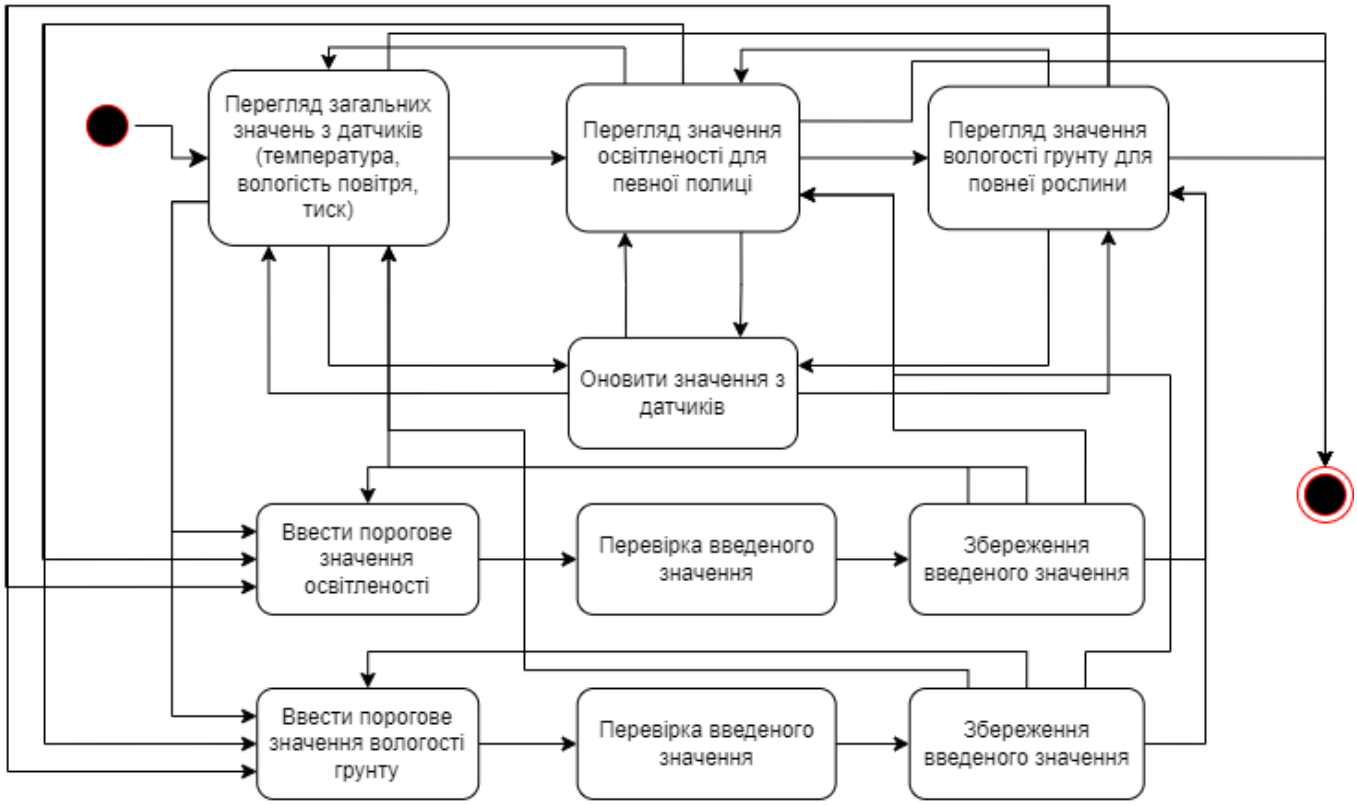


Рисунок 2.3 – Діаграма станів користувача

Згідно з представленого сценарію станів користувача були розроблені наступні макети екранних форм (рис. 2.4).

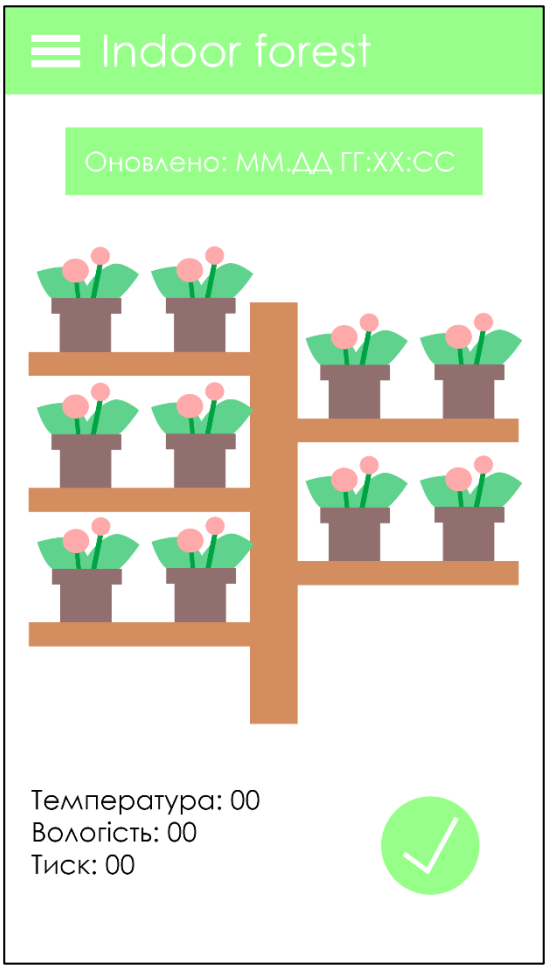


Рисунок 2.4 – Головна форма, початковий вигляд

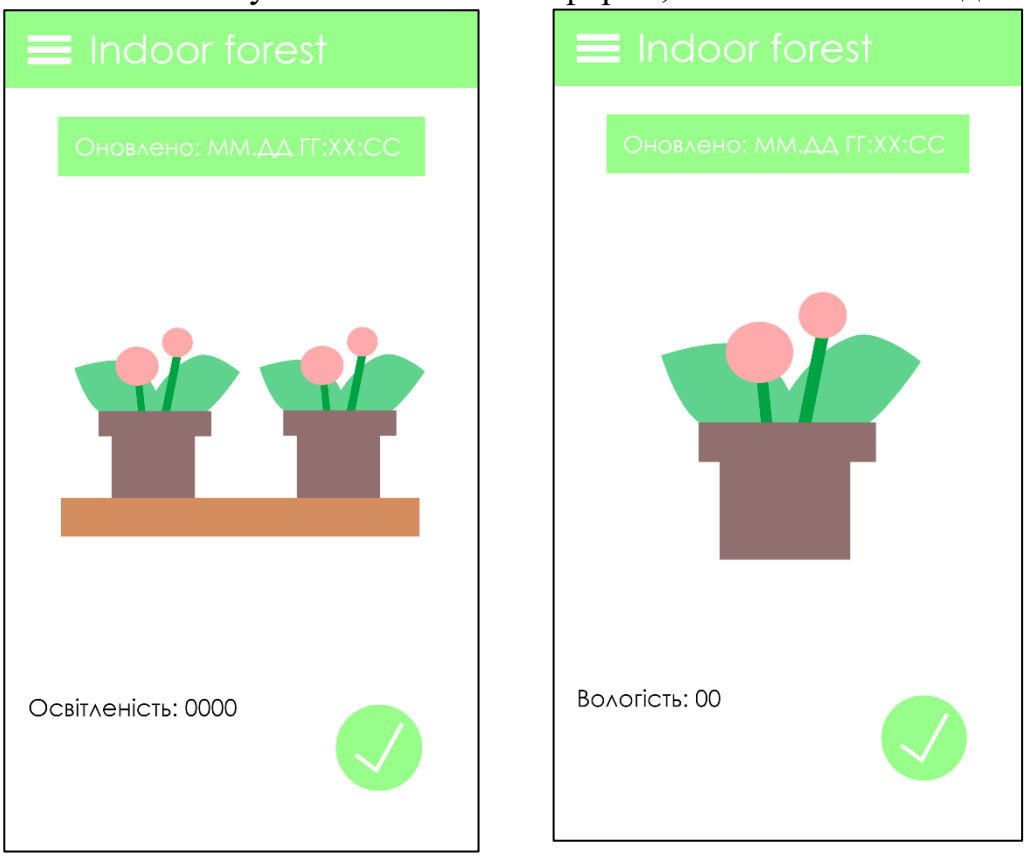


Рисунок 2.5 – Головна форма, відображення окремої полиці і окремої рослини

На рис. 2.4-2.5 представлені макети головної форми застосунку. У верхній частині форми розташована панель з назвою проекту Indoor forest. На початку цієї панелі розташована кнопка з трьома смужками (☰) для відкриття бокового меню навігації.

Основну частину форми займає інтерактивне зображення стенду. При натисканні на конкретну полицю користувач може переглянути інформацію про освітленість полиці. Зображення також відповідно змінюється і замість всього стенду відображає лише одну полицю с двома рослинами. При натисканні на конкретну рослину користувач може переглянути інформацію про вологість ґрунту цієї рослини. Уся інформація відображається внизу під зображенням. Отже, маємо 3 можливих варіантів вигляду головної форми:

- загальний – відображається весь стенд та інформація по загальним показникам (температура, вологість повітря і тиск);
- полиця – відображається одна полиця та інформація про її освітленість;
- рослина – відображається одна рослина та інформація про її вологість ґрунту.

На формі також розташовано дві кнопки, зовнішній вигляд та положення яких не залежить від обраного режиму перегляду.

Одна з них розташована над зображенням стенду та відповідає за ручне оновлення даних. При натисканні на неї застосунок отримує поточні значення з датчиків і оновлює дані у застосунку та на формі. На кнопці також відображається час останнього оновлення даних у форматі ММ.ДД ГГ:ХХ:СС.

Друга кнопка розташовується поруч з даними з датчиків і відповідає за відправлення запиту на оптимізацію показників. Зміст запиту відмінний в залежності від поточного режиму перегляду:

- у загальному режимі натискання на кнопку відправляє запит на оптимізацію значень освітленості усіх полиць і значень вологості ґрунту усіх рослин;

– у режимі полиці натискання на кнопку відправляє запит на оптимізацію значення освітленості конкретної полиці;
– у режимі рослини натискання на кнопку відправляє запит на оптимізацію значення вологості ґрунту конкретної полиці.

Рисунок 2.6 – Бокове меню навігації

При натисканні на кнопку з трьома смужками (☰) з лівого боку відкривається меню навігації (рис. 2.6). З нього користувач може перейти до форми налаштувань (рис. 2.7).

Налаштування на формі поділені на 2 розділи:

– загальні (порогові значення освітленості і вологості);
– автоматизація (ввімкнення/вимкнення автоматичного режиму, час оновлення даних).

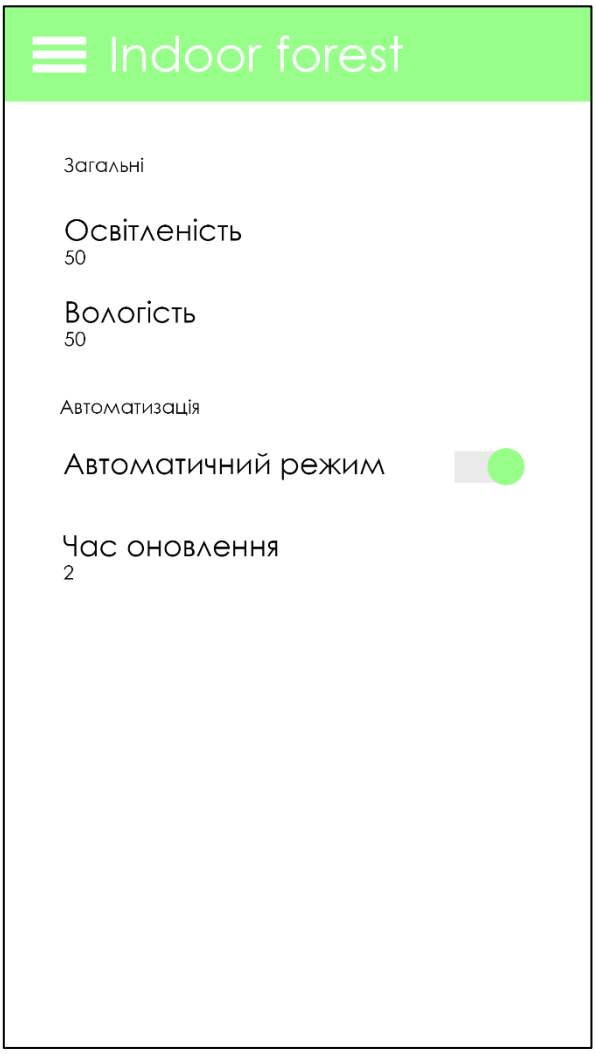


Рисунок 2.7 – Форма налаштувань

Опції освітленості, вологості та часу оновлення представлені кнопками, при натисканні на які відкривається діалогове вікно для введення нового значення. Для зручності поточні порогові значення освітленості і вологості виводяться під відповідними кнопками. Опція автоматичного режиму представлена перемикачем. Повернутися до головної форми користувач може за допомогою стрілки у верхній лівій частині форми.

Макет діалогового вікна для введення нових значень представлено на рис. 2.8. В верхній частині вікна розташована назва змінюваного значення. Під ним користувач бачить підказку стосовно обмежень для нового значення. Далі розташоване саме поле для введення значення. Після введення значення користувач має натиснути кнопку «Зберегти». Якщо нове значення не буде відповідати

зазначеним обмеженням, то на цьому ж діалоговому вікні користувач побачить повідомлення про помилку.

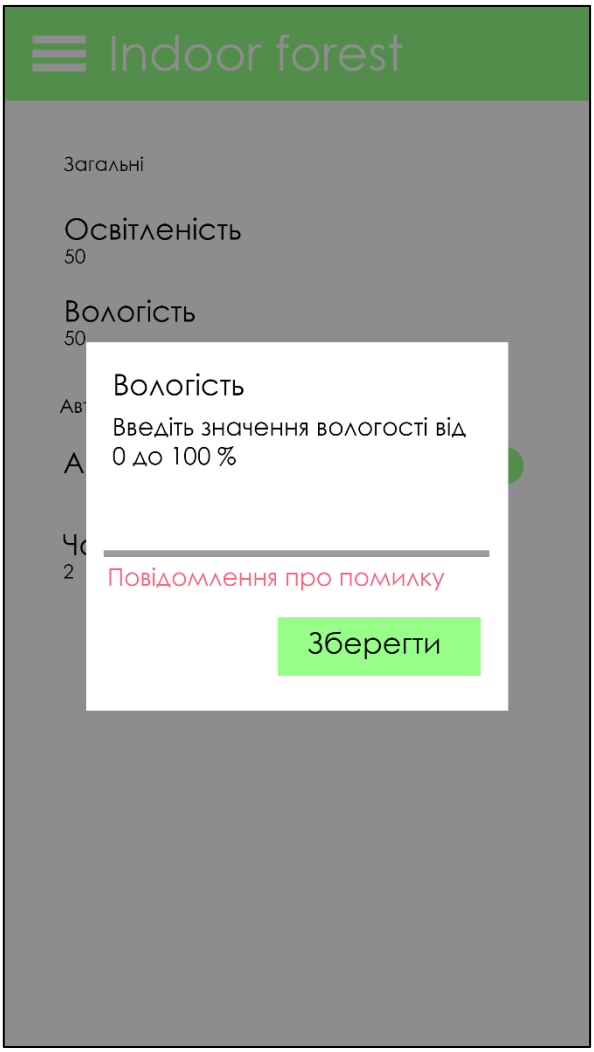


Рисунок 2.8 – Приклад діалогового вікна

Головним кольором застосунку було обрано зелений для асоціації проекту із рослинами. Для привернення уваги користувача натискання на кнопки супроводжується звуковим сигналом.

2.6 Проектування динаміки системи

Для представлення структурних особливостей передачі і прийому повідомлень між об’єктами було розроблено діаграми послідовностей [6] для наступних прецедентів:

– ручне оновлення даних (рис. 2.9);
– оптимізація усіх показників (рис. 2.10);
– введення нового порогового значення вологості ґрунту (рис. 2.11).

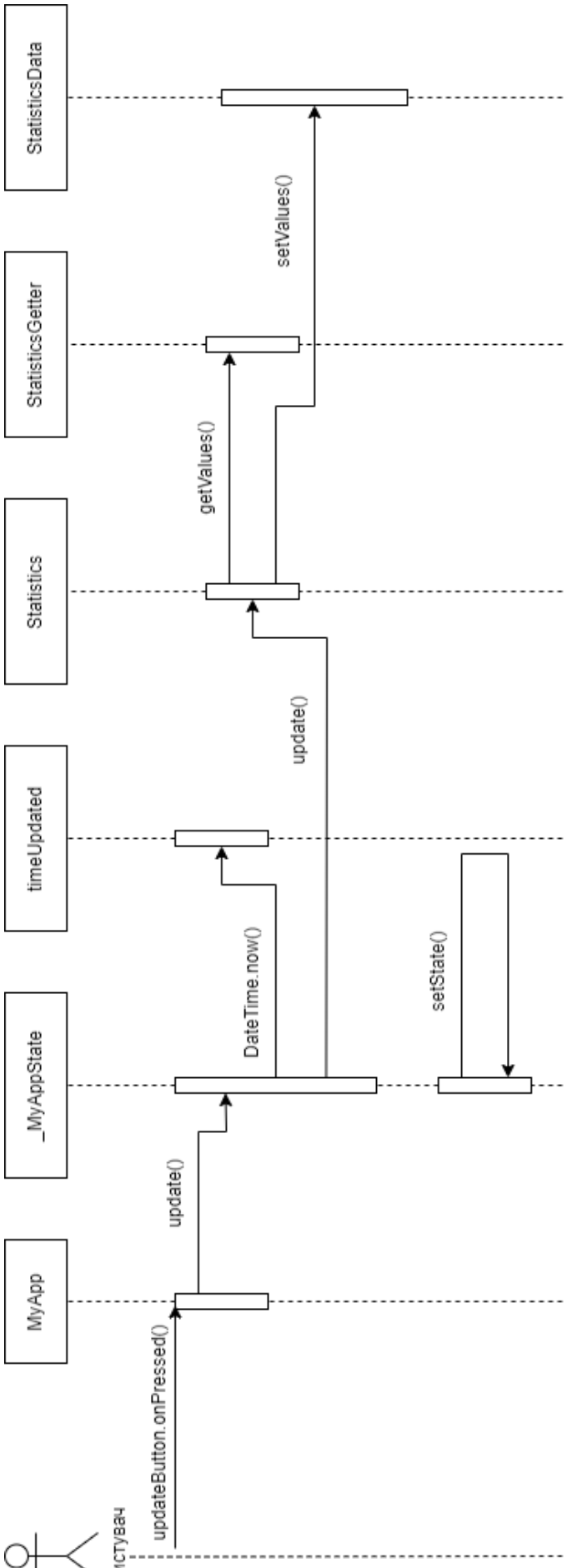


Рисунок 2.9 – Діаграма послідовностей для прецеденту «Ручне оновлення даних»

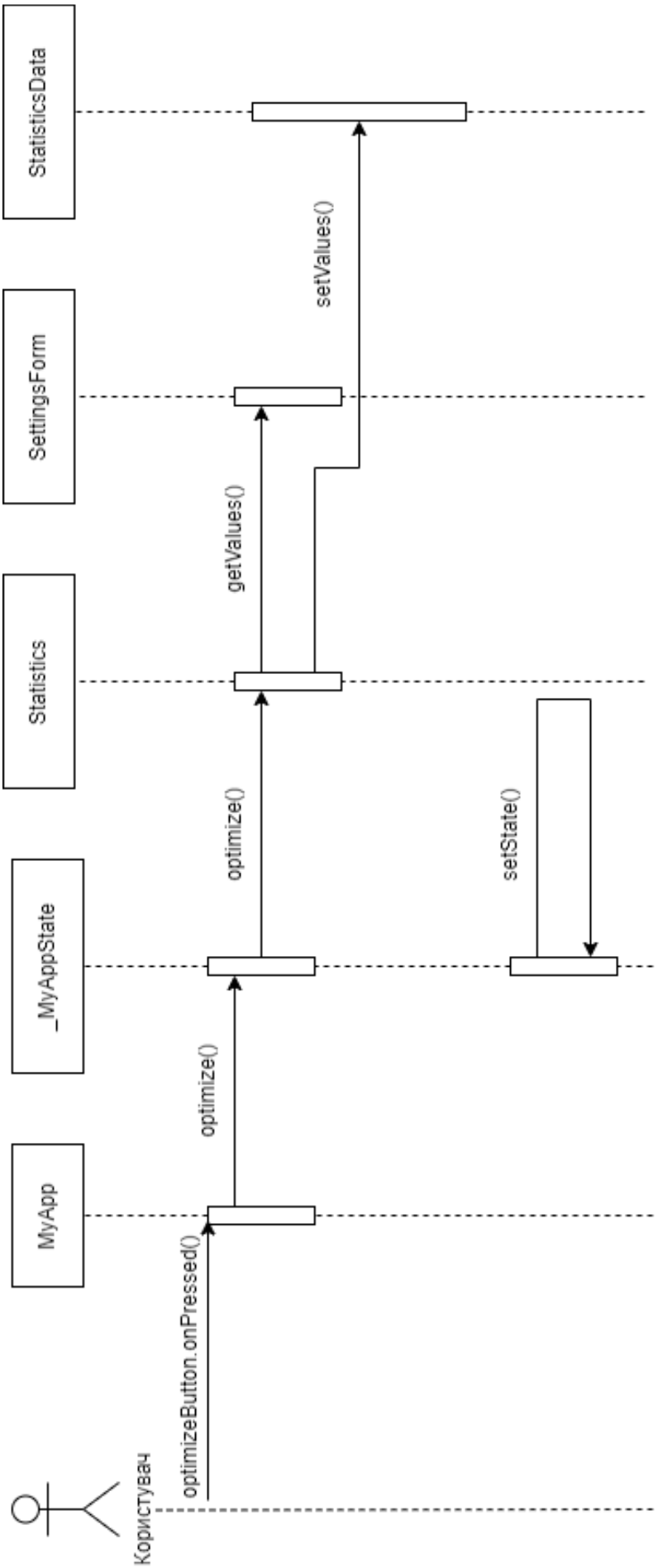


Рисунок 2.10 – Діаграма послідовностей для прецеденту «Оптимізація усіх показників»

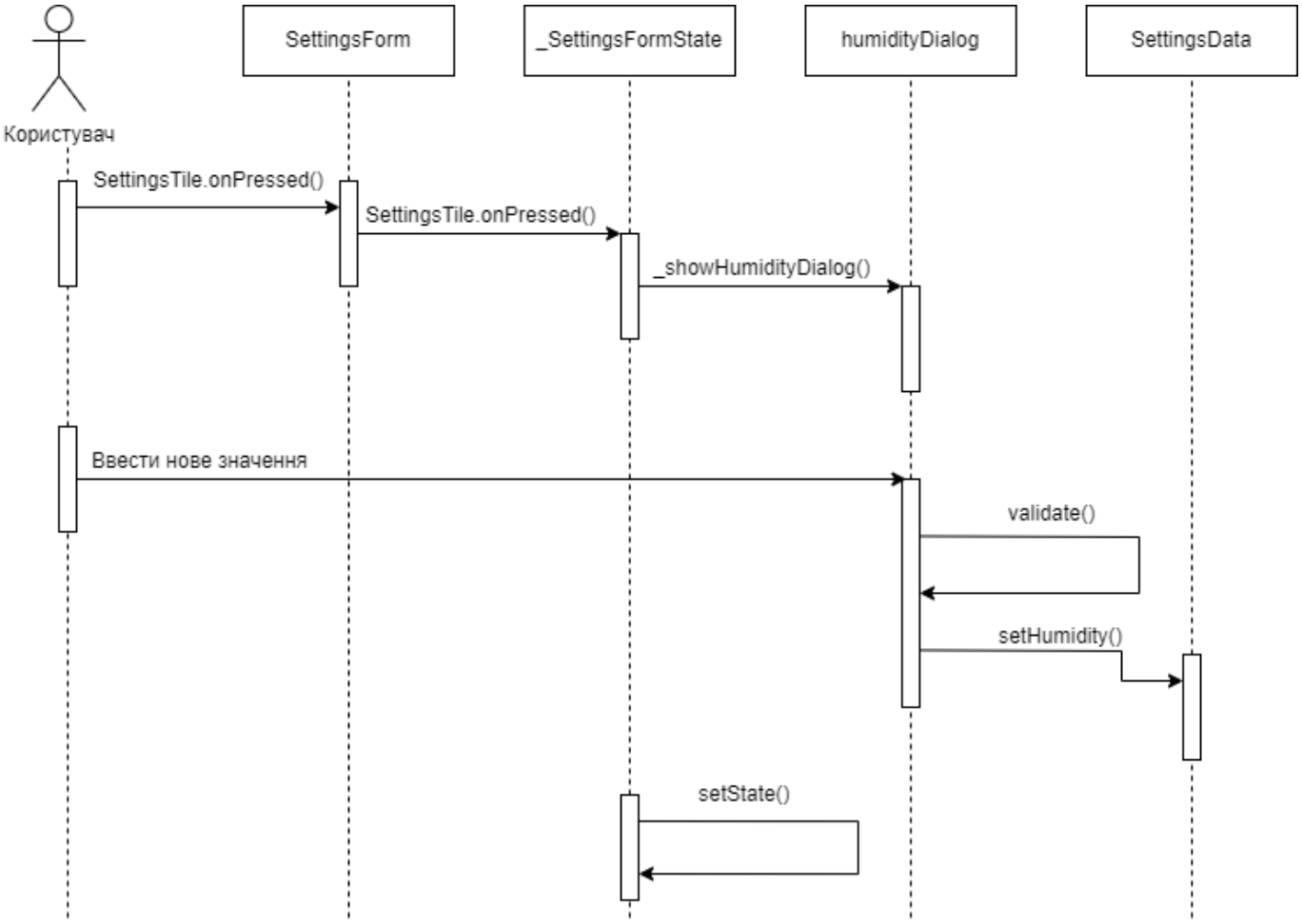


Рисунок 2.11 – Діаграма послідовностей для прецеденту «Введення нового значення вологості ґрунту»

Для кращого розуміння сценарію навігації користувача між представленнями застосунку були розроблені діаграми діяльності та станів [6] для прецеденту «Навігація між представленнями застосунку» (рис. 2.12-2.13).

Головна форма (MyApp) є основною формою застосунку, саме вона відкривається при його запуску та містить найбільш важливу інформацію, яку користувач буде частіше за все переглядати. Перейти до форми налаштувань можна за допомогою бокового меню навігації. Для цього користувач має натиснути кнопку з 3 смужками (☰) у верхній лівій частині головної форми, вона присутня на формі незалежно від поточного режиму відображення. Аби повернутися з форми налаштувань на головну форму користувач має натиснути кнопку зі стрілкою ліворуч (←) у верхній лівій частині форми. Відкрити бокове меню з форми налаштувань користувач не може. Закрити застосунок можна знаходячись на будь-якій формі.

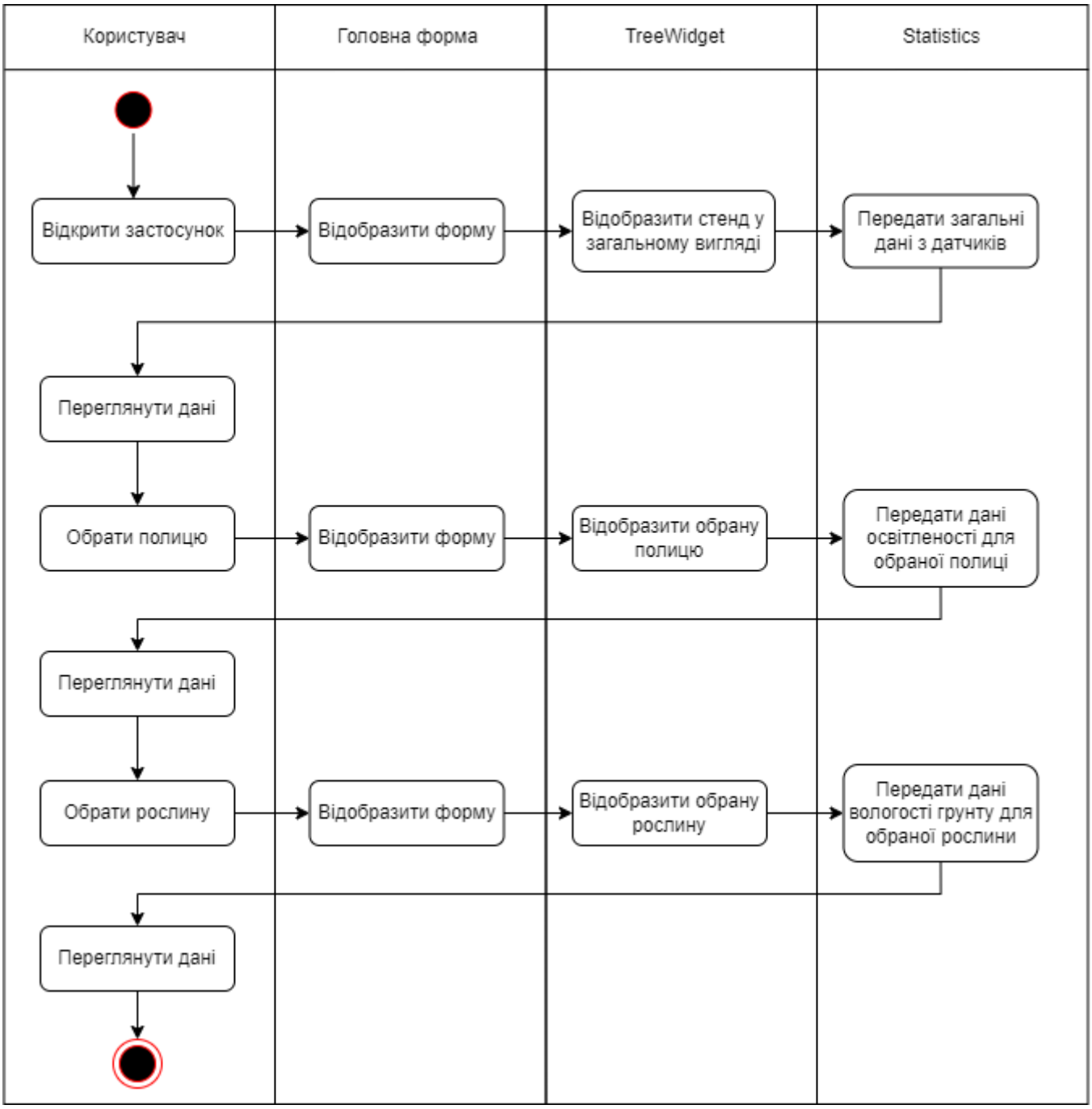


Рисунок 2.12 – Діаграма діяльності для прецеденту «Навігація між представленнями застосунку»

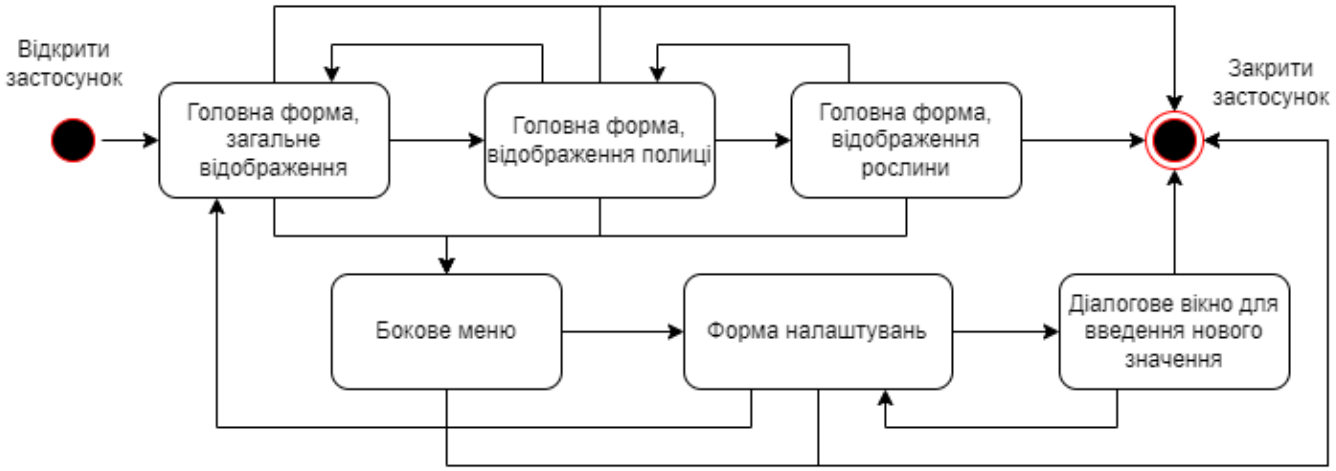


Рисунок 2.13 – Діаграма станів для прецеденту «Навігація між представленнями застосунку»

2.7 Проектування системи на фізичному рівні

На діаграмі артефактів (рис. 2.14) представлено фізичний проект системи. Головними двома артефактами є Indoor_forest.apk – мобільний додаток, що розробляється, та indoor_forest.py – скрипт, який прийматиме та відповідатиме на запити мобільного додатку.

Indoor_forest.apk є виконуваним файлом, результатом збірки усіх файлів з вихідним кодом та задіяними бібліотеками у інтегрованому середовищі розробки. Фізично він має бути встановлений на мобільний пристрій (телефон, планшет) користувача з операційною системою Android або iOS.

indoor_forest.py – скрипт, написаний мовою python, виконує роль серверу. Скрипт запускається на одно-платному комп’ютері Raspberry Pi, що розміщується на стенді та підключається до усіх датчиків та елементів стенду (лампи, насос). Саме до нього звертатиметься мобільний застосунок Indoor_forest.apk для отримання даних про показники датчиків та посилання запитів на керування елементами стенду (зміна рівню освітлення, ввімкнення/вимкнення поливу).

Зв’язок між двома пристроями (мобільним пристроєм та сервером) забезпечується мережею Wi-fi, що створюється одним із компонентів стенду. Таким чином, для роботи зі стендом користувачу не потрібне підключення до мережі Internet, йому необхідно підключитись до Wi-fi мережі стенду.

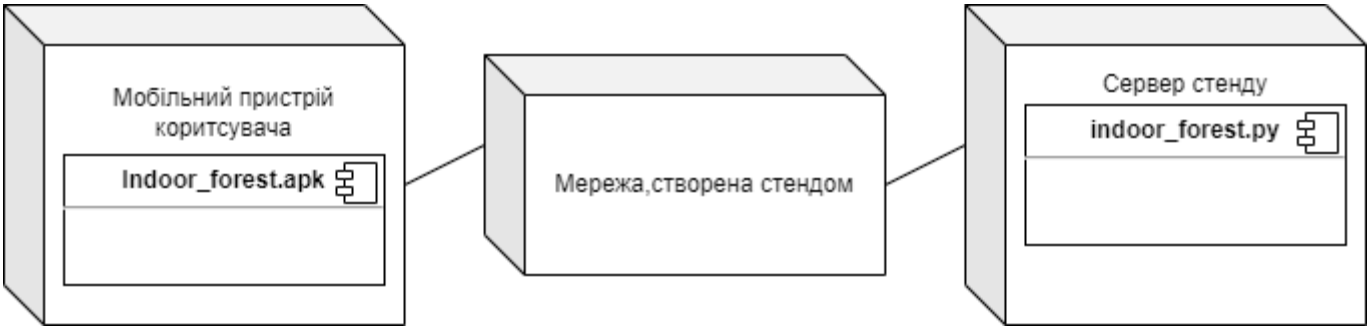


Рисунок 2.14 – Діаграма артефактів

Висновки до розділу 2

Після збору вимог було визначено функціональні вимоги до розроблюваного програмного забезпечення, описано вхідні та вихідні дані. Для більш наглядного представлення специфікації функціональних вимог було розроблено діаграму прецедентів.

Після порівняння декількох найбільш використовуваних для створення мобільних застосунків мов програмування та фреймворків, для розробки проекту було обрано мову Dart та фреймворк Flutter, зокрема завдяки наступним перевагам:

- простота у освоєнні;
- можливість писати єдиний код для обох платформ (Android та iOS);
- велика кількість готових бібліотек.

Оскільки Dart є об’єктно-орієнтованою мовою, було спроектовано класи, які будуть використовуватися у застосунку, зокрема було описано 7 класів логіки застосунку та 8 класів графічних представлень. На діаграмі класів було зображено спроектовані класи, їх атрибути та методи, а також зв’язки між спроектованими класами.

Під час проектування інтерфейсу користувача були розроблені макети графічних форм та компонентів застосунку, серед яких:

- головна форма;
- форма налаштувань;
- бокове навігаційне меню;
- діалогове вікно.

Для кращого розуміння динаміки системи та процесу взаємодії користувача з застосунком були розроблені діаграми для наступних прецедентів:

- ручне оновлення даних;
- оптимізація усіх показників;
- введення нового порогового значення вологості ґрунту.

Також були розроблені діаграма діяльності та діаграма станів для прецеденту для прецеденту «Навігація між представленнями застосунку».

Описано фізичні компоненти системи, зокрема Indoor_forest.apk та indoor_forest.py, а також представлено діаграму артефактів.

Усі спроектовані матеріали значно полегшують процес розробки застосунку та зменшують вірогідність допущення помилок чи несумісностей у фінальному програмному забезпеченні.

РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ

3.1 Опис використаних для розробки засобів

Для розробки програмного забезпечення було обрано інтегроване середовище розробки Android Studio.

Android Studio прийшло на зміну плаґіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0 [16].

Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), macOS і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Wear OS, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto). Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio.

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер

створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

Ручне тестування розробленого додатку було виконано за допомогою менеджеру пристроїв, що представлено у Android Studio. Зокрема, було використано один із стандартних образів під назвою Nexus 6. Характеристики віртуального пристрою Nexus 6 представлено у таблиці 3.1.

Таблиця 3.1 – Характеристики Nexus 6

Версія Android	9.0
API level	28
Розмірність екрану	1440 × 2560
Орієнтація екрану	Портрет
Кількість ядер	4
RAM	1536

3.2 Структура вихідного коду

Загальна структура розробленого проекту представлена на рис. 3.1.

Каталоги «.dart_tool» та «.idea» зберігають системні файли, необхідні для роботи середовища. Каталоги «android» та «ios» зберігають допоміжні файли, необхідні для компіляції коду для операційних систем Android та iOS відповідно. Каталог «build» зберігає скомпільовані допоміжні файли. Основні файли з вихідним кодом зберігаються у каталозі «lib». В окрему директорію «svg» були збережені графічні матеріали для застосунку.

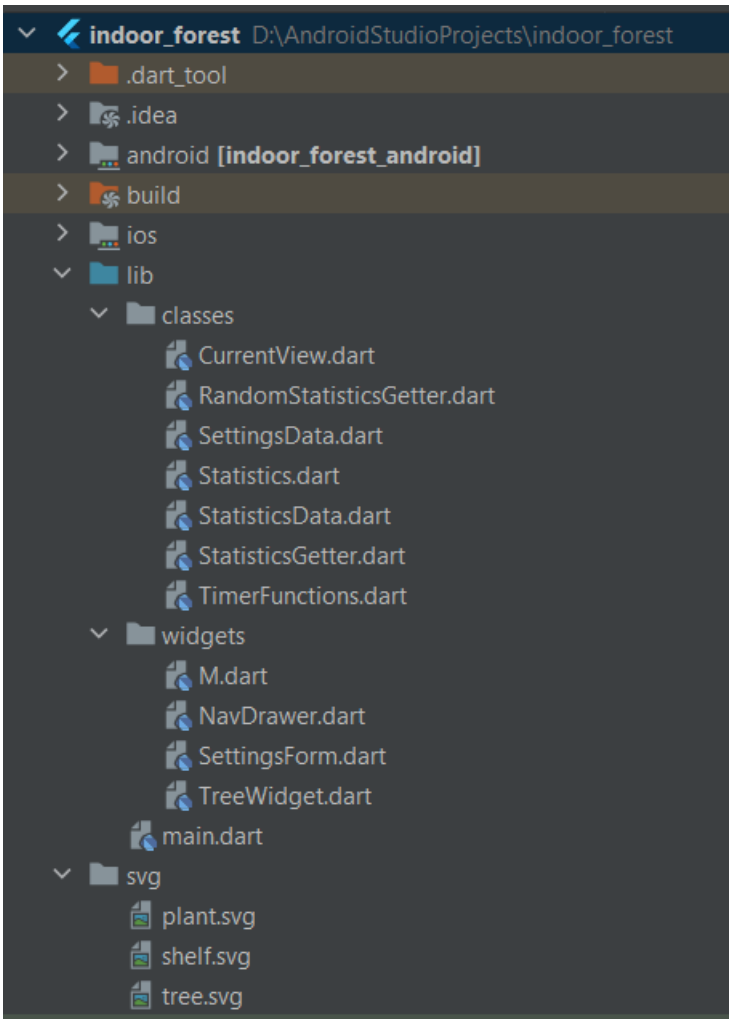


Рисунок 3.1 – Структура проекту

У директорії «lib» були додатково створені директорії «classes» та «widgets» для більш зручного та наглядного розміщення файлів з кодом. У директорії «classes» розміщено файли з описами класами логіки. У директорії «widgets» розміщено файли з описом графічних компонентів. Всього було створено 12 файлів з вихідних кодом:

- CurrentView.dart – зберігає опис переліку CurrentView;
- RandomStatisticsGenerator.dart – зберігає код класу RandomStatisticsGetter;
- SettingsData.dart – зберігає код класу SettingsData;
- Statistics.dart – зберігає код класу Statistics;
- StatisticsData.dart – зберігає код класу StatisticsData;
- StatisticsGetter.dart – зберігає опис абстрактного класу StatisticsGetter;
- TimerFunctions.dart – зберігає код класу TimerFunctions;
- M.dart – зберігає код класу CustomStyles;
- NavDrawer.dart – зберігає код класу NawDrawer;

– SettingsForm.dart – зберігає коди класів SettingsForm та _SettingsFormState;
– TreeWidget.dart – зберігає коди класів TreeWidget та _TreeWidgetState;
– main.dart – зберігає коди класів MyApp та _MyAppState, а також головний метод main().

У директорії «svg» зберігаються розроблені для застосунку 3 векторні зображення:

– plant.svg (рис. 3.2);
– shelf.svg (рис. 3.3);
– tree.svg (рис. 3.4).

Рисунок 3.2 – plant.svg

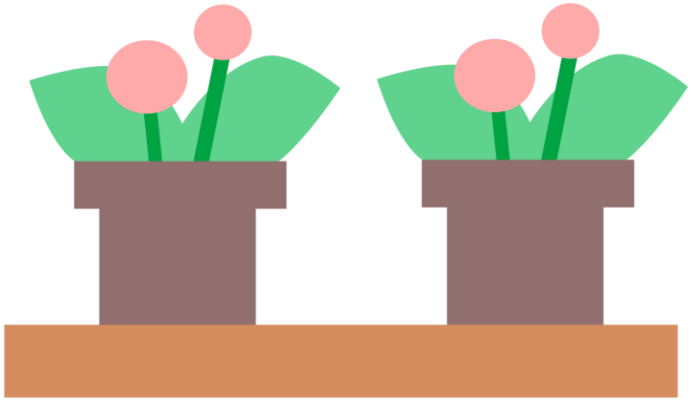


Рисунок 3.3 – shelf.svg

3.3 Розроблені графічні компоненти

Для розробки графічних компонентів застосунку були використані стандартні засоби фреймворку Flutter та бібліотека settings_ui.

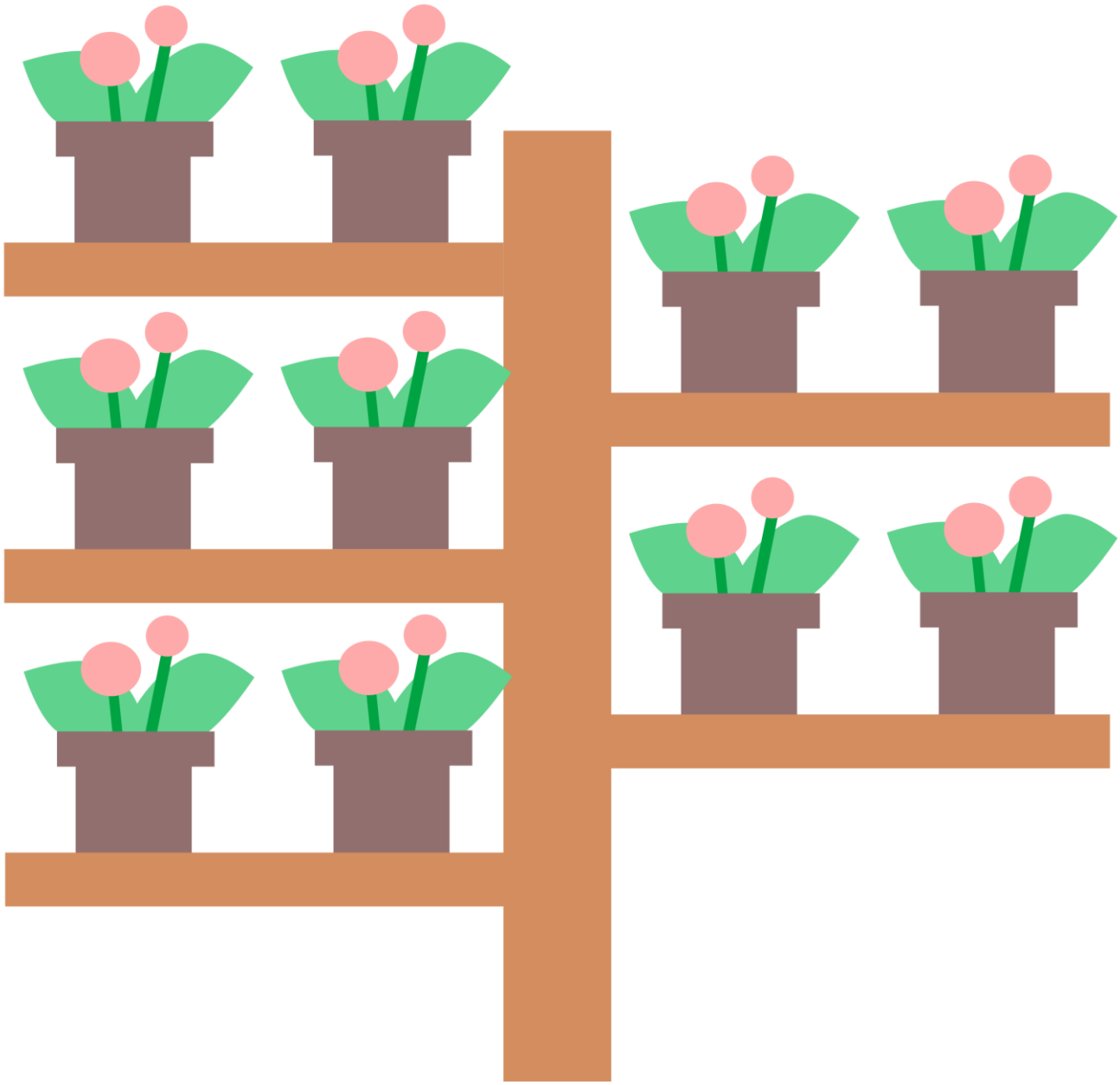


Рисунок 3.4 – tree.svg

Зокрема, були використані такі компоненти:

- StatelessWidget;
- StatefulWidget;
- State<>;
- Scaffold;
- AppBar;
- Text;

- OrientalBuilder;
- ElevatedButton;
- Icon;
- Column;
- Expanded;
- Container;
- Center;
- Row;
- Form;
- AlterDialog;
- SingleChildScrollView;
- TextFormField;
- SettingsList;
- SettingTile;
- SettingSection;
- ListView;
- DrawerHeader;
- BoxDecoration;
- ListTile.

Для відображення тексту було обрано сімейство шрифтів Calibri. Головним кольором було обрано один із стандартних кольорів, представлених у класі Colors, Colors.greenAccent ().

Зображення стенду, полиці та рослини розроблені у форматі векторної графіки svg. Це зроблено для збереження якості зображення на пристроях з різною розмірністю екранів. Робота з файлами формату svg забезпечується стандартниою бібліотекою flutter_svg.

Аби зробити графічний інтерфейс більш зрозумілим та дружнім до користувача більшість кнопок у застосунку відображаються з іконками (Icon), що мають визивати асоціації із дією, що кнопка виконує. Серед таких іконок є наступні:

– Icons.update ( ) – на кнопці для оновлення даних з датчиків;

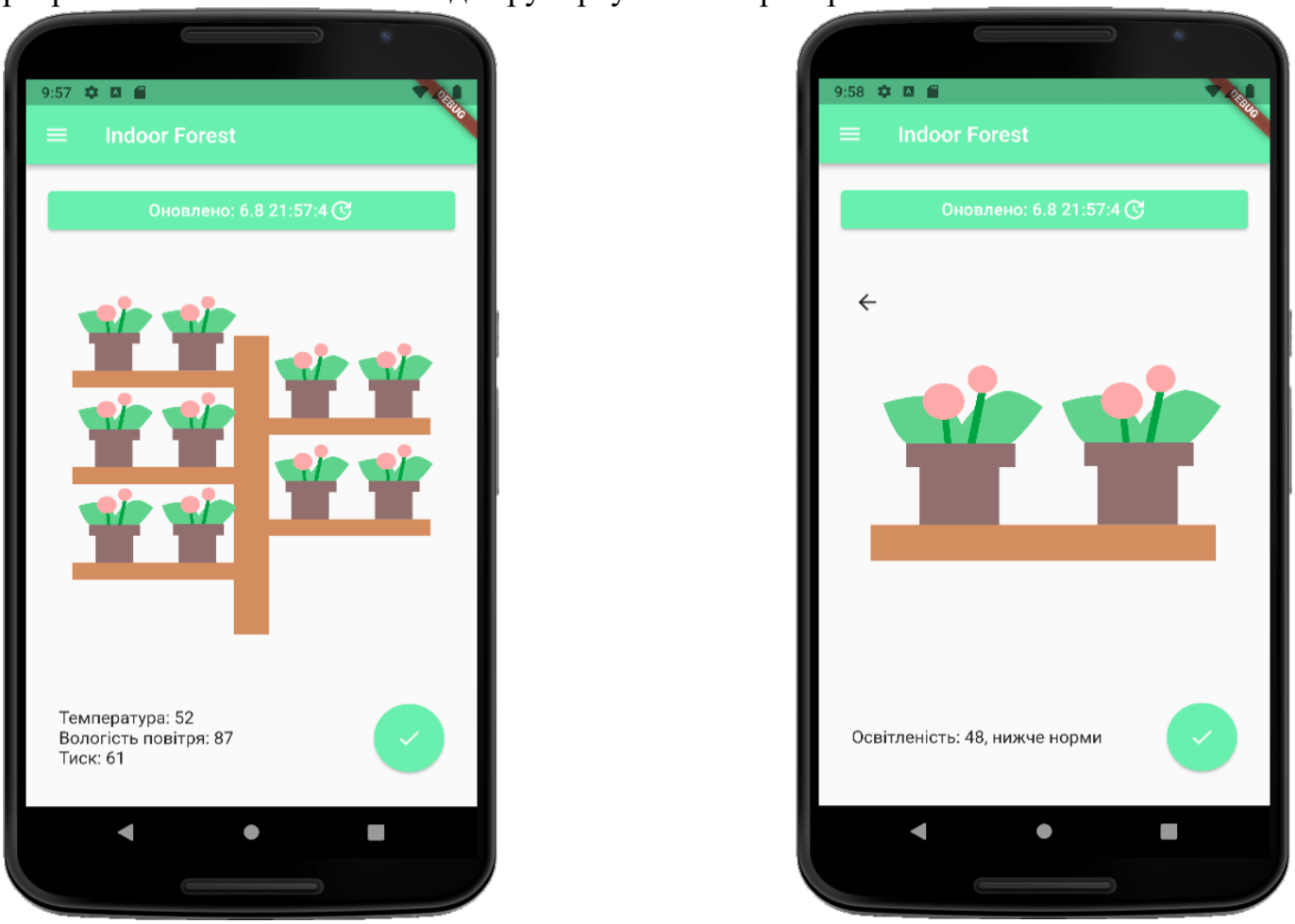
– Icons.settings ( ) – на кнопці переходу до форми налаштувань;

– Icons.light_mode_outlined ( ) – на кнопці зміни порогового значення освітленості;

– Icons.water_drop_outlined ( ) – на кнопці зміни порогового значення вологості;

– Icons.timer_outlined ( ) – на кнопці зміни часу оновлення даних.

Скріншоти розроблених екранних форм представлено на рис. 3.5-3.8. Скріншоти було зроблено за допомогою вбудованого в інтегроване середовище розробки Android Studio менеджеру віртуальних пристроїв.



а б

Рисунок 3.5 – Головна форма

а – загальне представлення

б – представлення полиці

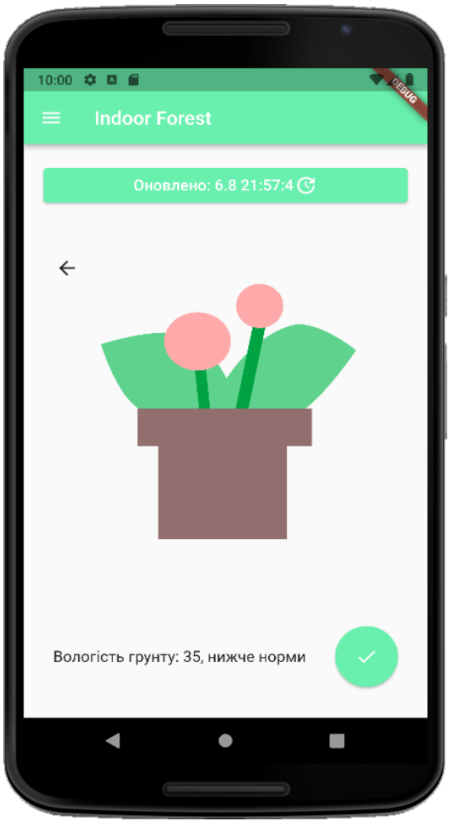
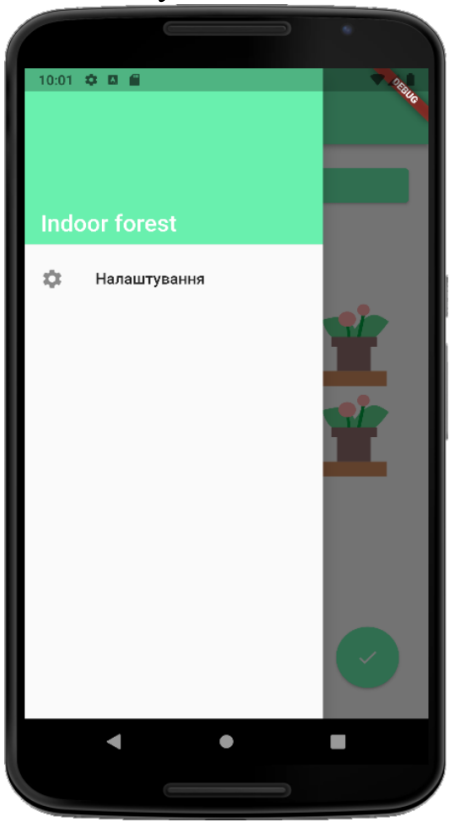
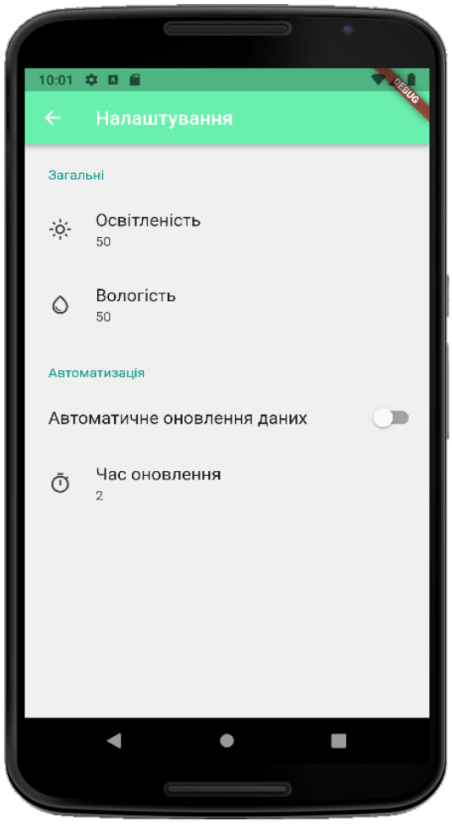


Рисунок 3.6 – Головна форма застосунку, представлення полиці



а



б

Рисунок 3.7 – Екранні форми

а – бокове меню навігації

б – форма налаштувань

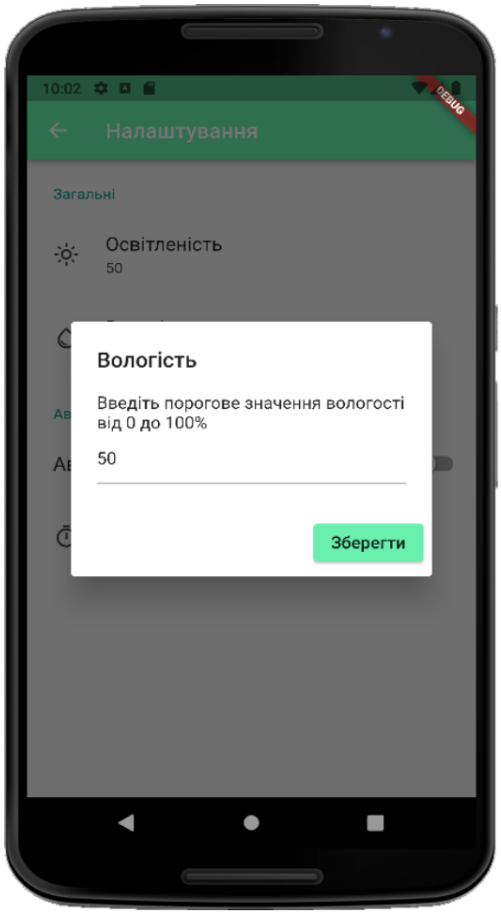
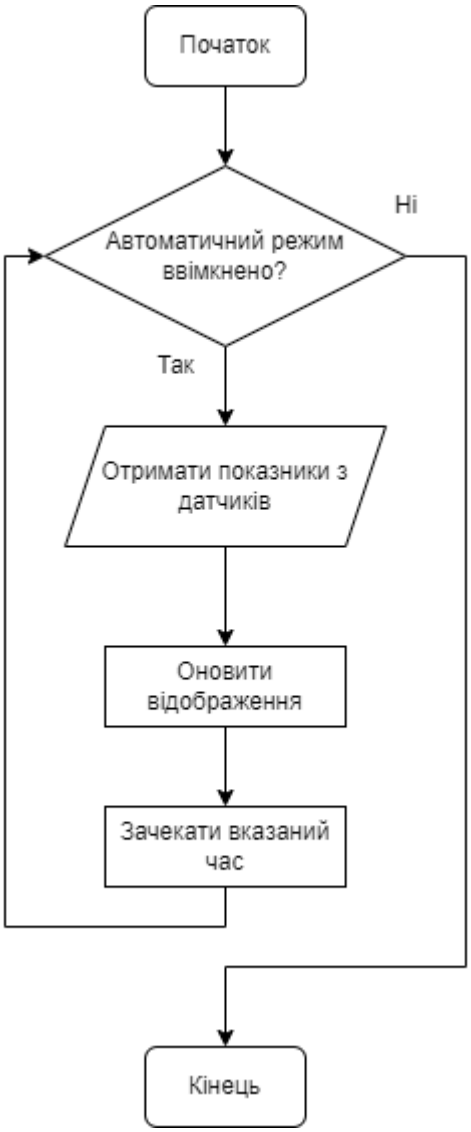
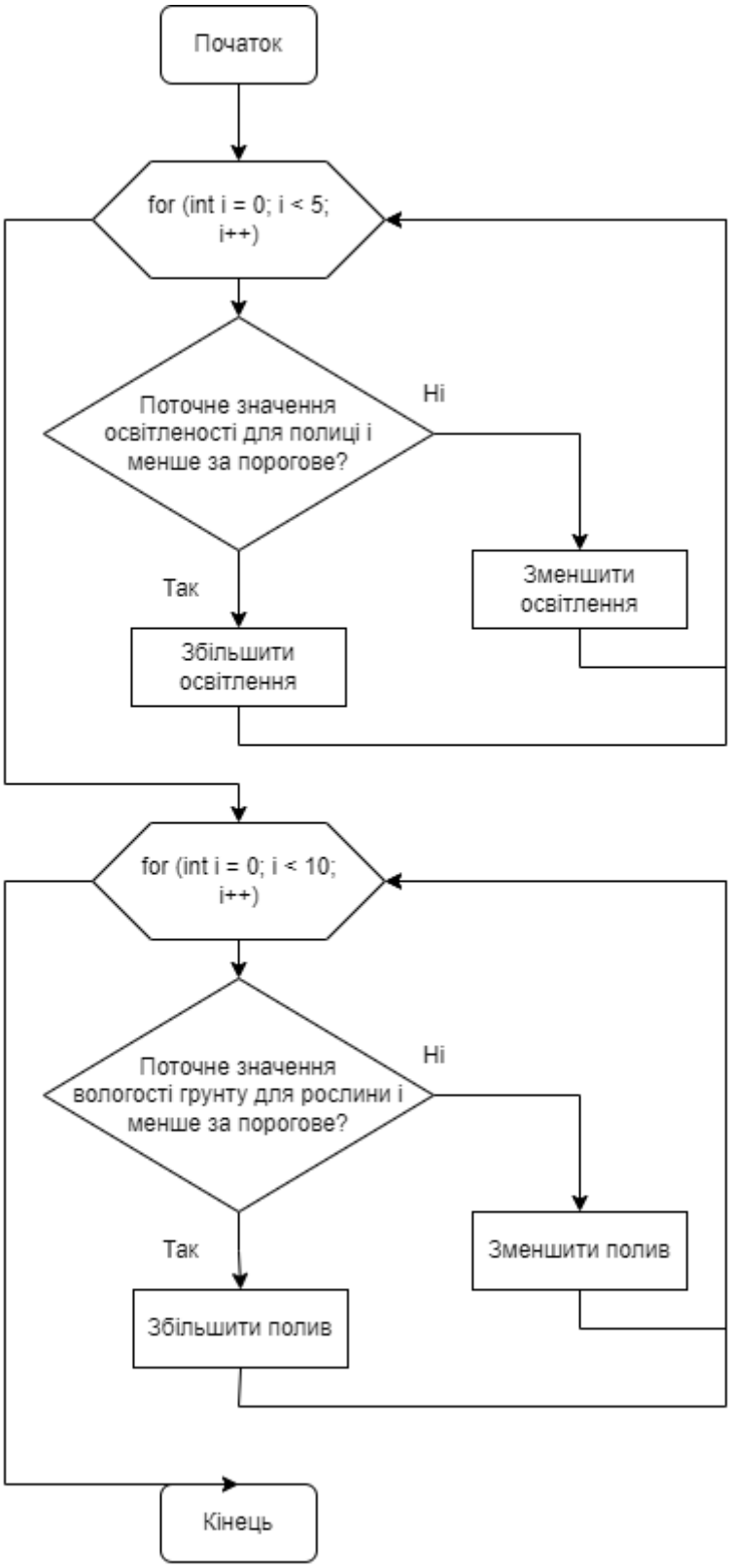


Рисунок 3.8 – Діалогове вікно

3.4 Опис використаних алгоритмів

Логіка застосунку побудована навколо його функціонального призначення – отримання даних з датчиків та керування елементами стенду. Для цих дій були розроблені відповідні алгоритми. Частково ці дії виконуються користувачем: користувач посилає запит на оновлення даних або користувач посилає команду на керування одним елементом стенду (конкретною лампою або конкретним клапаном насосу). В такому випадку їх алгоритм дуже простий і складається з 1-3 дій, однією з яких є отримання запиту від користувача. Більш складними є алгоритми організації автоматичної роботи додатку – автоматичного оновлення даних та керування декількома елементами стенду одночасно. У цих випадках застосунок має сам приймати рішення щодо виконуваних дій. Для цих випадків були розроблені відповідні блок-схеми, що представляють дії, які виконує застосунок. Вони представлені на рис. 3.9-3.10.





а б

Рисунок 3.9 – Блок-схеми алгоритмів

а – автоматичного оновлення даних

б – оптимізації показників усіх датчиків

Висновки до розділу 3

На основі спроектованих класів, сценаріїв та макетів графічних форм застосунку було розроблено робочу версію застосунку, яка відповідає визначеним функціональним вимогам та виконує встановлене функціональне призначення.

Було описано використані засоби розробки, зокрема:

- інтегроване середовище розробки Android Studio;
- вбудований в Android Studio менеджер віртуальних пристроїв;
- стандартні засоби фреймворку Flutter;
- додаткові бібліотеки для фреймворку Flutter.

Була описана структура проекту та використані алгоритми. Додатково були представлені скріншоти розроблених графічних форм застосунку.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

4.1 Огляд стратегій тестування

Тестування застосовується для визначення відповідності предмета випробування заданим специфікаціям. До завдань тестування не належить визначення причин невідповідності заданим вимогам (специфікаціям). Тестування – один з розділів діагностики [17].

Тестування застосовується в техніці, медицині, психіатрії, освіті для визначення придатності об'єкта тестування для виконання тих чи інших функцій. Якість тестування і достовірність його результатів значною мірою залежить від методів тестування та складу тестів.

Процес тестування включає:

- подачу тестового набору;
- визначення реакції об'єкта тестування на тестовий набір;
- оцінку реакції і висновки.

Тестовий набір складається з окремих тестів і розробляється таким чином, щоб забезпечити повне або значне покриття множини ймовірних впливів на об'єкт тестування. Цим, також, визначається складність розробки як окремих тестів, так і тестових наборів.

Тестування програмного забезпечення – процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Класифікація тестувань програмного забезпечення за ознаками:

- за ступенем автоматизації:
  - ручне тестування (manual testing);
  - автоматизоване тестування (automated testing);
  - напівавтоматизоване тестування (semiautomated testing);
- за знанням системи:
  - тестування чорної скриньки (black box);

- тестування білої скриньки (white box);
- тестування сірої скриньки (grey box);

– за ступенем ізольованості:

- тестування компонентів;
- інтеграційне тестування;
- системне тестування.

Тестування методом «чорного ящика», також відоме як тестування, засноване на специфікації або тестування поведінки – техніка тестування, заснована на роботі виключно з зовнішніми інтерфейсами тестованої системи. Таким чином, тестувальник не має уявлення про структуру та внутрішній пристрій системи. Цей тип тестування концентрується на тому, що програма робить, а не на те, як вона це робить [19].

Тестування методом білого ящика – метод тестування програмного забезпечення, який передбачає, що внутрішня структура/пристрій/реалізація системи відомі тестувальнику. Він обирає вхідні значення, ґрунтуючись на знанні коду, який буде їх обробляти. Точно так само він знає, яким повинен бути результат цієї обробки. Знання всіх особливостей тестованої програми та її реалізації – обов’язкові для цієї техніки. Тестування білого ящика – поглиблення у код системи, за межі її зовнішніх інтерфейсів [19].

4.2 Опис розроблених тестів

Більшість функцій логіки програми виконуються лінійно, оскільки у них немає умовних розгалужень, тому для них тестування методами білої та чорної скриньки є скриньки є недоцільними. Серед цих функцій зокрема функції отримання показників з датчиків та функції керування елементами стенду. Для перевірки таких функцій необхідно виконати інтеграційне тестування. Перевірку коректності виконання функцій відображення графічних компонентів теж складно автоматизувати, тому для них краще виконати ручне тестування [20].

Для тестування методами білої та чорної скриньки було обрано функції validator класу TextFormField, що відповідають за валідацію введеного у текстове поле значення. Специфікація функцій представлена у таблиці 4.1.

Таблиця 4.1 – Специфікація тестованих функцій

Опис призначення функції	Отримує введене у тестове поле значення та повертає повідомлення про помилку, якщо значення є некоректним, або null.
Вхідні дані у явному вигляді	String value – введене значення у текстове поле.
Вхідні дані у неявному вигляді	Функція не приймає значень у неявному вигляді.
Вихідні дані	Об’єкт типу String з повідомленням про помилку, або null.

Всього у програмі наявні 3 такі функції: для валідації значення освітлення, значення вологості ґрунту та часу оновлення даних у автоматичному режимі. Вони передаються як атрибути класів TextFormField та не мають назв, тому для зручності у цьому розділі вони будуть називатись validateLuminosity(), validateSoilHumidity() та validateTime() відповідно.

Код функції validateLuminosity() представлено далі:

```
var i = int.tryParse(value);
if (i == null) return "Введіть нове значення";
if (i < 0) return "Значення має бути додатнім числом";
if (i > 2000)
  return "Значення не може бути більше 2000";
return null;
```

Для цієї функції було розроблено наступні тести:

1. Вхід: value – 1000. Вихід: null.
2. Вхід: value – -1000. Вихід: "Значення має бути додатнім числом".
3. Вхід: value – 3000. Вихід: "Значення не може бути більше 2000".
4. Вхід: value – «». Вихід: "Введіть нове значення".

У таблиці 4.2 представлено результат тестування білої скриньки методом покриття операторів для функції validateLuminosity().

Таблиця 4.2 – Метод покриття операторів для функції validateLuminosity()

Таблиця покриття операторів							
Тест	var i = int.tryParse(value);	if (i == null)	return "Введіть нове значення ";	return "Значення має бути додатнім числом";	if (i > 2000)	return "Значення не може бути більше 2000";	return null;
1	+	+	-	-	+	-	+
2	+	+	-	+	-	-	-
3	+	+	-	-	+	+	-
4	+	+	+	-	-	-	-

У таблиці 4.3 представлено результат тестування білої скриньки методом покриття рішень і умов для функції validateLuminosity().

Таблиця 4.3 – Метод покриття рішень і умов для функції validateLuminosity()

Таблиця покриття рішень і умов			
Тест	i == null	i < 0	i > 2000
1	-	-	-
2	-	+	-
3	-	-	+
4	+	-	-

У таблиці 4.4 представлено виділені класи еквівалентності для функції validateLuminosity().

Метод граничних умов для даної функції є недоцільним, оскільки вона приймає лише один параметр у форматі String, який сам по собі обмежень не має.

Таблиця 4.4 – Класи еквівалентності для функції validateLuminosity()

Класи еквівалентності		
Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
String value	Рядок, що може бути конвертований у ціле число діапазоном від 0 до 2000 (1)	Рядок, що може бути конвертований у ціле число менше 0 (2)
		Рядок, що може бути конвертований у ціле число більше 2000 (3)
		Рядок, що не може бути конвертований у ціле число (4)

Виконаємо тестування методом припущення про помилку. Висунемо ряд припущень:

1. Якщо користувач введе рядок, що не може бути конвертований у ціле число (рядок, що містить літери, спеціальні символи), функція поверне «Введіть нове значення».
2. Якщо користувач введе рядок, що конвертується у ціле число менше 0, то функція поверне «Значення має бути додатнім числом».
3. Якщо користувач введе рядок, що конвертується у ціле число більше 2000, то функція поверне «Значення не може бути більше 2000».

У таблиці 4.5 представлено результат тестування чорної скриньки методом причин та наслідків для функції validateLuminosity().

Таблиця 4.5 – Метод причин та наслідків для функції validateLuminosity()

Причини та наслідки						
Причини та наслідки	Умовне позначення	Значення	Помітки про присутність			
			1	2	3	4
Причини	п1	Рядок value може бути конвертований у ціле число діапазоном від 0 до 2000	*			
	п2	Рядок value може бути конвертований у ціле число менше 0		*		
	п3	Рядок value може бути конвертований у ціле число більше 2000			*	
	п4	Рядок value не може бути конвертований у ціле число				*
Наслідки	н1	Функція повертає null	*			
	н2	Функція повертає «Значення має бути додатнім числом»		*		
	н3	Функція повертає «Значення не може бути більше 2000»			*	
	н4	Функція повертає «Введіть нове значення»				*

Розроблена функціональна діаграма представлена на рис. 4.1.

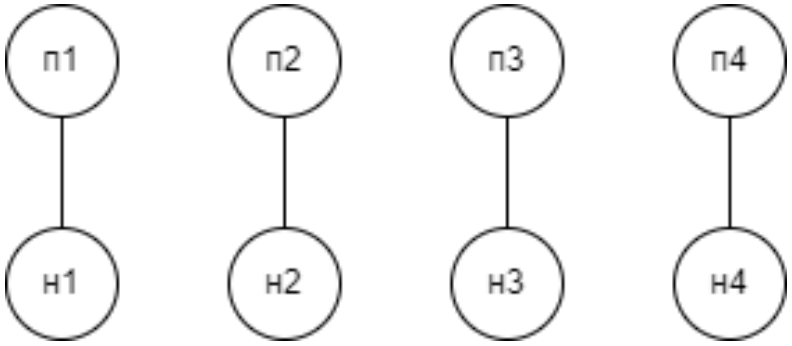


Рисунок 4.1 – Функціональна діаграма для функції validateLuminosity()

Усі визначені класи еквівалентності, припущення, причини та наслідки покриваються розробленими тестами.

Код функції validateSoilHumidity() представлено далі:

```
var i = int.tryParse(value);
if (i == null) return "Введіть нове значення";
if (i < 0) return "Значення має бути додатнім числом";
if (i > 100) return "Значення не може бути більше 100";
return null;
```

Для цієї функції було розроблено наступні тести:

1. Вхід: value – 50. Вихід: null.
2. Вхід: value – -50. Вихід: "Значення має бути додатнім числом".
3. Вхід: value – 300. Вихід: "Значення не може бути більше 100".
4. Вхід: value – «». Вихід: "Введіть нове значення".

У таблиці 4.6 представлено результат тестування білої скриньки методом покриття операторів для функції validateSoilHumidity().

Таблиця 4.6 – Метод покриття операторів для функції validateSoilHumidity()

Таблиця покриття операторів							
Тест	var i = int.tryParse(value);	if (i == null)	return "Введіть нове значення";	return "Значення має бути додатнім числом";	if (i > 100)	return "Значення не може бути більше 100";	return null;
1	+	+	-	-	+	-	+
2	+	+	-	+	-	-	-
3	+	+	-	-	+	+	-
4	+	+	+	-	-	-	-

У таблиці 4.7 представлено результат тестування білої скриньки методом покриття рішень і умов для функції validateSoilHumidity().

Таблиця 4.7 – Метод покриття рішень і умов для функції validateSoilHumidity()

Таблиця покриття рішень і умов			
Тест	i == null	i < 0	i > 100
1	-	-	-
2	-	+	-
3	-	-	+
4	+	-	-

У таблиці 4.8 представлено виділені класи еквівалентності для функції validateSoilHumidity().

Таблиця 4.8 – Класи еквівалентності для функції validateSoilHumidity()

Класи еквівалентності		
Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
String value	Рядок, що може бути конвертований у ціле число діапазоном від 0 до 100 (1)	Рядок, що може бути конвертований у ціле число менше 0 (2)
		Рядок, що може бути конвертований у ціле число більше 100 (3)
		Рядок, що не може бути конвертований у ціле число (4)

Метод граничних умов для даної функції є недоцільним, оскільки вона приймає лише один параметр у форматі String, який сам по собі обмежень не має.

Виконаємо тестування методом припущення про помилку. Висунемо ряд припущень:

1. Якщо користувач введе рядок, що не може бути конвертований у ціле число (рядок, що містить літери, спеціальні символи), функція поверне «Введіть нове значення».
2. Якщо користувач введе рядок, що конвертується у ціле число менше 0, то функція поверне «Значення має бути додатнім числом».
3. Якщо користувач введе рядок, що конвертується у ціле число більше 100, то функція поверне «Значення не може бути більше 100».

У таблиці 4.9 представлено результат тестування чорної скриньки методом причин та наслідків для функції validateSoilHumidity().

Таблиця 4.9 – Метод причин та наслідків для функції validateSoilHumidity()

Причини та наслідки						
Причини та наслідки	Умовне позначення	Значення	Помітки про присутність			
			1	2	3	4
Причини	п1	Рядок value може бути конвертований у ціле число діапазоном від 0 до 100	*			
	п2	Рядок value може бути конвертований у ціле число менше 0		*		
	п3	Рядок value може бути конвертований у ціле число більше 100			*	
	п4	Рядок value не може бути конвертований у ціле число				*
Наслідки	н1	Функція повертає null	*			
	н2	Функція повертає «Значення має бути додатнім числом»		*		
	н3	Функція повертає «Значення не може бути більше 100»			*	
	н4	Функція повертає «Введіть нове значення»				*

Розроблена функціональна діаграма представлена на рис. 4.2.

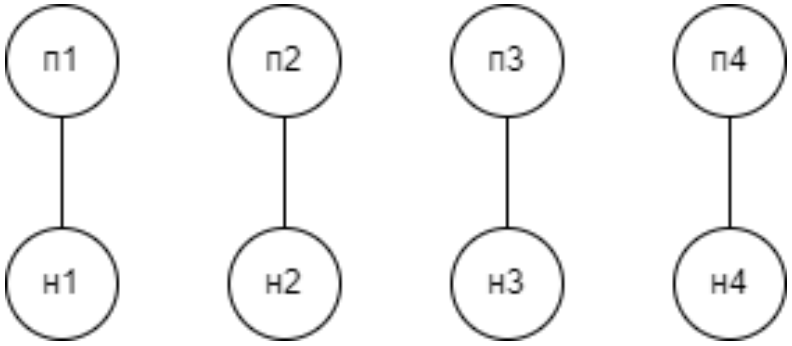


Рисунок 4.2 – Функціональна діаграма для функції validateSoilHumidity()

Усі визначені класи еквівалентності, припущення, причини та наслідки покриваються розробленими тестами.

Код функції validateTime() представлено далі:

```
var i = int.tryParse(value);
if (i == null) return "Введіть нове значення";
if (i < 0) return "Значення має бути додатнім числом";
if (i > 604800) return "Значення не може бути більше 604800
с (1 тиждень)";
return null;
```

Для цієї функції було розроблено наступні тести:

1. Вхід: value – 2. Вихід: null.
2. Вхід: value – -2. Вихід: "Значення має бути додатнім числом".
3. Вхід: value – 604801. Вихід: "Значення не може бути більше 604800".
4. Вхід: value – «». Вихід: "Введіть нове значення".

У таблиці 4.10 представлено результат тестування білої скриньки методом покриття операторів для функції validateTime().

Таблиця 4.10 – Метод покриття операторів для функції validateTime()

Таблиця покриття операторів							
Тест	var i = int.tryParse(value);	if (i == null)	return "Введіть нове значення";	return "Значення має бути додатнім числом";	if (i > 604800)	return "Значення не може бути більше 604800 (1 тиждень)";	return null;
1	+	+	-	-	+	-	+
2	+	+	-	+	-	-	-
3	+	+	-	-	+	+	-
4	+	+	+	-	-	-	-

У таблиці 4.11 представлено результат тестування білої скриньки методом покриття рішень і умов для функції validateTime().

Таблиця 4.11 – Метод покриття рішень і умов для функції validateTime()

Таблиця покриття рішень і умов			
Тест	i == null	i < 0	i > 604800
1	-	-	-
2	-	+	-
3	-	-	+
4	+	-	-

У таблиці 4.12 представлено виділені класи еквівалентності для функції validateSoilHumidity().

Таблиця 4.12 – Класи еквівалентності для функції validateTime()

Класи еквівалентності		
Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
String value	Рядок, що може бути конвертований у ціле число діапазоном від 0 до 604800 (1)	Рядок, що може бути конвертований у ціле число менше 0 (2)
		Рядок, що може бути конвертований у ціле число більше 604800 (3)
		Рядок, що не може бути конвертований у ціле число (4)

Метод граничних умов для даної функції є недоцільним, оскільки вона приймає лише один параметр у форматі String, який сам по собі обмежень не має.

Виконаємо тестування методом припущення про помилку. Висунемо ряд припущень:

1. Якщо користувач введе рядок, що не може бути конвертований у ціле число (рядок, що містить літери, спеціальні символи), функція поверне «Введіть нове значення».
2. Якщо користувач введе рядок, що конвертується у ціле число менше 0, то функція поверне «Значення має бути додатнім числом».
3. Якщо користувач введе рядок, що конвертується у ціле число більше 604800, то функція поверне «Значення не може бути більше 604800 (1 тиждень)».

У таблиці 4.13 представлено результат тестування чорної скриньки методом причин та наслідків для функції validateTime().

Таблиця 4.13 – Метод причин та наслідків для функції validateTime()

Причини та наслідки						
Причини та наслідки	Умовне позначення	Значення	Помітки про присутність			
			1	2	3	4
Причини	п1	Рядок value може бути конвертований у ціле число діапазоном від 0 до 604800	*			
	п2	Рядок value може бути конвертований у ціле число менше 0		*		
	п3	Рядок value може бути конвертований у ціле число більше 604800			*	
	п4	Рядок value не може бути конвертований у ціле число				*
Наслідки	н1	Функція повертає null	*			
	н2	Функція повертає «Значення має бути додатнім числом»		*		
	н3	Функція повертає «Значення не може бути більше 604800 (1 тиждень)»			*	
	н4	Функція повертає «Введіть нове значення»				*

Розроблена функціональна діаграма представлена на рис. 4.3.

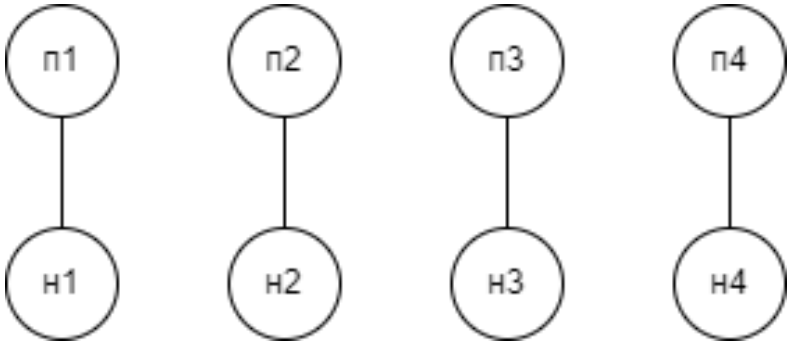


Рисунок 4.3 – Функціональна діаграма для функції validateTime()

Усі визначені класи еквівалентності, припущення, причини та наслідки покриваються розробленими тестами.

4.3 Результат виконання тестування та аналіз виявлених помилок

Розроблені тести покривають усі можливі умови та результати роботи функцій. Згідно до них були розроблені та виконані відповідні unit-тести. Додатково було проведено ручне тестування за допомогою вбудованого в Android studio менеджеру віртуальних пристроїв. Виявлені помилки були виправлені. Робота застосунку відповідає зазначеним вимогам.

Висновки до розділу 4

Для тестування методами білої та чорної скриньки були обрані 3 функції:

- валідації значення освітленості;
- валідації значення вологості ґрунту;
- валідації часу оновлення.

Для розроблення найбільш ефективних тестів були використані наступні методи:

- покриття операторів;
- покриття умов і рішень;
- класів еквівалентності;
- припущення про помилку;
- функціональних діаграм.

Розроблені тести дозволили покрити усі можливі умови та результати виконання функцій. Усі знайдені помилки були виправлені.

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було розроблено мобільний застосунок для навчально-діючого STEM-стенду. Застосунок дозволяє переглядати поточні дані з датчиків на стенді, встановлювати порогові значення освітленості та вологості ґрунту, а також керувати елементами стенду, зокрема освітленням та поливом. Застосунок розроблено мовою Dart з використанням фреймворку Flutter для операційних систем Android та iOS.

Для визначення функціональних вимог було розглянуто існуючі аналоги, проведено презентації проекту та опитування серед цільової аудиторії.

На етапі проектування були визначені функціональні вимоги, обрано мову програмування, розроблено діаграми класів, станів користувача, послідовностей для прецедентів «Ручне оновлення даних», «Оптимізація усіх показників», «Введення нового порогового значення вологості ґрунту», діяльності та станів для прецеденту «Навігація між представленнями застосунку», артефактів. Також представлено макети екранних форм застосунку.

Для розробки застосунку було використано інтегроване середовище розробки Android Studio. Зокрема було розроблено 2 екранні форми та 15 класів. Структуру проекту, використані програмні компоненти фреймворку Flutter та скріншоти розроблених екранних форм представлено у 3 розділі.

Було виконано тестування функцій валідації нових введених значень освітленості, вологості ґрунту та часу оновлення методами білої та чорної скриньок. Для перевірки коректності роботи графічного інтерфейсу застосунку було виконано ручне тестування. Усі виявлені помилки були виправлені.

Список літератури

1. Горячкін, В. М. Навчальний посібник для підготовки кваліфікаційної роботи на здобуття ОС Бакалавр [Текст] / В. М. Горячкін, О. В. Горбова, О. С. Куроп'ятник. – Дніпро : Український державний університет науки і технологій, 2022. – 137с.
2. Розпорядження : офіц. текст : [Про схвалення Концепції розвитку природничо-математичної освіти (STEM-освіти) від 5 серпня 2020 р. № 960-р]. – Київ : Кабінет міністрів України, 2020. – 11 с.
3. Home – eWeLink [Електронний ресурс] / Режим доступу : URL : https://ewelink.cc/. – Дата звернення: 2 травня 2022.
4. AquaPot E-mode 4 [Електронний ресурс] / Режим доступу : URL : https://e-mode.pro/catalog/sistemy/aquapot_e_mode_4/. – Дата звернення: 2 травня 2022.
5. Ray, L. How to Construct a Follow Up Document for an Implementation Project [Електронний ресурс] / Linda Ray // Small Business. Режим доступу : URL : https://smallbusiness.chron.com/convert-boardmaker-pdf-58196.html/. – Дата звернення: 18 травня 2022.
6. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language 3rd Edition [Текст] / Martin Fowler. – Boston : Addison-Wesley Professional; 3rd edition (September 15, 2003). – 208 с.
7. Языки программирования: что это такое, зачем нужны и какой выбрать новичку [Електронний ресурс] / Режим доступу : URL : https://skillbox.ru/media/code/yazyki_programmirovaniya_chto_eto_takoe/. – Дата звернення: 18 травня 2022.
8. Eckel, B. Atomic Kotlin [Текст] / Bruce Eckel, Svetlana Isakova. – Boston : Mindview LLC, 2021. – 636 с.
9. Основные языки программирования для разработки мобильных приложений [Електронний ресурс] / Режим доступу : URL : https://appcraft.pro/blog/yazyki-dlya-razrabotki-mobilnykh-prilozhenij/. – Дата звернення: 18 травня 2022.

10. Усов, В. А. Swift. Основы разработки приложений под iOS, iPadOS и macOS. 6-е изд. дополненное и переработанное [Текст] / В. А. Усов. – Москва : Прогресс книга, 2021. – 544 с.
11. Блох, Д. Java. Эффективное программирование [Текст] / Джошуа Блох. – Москва : Лори, 2014. – 310 с.
12. Kochan, S. G. Programming in Objective-C [Текст] / Stephen G. Kochan. – Boston : Addison-Wesley Professional, 2012. – 544 с.
13. Bracha, G. The Dart Programming Language [Текст] / Gilad Bracha. – Boston : Addison-Wesley Professional, 2015. – 224 с.
14. Флэнаган, Д. JavaScript. Полное руководство [Текст] / Флэнаган Дэвид. – Boston : Диалектика-Вильямс, 2021. – 720 с.
15. Об'єктно-орієнтоване програмування [Електронний ресурс] / Режим доступу : URL : https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F/. – Дата звернення: 29 травня 2022.
16. Android Studio [Електронний ресурс] / Режим доступу : URL : https://uk.wikipedia.org/wiki/Android_Studio/. – Дата звернення: 29 травня 2022.
17. Тестування [Електронний ресурс] / Режим доступу : URL : https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F/. – Дата звернення: 29 травня 2022.
18. Тестирование программного обеспечения [Електронний ресурс] / Режим доступу : URL : https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B

5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F/. – Дата звернення: 03 червня 2022.

19. White/black/grey box-тестування [Електронний ресурс] / Режим доступу : URL : https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/. – Дата звернення: 03 червня 2022.
20. Ручне та автоматизоване тестування [Електронний ресурс] / Режим доступу : URL : https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya/. – Дата звернення: 03 червня 2022.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського державного університету науки і технологій
Анатолій РАДКЕВИЧ
18.02.22

ВЧАЛЬНО-ДІЮЧИЙ СТЕНД «ЕКОКЛІМАТИЧНА СТАНЦІЯ З ВИРОЩУВАННЯ БІОРІЗНОМАНІТТЯ» INDOOR FOREST

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.94115-01-ЛЗ

Представники підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
18.02.22

Керівник розробки
Олександр ІВАНОВ
18.02.22

Виконавець
Марина СТЕШЕНКО
18.02.22

Нормоконтролер
Олена КУРОП'ЯТНИК
18.02.22

2022

ЗАТВЕРДЖЕНО
44165850.94115-01-ЛЗ

НАВЧАЛЬНО-ДІЮЧИЙ СТЕНД «ЕКОКЛІМАТИЧНА СТАНЦІЯ З ВИРОЩУВАННЯ БІОРІЗНОМАНІТТЯ» INDOOR FOREST

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01237-01
Листів 12

2022

ЗМІСТ

Вступ…..... 4
2. Підстави для розробки..... 5
3. Призначення розробки..... 6
4. Вимоги до програмного продукту..... 7
4.1. Вимоги до функціональних характеристик..... 7
4.2. Вимоги до надійності 8
4.3. Вимоги експлуатації..... 8
4.4. Вимоги до складу та параметрів технічних засобів..... 9
4.5. Вимоги до інформаційної та програмної сумісності..... 9
4.6. Вимоги до маркування і упаковки..... 9
4.7. Вимоги до транспортування та зберігання..... 9
5. Вимоги до програмної документації..... 10
6. Стадії та етапи розробки..... 11
7. Порядок контролю приймання 12

ВСТУП

Сучасний світ важко уявити без інновацій та технологій, яким знайшли застосування у багатьох сферах людської діяльності. Задля покращення рівня життя та забезпечення конкурентоспроможності нашої держави необхідно враховувати ці зміни та відповідно адаптуватися до них. Саме тому важливими факторами розвитку економіки є наукоємні та високотехнологічні галузі. На жаль, зараз спостерігається втрата популярності предметів природничої, технологічної та математичної освітніх галузей. Перед сферою освіти зараз постає завдання розвитку цих галузей та заохочення учнів та студентів до їх вивчення. Одним з пріоритетів розвитку сфери освіти зараз є природничо-математична освіта STEM.

Природничо-математична освіта (STEM-освіта) – цілісна система природничої і математичної освітніх галузей, метою якої є розвиток особистості через формування компетентностей, природничо-наукової картини світу, світоглядних позицій і життєвих цінностей з використанням трансдисциплінарного підходу до навчання, що базується на практичному застосуванні наукових, математичних, технічних та інженерних знань для розв’язання практичних проблем для подальшого використання цих знань і вмінь у професійній діяльності.

STEM-лабораторія – навчальний кабінет або приміщення закладу освіти, оснащене сучасними засобами навчання та обладнанням для залучення здобувачів освіти до навчально-дослідницької, дослідницько-експериментальної, конструкторської, винахідницької та пошукової діяльності відповідно до стандартів освіти, освітніх та навчальних програм з використанням проектних технологій в освітньому процесі;

STEM-центр – структурний підрозділ закладу освіти, утворений з метою забезпечення природничо-математичної освіти (STEM-освіти), організації та взаємодії заінтересованих осіб.

Для забезпечення науково-методичної підтримки природничо-математичної освіти (STEM-освіти) важливе значення має розроблення інтегрованих навчальних програм для всіх типів закладів освіти для викладання спеціальних курсів, факультативів, організації роботи гуртків з робототехніки, інженерії, природничих та аграрних дисциплін, сучасних наукових напрямів, новітніх технологій з урахуванням кращого національного та міжнародного досвіду.

Метою проекту є створення навчально-діючого STEM-стенду для використання на уроках фізики, біології, хімії, екології та залучення здобувачів освіти до дослідницько-експериментальної, конструкторської діяльності.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 08.12.21 №77ст ректора Українського державного університету науки і технологій «Про призначення наукових керівників та затвердження тем бакалаврських робіт» за спеціальністю 121 «Інженерія програмного забезпечення» факультету «Комп’ютерних технологій і систем» по кафедрі «Комп’ютерні інформаційні технології».

Тема дипломної роботи – «Навчально-діючий стенд «Екокліматична станція з вирощування біорізноманіття» INDOOR FOREST». Керівник – доцент Іванов О. П.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – програмний продукт є мобільним додатком до стенду з системою поливу та освітлення для 10 рослин. Він має отримувати дані з датчиків та керувати елементами системи поливу та освітлення згідно отриманих даних. Також має надавати користувачу можливість переглядати поточні дані та встановлювати порогові значення.

Експлуатаційне призначення – стенд та додаток до нього виконують навчально-демонстраційну роль на дисциплінах фізики, хімії, біології та екології. Вони використовуються для вивчення рослин, їхнього життєвого циклу, будови, особливостей умов їх зберігання, застосування сучасних технологій для догляду за ними, принципу роботи датчиків, клапанів, сонячних батарей.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

4.1. Вимоги до функціональних характеристик

Програмний продукт повинен забезпечувати можливість встановити та змінювати порогові значення даних з датчиків. До них відносяться:

- порогове значення вологості ґрунту – встановлюється в відсотках (%), діапазон значень від 0 до 100;
- порогове значення освітленості – встановлюється в люксах (лк), діапазон значень від 0 до 2000.

Програмний продукт має отримувати дані з датчиків та надавати можливість їх перегляду. До датчиків, з якими буде працювати додаток, відносяться:

- датчик вологості ґрунту (вимірюється в відсотках, %);
- датчик освітленості (вимірюється в люксах, лк);
- датчик температури (вимірюється в градусах Цельсія, °С), тиску (вимірюється в Паскалях, Ра) та вологості повітря (вимірюється в відсотках, %).

Згідно отриманих даних програмний продукт має виконувати керування елементами системи поливу та освітлення. Рішення щодо виконуваних команд продукт приймає базуючись на наступних правилах:

1. Якщо фактичне значення вологості ґрунту менше за встановлене порогове, то вмикається полив до збільшення значення до необхідного.
2. Якщо фактичне значення освітленості менше за встановлене порогове, то збільшується освітлення до необхідного.
3. Якщо фактичне значення освітленості більше за встановлене порогове, то зменшується освітлення до необхідного.

Програмний продукт має отримувати дані з датчиків із заданою періодичністю:

- під час вимкненого поливу дані про полив отримуються раз у 1 годину;
- під час ввімкненого поливу дані про полив отримуються раз у 1 хвилину;
- дані про освітлення отримуються раз у 1 годину.

Вимоги до вхідних даних

Програмний додаток через зазначені проміжки часу отримує наступні дані з датчиків:

- вологість ґрунту, %;
- освітленість, лк;
- температуру повітря, °С;
- повітряний тиск, Ра;
- вологість повітря, %.

Також вхідними даними є порогові значення:

- вологості ґрунту, %;
- освітленості, лк.

Вимоги до вихідних даних

Вихідними даними є дані з датчиків, які оновлюються з заданою періодичністю:

- вологість ґрунту, %;
- освітленість, лк;
- температуру повітря, °С;
- повітряний тиск, Ра;
- вологість повітря, %.

Команди керування елементами стенду:

- ввімкнення поливу;
- вимкнення поливу;
- збільшення освітлення;
- зменшення освітлення.

4.2. Вимоги до надійності

Вимоги до надійності наступні:

- при оновленні даних забезпечити показ відповідних повідомлень про стан роботи програми та результати виконання операцій;
- наявність архівної копії тексту програми на зовнішньому носії;
- наявність резервної копії бази даних на зовнішньому носії.

4.3. Вимоги експлуатації

Програмний продукт повинен використовуватись у приміщеннях які відповідають умовам роботи ЕОМ, а саме мають такі кліматичні, санітарні та гігієнічні умови, які відповідають ДНАОП 0.00-1.13-99 (див. табл. 3.1).

Таблиця 3.1. Кліматичні умови

Пора року	Категорія робіт згідно з ГОСТ 12.01-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		Оптимальна	Оптимальна	Оптимальна
Холодна	легка-1-а	22-24	40-60	0,1
	легка-1-б	21-23	40-60	0,1
Тепла	легка-1-а	23-25	40-60	0,1
	легка-1-б	22-24	40-60	0,2

Працювати з програмою може людина, що має навички роботи з мобільними пристроями та ознайомлена з керівництвом користувача програмного продукту.

4.4. Вимоги до складу та параметрів технічних засобів

Продукт, що розробляється повинен використовуватись на мобільних пристроях, що мають наступні характеристики:

- діагональ екрану – 5.5;
- роздільна здатність дисплею – HD (1280x720);
- оперативна пам’ять – 2 ГБ;
- вбудована пам’ять – 16 ГБ;
- операційна система – ANDROID;
- частота процесора – 1.6 ГГц;
- комунікації – microUSB.

4.5. Вимоги до інформаційної та програмної сумісності

Програмний продукт розробляється для всіх видів операційних систем сімейства «ANDROID» починаючи від версії 6 та наступні версії.

4.6. Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

На упаковці повинно бути вказана назва продукту, номер версії, мінімальні системні вимоги.

На зворотній стороні упаковки вказується розробник та його юридична адреса.

4.7. Вимоги до транспортування та зберігання

Транспортування повинне забезпечувати збереження програмного продукту його цілісність і запобігання несанкціонованого доступу до нього. Програмний виріб міститься на фізичному носії та переданий через microUSB порт.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- специфікація;
- текст програми;
- опис програми;
- керівництво користувача з користуванням стендом.

Вся документація програмного додатку повинна задовольняти вимоги до програмної документації.

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиці 6.1. Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	31.01.22 -18.02.22
Робочий проект	Програмування та відлагодження програми.	19.02.22 - 20.05.22
	Тестування програми	20.05.22 - 27.05.22
	Розробка, узгодження і затвердження програмної документації.	27.05.22 - 12.06.22

7. ПОРЯДОК КОНТРОЛЮ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Іванов О. П.

Додаток Б

ЗАТВЕРДЖЕНО

44165850.01237-01 12-01 ЛЗ

НАВЧАЛЬНО-ДІЮЧИЙ СТЕНД "ЕКОКЛІМАТИЧНА СТАНЦІЯ З ВИРОЩУВАННЯ БІОРІЗНОМАНІТТЯ" INDOOR FOREST

Текст програми

44165850.01237-01 12-01 ЛЗ

Листів 17

2022

АНОТАЦІЯ

Документ 44165850.01237-01 12-01 «Навчально-діючий стенд "Екокліматична станція з вирощування біорізноманіття" INDOOR FOREST. Текст програми» входить до складу програмної документації програми, що є мобільним додатком до стенду з системою поливу та освітлення для 10 рослин та має отримувати дані з датчиків, керувати елементами системи поливу та освітлення, а також має надавати користувачу можливість переглядати поточні дані та встановлювати порогові значення.

У даному документі представлений вихідний текст програми. Програма написана мовою Dart з використанням фреймворку Flutter у інтегрованому середовищі розробки Android Studio.

ЗМІСТ

1. Зміст файлу CurrentView.dart. 4

2. Зміст файлу RandomStatisticsGetter.dart..... 4

3. Зміст файлу SettingsData.dart..... 4

4. Зміст файлу Statistics.dart..... 5

5. Зміст файлу StatisticsData.dart..... 6

6. Зміст файлу StatisticsGetter.dart..... 6

7. Зміст файлу TimerFunctions.dart..... 6

8. Зміст файлу M.dart..... 7

9. Зміст файлу NavDrawer.dart..... 8

10. Зміст файлу SettingsForm.dart..... 10

11. Зміст файлу TreeWidget.dart..... 11

12. Зміст файлу main.dart..... 13

1. Зміст файлу CurrentView.dart

```
enum CurrentView { GENERAL, SHELF, PLANT }
```

2. Зміст файлу RandomStatisticsGetter.dart

```
import 'dart:math';

import 'package:indoor_forest/classes/StatisticsGetter.dart';

class RandomStatisticsGetter extends StatisticsGetter {
  Random random = Random();

  @override
  int getHumidity() {
    return random.nextInt(100);
  }

  @override
  List<int> getLuminosity() {
    return <int>[
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100)
    ];
  }

  @override
  int getPressure() {
    return random.nextInt(100);
  }

  @override
  List<int> getSoilHumidity() {
    return <int>[
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100),
      random.nextInt(100)
    ];
  }

  @override
  int getTemperature() {
    return random.nextInt(100);
  }
}
```

3. Зміст файлу SettingsData.dart

```
class SettingsData{
  static int luminosityLimit = 50;
  static int humidityLimit = 50;
  static int timer = 2;
}
```

4. Зміст файлу Statistics.dart

```
import 'package:indoor_forest/classes/SettingsData.dart';
import 'package:indoor_forest/classes/StatisticsData.dart';
import 'package:indoor_forest/classes/StatisticsGetter.dart';

class Statistics {
  StatisticsData data;
  StatisticsGetter getter;

  Statistics(this.data, this.getter);

  void update() {
    data.humidity = getter.getHumidity();
    data.temperature = getter.getTemperature();
    data.pressure = getter.getPressure();
    data.luminosity = getter.getLuminosity();
    data.soilHumidity = getter.getSoilHumidity();
  }

  int getTemperature() {
    return data.temperature;
  }

  int getHumidity() {
    return data.humidity;
  }

  int getPressure() {
    return data.pressure;
  }

  int getLuminosity(int shelfIndex){
    return data.luminosity[shelfIndex];
  }

  int getSoilHumidity(int plantIndex){
    return data.soilHumidity[plantIndex];
  }

  void changeLuminosity(int shelfIndex){
    data.luminosity[shelfIndex] = SettingsData.luminosityLimit;
  }

  void changeHumidity(int plantIndex){
    data.soilHumidity[plantIndex] = SettingsData.humidityLimit;
  }

  void optimize(){
    for (int i = 0; i < data.luminosity.length; i++){
      data.luminosity[i] = SettingsData.luminosityLimit;
    }
    for (int i = 0; i < data.soilHumidity.length; i++){
      data.soilHumidity[i] = SettingsData.humidityLimit;
```

```
    }
  }
}
```

5. Зміст файлу StatisticsData.dart

```
class StatisticsData {
  int temperature;
  int pressure;
  int humidity;
  List<int> luminosity;
  List<int> soilHumidity;

  StatisticsData(this.temperature, this.pressure, this.humidity, this.luminosity,
      this.soilHumidity);

}
```

6. Зміст файлу StatisticsGetter.dart

```
abstract class StatisticsGetter{
  int getTemperature();
  int getHumidity();
  int getPressure();
  List<int> getLuminosity();
  List<int> getSoilHumidity();
}
```

7. Зміст файлу TimerFunctions.dart

```
import 'dart:async';

import 'package:indoor_forest/classes/SettingsData.dart';

class TimerFunctions {
  bool _autoUpdate = false;
  bool _timerInitialized = false;
  final void Function() updateFunction;
  late Timer updateTimer;

  TimerFunctions(this.updateFunction);

  bool getAutoUpdate() {
    return _autoUpdate;
  }

  void setAutoUpdate(bool value) {
    _autoUpdate = value;
    restart();
  }

  void restart() {
    if (_timerInitialized) {
      cancelTimer();
    }
    startTimer();
  }
```

```
  void startTimer() {
    _timerInitialized = true;
    updateTimer =
        Timer.periodic(Duration(seconds: SettingsData.timer), (timer) {
      if (!_autoUpdate) {
        updateTimer.cancel();
      } else {
        updateFunction.call();
      }
    });
  }

  void cancelTimer() {
    updateTimer.cancel();
  }
}
```

8. Зміст файлу M.dart

```
import 'package:flutter/material.dart';

class CustomStyles {
  static Color mainColor = Colors.greenAccent;

  static TextStyle simpleText() {
    return TextStyle(
      color: Colors.black87,
      fontSize: 16,
    );
  }

  static TextStyle simpleTextWhite() {
    return TextStyle(
      color: Colors.white,
      fontSize: 16,
    );
  }

  static TextStyle headerText() {
    return TextStyle(
      color: Colors.white,
      fontSize: 24,
    );
  }

  static TextStyle headerTextBlack() {
    return TextStyle(
      color: Colors.black87,
      fontSize: 24,
    );
  }

  static TextStyle subText() {
    return TextStyle(
      color: Colors.teal,
      fontSize: 14,
    );
  }
```

```
  static simpleButtonStyle() {
    return ElevatedButton.styleFrom(
      primary: Colors.greenAccent,
    );
  }

  static simpleCircleButtonStyle() {
    return ElevatedButton.styleFrom(
      primary: Colors.greenAccent,
        shape: const CircleBorder(),
        padding: const EdgeInsets.all(20)
    );
  }
}
```

9. Зміст файлу NavDrawer.dart

```
import 'package:flutter/material.dart';
import 'package:indoor_forest/widgets/M.dart';

class NavDrawer extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Drawer(
      child: ListView(
        padding: EdgeInsets.zero,
        children: <Widget>[
          DrawerHeader(
            decoration: BoxDecoration(
              color: Colors.greenAccent
            ),
            child: Container(
              alignment: Alignment.bottomLeft,
              child: Text(
                'Indoor forest',
                style: CustomStyles.headerText(),
              ),
            ),
          ),
          ListTile(
            leading: Icon(Icons.settings, color: Colors.black45,),
            title: Text('Налаштування', style: CustomStyles.simpleText(),),
            onTap: () => {
              Navigator.pushNamed(context, "/settings")
            },
          ),
        ],
      ),
    );
  }
}
```

10. Зміст файлу SettingsForm.dart

```
import 'package:flutter/material.dart';
import
'package:indoor_forest/classes/Settings
Data.dart';
import
'package:indoor_forest/classes/TimerFun
```

```
ctions.dart';
import
'package:settings_ui/settings_ui.dart';

import 'M.dart';

class SettingsForm extends
StatefulWidget {
  const SettingsForm({Key? key,
required this.timerFunctions}) :
super(key: key);
  final TimerFunctions timerFunctions;

  @override
  State<SettingsForm> createState() =>
_SettingsFormState(timerFunctions);
}

class _SettingsFormState extends
State<SettingsForm> {

_SettingsFormState(this.timerFunctions)
;

  TimerFunctions timerFunctions;

  final luminosityController =
      TextEditingController(text:
"${SettingsData.luminosityLimit}");
  final humidityController =
      TextEditingController(text:
"${SettingsData.humidityLimit}");
  final timerController =
  TextEditingController(text:
"${SettingsData.timer}");

  final _formLuminosityKey =
GlobalKey<FormState>();
  final _formHumidityKey =
GlobalKey<FormState>();
  final _formTimerKey =
GlobalKey<FormState>();

  Future<void> _showLuminosityDialog()
async {
    return showDialog<void>(
      context: context,
      builder: (BuildContext context) {
        return Form(
          key: _formLuminosityKey,
          child: AlertDialog(
            title: Text('Освітленість',
style: CustomStyles.simpleText()),
            content:
SingleChildScrollView(
              child: ListBody(
                children: <Widget>[
                  Text(
                      'Введіть порогове
значення освітленості від 0 до 2000
Л'),
                  TextFormField(
                    controller:
luminosityController,
                    validator: (value)
{
                      if (value !=
null) {
                        var i =
int.tryParse(value);
                        if (i == null)
return "Введіть нове значення";
                        if (i < 0)
return "Значення має бути додатнім
числом";
                        if (i > 2000)
                          return
"Значення не може бути більше 2000";
                        return null;
                      }
                    },
                  ),
                ],
              ),
            ),
            actions: <Widget>[
              ElevatedButton(
                  style:
ElevatedButton.styleFrom(
                    primary:
Colors.greenAccent,
                  ),
                  onPressed: () {
                    setState(() {
                      if
(_formLuminosityKey.currentState!.valid
ate()) {

SettingsData.luminosityLimit =

int.parse(luminosityController.text);

Navigator.of(context).pop();
                      }
                    });
                  },
                  child:
Text("Зберегти", style:
CustomStyles.simpleText()))
            ],
          ),
        );
      },
    );
  }

  Future<void> _showHumidityDialog()
async {
    return showDialog<void>(
      context: context,
      builder: (BuildContext context) {
        return Form(
```

```
          key: _formHumidityKey,
          child: AlertDialog(
            title: const
Text('Вологість'),
            content:
SingleChildScrollView(
              child: ListBody(
                children: <Widget>[
                  Text('Введіть
порогове значення вологості від 0 до
100%'),
                  TextFormField(
                    validator: (value)
{
                      if (value !=
null) {
                        var i =
int.tryParse(value);
                        if (i == null)
return "Введіть нове значення";
                        if (i < 0)
return "Значення має бути додатнім
числом";
                        if (i > 100)
return "Значення не може бути більше
100";
                        return null;
                      }
                    },
                    controller:
humidityController,
                  ),
                ],
              ),
            ),
            actions: <Widget>[
              ElevatedButton(
                  style:
ElevatedButton.styleFrom(
                    primary:
Colors.greenAccent,
                  ),
                  onPressed: () {
                    setState(() {
                      if
(_formHumidityKey.currentState!.validat
e()) {

SettingsData.humidityLimit =

int.parse(humidityController.text);

Navigator.of(context).pop();
                      }
                    });
                  },
                  child:
Text("Зберегти", style:
CustomStyles.simpleText()))
            ],
          ),
        );
      },
    );
  }

  Future<void> _showTimerDialog() async
{
    return showDialog<void>(
      context: context,
      builder: (BuildContext context) {
        return Form(
          key: _formTimerKey,
          child: AlertDialog(
            title: const Text('Час
оновлення'),
            content:
SingleChildScrollView(
              child: ListBody(
                children: <Widget>[
                  Text('Введіть час
оновлення у секундах'),
                  TextFormField(
                    validator: (value)
{
                      if (value !=
null) {
                        var i =
int.tryParse(value);
                        if (i == null)
return "Введіть нове значення";
                        if (i < 0)
return "Значення має бути додатнім
числом";
                        if (i > 604800)
return "Значення не може бути більше
604800 с (1 тиждень)";
                        return null;
                      }
                    },
                    controller:
timerController,
                  ),
                ],
              ),
            ),
            actions: <Widget>[
              ElevatedButton(
                  style:
ElevatedButton.styleFrom(
                    primary:
Colors.greenAccent,
                  ),
                  onPressed: () {
                    setState(() {
                      if
(_formTimerKey.currentState!.validate()
) {

SettingsData.timer =

int.parse(timerController.text);
```

```
timerFunctions.restart();

Navigator.of(context).pop();
                        }
                      });
                    },
                    child:
Text("Зберегти", style:
CustomStyles.simpleText()))
              ],
            ),
          );
        },
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Налаштування"),
        backgroundColor:
Colors.greenAccent,
      ),
      body: SettingsList(
        sections: [
          SettingsSection(title:
Text('Загальні', style:
CustomStyles.subText(),), tiles:
<SettingsTile>[
            SettingsTile.navigation(
              leading:
Icon(Icons.light_mode_outlined),
              title:
Text('Освітленість'),
              value:
Text("${SettingsData.luminosityLimit}")
,
              onPressed: (context) =>
_showLuminosityDialog(),
            ),
            SettingsTile.navigation(
              leading:
Icon(Icons.water_drop_outlined),
              title: Text('Вологість'),
              value:
Text("${SettingsData.humidityLimit}"),
              onPressed: (context) =>
_showHumidityDialog(),
            )
          ]),
          SettingsSection(
            title:
Text("Автоматизація", style:
CustomStyles.subText(),),
            tiles: [
              SettingsTile.switchTile(
                activeSwitchColor:
CustomStyles.mainColor,
                  initialValue:
timerFunctions.getAutoUpdate(),
                  onToggle: (value) {
                    setState(() {

timerFunctions.setAutoUpdate(value);
                    });
                  },
                  title:
Text("Автоматичне оновлення даних")),
              SettingsTile.navigation(
                leading:
Icon(Icons.timer_outlined),
                title: Text('Час
оновлення'),
                value:
Text("${SettingsData.timer}"),
                onPressed: (context) =>
_showTimerDialog(),
              )
            ],
          ),
        ],
      ),
    );
  }
}
```

11. Зміст файлу TreeWidget.dart

```
import 'package:flutter/material.dart';
import
'package:flutter_svg/flutter_svg.dart';

import '../classes/CurrentView.dart';

class TreeWidget extends StatefulWidget
{
  final void Function(CurrentView)
changeViewFunction;
  final CurrentView Function()
getViewFunction;
  final void Function(int)
setShelfIndex;
  final void Function(int)
setPlantIndex;

  const TreeWidget(
      {Key? key,
      required this.changeViewFunction,
      required this.getViewFunction,
      required this.setShelfIndex,
      required this.setPlantIndex})
      : super(key: key);
```

```
  @override
  State<TreeWidget> createState() =>
_TreeWidgetState(
      changeViewFunction,
getViewFunction, setShelfIndex,
setPlantIndex);
}

class _TreeWidgetState extends
State<TreeWidget> {
  void Function(CurrentView)
changeViewFunction;
  CurrentView Function()
getViewFunction;
  void Function(int) setShelfIndex;
  void Function(int) setPlantIndex;
  int _shelfIndex = -1;


_TreeWidgetState(this.changeViewFunctio
n, this.getViewFunction,
      this.setShelfIndex,
this.setPlantIndex);

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
        builder: (BuildContext context,
BoxConstraints constraints) {
      return Stack(
          fit: StackFit.expand,
          children:
getTreeButtons(getViewFunction(), 0,
constraints));
    });
  }

  List<Positioned> getTreeButtons(
      CurrentView view, int index,
BoxConstraints constraints) {
    var heightH = constraints.maxHeight
/ 100;
    var widthH = constraints.maxWidth /
100;
    List<Positioned> buttons = [];
    switch (view) {
      case CurrentView.GENERAL:
        buttons += [
          Positioned(
              child: Container(
            child:
SvgPicture.asset('svg/tree.svg',
semanticsLabel: 'Tree'),
          )),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  _shelfIndex = 0;

setShelfIndex.call(0);

changeViewFunction(CurrentView.SHELF);
                },
            ),
            top: 10 * heightH,
            left: 2 * widthH,
            height: 20 * heightH,
            width: 45 * widthH,
          ),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  _shelfIndex = 1;

setShelfIndex.call(1);

changeViewFunction(CurrentView.SHELF);
                },
            ),
            top: 22 * heightH,
            right: 2 * widthH,
            height: 20 * heightH,
            width: 45 * widthH,
          ),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  _shelfIndex = 2;

setShelfIndex.call(2);

changeViewFunction(CurrentView.SHELF);
                },
            ),
            top: 35 * heightH,
            left: 2 * widthH,
            height: 20 * heightH,
            width: 45 * widthH,
          ),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  _shelfIndex = 3;

setShelfIndex.call(3);

changeViewFunction(CurrentView.SHELF);
                },
            ),
            top: 48 * heightH,
            right: 2 * widthH,
            height: 20 * heightH,
            width: 45 * widthH,
          ),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  _shelfIndex = 4;

setShelfIndex.call(4);
```

```
changeViewFunction(CurrentView.SHELF);
                },
            ),
            top: 60 * heightH,
            left: 2 * widthH,
            height: 20 * heightH,
            width: 45 * widthH,
          )
        ];
        break;
      case CurrentView.SHELF:
        buttons += [
          Positioned(
              child: Container(
            child:
SvgPicture.asset('svg/shelf.svg',
semanticsLabel: 'Shelf'),
          )),
          Positioned(
              child: IconButton(
                icon:
Icon(Icons.arrow_back),
                onPressed: () {
                  _shelfIndex = -1;
                  setShelfIndex.call(-
1);

changeViewFunction(CurrentView.GENERAL)
;
                },
              ),
              top: 0),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  int plantIndex = 2 *
_shelfIndex;

setPlantIndex.call(plantIndex);

changeViewFunction(CurrentView.PLANT);
                },
            ),
            top: 30 * heightH,
            left: 7 * widthH,
            height: 40 * heightH,
            width: 40 * widthH,
          ),
          Positioned(
            child: GestureDetector(
                onTap: () {
                  int plantIndex = 2 *
_shelfIndex + 1;

setPlantIndex.call(plantIndex);

changeViewFunction(CurrentView.PLANT);
                },
            ),
            top: 30 * heightH,
            right: 7 * widthH,
            height: 40 * heightH,
            width: 40 * widthH,
          )
        ];
        break;
      case CurrentView.PLANT:
        buttons += [
          Positioned(
              child: Container(
            child:
SvgPicture.asset('svg/plant.svg',
semanticsLabel: 'Plant'),
          )),
          Positioned(
              child: IconButton(
                icon:
Icon(Icons.arrow_back),
                onPressed: () {
                  setPlantIndex.call(-
1);

changeViewFunction(CurrentView.SHELF);
                },
              ),
              top: 0),
        ];
        break;
    }
    return buttons;
  }
}
```

12. Зміст файлу main.dart

```
import 'package:flutter/material.dart';
import
'package:indoor_forest/classes/RandomSt
atisticsGetter.dart';
import
'package:indoor_forest/classes/Settings
Data.dart';
import
'package:indoor_forest/classes/Statisti
cs.dart';
import
'package:indoor_forest/classes/Statisti
csData.dart';
import
'package:indoor_forest/classes/TimerFun
ctions.dart';
import
'package:indoor_forest/widgets/M.dart';
import
'package:indoor_forest/widgets/NavDrawe
```

```
r.dart';
import
'package:indoor_forest/widgets/Settings
Form.dart';
import
'package:indoor_forest/widgets/TreeWidg
et.dart';

import 'classes/CurrentView.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatefulWidget {
  MyApp({Key? key})
      : state = _MyAppState(),
        super(key: key);

  _MyAppState state;

  void setView(CurrentView view) {
    state.setView(view);
  }

  @override
  State<MyApp> createState() => state;
}

class _MyAppState extends State<MyApp>
{
  late TimerFunctions timerFunctions;
  late SettingsForm settingsForm;

  _MyAppState() {
    this.timerFunctions =
TimerFunctions(update);
    settingsForm =
SettingsForm(timerFunctions:
timerFunctions);
  }

  DateTime timeUpdated =
DateTime.now();
  Statistics statistics = Statistics(
      StatisticsData(0, 0, 0, [0, 0, 0,
0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0]),
      RandomStatisticsGetter());

  CurrentView _view =
CurrentView.GENERAL;
  int _shelfIndex = 0;
  int _plantIndex = 0;

  void setView(CurrentView newView) {
    setState(() {
      _view = newView;
    });
  }

  void setShelfIndex(int index) {
    _shelfIndex = index;
  }

  void setPlantIndex(int index) {
    _plantIndex = index;
  }

  CurrentView getView() {
    return _view == null ?
CurrentView.GENERAL : _view;
  }

  int getShelfIndex() {
    return _shelfIndex;
  }

  int getPlantIndex() {
    return _plantIndex;
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        routes: {"/settings": (context)
=> settingsForm},
        home: Scaffold(
          drawer: NavDrawer(),
          appBar: AppBar(
            title: Text("Indoor
Forest"),
            backgroundColor:
Colors.greenAccent,
          ),
          body: OrientationBuilder(
              builder: (context,
orientation) {
                return orientation ==
Orientation.portrait ?
_portraitView() : _landscapeView();
              }
          ),
        ));
  }

  List<Text> getDataList() {
    List<Text> data = [];
    switch (_view) {
      case CurrentView.GENERAL:
        data = [
          Text(
            "Температура:
${statistics.data.temperature}",
            style:
CustomStyles.simpleText(),
          ),
          Text("Вологість повітря:
${statistics.data.humidity}",
              style:
CustomStyles.simpleText()),
          Text("Тиск:
${statistics.data.pressure}",
```

```
              style:
CustomStyles.simpleText())
        ];
        break;
      case CurrentView.SHELF:
        var value =
statistics.data.luminosity[_shelfIndex]
;
        var message =
            "Освітленість:
${statistics.data
            .luminosity[_shelfIndex]}${
checkValue(
            value,
SettingsData.luminosityLimit)}";
        data = [
          Text(message, style:
CustomStyles.simpleText()),
        ];
        break;
      case CurrentView.PLANT:
        var value =
statistics.data.soilHumidity[_plantInde
x];
        var message =
            "Вологість ґрунту:
${statistics.data
            .soilHumidity[_plantIndex]}
${checkValue(
            value,
SettingsData.humidityLimit)}";
        data = [
          Text(message, style:
CustomStyles.simpleText()),
        ];
        break;
    }
    return data;
  }

  List<Widget> getButtons() {
    List<Widget> buttons = [];
    switch (_view) {
      case CurrentView.SHELF:
        buttons = [
          ElevatedButton(
              onPressed: () {
                setState(() {

statistics.changeLuminosity(_shelfIndex
);
                });
              },
              child: Icon(Icons.check),
              style:
CustomStyles.simpleCircleButtonStyle())
,
        ];
        break;
      case CurrentView.PLANT:
        buttons = [
          ElevatedButton(
              onPressed: () {
                setState(() {

statistics.changeHumidity(_plantIndex);
                });
              },
              child: Icon(Icons.check),
              style:
CustomStyles.simpleCircleButtonStyle())
,
        ];
        break;
      case CurrentView.GENERAL:
        buttons = [
          ElevatedButton(
              onPressed: () {
                setState(() {

statistics.optimize();
                });
              },
              child: Icon(Icons.check),
              style:
CustomStyles.simpleCircleButtonStyle())
,
        ];
        break;
    }
    return buttons;
  }

  String checkValue(int actual, int
limit) {
    return actual >= limit ? "
задовільно" : ", нижче норми";
  }

  void update() {
    setState(() {
      timeUpdated = DateTime.now();
      statistics.update();
    });
  }

  Widget _portraitView() {
    return Column(
        mainAxisAlignment:
MainAxisAlignment.spaceAround,
        crossAxisAlignment:
CrossAxisAlignment.stretch,
        children: [
          Expanded(
            flex: 2,
            child: Container(
              padding:
EdgeInsets.all(20),
              child: Center(
                child: _updateButton(),
              ),
            ),
```

```
          ),
          Expanded(
            flex: 9,
            child: Container(
              padding:
EdgeInsets.all(20),
              child: TreeWidget(
                changeViewFunction:
setView,
                getViewFunction:
getView,
                setShelfIndex:
setShelfIndex,
                setPlantIndex:
setPlantIndex,
              ),
            ),
          ),
          Expanded(
            flex: 3,
            child: Container(
              padding:
EdgeInsets.all(30),
              child: Row(
                crossAxisAlignment:
CrossAxisAlignment.center,
                mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                children: <Widget>[
                  Column(
                    children:
getDataList(),
                    mainAxisAlignment:
MainAxisAlignment.center,
                    crossAxisAlignment:
CrossAxisAlignment.start,
                  ),
                ] + getButtons(),
              ),
            ),
          ),
        ]);
  }

  Widget _landscapeView(){
    return Row(
      mainAxisAlignment:
MainAxisAlignment.spaceAround,
      crossAxisAlignment:
CrossAxisAlignment.stretch,
      children: [
        Expanded(
          flex: 5,
          child: Container(
            padding:
EdgeInsets.all(20),
            child: TreeWidget(
              changeViewFunction:
setView,
              getViewFunction: getView,
              setShelfIndex:
setShelfIndex,
              setPlantIndex:
setPlantIndex,
            ),
          ),
        ),
        Expanded(
          flex: 5,
          child: Container(
            padding:
EdgeInsets.all(20),
            child: Column(
              crossAxisAlignment:
CrossAxisAlignment.start,
              mainAxisAlignment:
MainAxisAlignment.spaceBetween,
              children: [
                _updateButton(),
                Column(
                  children:
getDataList(),
                  mainAxisAlignment:
MainAxisAlignment.start,
                  crossAxisAlignment:
CrossAxisAlignment.start,
                ),
                Row(
                  children:
getButtons(),
                  mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
                )
              ],
            ),
          ),
        ),
      ],
    );
  }

  Widget _updateButton(){
    return ElevatedButton(
      style: ElevatedButton.styleFrom(
        primary: Colors.greenAccent,
      ),
      onPressed: () {
        update();
      },
      child: Row(
        mainAxisAlignment:
MainAxisAlignment.center,
        children: [
          Text(
              "Оновлено: ${timeUpdated
                  .month}.${timeUpdated
.day} ${timeUpdated
                  .hour}:${timeUpdated
                  .minute}:${timeUpdate
d.second}",
              style:
CustomStyles.simpleTextWhite()),
```

```
  Icon(
    Icons.update,
    color: Colors.white,
  ),
],
```

```
        ),
      );
    }
  }
```