Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

**ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ДОГЛЯДУ ЗА РОСЛИНАМИ НА ОСНОВІ ІГРОВОЇ МОДЕЛІ ВЗАЄМОДІЇ**
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-2
Шевченко О. О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл. Кобилін І. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шевченку Олегу Олександровчу
(прізвище, ім'я, по батькові)

1. Тема роботи Проєктування та реалізація мобільного застосунку для догляду за рослинами на основі ігрової моделі взаємодії

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 27 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова програмування Dart, фреймворк Flutter, мова програмування Go, база даних PostgreSQL, платформа Supabase, сервіс Firebase Cloud Messaging, середовище розробки Visual Studio Code.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних мобільних застосунків для догляду за кімнатними рослинами та визначення основних проблем користувачів.
2. Проєктування структури мобільного застосунку, моделі даних та сценаріїв взаємодії користувача із системою.
3. Програмна реалізація мобільного застосунку для догляду за рослинами з функціями нагадувань, рекомендацій, check-in взаємодії та ігрової моделі.
4. Тестування розробленого мобільного застосунку та аналіз результатів роботи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми догляду за кімнатними рослинами, постановка задачі, етапи розроблення застосунку, функціональна модель мобільного застосунку, модель даних, тестування застосунку, висновки, апробація результатів роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

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

**КАЛЕНДАРНИЙ ПЛАН**

| № з/п | Назва етапів роботи | Строк / терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 13.04.2026 | |
| 2 | Аналіз завдання, підбір літератури | 14.04.26-19.04.26 | |
| 3 | Аналіз літератури з проблеми, що вирішується | 19.04.26-25.04.26 | |
| 4 | Аналіз існуючих методів розпізнавання | 25.04.26-27.04.26 | |
| 5 | Формування кроків вирішення проблеми | 27.04.26-09.05.26 | |
| 6 | Програмна реалізація | 30.04.26-25.05.26 | |
| 7 | Аналіз отриманих результатів | 25.05.26-04.06.26 | |
| 8 | Оформлення пояснювальної записки | 05.06.26-09.06.26 | |
| 9 | Перевірка на нормоконтроль | 10.06.26-19.06.26 | |
| 10 | Перевірка на плагіат | 11.06.26-30.06.26 | |
| 11 | Рецензування | 20.06.26-30.06.26 | |
| 12 | Підготовка презентації та доповіді | 20.06.26-30.06.26 | |
| 13 | Занесення роботи в електронний архів | 30.06.26-09.07.26 | |
| 14 | Попередній захист кваліфікаційної роботи | 30.06.26-09.07.26 | |

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ ст. викл. Кобилін І. О.
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 102 с., 8 табл., 19 рис., 32 джерела.

ГЕЙМІФІКАЦІЯ, ДОГЛЯД ЗА РОСЛИНАМИ, МОБІЛЬНИЙ ЗАСТОСУНОК, НАГАДУВАННЯ, РОЗПІЗНАВАННЯ РОСЛИН, DART, FLUTTER.

Об’єктом роботи є процес цифрової підтримки користувача під час догляду за кімнатними рослинами.

Метою роботи є проєктування та реалізація мобільного застосунку для догляду за рослинами, що забезпечує додавання рослин, перегляд рекомендацій, нагадування про полив, та елементи ігрової взаємодії.

Під час роботи використано методи аналізу предметної області, проєктування мобільних застосунків, організації даних, елементи гейміфікації та підходи до розпізнавання рослин за зображенням.

Практична цінність роботи полягає у створенні мобільного застосунку, який поєднує інформаційні, функціональні та мотиваційні елементи підтримки догляду за кімнатними рослинами.

Розроблений мобільний застосунок MoriMori автоматизує догляд за рослинами за допомогою профілів, нагадувань, історії дій, розпізнавання за фото та інтерактивного маскота.

Область застосування: догляд за кімнатними рослинами, мобільні сервіси підтримки користувача, цифрові системи формування звичок.

DART, FLUTTER, GAMIFICATION, MOBILE APPLICATION, PLANT CARE, PLANT RECOGNITION, REMINDERS.

The object of the work is the process of digital user support during indoor plant care.

The goal of the work is to design and implement a mobile application for plant care that provides plant profile creation, care recommendations, watering reminders, care history tracking, and game interaction elements.

During the work, methods of domain analysis, mobile application design, data organization, gamification elements, and approaches to plant recognition from images were used.

The developed MoriMori application combines informational and motivational tools for indoor plant care, providing profile creation, reminders, action tracking, history logs, photo-based recognition, and mascot interaction.

Application areas: indoor plant care, mobile user support services, digital habit-forming systems.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface, інтерфейс прикладного програмування, що використовується для обміну даними між клієнтською та серверною частинами застосунку

Check-in – механіка щоденної взаємодії користувача із рослиною

CRUD – Create, Read, Update, Delete, базові операції створення, перегляду, оновлення та видалення даних

Dart – мова програмування, що використовується для розроблення мобільного застосунку MoriMori

Flutter – фреймворк для створення кросплатформених мобільних застосунків

Go – мова програмування, що використовується для реалізації серверної частини застосунку

IAP – In-App Purchases, покупки всередині застосунку

JWT – JSON Web Token, формат токена для підтвердження автентифікації користувача

MVP – Minimum Viable Product, мінімально життєздатна версія програмного продукту

Rescue Mode – спеціальний режим підтримки рослини у застосунку MoriMori

Supabase – сервісна платформа для роботи з базою даних, авторизацією та сховищем файлів

Timeline – історія подій і дій догляду за рослиною

# ВСТУП

У сучасних умовах мобільні застосунки стали одним із найзручніших інструментів для підтримки повсякденних процесів користувача. Вони використовуються для планування задач, контролю звичок, отримання рекомендацій, збереження інформації та взаємодії з цифровими сервісами. Одним із напрямів, де мобільні технології можуть бути корисними, є догляд за кімнатними рослинами.

Догляд за рослинами потребує регулярності, уважності та базових знань про потреби конкретного виду рослини. Користувач повинен враховувати частоту поливу, освітлення, температуру, вологість, стан ґрунту та зовнішній вигляд рослини. На практиці багато користувачів, особливо без значного досвіду, стикаються з проблемами нерегулярного догляду, неправильного поливу або відсутності розуміння, як реагувати на зміни стану рослини.

Актуальність роботи полягає в необхідності створення зручного цифрового інструменту, який допомагає користувачу не лише отримувати інформацію про рослину, а й підтримувати регулярну взаємодію з нею. Звичайні нагадування або довідники частково вирішують проблему, але не завжди формують звичку догляду. Тому доцільним є використання ігрової моделі взаємодії, яка може підвищувати залученість користувача за допомогою маскота, прогресу, рівнів, нагород та елементів кастомізації.

# 1 АНАЛІЗ МЕТОДІВ І ЗАСОБІВ СТВОРЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ДОГЛЯДУ ЗА РОСЛИНАМИ

## 1.1 Аналіз предметної області догляду за кімнатними рослинами

Догляд за кімнатними рослинами є процесом, що поєднує регулярне виконання практичних дій, спостереження за станом рослини та використання базових знань про її потреби [1, 2]. До основних дій догляду належать полив, контроль освітлення, пересаджування, підживлення, очищення листя, спостереження за змінами зовнішнього вигляду та своєчасне реагування на ознаки захворювань або неправильних умов утримання. Незважаючи на те, що ці дії можуть здаватися простими, на практиці користувачі часто стикаються з проблемою нерегулярності догляду, недостатньої обізнаності та складності у визначенні реального стану рослини.

Особливо актуальною ця проблема є для користувачів без значного досвіду вирощування кімнатних рослин. Новачки часто не знають, як часто потрібно поливати конкретну рослину, де її краще розмістити, які ознаки свідчать про нестачу світла, надлишок води або початок захворювання. У результаті догляд часто ґрунтується не на системному підході, а на випадкових діях: користувач може поливати рослину занадто часто, забувати про неї на тривалий час або не помічати поступового погіршення її стану.

Однією з ключових особливостей догляду за рослинами є залежність правильних дій від конкретного виду рослини. Різні рослини мають різні вимоги до освітлення, частоти поливу, температури, вологості та ґрунту. Наприклад, рослини, які пристосовані до сухого клімату, можуть негативно реагувати на надмірний полив, тоді як тропічні рослини часто потребують більшої вологості та стабільніших умов. Тому універсальні поради не завжди є достатніми, а користувачу потрібна більш персоналізована інформація.

Окремою проблемою є те, що догляд за рослинами потребує не одноразової дії, а регулярної поведінки [3, 4]. Користувач повинен не лише один раз дізнатися інформацію про рослину, а й повертатися до застосунку, виконувати відмітки про догляд, переглядати нагадування та підтримувати звичку догляду. Саме тому звичайний довідник або одноразова функція розпізнавання рослини не завжди вирішує проблему повністю [5]. Для практичної користі потрібна система, яка допомагає користувачу регулярно взаємодіяти з рослиною та отримувати зворотний зв’язок.

Для узагальнення основних складових цифрової підтримки догляду за кімнатними рослинами доцільно виділити кілька взаємопов’язаних напрямів: ідентифікацію рослини, надання рекомендацій, планування нагадувань, фіксацію виконаних дій, аналіз стану рослини та підтримку повторної взаємодії користувача. Така структура показує, що мобільний застосунок для догляду за рослинами має поєднувати не одну окрему функцію, а кілька взаємопов’язаних модулів. Узагальнену схему цифрової підтримки догляду за рослинами наведено на рисунку 1.1.

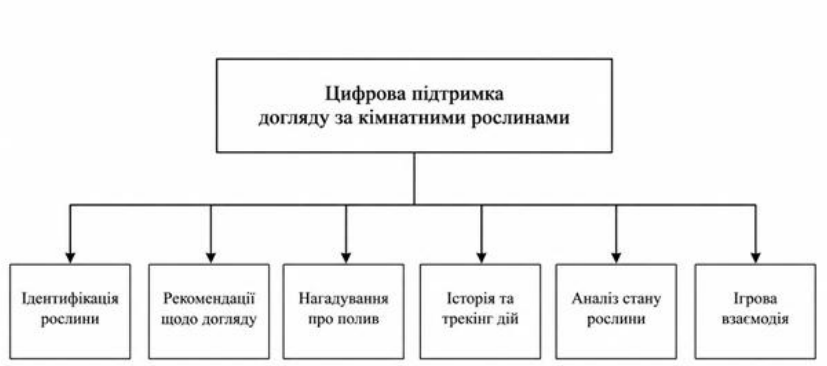


Рисунок 1.1 – Узагальнена схема цифрової підтримки догляду за кімнатними рослинами

Мобільний застосунок є зручним засобом для вирішення цієї проблеми, оскільки смартфон постійно знаходиться поруч із користувачем і може виконувати функції персонального помічника. За допомогою мобільного

застосунку можна надсилати нагадування, зберігати історію догляду, надавати рекомендації, фіксувати стан рослини та створювати персоналізовану взаємодію. Крім того, мобільний застосунок може використовувати камеру смартфона для розпізнавання рослини або аналізу її зовнішнього вигляду.

Важливо також враховувати, що користувачі не завжди мають мотивацію регулярно відкривати застосунок, якщо він виконує лише утилітарну функцію. Якщо система тільки нагадує про полив, вона може сприйматися як звичайний календар або список задач. У такому випадку користувач може швидко втратити інтерес або почати ігнорувати сповіщення. Тому доцільно доповнювати функціональні можливості елементами взаємодії, які підвищують емоційну залученість користувача.

Таким чином, предметна область догляду за кімнатними рослинами включає не тільки інформаційні задачі, пов’язані з пошуком рекомендацій, а й поведінкові задачі, пов’язані з формуванням регулярної взаємодії користувача з рослиною. Саме тому доцільним є створення мобільного застосунку, який поєднує функції розпізнавання рослин, надання рекомендацій, нагадувань, історії догляду та ігрової моделі взаємодії [6–9].

## 1.2 Аналіз існуючих типів мобільних застосунків для догляду за рослинами

Сучасні мобільні застосунки для догляду за рослинами можна умовно поділити на кілька груп. До першої групи належать застосунки для розпізнавання рослин за фотографією. Основна функція таких застосунків полягає у визначенні виду рослини за зображенням, отриманим з камери смартфона або завантаженим із галереї. Такі рішення є корисними на етапі

первинного визначення рослини, однак вони не завжди забезпечують подальший супровід користувача в процесі догляду.

До другої групи належать застосунки-нагадувачі. Їхня основна задача полягає у створенні розкладу поливу, пересаджування або підживлення. Такі застосунки допомагають уникнути пропусків у догляді, однак часто мають обмежену функціональність. Вони можуть нагадувати користувачу про певну дію, але не завжди пояснюють, чому саме ця дія потрібна, як змінити графік залежно від стану рослини або як реагувати на проблеми.

До третьої групи належать інформаційні довідники про рослини. Вони містять дані про види рослин, умови освітлення, частоту поливу, температуру, вологість, тип ґрунту та інші параметри [10, 11]. Такі рішення мають високу інформаційну цінність, але часто є пасивними: користувач повинен самостійно шукати потрібну інформацію та переносити її у власну практику догляду.

До четвертої групи можна віднести застосунки з елементами трекінгу. Вони дозволяють користувачу фіксувати виконані дії, наприклад полив, пересаджування або зміну стану рослини. Трекінг є важливим елементом, оскільки дає змогу бачити історію взаємодії з рослиною та аналізувати, які дії вже були виконані. Проте сам по собі трекінг не гарантує, що користувач буде регулярно повертатися до застосунку [11–13].

Прикладами сучасних мобільних рішень у сфері догляду за рослинами є PlantIn, Planta, PictureThis та PlantNet. Для детальнішого порівняння існуючих мобільних застосунків було виділено основні функціональні критерії, які є важливими для розроблюваного застосунку: розпізнавання рослин, наявність рекомендацій, нагадування, історія догляду, діагностика проблем та ігрова взаємодія. Порівняння існуючих рішень наведено в таблиці 1.1.

Такі застосунки демонструють різні підходи до вирішення проблеми: одні більше зосереджені на розпізнаванні рослин, інші − на нагадуваннях, порадах щодо догляду або діагностиці можливих проблем [14, 15].

Таблиця 1.1 – Порівняння мобільних застосунків для догляду за рослинами

| **Застосунок** | **Розпізнавання рослин** | **Рекомендації з догляду** | **Нагадування** | **Історія догляду** | **Ігрова взаємодія** |
|---|---|---|---|---|---|
| PlantIn | Є | Є | Є | Частково | Обмежена |
| Planta | Частково | Є | Є | Є | Обмежена |
| PictureThis | Є | Є | Частково | Частково | Обмежена |
| PlantNet | Є | Частково | Немає | Немає | Немає |
| MoriMori | Є | Є | Є | Є | Є |

Наприклад, PlantNet насамперед асоціюється з розпізнаванням рослин за фотографією, PictureThis і PlantIn поєднують ідентифікацію з рекомендаціями, а Planta орієнтована на супровід користувача в догляді та нагадування.

Аналіз цих застосунків показує, що більшість із них будують взаємодію навколо конкретної функції: визначити рослину, показати інформацію або нагадати про полив. Такий підхід є практичним, але не завжди достатнім для формування довгострокової звички. Користувач може отримати корисну інформацію, однак після цього не обов’язково буде регулярно повертатися до застосунку. Тому важливо не лише реалізувати функціональні можливості, а й продумати модель повторної взаємодії.

Окрему цінність мають застосунки, які використовують ігрові механіки [16–18]. У таких системах взаємодія з користувачем будується не лише через функціональні кнопки та нагадування, а й через емоційні та мотиваційні елементи: рівні, досягнення, віртуальні персонажі, нагороди, прогрес та кастомізацію. Ігрова модель взаємодії може підвищити залученість користувача та зробити процес догляду менш механічним.

У контексті розроблюваного застосунку доцільно використати комбінований підхід. З одного боку, застосунок повинен мати практичну цінність: допомагати додати рослину, отримати базову інформацію, зафіксувати дію догляду та переглянути історію. З іншого боку, система повинна мати просту ігрову модель, яка не ускладнює застосунок, але створює додаткову мотивацію для регулярного використання.

Аналіз існуючих рішень показує, що більшість застосунків зосереджені на окремих функціях: розпізнаванні рослини, довідковій інформації, нагадуваннях або трекінгу. Проте для повноцінної підтримки користувача доцільно поєднати ці підходи в одному мобільному застосунку. Особливо важливо не лише надати інформацію, а й створити модель взаємодії, яка допомагає користувачу підтримувати регулярний догляд.

Таким чином, перспективним напрямом є створення застосунку, який поєднує функції догляду за рослинами з ігровими елементами. Такий підхід дозволяє розглядати рослину не лише як об’єкт догляду, а як частину інтерактивної взаємодії користувача із системою.

## 1.3 Методи розпізнавання рослин за зображенням

Однією з важливих функцій мобільного застосунку для догляду за рослинами є можливість розпізнавання рослини за фотографією. Така функція дозволяє користувачу швидко визначити вид рослини та отримати базові рекомендації щодо догляду. Для новачків це особливо важливо, оскільки користувач може не знати точної назви рослини, але потребує інформації про полив, освітлення та інші умови утримання.

З технічної точки зору розпізнавання рослин належить до задач комп’ютерного зору [3, 9, 18]. Вхідними даними є зображення рослини, а результатом роботи системи є визначення її виду або формування списку найбільш імовірних варіантів. Для виконання таких задач можуть

застосовуватися методи класифікації зображень, нейронні мережі, згорткові нейронні мережі, мультимодальні моделі або спеціалізовані API для розпізнавання рослин.

Класичний підхід до розпізнавання зображень передбачає виділення ознак, які характеризують об’єкт. У випадку рослин такими ознаками можуть бути форма листя, колір, розмір, текстура, контури, розташування жилок, форма стебла або квітки [9, 10]. Проте ручне виділення ознак має обмеження, оскільки рослини можуть суттєво відрізнятися залежно від умов освітлення, кута фотографування, віку рослини та якості зображення.

Сучасні підходи частіше використовують нейронні мережі, які здатні автоматично виділяти важливі ознаки зображення. Згорткові нейронні мережі добре підходять для задач класифікації зображень, оскільки вони аналізують просторову структуру зображення та можуть виявляти локальні візуальні патерни. У контексті розпізнавання рослин це дозволяє моделі враховувати форму листя, контури, текстуру та інші візуальні характеристики [19, 20].

У практичній реалізації мобільного застосунку не обов’язково навчати власну модель розпізнавання з нуля [9, 18, 19]. Це потребувало б великого набору зображень, часу на навчання, обчислювальних ресурсів і додаткового оцінювання точності. Для студентського проєкту більш доцільним є використання готової моделі, зовнішнього API або спрощеного програмного модуля, який приймає фото рослини та повертає результат ідентифікації. Такий підхід дозволяє зосередитися не на складному навчанні моделі, а на інтеграції функції розпізнавання в загальний сценарій користувача.

Також можуть використовуватися мультимодальні моделі, які працюють не лише із зображенням, а й з текстовим описом. Такий підхід є корисним, якщо система повинна не просто визначити вид рослини, а й сформувати пояснення або рекомендацію для користувача. Наприклад, після аналізу фотографії система може повернути не лише назву рослини, а й коротку інформацію про умови догляду.

У мобільному застосунку розпізнавання рослини може бути реалізоване як окремий модуль. Користувач робить фото рослини, після чого зображення передається на обробку. Система аналізує зображення та повертає результат: ймовірний вид рослини, базовий опис, рекомендації щодо освітлення, частоти поливу та можливих особливостей догляду. Отримана інформація може бути збережена у профілі рослини та використовуватися для формування подальших нагадувань.

Важливим обмеженням такого підходу є залежність якості розпізнавання від якості фотографії. Погане освітлення, розмиття, перекриття листя, неправильний ракурс або наявність кількох рослин на одному зображенні можуть зменшити точність визначення [9, 10]. Тому в застосунку доцільно передбачити можливість ручного уточнення результату або повторного завантаження фотографії.

Для розроблюваного застосунку достатньо реалізувати базовий сценарій роботи з розпізнаванням: отримання або завантаження зображення, передавання його на обробку, отримання результату та збереження інформації про рослину в профілі користувача. Такий підхід є прийнятним для кваліфікаційної роботи, оскільки демонструє практичне застосування методів комп’ютерного зору без необхідності створювати повноцінну дослідницьку систему навчання моделей.

Отже, модуль розпізнавання рослин є важливою частиною застосунку, оскільки він зменшує вхідний бар’єр для користувача. Користувачу не потрібно самостійно шукати назву рослини, а система може автоматично допомогти з первинною ідентифікацією та подальшими рекомендаціями.

## 1.4 Використання ігрової моделі взаємодії у мобільному застосунку

Ігрова модель взаємодії використовується в мобільних застосунках для підвищення залученості користувача, формування звички та створення

емоційного зв’язку із системою [7, 14, 15]. У контексті догляду за рослинами це особливо актуально, оскільки користувачу потрібно регулярно повертатися до застосунку та виконувати повторювані дії. Якщо ці дії подані лише як список обов’язків, користувач може швидко втратити інтерес. Ігрові елементи дозволяють зробити процес догляду більш зрозумілим, мотивуючим і персоналізованим.

До основних елементів ігрової моделі можна віднести рівні, прогрес, досягнення, нагороди, віртуальну валюту, кастомізацію та персонажів [7, 14, 15].

Основні елементи ігрової моделі у застосунку MoriMori та їх призначення наведено в таблиці 1.2.

Таблиця 1.2 – Елементи ігрової моделі взаємодії в застосунку MoriMori

| **Елемент** | **Призначення** | **Зв’язок із доглядом** |
|---|---|---|
| Маскот | Візуальний персонаж для комунікації з користувачем | Реагує на стан рослини та дії користувача |
| XP | Нарахування досвіду за виконані дії | Нараховується за check-in, полив, rescue-дії |
| Рівні | Відображення довгострокового прогресу | Показують регулярність взаємодії |
| Streak | Фіксація послідовних днів активності | Підтримує звичку догляду |
| Кастомізація | Персоналізація маскота або інтерфейсу | Виступає винагородою за активність |
| Медитація з рослиною | Коротка спокійна взаємодія | Підсилює емоційний зв’язок із рослиною |

У розроблюваному застосунку важливу роль відіграє маскот – віртуальний персонаж, через якого система може комунікувати з користувачем. Маскот може відображати стан взаємодії, реагувати на

виконані дії, мотивувати користувача та створювати більш дружній інтерфейс [1, 2].

Рівні та прогрес можуть використовуватися для візуалізації активності користувача. Наприклад, користувач може отримувати прогрес за регулярний догляд, виконання відміток про стан рослини, своєчасний полив або використання функцій застосунку. Така система дозволяє показати, що навіть невеликі дії мають значення та поступово накопичуються в загальний результат.

Система досягнень, прогресу та елементи кастомізації можуть використовуватися для персоналізації маскота. Такі механіки доповнюють основні сценарії догляду та сприяють формуванню звички регулярної взаємодії з рослиною. Користувач може отримувати умовну валюту або бонуси за виконання дій догляду, а потім використовувати їх для персоналізації персонажа. Це створює додатковий мотив повертатися до застосунку та підтримувати регулярність дій.

Окремим елементом взаємодії може бути таймер короткої спокійної взаємодії з рослиною. У застосунку така функція може бути подана як режим короткого відпочинку або «мікрочіл». Її задача полягає не в безпосередньому догляді за рослиною, а у створенні емоційного контакту користувача з рослиною та застосунком [16, 17]. Такий елемент може підвищити суб’єктивну цінність продукту та зробити взаємодію менш утилітарною.

При цьому ігрова модель не повинна ускладнювати основні сценарії використання. Якщо користувач хоче швидко переглянути стан рослини або відмітити полив, він повинен мати змогу зробити це без зайвих кроків. Тому гейміфікація має виконувати допоміжну роль: підтримувати мотивацію, але не заважати основним функціям догляду [7, 14]. Для цього доцільно використовувати прості механіки: прогрес, рівень, реакцію маскота та базову кастомізацію.

Також важливо, щоб ігрові елементи були пов’язані з реальними діями користувача. Якщо користувач отримує прогрес або нагороду, це має бути

пов’язано з виконанням корисної дії: додаванням рослини, поливом, переглядом рекомендації, фіксацією стану або поверненням до застосунку [21, 22]. Такий підхід дозволяє уникнути ситуації, коли гейміфікація існує окремо від основної мети продукту.

Ігрова модель взаємодії повинна бути збалансованою. Її мета полягає не в тому, щоб перетворити застосунок на гру, а в тому, щоб підтримати основну задачу − регулярний і більш усвідомлений догляд за рослинами. Тому гейміфікація має бути пов’язана з реальними діями користувача: поливом, відміткою стану, переглядом рекомендацій, фіксацією стану рослини або виконанням нагадувань.

Таким чином, ігрова модель взаємодії є важливим засобом підвищення залученості користувача. Вона дозволяє зробити догляд за рослинами не лише функціональним процесом, а й більш емоційною та регулярною взаємодією. Для розроблюваного застосунку це є однією з ключових відмінностей від звичайних нагадувачів або довідників.

## 1.5 Постановка задачі

На основі аналізу предметної області, існуючих типів мобільних застосунків і методів розпізнавання рослин було визначено необхідність створення мобільного застосунку для догляду за кімнатними рослинами з ігровою моделлю взаємодії. Такий застосунок повинен не лише надавати користувачу інформацію про догляд, а й допомагати підтримувати регулярну взаємодію з рослиною.

Об’єктом роботи є процес цифрової підтримки користувача під час догляду за кімнатними рослинами.

Метою роботи є проєктування та реалізація мобільного застосунку для догляду за рослинами, що забезпечує додавання рослин, перегляд рекомендацій, нагадування про полив, та елементи ігрової взаємодії.

Розроблюваний застосунок має забезпечувати додавання рослини, перегляд рекомендацій щодо догляду, формування нагадувань про полив, фіксацію виконаних дій, ведення історії догляду та взаємодію користувача з маскотом. Ігрова модель взаємодії має використовуватися як допоміжний механізм для підвищення регулярності користування застосунком і підтримки звички догляду за рослинами.

Для досягнення поставленої мети необхідно вирішити такі завдання:

− проаналізувати предметну область догляду за кімнатними рослинами;

− дослідити існуючі типи мобільних застосунків для догляду за рослинами;

− розглянути методи розпізнавання рослин за зображенням;

− визначити основні функціональні можливості мобільного застосунку;

− спроєктувати структуру мобільного застосунку та основні сценарії взаємодії користувача;

− реалізувати основні екрани застосунку;

− реалізувати базову логіку додавання рослини, перегляду інформації про догляд, нагадувань і трекінгу;

− передбачити можливість використання модуля розпізнавання рослини за зображенням;

− реалізувати базові елементи ігрової моделі взаємодії, зокрема маскота, прогрес користувача та кастомізацію;

− перевірити працездатність основних функцій застосунку.

Розроблюваний застосунок має забезпечувати користувачу можливість додати рослину, отримати інформацію про її догляд, фіксувати виконані дії, переглядати історію догляду та взаємодіяти з маскотом. Ігрова модель взаємодії має використовуватися як допоміжний механізм для підвищення регулярності користування застосунком і формування звички догляду за рослинами. Основний акцент робиться на проєктуванні та реалізації мобільного застосунку, який може інтегрувати функцію розпізнавання рослин як окремий модуль або через зовнішній сервіс [14].

# 2 ПРОЄКТУВАННЯ МОДЕЛЕЙ ТА АЛГОРИТМІВ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ДОГЛЯДУ ЗА РОСЛИНАМИ

На основі аналізу предметної області, виконаного у першому розділі, було визначено основні функціональні вимоги до системи, побудовано загальну модель роботи застосунку, описано структуру даних, алгоритм створення профілю рослини, алгоритм розпізнавання рослини за зображенням, модель формування рекомендацій щодо догляду, модель нагадувань, механіку check-in, rescue-режим та ігрову модель взаємодії користувача з маскотом.

Особливістю розроблюваного застосунку є поєднання утилітарної функції догляду за рослиною з поведінковою моделлю повторної взаємодії [11, 23, 24]. Застосунок не лише зберігає інформацію про рослину та нагадує про полив, а й підтримує користувача через регулярні короткі дії, історію догляду, маскота, прогрес, рівні та елементи кастомізації. Такий підхід дозволяє розглядати застосунок як систему цифрової підтримки користувача, а не лише як довідник або календар нагадувань. У межах роботи основна увага приділяється моделюванню логіки мобільного застосунку. Безпосередній опис програмної реалізації, вибір технологічного стеку, структура коду, приклади екранів і тестування наведено у третьому розділі.

## 2.1 Визначення функціональних вимог до мобільного застосунку

Мобільний застосунок MoriMori орієнтований на користувачів, які мають кімнатні рослини та потребують зручного інструменту для організації догляду. Основним користувачем системи є людина без глибоких знань у сфері рослинництва, яка хоче зрозуміти, як доглядати за конкретною

рослиною, коли її поливати, де краще розмістити, як фіксувати виконані дії та як не втрачати регулярність догляду.

На основі аналізу предметної області було визначено такі основні проблеми користувача:

– користувач не завжди знає точний вид рослини;

– користувач не має достатньої інформації про умови догляду;

– догляд часто виконується нерегулярно;

– користувач може забувати про полив або інші дії;

– користувач не завжди розуміє, чи рослина перебуває у нормальному стані;

– звичайні нагадування не завжди формують довгострокову звичку;

– користувач може втрачати емоційний інтерес до рослини після першого періоду використання.

З урахуванням цих проблем було сформовано перелік функціональних вимог до мобільного застосунку наведені в таблиці 2.1

Таблиця 2.1 – Елементи ігрової моделі взаємодії в застосунку MoriMori

| **№** | **Назва вимоги** | **Опис вимоги** | **Очікуваний результат** |
|---|---|---|---|
| **1** | **2** | **3** | **4** |
| F1 | Реєстрація та авторизація користувача | Система повинна забезпечувати створення облікового запису, вхід користувача та доступ до персоналізованих даних. | Користувач отримує доступ до власного профілю, рослини, історії догляду та ігрового прогресу. |
| F2 | Створення профілю рослини | Користувач повинен мати можливість додати рослину, вказати її назву, тип, фото, дату додавання. | Створюється профіль рослини, який надає рекомендації, нагадування і трекінг. |

Продовження таблиці 2.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| F3 | Розпізнавання рослини за зображенням | Система повинна дозволяти користувачу зробити або завантажити фото рослини для визначення її ймовірного виду. | Користувач отримує результат розпізнавання та може підтвердити або уточнити вид рослини. |
| F4 | Надання рекомендацій щодо догляду | Система повинна надавати інформацію про полив, освітлення, розміщення та базові умови догляду за рослиною. | Користувач отримує зрозумілі рекомендації щодо догляду за конкретною рослиною. |
| F5 | Формування нагадувань про полив | Система повинна визначати дату наступного поливу на основі дати останнього поливу та інтервалу догляду. | Користувач отримує нагадування про необхідність поливу рослини. |
| F6 | Щоденна check-in взаємодія | Користувач повинен мати можливість швидко вказати поточний стан рослини: все добре, є проблема або стан невідомий. | Система фіксує стан рослини, оновлює історію догляду та запускає відповідний сценарій взаємодії. |
| F7 | Rescue-режим | Система повинна допомогти уточнити симптоми та сформувати базовий план дій. | Користувач отримує послідовність простих дій для підтримки або відновлення стану рослини. |

Продовження таблиці 2.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| F8 | Збереження історії догляду | Система повинна зберігати події догляду, зокрема полив, фото, нотатки, check-in та rescue-події. | Користувач може переглядати історію взаємодії з рослиною та відстежувати зміни її стану. |
| F9 | Ігрова модель взаємодії | Застосунок повинен використовувати маскота, досвід, рівні, streak-механіку та елементи кастомізації. | Користувач отримує додаткову мотивацію для регулярного догляду за рослиною. |
| F10 | Медитація з рослиною | Система повинна мати таймер короткої спокійної взаємодії користувача з рослиною. | Користувач може виконати коротку спокійну сесію, що підтримує емоційний зв’язок із рослиною. |
| F11 | Профіль користувача | Користувач повинен мати можливість переглядати власний прогрес, рівень, статистику взаємодії та налаштування. | Користувач бачить персональні дані, ігровий прогрес і параметри свого акаунта. |
| F12 | Оброблення відсутності інтернет-з’єднання | Система повинна повідомляти користувача про недоступність функцій, які потребують мережі. | Користувач отримує зрозуміле повідомлення замість технічної помилки. |

Окрім функціональних вимог, було визначено нефункціональні вимоги. Застосунок повинен мати простий інтерфейс, швидко відкривати основні

екрани, підтримувати мобільні пристрої, коректно зберігати дані користувача, захищати персональні дані, забезпечувати стабільну роботу основних сценаріїв і не перевантажувати користувача зайвими діями.

Нефункціональні вимоги можна сформулювати так:

– інтерфейс повинен бути зрозумілим для користувача без технічної підготовки;

– основні дії мають виконуватися за мінімальну кількість кроків;

– дані користувача повинні бути захищені через механізм авторизації;

– система повинна коректно обробляти помилки мережі.

Таким чином, функціональні та нефункціональні вимоги визначають загальну логіку розроблення застосунку. Основним критерієм якості системи є не лише наявність окремих функцій, а їх поєднання у цілісний сценарій: користувач додає рослину, отримує рекомендації, регулярно виконує догляд, бачить історію взаємодії та отримує додаткову мотивацію через ігрову модель [25, 26].

## 2.2 Загальна модель роботи мобільного застосунку

Загальна модель роботи мобільного застосунку MoriMori будується на взаємодії користувача, клієнтської частини, серверної частини, бази даних та зовнішніх сервісів. Користувач взаємодіє з інтерфейсом мобільного застосунку, мобільний застосунок обробляє локальний стан і надсилає запити до серверної частини, серверна частина виконує бізнес-логіку, звертається до бази даних або зовнішніх API, після чого повертає результат у мобільний застосунок [19–24, 27, 28].

У межах проєктування система поділяється на такі основні компоненти:

– користувацький інтерфейс;

– модуль керування станом;

– модуль навігації;

– модуль авторизації;

– модуль рослини;

– модуль рекомендацій;

– модуль нагадувань;

– модуль check-in;

– модуль rescue-режиму;

– модуль історії догляду;

– модуль гейміфікації;

– модуль маскота;

– серверна частина;

– база даних;

– зовнішні сервіси розпізнавання рослин;

– сервіси сповіщень.

На рисунку 2.1 доцільно подати загальну функціональну модель застосунку MoriMori. На рисунку треба показати користувача, мобільний застосунок, backend API, базу даних, сховище файлів, сервіс розпізнавання рослин і сервіс сповіщень. Основний потік даних: користувач взаємодіє з мобільним застосунком, застосунок надсилає HTTP-запит до backend, backend обробляє запит, звертається до бази даних або зовнішнього сервісу, після чого повертає відповідь у застосунок. Загальну функціональну модель мобільного застосунку MoriMori наведено на рисунку 2.1.

У проєктній моделі застосунок розглядається як система зі станами. Основними станами користувача є:

– неавторизований користувач;

– авторизований користувач без створеної рослини;

– користувач зі створеною рослиною;

– користувач у стані щоденного догляду;

– користувач у rescue-сценарії;

– користувач у режимі перегляду історії;

– користувач у режимі взаємодії з маскотом або кастомізації.

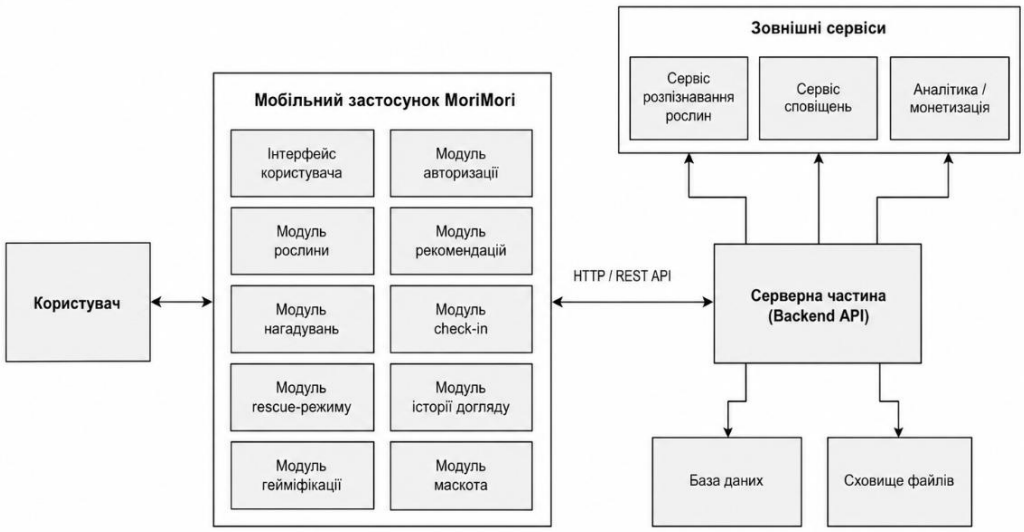


Рисунок 2.1 – Загальна функціональна модель мобільного застосунку MoriMori

У проєктній моделі застосунок розглядається як система зі станами. Основними станами користувача є:

– неавторизований користувач;

– авторизований користувач без створеної рослини;

– користувач зі створеною рослиною;

– користувач у стані щоденного догляду;

– користувач у rescue-сценарії;

– користувач у режимі перегляду історії;

– користувач у режимі взаємодії з маскотом або кастомізації.

Перехід між станами відбувається залежно від дій користувача та результатів оброблення даних. Наприклад, після авторизації користувач переходить до onboarding-сценарію. Після створення профілю рослини користувач отримує доступ до головного екрана. Після вибору варіанту «щось не так» у check-in система переводить користувача до rescue-режиму. У спрощеному вигляді повний цикл роботи застосунку можна подати так:

Крок 1. Користувач проходить авторизацію.

Крок 2. Користувач створює профіль рослини або додає фото для розпізнавання.

Крок 3. Система формує або отримує базові рекомендації щодо догляду.

Крок 4. Для рослини визначається дата наступного поливу.

Крок 5. Користувач отримує нагадування або самостійно відкриває застосунок.

Крок 6. Користувач виконує check-in і вказує стан рослини.

Крок 7. Якщо стан нормальний, система оновлює прогрес, історію та ігрові показники.

Крок 8. Якщо є проблема, система запускає rescue-режим.

Крок 9. Усі важливі дії зберігаються в історії догляду.

Крок 10. Маскот реагує на дії користувача та підтримує повторну взаємодію.

Така модель дозволяє поєднати інформаційні, функціональні та ігрові елементи в одному сценарії. Користувач отримує не лише довідкову інформацію, а й регулярний цикл взаємодії з рослиною.

## 2.3 Модель даних системи

Для роботи мобільного застосунку необхідно зберігати дані про користувача, рослину, рекомендації, нагадування, події догляду, check-in, rescue-сценарії та ігровий прогрес [23, 25]. Модель даних повинна бути достатньо простою для реалізації MVP, але водночас такою, що дозволяє розширювати застосунок у майбутньому.

Основними сутностями системи є User, Profile, Plant, CareRecommendation, CheckIn, TimelineEvent, RescueSession, RescueTask, NotificationSchedule, GamificationState, MascotState та ShopItem. Для забезпечення роботи мобільного застосунку було визначено набір основних

сутностей моделі даних. Їх призначення, ключові поля та зв’язки наведено в таблиці 2.2 та в таблиці 2.3.

Таблиця 2.2 – Основні сутності моделі даних мобільного застосунку MoriMori

| **Сутність** | **Призначення** | **Основні поля** |
|---|---|---|
| **1** | **2** | **3** |
| User | Зберігає базову інформацію про користувача системи. | user_id, email, auth_provider, created_at |
| Profile | Описує профіль користувача, його прогрес і персональні налаштування. | profile_id, user_id, display_name, level, xp, streak, created_at, updated_at |
| Plant | Зберігає основну інформацію про рослину користувача. | plant_id, user_id, name, species, avatar_index, photo_url, watering_interval_days, last_watered_at, plant_status, created_at |
| CareRecommendation | Містить рекомендації щодо догляду за конкретною рослиною. | recommendation_id, plant_id, light_level, watering_amount, watering_frequency, placement_advice, care_notes |
| CheckIn | Фіксує коротку регулярну взаємодію користувача з рослиною. | check_in_id, plant_id, user_id, mood_option, note, created_at |

Продовження таблиці 2.2

| 1 | 2 | 3 |
|---|---|---|
| TimelineEvent | Зберігає події історії догляду за рослиною. | event_id, plant_id, user_id, event_type, description, photo_url, created_at |
| RescueSession | Описує сесію допомоги рослині у випадку проблемного стану. | rescue_session_id, plant_id, status, symptoms, created_at, completed_at |
| RescueTask | Зберігає окрему дію, яку користувач має виконати в межах rescue-сесії. | task_id, rescue_session_id, title, description, is_completed, due_at |
| NotificationSchedule | Описує заплановані нагадування для користувача. | notification_id, plant_id, notification_type, scheduled_at, status |
| GamificationState | Зберігає ігровий прогрес користувача. | user_id, xp, level, streak, coins, milestones |
| MascotState | Описує поточний стан маскота та його налаштування. | mascot_id, user_id, mood, selected_skin, selected_accessory, last_reaction |

У межах MVP система може підтримувати одну активну рослину для користувача. Таке обмеження спрощує початкову реалізацію, оскільки головний екран, нагадування, check-in, rescue-режим і маскот працюють навколо однієї рослини.

Таблиця 2.3 – Зв’язки між сутностями моделі даних мобільного застосунку MoriMori

| **Сутність** | **Зв’язки** |
|---|---|
| User | Пов’язана з Profile та Plant. |
| Profile | Пов’язана з User, GamificationState та MascotState. |
| Plant | Пов’язана з User, CareRecommendation, CheckIn, TimelineEvent, RescueSession та NotificationSchedule. |
| CareRecommendation | Пов’язана з Plant. |
| CheckIn | Пов’язана з User, Plant та TimelineEvent. |
| TimelineEvent | Пов’язана з User та Plant. |
| RescueSession | Пов’язана з Plant та RescueTask. |
| RescueTask | Пов’язана з RescueSession. |
| NotificationSchedule | Пов’язана з Plant. |
| GamificationState | Пов’язана з Profile та MascotState. |
| MascotState | Пов’язана з Profile та GamificationState. |

User – сутність користувача. Основні поля: user_id, email, created_at, auth_provider. Зв’язана з Profile і Plant.

Profile – профіль користувача. Основні поля: profile_id, user_id, display_name, level, xp, streak, created_at, updated_at. Зв’язаний з User і GamificationState.

Plant – профіль рослини. Основні поля: plant_id, user_id, name, species, avatar_index, photo_url, watering_interval_days, last_watered_at, plant_status, created_at. Зв’язана з User, CheckIn, TimelineEvent, RescueSession і NotificationSchedule.

CareRecommendation – рекомендації щодо догляду. Основні поля: recommendation_id, plant_id, light_level, watering_amount, watering_frequency, placement_advice, care_notes. Зв’язана з Plant.

CheckIn – щоденна взаємодія користувача з рослиною. Основні поля: check_in_id, plant_id, user_id, mood_option, note, created_at. Зв’язана з Plant і TimelineEvent.

TimelineEvent – подія історії догляду. Основні поля: event_id, plant_id, user_id, event_type, description, photo_url, created_at. Зв’язана з Plant.

RescueSession – сесія допомоги рослині. Основні поля: rescue_session_id, plant_id, status, symptoms, created_at, completed_at. Зв’язана з Plant і RescueTask.

RescueTask – окрема задача в rescue-режимі. Основні поля: task_id, rescue_session_id, title, description, is_completed, due_at. Зв’язана з RescueSession.

NotificationSchedule – розклад нагадувань. Основні поля: notification_id, plant_id, notification_type, scheduled_at, status. Зв’язана з Plant.

GamificationState – ігровий стан користувача. Основні поля: user_id, xp, level, streak, coins, milestones. Зв’язана з Profile.

MascotState – стан маскота. Основні поля: mascot_id, user_id, mood, selected_skin, selected_accessory, last_reaction. Зв’язана з Profile і GamificationState.

У майбутньому модель можна розширити до підтримки кількох рослин шляхом додавання списку Plant, окремих нагадувань і окремих історій догляду для кожної рослини.

Зв’язки між сутностями можна описати так:

– один користувач має один профіль;

– один користувач може мати одну або кілька рослин;

– одна рослина має багато подій історії догляду;

– одна рослина має багато check-in записів;

– одна рослина може мати активну або завершену rescue-сесію;

– одна rescue-сесія має кілька rescue-задач;

– одна рослина має один або кілька записів нагадувань;

– один користувач має один ігровий стан;

– один користувач має один поточний стан маскота;

– один користувач може мати багато придбаних елементів кастомізації.

На рисунку 2.2 доцільно подати ER-модель даних застосунку MoriMori. Щоб не перевантажувати роботу, достатньо показати лише основні сутності: User, Profile, Plant, CheckIn, TimelineEvent, RescueSession, RescueTask, NotificationSchedule, GamificationState.

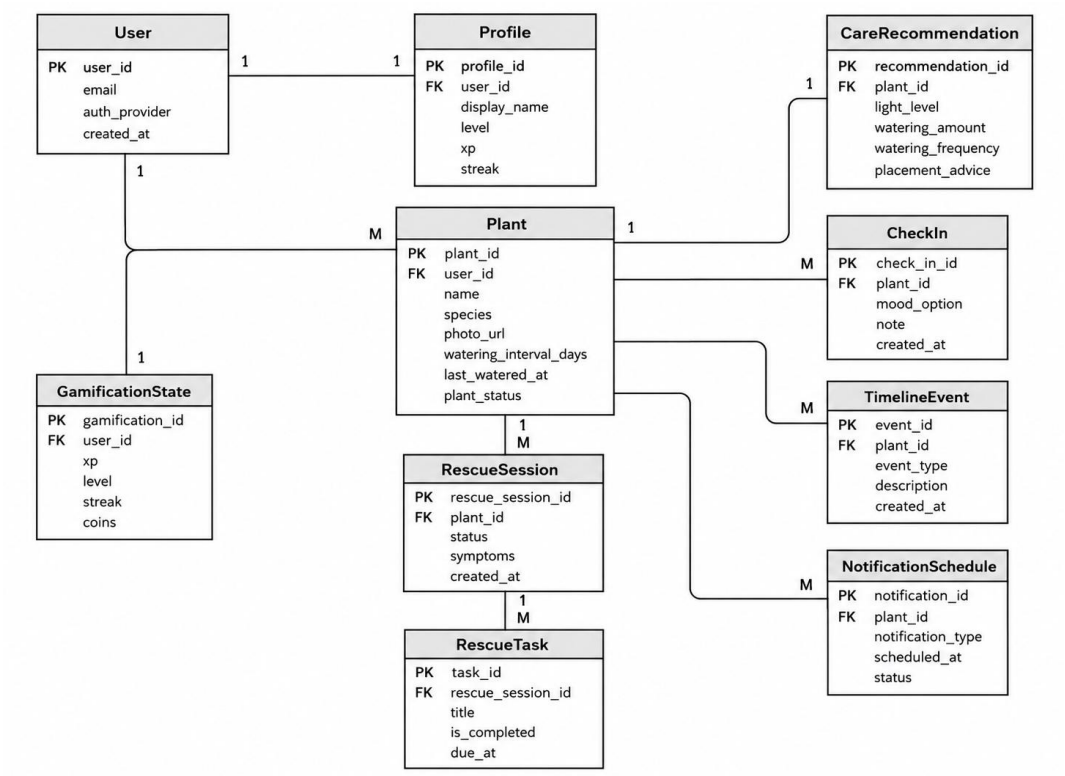


Рисунок 2.2 – Спрощена ER-модель даних мобільного застосунку MoriMori

Модель даних є основою для подальшого проєктування алгоритмів. Наприклад, дата останнього поливу та інтервал поливу використовуються для розрахунку наступного нагадування; check-in впливає на стан рослини та ігровий прогрес; rescue-сесія створюється на основі негативного або невпевненого стану рослини; timeline зберігає історію всіх важливих дій користувача.

## 2.4 Алгоритм створення профілю рослини

Створення профілю рослини є одним із ключових сценаріїв мобільного застосунку, оскільки без нього неможливо формувати персоналізовані рекомендації, нагадування, історію догляду та ігрову взаємодію [11, 12, 27]. Профіль рослини містить основну інформацію, яка потрібна для подальшої роботи системи: назву, вид, фото, інтервал поливу, дату останнього поливу, умови розміщення та початковий стан.

Алгоритм створення профілю рослини може працювати у двох режимах:

– ручне створення профілю;

– створення профілю з використанням розпізнавання рослини за фото.

У ручному режимі користувач самостійно вводить назву рослини або обирає тип із запропонованого списку. У режимі розпізнавання користувач додає фото, система визначає ймовірний вид рослини, після чого користувач може підтвердити або змінити результат.

Алгоритм створення профілю рослини має такий вигляд:

Крок 1. Користувач відкриває onboarding-сценарій.

Крок 2. Система пропонує створити першу рослину.

Крок 3. Користувач вводить ім’я рослини або обирає варіант розпізнавання за фото.

Крок 4. Якщо користувач додає фото, система передає зображення на оброблення.

Крок 5. Система отримує результат розпізнавання або дозволяє користувачу ввести дані вручну.

Крок 6. Користувач уточнює базові параметри: місце розташування, умови освітлення, дату останнього поливу, бажаний інтервал поливу.

Крок 7. Система перевіряє коректність введених даних.

Крок 8. Якщо дані коректні, створюється профіль рослини.

Крок 9. На основі параметрів рослини формується дата наступного поливу.

Крок 10. Створюється перший запис у timeline про додавання рослини.

Крок 11. Користувач переходить на головний екран застосунку.

Для забезпечення коректності роботи алгоритму вводяться такі умови:

– назва рослини не повинна бути порожньою;

– дата останнього поливу не може бути пізнішою за поточну дату;

– інтервал поливу повинен бути додатним числом;

– якщо інтервал поливу не визначено, використовується стандартне значення;

– якщо користувач не додав фото, профіль створюється без зображення;

– якщо система не змогла розпізнати рослину, користувач повинен мати можливість ввести дані вручну;

– якщо у користувача вже є активна рослина в MVP-версії, система повинна або оновити її, або попередити про обмеження.

Для MVP доцільно прийняти базовий інтервал поливу за замовчуванням:

$$I_0 = 7, \tag{2.1}$$

де $I_0$ – стандартний інтервал поливу, який використовується, якщо для рослини немає точних даних.

Якщо для рослини доступні рекомендації з бази або зовнішнього сервісу, інтервал поливу визначається не за замовчуванням, а на основі отриманої інформації:

$$I_W = I_{rec}, \tag{2.2}$$

де $I_w$ – інтервал поливу для конкретної рослини;

$I_{rec}$ – рекомендований інтервал поливу, отриманий із джерела даних.

Якщо рекомендація відсутня, використовується стандартне значення:

$$I_W = I_0 . \tag{2.3}$$

Приклад розрахунку. Якщо користувач додав рослину, для якої немає точних даних про полив, система встановлює $I_w$ = 7 днів. Якщо останній полив було виконано 01.05.2026, то наступна рекомендована дата поливу:

$T_{next}$ = 01.05.2026 + 7 днів = 08.05.2026.

Формально дата наступного поливу визначається так:

$$I_W = I_0 , \tag{2.4}$$

де $T_{next}$ – дата наступного поливу;

$T_{last}$ – дата останнього поливу;

$I_w$ – інтервал поливу у днях.

На рисунку 2.3 подана блок-схема алгоритму створення профілю рослини, яка детально відображає логіку початкової взаємодії користувача з системою. Схема демонструє розгалуження процесу на вибір між ручним введенням параметрів та автоматичним розпізнаванням виду за фотографією. Після цього алгоритм передбачає обов'язкову перевірку коректності даних, безпосереднє створення профілю в базі та автоматичне формування індивідуального календаря нагадувань, завершуючись переходом на головний екран.

Таким чином, цей алгоритм забезпечує базову персоналізацію застосунку під конкретного користувача та його зелені насадження. Отримані відомості надалі динамічно використовуються для генерації рекомендацій, автоматичного надсилання нагадувань, ведення хронологічної історії догляду, фіксації щоденних відміток (check-in) та запуску елементів гейміфікації.

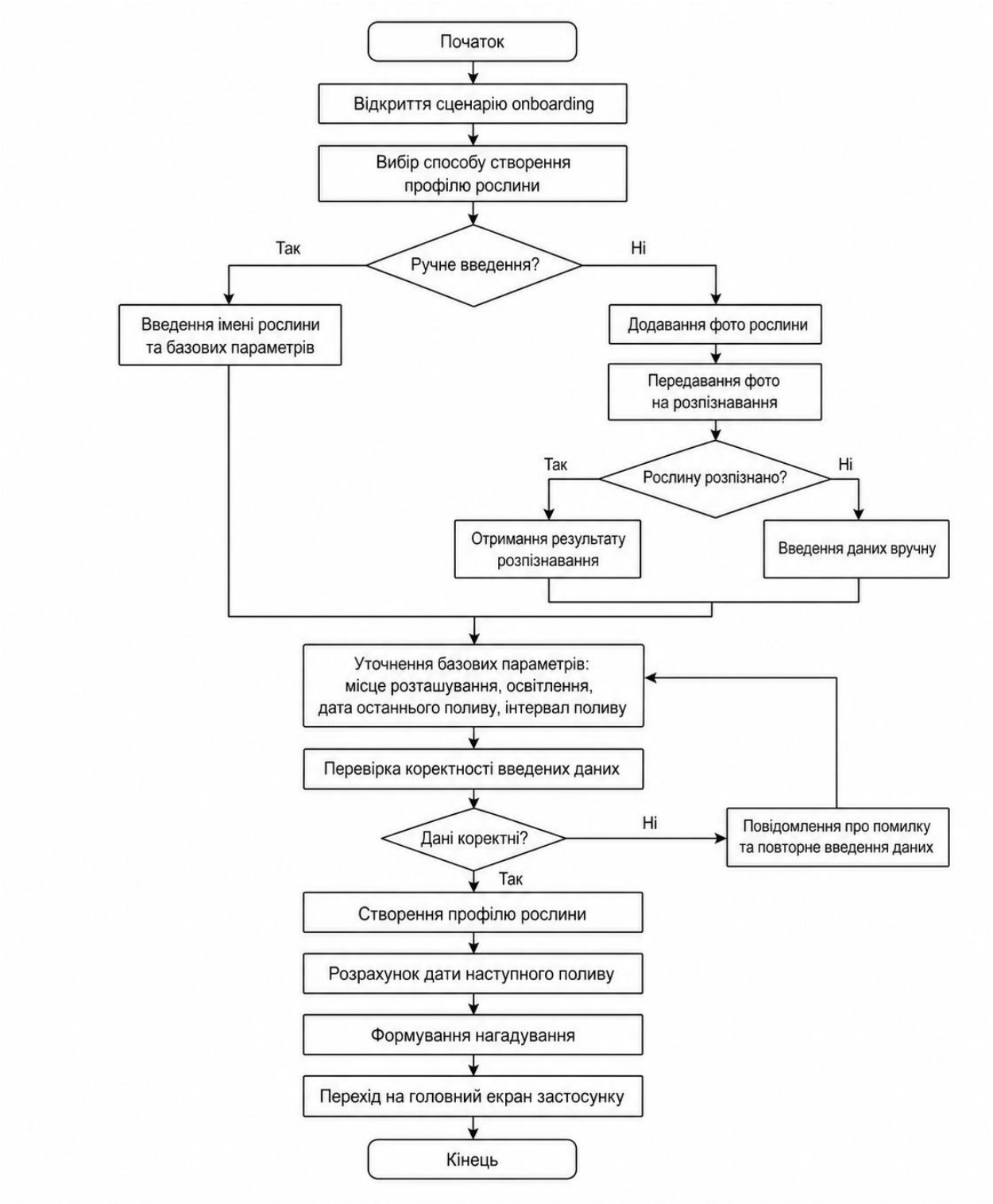


Рисунок 2.3 – Алгоритм створення профілю рослини в мобільному застосунку MoriMori

## 2.5 Алгоритм розпізнавання рослини за зображенням

Розпізнавання рослини за зображенням призначене для зниження вхідного бар’єра для користувача. Якщо користувач не знає точну назву рослини, він може зробити фото або завантажити зображення з галереї. Система аналізує зображення та повертає ймовірний результат ідентифікації. У межах цієї роботи не ставиться задача навчання власної моделі машинного навчання з нуля. Такий підхід потребував би великого набору зображень, ручної розмітки, навчання, валідації та оцінювання точності моделі. Для розроблюваного мобільного застосунку доцільніше використати готовий сервіс або зовнішнє API розпізнавання рослин [9, 29, 30]. Основна задача полягає в інтеграції цього модуля у користувацький сценарій застосунку. Алгоритм розпізнавання рослини має такий вигляд:

Крок 1. Користувач обирає функцію розпізнавання рослини.

Крок 2. Користувач робить фото або завантажує зображення з галереї.

Крок 3. Система перевіряє наявність зображення та його формат.

Крок 4. Зображення передається на серверну частину або безпосередньо до сервісу розпізнавання.

Крок 5. Сервіс розпізнавання повертає список можливих варіантів рослини з імовірностями.

Крок 6. Система вибирає результат із найбільшою імовірністю.

Крок 7. Якщо імовірність достатня, результат пропонується користувачу для підтвердження.

Крок 8. Якщо імовірність середня, система показує кілька варіантів.

Крок 9. Якщо імовірність низька, система пропонує повторити фото або ввести дані вручну.

Крок 10. Після підтвердження результату дані зберігаються у профілі рослини. Результат роботи модуля розпізнавання можна подати у вигляді множини:

$$R = \{r_{1,2}, ..., r_n\}, \quad (2.5)$$

де $R$ – множина можливих результатів розпізнавання;

$r_i$ – окремий результат розпізнавання;

$n$ – кількість варіантів, отриманих від сервісу.

Кожен результат $r_i$ має назву рослини та значення імовірності:

$$r_i = \{name_i, p_i\}, \quad (2.6)$$

де *name* – назва рослини;

$p_i$ – імовірність відповідності зображення цьому виду рослини.

Основним результатом вважається варіант із найбільшою імовірністю:

$$r_{best} = \arg\max p_i. \quad (2.7)$$

Для прийняття рішення щодо якості результату вводяться порогові значення. У межах MVP можна використати такі умови:

– якщо $r_{best}$ >= 0,70, результат вважається достатньо надійним;

– якщо 0,40 <= $r_{best}$ < 0,70, результат потребує підтвердження користувача;

– якщо $r_{best}$ < 0,40, результат вважається недостатньо надійним.

Ці пороги не є біологічною або науковою оцінкою виду рослини, а використовуються як прикладна логіка інтерфейсу для прийняття рішення про подальшу дію користувача.

Приклад розрахунку. Нехай сервіс розпізнавання повернув три варіанти:

– $r_1$ = *{Monstera deliciosa; 0,82};*

– $r_2$ = *{Philodendron; 0,11};*

– $r_3$ = *{Epipremnum aureum; 0,07}.*

Тоді:

$r_{best} = 0{,}82$.

Оскільки 0,82 >= 0,70, система показує користувачу Monstera deliciosa як основний результат і пропонує підтвердити його.

Інший приклад. Якщо $p_{best} = 0{,}53$, то результат не є достатньо надійним. У такому випадку система повинна показати кілька найімовірніших варіантів і надати користувачу можливість вибору. Якщо $r_{best} = 0{,}25$, система повинна запропонувати повторити фото або ввести назву рослини вручну.

Для зменшення повторної обробки однакових фото може використовуватися хеш зображення:

$$H_{photo} = SHA256(B_{photo}), \tag{2.8}$$

де $H_{photo}$ – хеш зображення;

$B_{photo}$ – байтове представлення фотографії.

Якщо фото з таким хешем уже було оброблене, система може використати збережений результат замість повторного звернення до зовнішнього сервісу. Це дозволяє зменшити кількість зайвих запитів і підвищити ефективність роботи системи. Для коректної роботи алгоритму розпізнавання визначено такі умови:

– зображення повинно бути у підтримуваному форматі;

– фото не повинно бути порожнім або пошкодженим;

– бажано, щоб на фото була одна рослина;

– якість фото повинна дозволяти побачити листя, стебло або інші характерні ознаки;

– для розпізнавання потрібне інтернет-з’єднання;

– користувач завжди повинен мати можливість змінити або уточнити результат.

Обмеження алгоритму пов’язані з тим, що якість розпізнавання залежить від освітлення, ракурсу, стану рослини, наявності фону, кількості рослин на зображенні та схожості видів. Тому результат розпізнавання не повинен автоматично вважатися остаточним без можливості підтвердження користувачем.

Таким чином, алгоритм розпізнавання рослини виконує допоміжну роль у загальній системі. Він спрощує створення профілю рослини, але не замінює ручне уточнення даних користувачем.

## 2.6 Модель формування рекомендацій щодо догляду

Рекомендації щодо догляду є основою практичної цінності мобільного застосунку. Користувач повинен отримувати зрозумілу інформацію про те, як доглядати за конкретною рослиною, коли її поливати, де краще розмістити та як реагувати на зміну стану.

У межах застосунку рекомендації формуються на основі таких даних:

– вид або тип рослини;

– дата останнього поливу;

– рекомендований інтервал поливу;

– місце розміщення рослини;

– умови освітлення;

– поточний стан рослини;

– історія взаємодії користувача з рослиною;

– результат check-in;

– наявність активної rescue-сесії.

Основним параметром для розрахунку нагадування є інтервал поливу. Він може бути отриманий із зовнішнього джерела даних, визначений за типом рослини або встановлений користувачем вручну [27].

Кількість днів після останнього поливу визначається за формулою:

$$D = floor((T_{now} - T_{last}) / 1день), \quad (2.9)$$

де $D$ – кількість повних днів після останнього поливу;

$T_{now}$ – поточна дата;

$T_{last}$ – дата останнього поливу;

*floor* – функція округлення до меншого цілого.

Кількість днів до наступного поливу:

$$D_{left} - I_W - D, \quad (2.10)$$

де $D_{left}$ – кількість днів до наступного поливу.

Якщо $D_{left} > 0$, полив ще не потрібен. Якщо $D_{left} = 0$, полив рекомендовано виконати сьогодні. Якщо $D_{left} < 0$, полив прострочено.

Для відображення умовного рівня води в інтерфейсі використовується нормоване значення:

$$W = \max(0, \min(1, 1 - d / I_w)), \quad (2.11)$$

де $W$ – умовний рівень води;

$D$ – кількість днів після останнього поливу;

$I_w$ – інтервал поливу.

Значення $W$ належить до діапазону від 0 до 1. Якщо $W$ = 1, рослину нещодавно поливали. Якщо $W$ наближається до 0, система повинна повідомити користувача про необхідність поливу.

Рівень терміновості поливу можна визначити так:

$$U = \max(0, \min(1, D / I_W)), \quad (2.12)$$

де $U$ – рівень терміновості поливу.

Якщо $U < 0{,}7$, полив не є терміновим. Якщо $0{,}7 \leq U < 1$, система може показати м’яке попередження. Якщо $U \geq 1$, система повинна показати, що полив уже потрібен.

Приклад розрахунку. Нехай останній полив був 4 дні тому, а інтервал поливу становить 7 днів.

$D = 4$;

$I_W = 7$.

Тоді умовний рівень води:

$W = 1 - 4 / 7 = 0{,}43$.

Рівень терміновості:

$U = 4 / 7 = 0{,}57$.

Оскільки $U < 0{,}7$, система може показати, що рослина ще не потребує термінового поливу, але рівень води вже зменшується.

Інший приклад. Якщо $D = 8$, $I_W = 7$, тоді:

$W = max(0, 1 - 8 / 7) = 0$;

$U = min(1, 8 / 7) = 1$.

У цьому випадку система повинна показати, що полив прострочено.

Окрім поливу, рекомендації можуть містити інформацію про освітлення. У MVP-версії освітлення можна задавати через прості категорії:

– низький рівень освітлення;

– середній рівень освітлення;

– яскраве розсіяне світло;

– пряме сонячне світло.

Для кожної рослини можна зберігати рекомендований рівень освітлення $L_{rec}$ і фактичний рівень $L_{fact}$, який вводить користувач або визначає за допомогою допоміжного сценарію. Якщо $L_{fact}$ не відповідає $L_{rec}$, система формує рекомендацію щодо зміни розміщення рослини.

У спрощеному вигляді перевірку освітлення можна подати так:

$$C_{light} = 0, якщо\, L_{fact} \neq L_{rec}, \quad (2.13)$$

де $C_{light}$ – ознака відповідності фактичних умов освітлення рекомендованим.

У практичній реалізації це правило може бути не бінарним, а категоріальним. Наприклад, якщо рослина потребує яскравого розсіяного світла, а користувач вказав середній рівень освітлення, система може не показувати критичну помилку, а лише рекомендувати поставити рослину ближче до вікна.

Загальний стан рослини в системі може набувати таких значень:

– healthy – рослина в нормальному стані;

– thirsty – рослина потребує поливу;

– struggling – користувач повідомив про проблему;

– recovering – рослина перебуває у rescue-режимі;

– unknown – стан не визначено.

Стан рослини визначається на основі водного показника, результатів check-in і наявності rescue-сесії. Спрощене правило визначення стану:

– якщо активна rescue-сесія існує, status = recovering;

– інакше якщо користувач обрав «щось не так», status = struggling;

– інакше якщо $W < 0{,}25$, status = thirsty;

– інакше status = healthy.

Таке правило не є біологічною діагностикою, але воно достатнє для інтерфейсної логіки MVP. Його задача – допомогти користувачу швидко зрозуміти, що відбувається з рослиною, і яку дію потрібно виконати.

## 2.7 Модель нагадувань про полив і повторну взаємодію

Нагадування є одним із ключових механізмів підтримки регулярного догляду за рослиною. У застосунку нагадування повинні бути пов’язані не з

довільним календарем, а з параметрами конкретної рослини та історією дій користувача.

Основним типом нагадування є нагадування про полив. Воно формується на основі дати останнього поливу та рекомендованого інтервалу поливу.

Дата нагадування про полив визначається так:

$$T_{re\min der} = T_{last} + I_w, \tag{2.14}$$

де $T_{re\min der}$ – дата нагадування;

$T_{last}$ – дата останнього поливу;

$I_w$ – інтервал поливу.

Якщо користувач виконує полив, дата останнього поливу оновлюється, а система планує нове нагадування:

$$T_{last\_new} = T_{now}. \tag{2.15}$$

$$T_{re\min der,new} = T_{last,new} + I_w. \tag{2.16}$$

Якщо користувач не виконує полив у рекомендований день, система може показати повторне нагадування. Щоб не створювати надмірного тиску на користувача, повторні нагадування повинні бути обмежені. Наприклад, достатньо одного додаткового нагадування наступного дня.

Для повернення користувача до застосунку після тривалої неактивності використовується затримка повторної взаємодії:

$$D_{re} = \max(round(1{,}5 * I_W), 5), \tag{2.17}$$

де $D_{re}$ – кількість днів до нагадування про повернення;

$I_W$ – інтервал поливу.

Це правило означає, що система не повинна турбувати користувача занадто часто. Якщо рослина потребує поливу кожні 2 дні, мінімальна затримка повернення все одно становитиме 5 днів. Якщо рослина потребує поливу кожні 10 днів, затримка повернення становитиме 15 днів.

Отже, якщо користувач не взаємодіє із застосунком протягом 11 днів, система може надіслати м’яке повідомлення про повернення.

Для застосунку MoriMori доцільно передбачити такі типи нагадувань:

– watering_reminder – нагадування про полив;

– check_in_reminder – нагадування про коротку перевірку стану рослини;

– rescue_follow_up – нагадування про виконання задач rescue-режиму;

– reengagement_reminder – нагадування про повернення після неактивності;

– meditation_reminder – м’яке нагадування про коротку спокійну взаємодію з рослиною, якщо користувач сам увімкнув таку опцію.

Для кожного нагадування потрібно визначити:

– тип нагадування;

– час запланованого показу;

– пов’язану рослину;

– статус нагадування;

– канал показу;

– умову скасування або перенесення.

Умови роботи нагадувань:

– якщо користувач виконав полив раніше запланованої дати, старе нагадування скасовується;

– якщо користувач змінив інтервал поливу, нагадування перераховується;

– якщо користувач вимкнув дозвіл на сповіщення, застосунок показує нагадування лише всередині інтерфейсу;

– якщо немає інтернет-з’єднання, локальні нагадування можуть працювати, але push-сповіщення або синхронізація можуть бути недоступними;

– якщо рослина перебуває в rescue-режимі, пріоритет можуть мати rescue-нагадування;

– якщо користувач видалив рослину, всі пов’язані нагадування мають бути скасовані.

На рисунку 2.4 доцільно подати алгоритм планування нагадування про полив. У схемі треба показати отримання дати останнього поливу, визначення інтервалу, розрахунок дати наступного поливу, перевірку дозволів на сповіщення, планування нагадування та оновлення після виконання поливу.

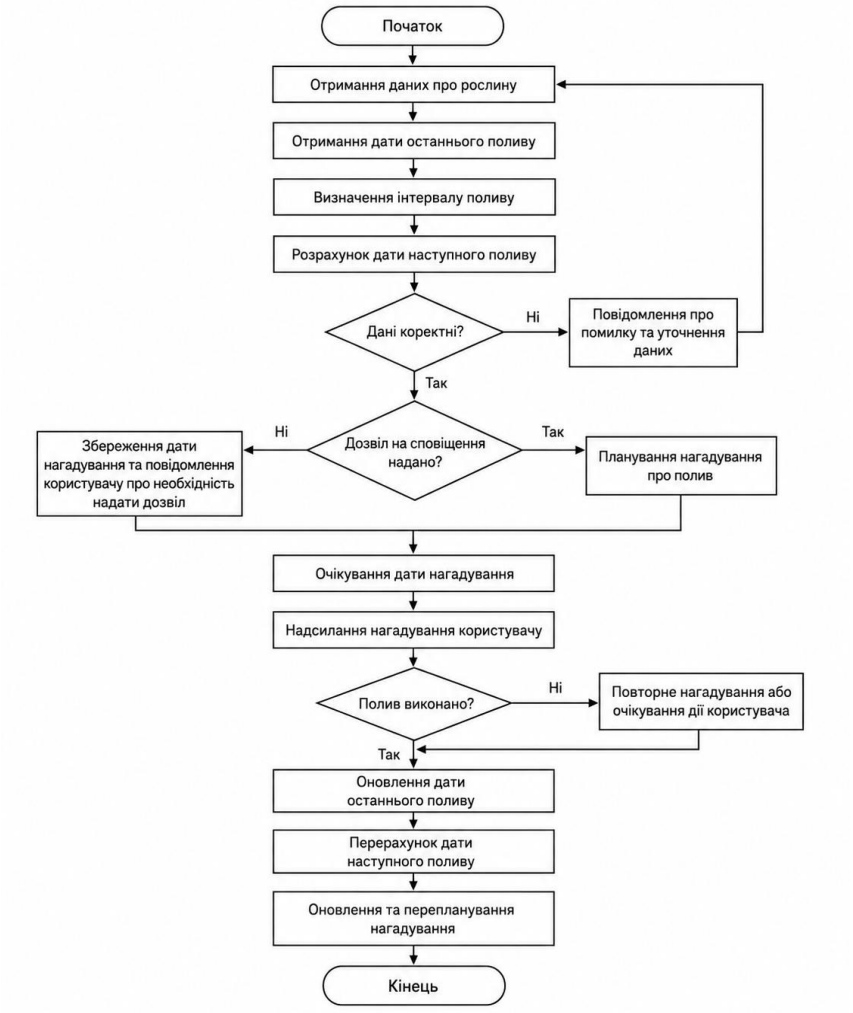


Рисунок 2.4 – Алгоритм планування нагадування про полив

Модель нагадувань повинна бути обережною, оскільки надмірна кількість сповіщень може знизити бажання користувача відкривати застосунок [11, 12, 24, 26, 29]. Тому нагадування мають бути прив’язані до реальних дій догляду та не дублюватися без потреби.

## 2.8 Модель check-in взаємодії користувача з рослиною

Check-in є короткою регулярною взаємодією користувача із застосунком, під час якої користувач повідомляє про поточний стан рослини. Ця механіка потрібна для того, щоб догляд не обмежувався лише автоматичними нагадуваннями. Користувач повинен мати простий спосіб регулярно повертатися до рослини, оцінювати її стан і фіксувати важливі зміни.

У застосунку передбачено три основні варіанти check-in:

– рослина почувається добре;

– з рослиною щось не так;

– користувач не впевнений у стані рослини.

Кожен варіант запускає окрему гілку логіки.

Якщо користувач обирає варіант «рослина почувається добре», система створює позитивний check-in, додає запис до історії догляду, оновлює streak і нараховує досвід.

Якщо користувач обирає варіант «з рослиною щось не так», система створює check-in із проблемним станом, оновлює статус рослини та пропонує перейти до rescue-режиму.

Якщо користувач обирає варіант “не впевнений”, система може поставити додаткові уточнювальні питання або запропонувати зробити фото рослини.

Алгоритм check-in має такий вигляд:

Крок 1. Користувач відкриває головний екран.

Крок 2. Система показує стан рослини та кнопку check-in.

Крок 3. Користувач обирає один із трьох варіантів стану.

Крок 4. Система створює запис check-in.

Крок 5. Система додає подію до timeline.

Крок 6. Якщо стан позитивний, оновлюється XP, streak і настрій маскота.

Крок 7. Якщо стан проблемний, запускається rescue-режим.

Крок 8. Якщо користувач не впевнений, система пропонує уточнення або фото.

Крок 9. Інтерфейс оновлює стан рослини.

Крок 10. Користувач повертається на головний екран.

Стан check-in можна подати як змінну C:

$$C \in \{good, problem, uncertain\}, \tag{2.18}$$

де *good* – рослина у нормальному стані;

*problem* – користувач повідомляє про проблему;

*uncertain* – користувач не впевнений у стані рослини.

Залежно від значення C система виконує різні дії:

– якщо $C = good$, то $status = healthy$;

– якщо $C = problem$, то $status = struggling$;

– якщо $C = uncertain$, то $status = unknown$ або система переходить до уточнення.

Для гейміфікації позитивний check-in може додавати досвід:

$$XP_{new} = XP_{old} + XP_{check}, \tag{2.19}$$

де $XP_{new}$ – нове значення досвіду;

$XP_{old}$ $XP_{new}$ – попереднє значення досвіду;

$XP_{check}$ $XP_{new}$ – кількість досвіду за виконання check-in.

$XP_{check}$ = 5 для звичайного check-in.

$XP_{water}$ = 10 для поливу.

$XP_{prescue}$ $XP_{rescue}$ = 15 для виконання rescue-задачі.

$XP_{meditation}$ = 3 для короткої медитації з рослиною.

Ці значення можуть бути змінені під час подальшого тестування, але для MVP вони дозволяють побудувати зрозумілу модель прогресу . Streak-механіка визначає кількість послідовних днів активності користувача.

Ця логіка дозволяє підтримувати регулярність, але не створює складної системи покарань. Якщо користувач пропустив день, застосунок не повинен агресивно карати його, а лише м’яко починати нову серію.

На рисунку 2.5 доцільно подати модель check-in взаємодії. У схемі треба показати три гілки: позитивний стан, проблемний стан і невпевненість. Позитивний стан веде до XP і timeline, проблемний $XP_{new}$ – до rescue-режиму, невпевненість $XP_{new}$ – до уточнювальних питань або фото.

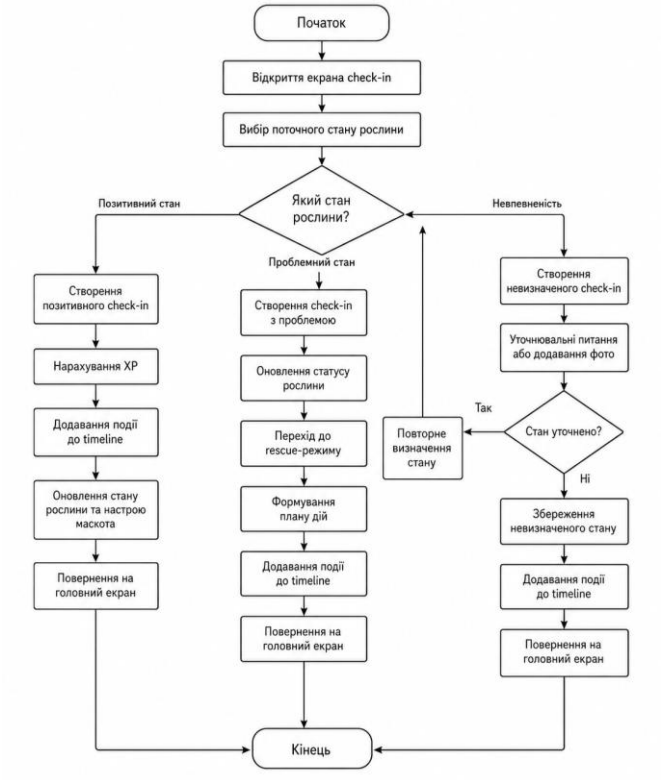


Рисунок 2.5 – Модель check-in взаємодії користувача з рослиною

Check-in є центральною поведінковою механікою застосунку. Саме він перетворює догляд із пасивного очікування нагадувань на активну взаємодію користувача з рослиною [11–13, 29].

## 2.9 Модель rescue-режиму

Rescue-режим призначений для ситуацій, коли користувач помічає проблему з рослиною або не впевнений у її стані. Основна задача rescue-режиму – провести користувача через простий сценарій уточнення проблеми та запропонувати базовий план дій.

Rescue-режим не є повноцінною медичною або ботанічною діагностикою. Він виконує роль цифрового помічника, який допомагає структурувати проблему та запропонувати перші кроки. Це важливо для новачків, які часто не знають, як реагувати на жовте листя, сухий ґрунт, надмірну вологість або зміну зовнішнього вигляду рослини.

Основними вхідними даними для rescue-режиму є:

– результат check-in;
– симптоми, які обрав користувач;
– фото рослини, якщо воно додане;
– інформація про останній полив;
– інтервал поливу;
– поточний стан рослини;
– історія останніх подій догляду.

Симптоми можуть включати такі варіанти:

– жовтіє листя;
– листя сохне;
– листя в’яне;
– ґрунт занадто сухий;
– ґрунт занадто вологий;

– з’явилися плями;

– рослина повільно росте;

– користувач не може визначити проблему.

Алгоритм rescue-режиму має такий вигляд:

Крок 1. Користувач повідомляє, що з рослиною є проблема.

Крок 2. Система створює rescue-сесію.

Крок 3. Користувач обирає симптоми або додає фото.

Крок 4. Система аналізує симптоми та дані про догляд.

Крок 5. Формується перелік можливих причин.

Крок 6. Система створює набір простих задач для користувача.

Крок 7. Користувач виконує задачі та відмічає їх виконання.

Крок 8. Події зберігаються в timeline.

Крок 9. Після виконання задач система пропонує повторний check-in.

Крок 10. Rescue-сесія завершується або залишається активною.

Статус rescue-сесії може набувати таких значень:

$$R_{status} \in \{active, completed, cancelled\}, \tag{2.20}$$

де *active* – rescue-сесія активна;

*completed* – користувач завершив план дій;

*cancelled* – сесію скасовано.

Для формування задач можна використати просту систему правил. Наприклад:

– якщо симптом = «ґрунт сухий» і $W < 0{,}25$, тоді запропонувати полив;

– якщо симптом = «ґрунт вологий» і $D < I_w / 2$, тоді запропонувати перевірити дренаж і не поливати найближчим часом;

– якщо симптом = «жовтіє листя» і ґрунт вологий, тоді запропонувати зменшити частоту поливу;

– якщо симптом = «листя сохне» і рослина стоїть під прямим сонцем, тоді запропонувати змінити розміщення.

Формально правило можна подати так:

$$if\ Condition = true, then\ task_i \in \mathrm{Re}\, scuePlan, \tag{2.21}$$

де *condition* – умова, яка перевіряється системою;

$task_i$ – задача, що додається до плану rescue;

*RescuePlan* – множина задач rescue-сценарію.

Rescue-план можна подати як множину задач:

$$P = \{t_1, t_2, ..., t_k\}, \tag{2.22}$$

де $P$ – план дій;

$t_i$ – окрема задача;

$k$ – кількість задач у плані.

Кожна задача має опис, статус і рекомендований час виконання:

$$t_i = \{title_i, description_i, status_i, due_{at,i}\}. \tag{2.23}$$

Для контролю виконання rescue-плану використовується коефіцієнт завершення:

$$Q = C_{done} / C_{all}, \tag{2.24}$$

де $Q$ – частка виконаних задач;

$C_{done}$ – кількість виконаних задач;

*Call Call* – загальна кількість задач.

Якщо $Q = 1$, rescue-план виконано повністю. Якщо $Q = 0$, жодну задачу не виконано. Якщо $0 < Q < 1$, план виконано частково.

Умови роботи rescue-режиму:

– rescue-сесія створюється лише для існуючої рослини;

– одночасно для однієї рослини бажано мати одну активну rescue-сесію;

– користувач може скасувати rescue-сесію;

– виконання rescue-задач повинно зберігатися в історії;

– після завершення rescue-сесії система повинна запропонувати повторний check-in;

– якщо даних недостатньо, система повинна пропонувати загальні безпечні дії, а не категоричні висновки.

Rescue-режим важливий для практичної цінності застосунку, оскільки користувач отримує не лише нагадування, а й підтримку в проблемних ситуаціях. Це посилює відмінність MoriMori від звичайного календаря поливу.

## 2.10 Ігрова модель взаємодії користувача з маскотом

Ігрова модель взаємодії у застосунку MoriMori використовується для підвищення регулярності користування, формування емоційного зв’язку та підтримки мотивації користувача [7, 14, 15]. Її задача полягає не в тому, щоб перетворити застосунок на повноцінну гру, а в тому, щоб зробити догляд за рослиною більш приємним і регулярним.

Основними елементами ігрової моделі є:

– маскот;

– досвід;

– рівні;

– streak;

– milestones;

– реакції маскота;

– коротка медитація з рослиною.

Маскот виконує роль візуального та комунікаційного посередника між користувачем і системою. Він може реагувати на стан рослини, виконані дії, пропуски догляду або завершення rescue-задач. Наприклад, якщо користувач виконав check-in, маскот може показати позитивну реакцію. Якщо рослина потребує поливу, маскот може м’яко звернути увагу користувача на це.

Досвід користувача формується на основі корисних дій. Загальне значення досвіду можна визначити так:

$$XP_{total} = XP_0 + \Sigma XP_i, \tag{2.25}$$

де $XP_{total}$ – загальна кількість досвіду;

$XP_0$ – початкове значення досвіду;

$XP_i$ – досвід, отриманий за окрему дію.

Для MVP можна використати такі значення досвіду:

– check-in – 5 XP;

– полив – 10 XP;

– додавання фото до timeline – 5 XP;

– виконання rescue-задачі – 15 XP;

– коротка медитація з рослиною – 3 XP;

– завершення onboarding – 20 XP.

Рівень користувача визначається за формулою:

$$Level = floor(XP_{total} / 100) + 1, \tag{2.26}$$

де *Level* – рівень користувача;

$XP_{total}$ – загальна кількість досвіду.

Прогрес до наступного рівня:

$$P_{Level} = (XP_{total} \bmod 100) / 100, \quad (2.27)$$

де $P_{Level}$ – прогрес до наступного рівня;

*mod* – операція знаходження остачі від ділення.

Streak використовується для відображення регулярності взаємодії. Він не повинен бути агресивною механікою покарання, оскільки догляд за рослинами має залишатися спокійним процесом. Тому streak краще використовувати як позитивне підкріплення, а не як джерело тиску.

Стан маскота можна визначати на основі стану рослини та дій користувача:

$$M_{mood} \in \{happy, calm, thirsty, worried, proud\}, \quad (2.28)$$

де $M_{mood}$ – поточний настрій маскота.

Правила зміни настрою маскота:

– якщо рослина у нормальному стані та користувач виконав check-in, $M_{mood} = happy$;

– якщо користувач запустив медитацію з рослиною, $M_{mood} = calm$;

– якщо $W < 0{,}25$, $M_{mood} = thirsty$;

– якщо активний rescue-режим, $M_{mood} = worried$;

– якщо користувач завершив важливу дію або milestone, $M_{mood} = proud$.

Milestone – це важлива подія довгострокової взаємодії. Наприклад:

– перша додана рослина;

– перший check-in;

– перший тиждень догляду;

– перший завершений rescue-план;

– 10 подій у timeline;

– 7 днів активності.

Milestone може нараховувати користувачу додатковий досвід або відкривати косметичний елемент. Важливо, щоб винагорода була пов’язана з реальною дією догляду, а не існувала окремо від основної функції застосунку.

Медитація з рослиною є окремим спокійним сценарієм. Вона може працювати як таймер короткої взаємодії, наприклад 1, 3 або 5 хвилин. Її задача – не стимулювати користувача до активної гри, а створити короткий момент уважності до рослини [16, 17]. Після завершення таймера система може додати невелику кількість XP і створити подію в timeline.

Умови ігрової моделі:

– ігрові елементи не повинні ускладнювати основні сценарії догляду;

– винагорода повинна нараховуватися за корисні дії;

– маскот повинен підтримувати користувача, а не створювати відчуття покарання;

– кастомізація повинна бути додатковою, а не обов’язковою;

– система рівнів повинна бути простою і зрозумілою;

– медитація повинна бути спокійною, без змагального тиску.

Таким чином, ігрова модель взаємодії посилює основну мету застосунку: підтримати регулярний догляд за рослиною. Маскот, XP, рівні, streak створюють додаткову мотивацію, але не замінюють практичну функцію догляду.

## 2.11 Узагальнений алгоритм роботи системи

Після проєктування окремих моделей доцільно описати узагальнений алгоритм роботи мобільного застосунку. Він показує, як усі елементи системи поєднуються в один користувацький сценарій.

Узагальнений алгоритм має такий вигляд:

Крок 1. Користувач відкриває мобільний застосунок.

Крок 2. Система перевіряє стан авторизації.

Крок 3. Якщо користувач не авторизований, система відкриває екран входу або реєстрації.

Крок 4. Якщо користувач авторизований, система перевіряє наявність активної рослини.

Крок 5. Якщо активної рослини немає, користувач проходить onboarding і створює профіль рослини.

Крок 6. Якщо рослина існує, система відкриває головний екран.

Крок 7. Система розраховує умовний рівень води, дату наступного поливу та стан рослини.

Крок 8. Система показує користувачу стан рослини, маскота, прогрес і швидкі дії.

Крок 9. Користувач може виконати одну з дій: полив, check-in, перегляд рекомендацій, додавання події, rescue-режим, медитація або кастомізація.

Крок 10. Після кожної дії система оновлює дані рослини, timeline, ігровий прогрес і стан маскота.

Крок 11. Якщо дія впливає на полив, система перераховує дату наступного нагадування.

Крок 12. Якщо користувач повідомляє про проблему, система запускає rescue-режим.

Крок 13. Якщо rescue-план виконано, система оновлює статус рослини та пропонує повторний check-in.

Крок 14. Дані синхронізуються із серверною частиною.

Крок 15. Користувач повертається до головного екрана.

На рисунку 2.6 доцільно подати узагальнений алгоритм роботи мобільного застосунку MoriMori. Щоб не перевантажувати роботу, достатньо

показати основні блоки: авторизація, onboarding, профіль рослини, рекомендації, нагадування, check-in, rescue-режим, timeline, гейміфікація.

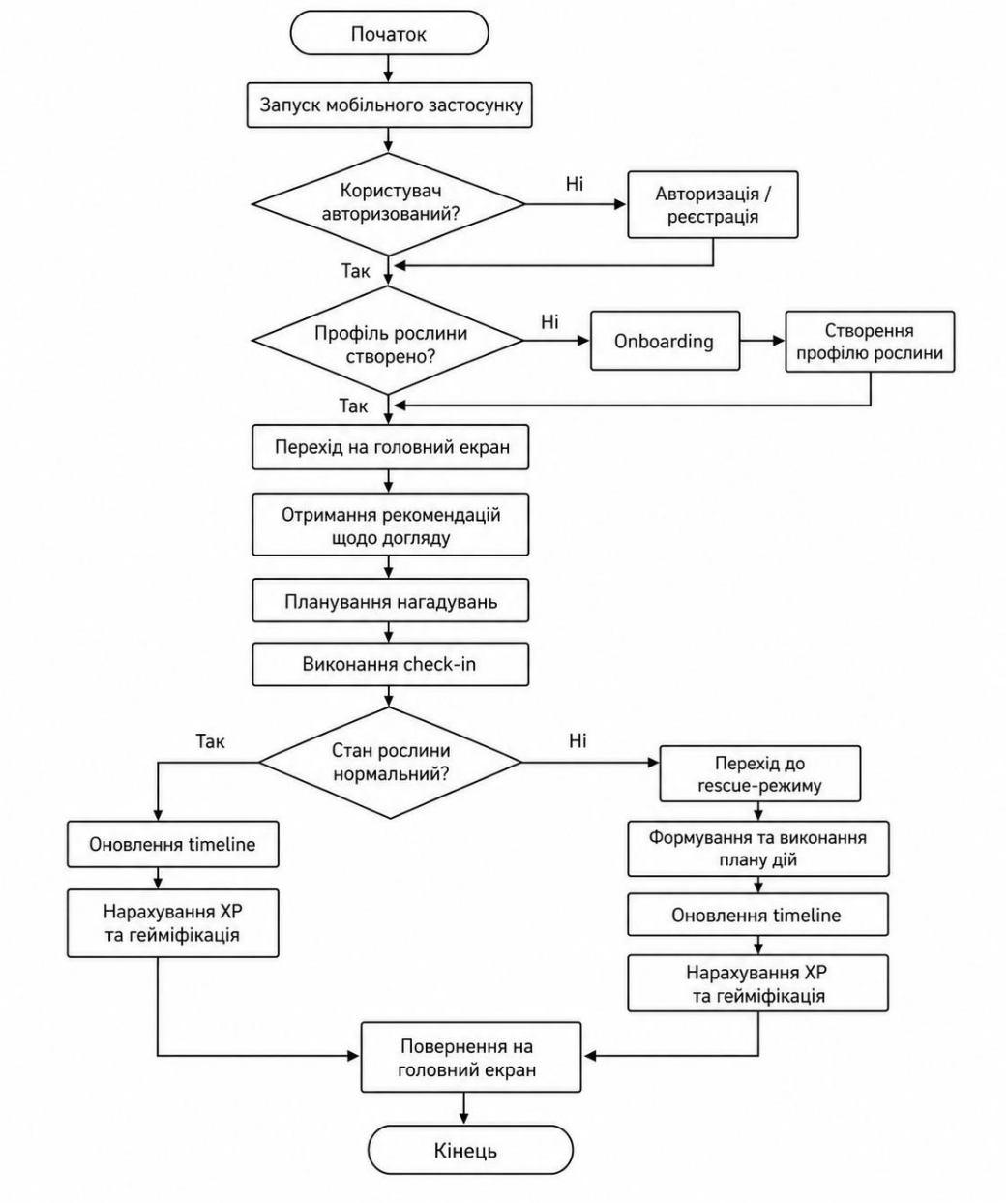


Рисунок 2.6 – Узагальнений алгоритм роботи мобільного застосунку MoriMori

Для перевірки узгодженості моделей можна описати основні вхідні та вихідні дані системи.

Вхідні дані:

– дані користувача;

– назва або фото рослини;

– дата останнього поливу;

– інтервал поливу;

– умови розміщення;

– результат check-in;

– симптоми проблеми;

– виконані дії користувача;

– фото для timeline або розпізнавання.

Вихідні дані:

– профіль рослини;

– рекомендації щодо догляду;

– дата наступного поливу;

– нагадування;

– стан рослини;

– події timeline;

– rescue-план;

– XP, рівень, streak;

– стан маскота;

– повідомлення користувачу.

Узагальнена модель показує, що застосунок MoriMori працює як система підтримки регулярної взаємодії з рослиною. Основна логіка не обмежується збереженням довідкової інформації. Система аналізує дії користувача, розраховує дату наступного поливу, формує нагадування, реагує на check-in, підтримує rescue-сценарій і використовує ігрові елементи для підвищення залученості.

Розроблені у другому розділі моделі та алгоритми є основою для програмної реалізації, описаної у третьому розділі.

# 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ДОГЛЯДУ ЗА РОСЛИНАМИ

## 3.1 Обґрунтування вибору технологій реалізації

Для реалізації мобільного застосунку MoriMori було обрано технологічний стек, який дозволяє створити кросплатформений застосунок, забезпечити зручну взаємодію користувача з інтерфейсом, організувати збереження даних, взаємодію з серверною частиною, надсилання сповіщень та підтримку ігрової моделі взаємодії. Оскільки застосунок орієнтований на щоденне використання, основними вимогами до технологій були стабільність, швидкість розроблення, підтримка мобільних платформ, можливість роботи з камерою, локальними та push-сповіщеннями, а також інтеграція з серверним API.

Основною технологією для реалізації клієнтської частини було обрано Flutter [19]. Flutter є фреймворком для створення кросплатформених мобільних застосунків з єдиною кодовою базою. Це дає можливість розробляти інтерфейс і бізнес-логіку одночасно для Android та iOS, не створюючи окремі нативні застосунки для кожної платформи. Для MoriMori це є важливим, оскільки застосунок має бути доступним для широкої аудиторії користувачів і підтримувати однакову логіку взаємодії на різних мобільних пристроях.

Мова програмування Dart була використана як основна мова клієнтської частини застосунку. Dart добре інтегрується з Flutter, підтримує об’єктно-орієнтований підхід, асинхронне програмування, роботу з HTTP-запитами, локальними даними та зовнішніми пакетами. Це дозволяє реалізувати складні сценарії взаємодії користувача із застосунком: авторизацію, додавання рослини, отримання інформації з сервера, оновлення стану інтерфейсу, завантаження фото та оброблення відповідей API.

Для керування станом у мобільному застосунку використано підхід на основі Provider та ChangeNotifier [21]. Такий підхід дозволяє відокремити бізнес-логіку від інтерфейсу користувача. Наприклад, окремі провайдери відповідають за авторизацію, рослини, профіль користувача, історію догляду, check-in, rescue-режим, діагностику та завантаження файлів. Це спрощує підтримку коду, оскільки кожен логічний модуль має власну зону відповідальності.

Для організації навігації між екранами використано GoRouter [22]. Цей інструмент дозволяє описувати маршрути застосунку декларативно та підтримувати складні сценарії переходів між екранами. У MoriMori це використовується для побудови onboarding-флоу, основного екрану, екранів check-in, rescue-сценарію, історії догляду, профілю та інших частин застосунку. Також навігація пов’язана зі станом авторизації користувача та доступністю інтернет-з’єднання.

Серверна частина застосунку реалізована мовою Go [20]. Вибір Go пояснюється високою продуктивністю, простотою розгортання, зручною роботою з HTTP-серверами та можливістю створення стабільного REST API. Серверна частина відповідає за оброблення запитів мобільного застосунку, роботу з користувачами, рослинами, історією догляду, check-in, rescue-сценаріями, завантаженням фото, push-сповіщеннями та іншими модулями.

Для маршрутизації HTTP-запитів у серверній частині використано бібліотеку chi. Вона дозволяє структурувати API за групами маршрутів і підключати middleware для авторизації, логування, оброблення помилок, CORS, rate limiting та інших технічних задач. У межах застосунку API має версіоновану структуру, що спрощує подальший розвиток і підтримку серверної частини.

Для роботи з базою даних використовується PostgreSQL через pgx, а також Supabase як інфраструктурне рішення для зберігання даних і файлів. У базі даних зберігаються дані користувача, профілю, рослини, історії догляду,

check-in, rescue-сесій, товарів , підписок, пристроїв для push-сповіщень та інших сутностей. Supabase Storage використовується для зберігання фото, пов’язаних із рослинами, аватарами або подіями в історії догляду.

Для авторизації використовується підхід на основі JWT-токенів. Мобільний застосунок зберігає токени у захищеному сховищі за допомогою flutter_secure_storage [25]. Це дозволяє зберігати access token та refresh token більш безпечно, ніж у звичайних локальних налаштуваннях. Такий підхід забезпечує захищений доступ до приватних API-методів, наприклад отримання даних користувача, рослини, історії догляду.

Для реалізації сповіщень використано Firebase Messaging та flutter_local_notifications [24, 26]. Firebase Messaging застосовується для push-сповіщень, а local notifications − для локальних нагадувань. Це важливо для MoriMori, оскільки нагадування є однією з ключових функцій застосунку. Користувач повинен отримувати м’які повідомлення про необхідність поливу, повернення до застосунку або виконання дії з догляду за рослиною.

Для роботи з фотографіями використовуються image_picker та camera. Ці пакети дозволяють користувачу додавати фото рослини, створювати записи в історії догляду, завантажувати зображення для першого фото або діагностики. Використання камери є важливим для застосунку, оскільки візуальна інформація про рослину допомагає краще фіксувати її стан і може бути використана для модуля розпізнавання або діагностики. Підхід до використання зображень у задачах ідентифікації рослин узгоджується з сучасними дослідженнями автоматичного розпізнавання рослин на основі мультимодального глибокого навчання [30].

Для аналітики використовується Amplitude. Аналітика потрібна для відстеження ключових дій користувача: проходження onboarding, додавання рослини, виконання check-in, запуск rescue-сценарію, здійснення покупок або повернення до застосунку. Це дозволяє в майбутньому аналізувати поведінку користувачів і покращувати продукт на основі фактичних даних.

Для монетизації і покупок використовується RevenueCat [31]. Це рішення спрощує роботу з підписками та in-app purchases на мобільних платформах. У MoriMori монетизація може бути пов’язана з косметичними елементами, темами, декораціями або іншими елементами кастомізації, не блокуючи базову функціональність догляду за рослиною.

Для отримання інформації про рослини використовується Perenual API [27]. Він може застосовуватися для отримання базових даних про рослину, рекомендацій щодо догляду, поливу та інших параметрів. У межах дипломної роботи це дозволяє не створювати власну повну базу ботанічної інформації, а інтегрувати зовнішнє джерело даних у сценарій роботи застосунку.

Таким чином, обраний технологічний стек дозволяє реалізувати MoriMori як повноцінний мобільний застосунок із клієнтською та серверною частинами. Flutter і Dart забезпечують швидку реалізацію мобільного інтерфейсу, Go та REST API відповідають за серверну логіку, Supabase і PostgreSQL − за збереження даних, Firebase − за сповіщення, а зовнішні сервіси дають можливість розширити функціональність застосунку без надмірного ускладнення власної реалізації. На рисунку 3.1 наведено загальну схему технологічного стеку застосунку MoriMori.

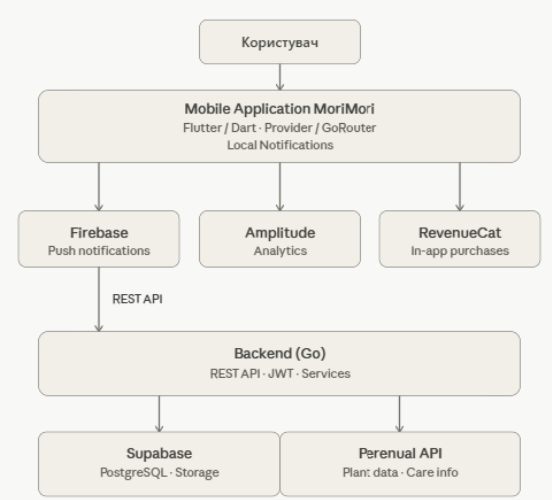


Рисунок 3.1 – Загальна схема технологічного стеку мобільного застосунку MoriMori

## 3.2 Загальна структура мобільного застосунку

Мобільний застосунок MoriMori має модульну структуру, що дозволяє розділити основні функції за зонами відповідальності. Такий підхід спрощує розроблення, тестування та подальший розвиток застосунку. Кожен функціональний модуль відповідає за окремий сценарій взаємодії користувача: авторизацію, створення профілю рослини, щоденний догляд, історію, rescue-режим, сповіщення, профіль користувача або взаємодію з backend.

Загальна структура застосунку складається з клієнтської частини, серверної частини та зовнішніх сервісів. Клієнтська частина реалізована на Flutter і відповідає за інтерфейс користувача, навігацію, локальні дії, відображення стану рослини, завантаження фото, сповіщення та взаємодію з API. Серверна частина реалізована на Go і відповідає за оброблення запитів, збереження даних, бізнес-логіку, роботу з базою даних, rescue-сесіями, timeline, push-сповіщеннями та інтеграціями.

Взаємодія між мобільним застосунком і сервером відбувається через REST API. Мобільний застосунок формує HTTP-запити до серверної частини, передає необхідні дані та отримує відповідь у форматі JSON. Такий підхід дозволяє відокремити інтерфейс користувача від серверної логіки. Клієнтська частина відповідає за зручну взаємодію, а серверна − за збереження, перевірку, оброблення та повернення даних.

Одним із ключових модулів є модуль авторизації. Він відповідає за реєстрацію, вхід користувача, оновлення токенів і доступ до захищених функцій. Після авторизації користувач отримує можливість працювати зі своїм профілем, додавати рослину, виконувати check-in, переглядати історію догляду та користуватися іншими персоналізованими функціями. На стороні мобільного застосунку за цю логіку відповідає AuthProvider, а на стороні backend − auth handler, auth service та middleware авторизації.

Модуль onboarding відповідає за первинне налаштування застосунку. У цьому сценарії користувач проходить кілька екранів: вітання, вибір типу рослини, введення імені, вибір аватара, додавання першого фото та перше повідомлення від рослини. Такий сценарій дозволяє не просто зареєструвати користувача, а одразу створити емоційний зв’язок із рослиною та маскотом. У майбутньому дані onboarding можуть використовуватися для персоналізованих рекомендацій щодо догляду.

Модуль рослини відповідає за збереження та відображення основної інформації про рослину. До такої інформації належать назва рослини, тип, avatar, інтервал поливу, стан рослини, дата останньої взаємодії, а також додаткові параметри догляду. Цей модуль є центральним, оскільки більшість інших частин застосунку пов’язані саме з активною рослиною користувача. Наприклад, check-in оновлює стан рослини, timeline зберігає події, а сповіщення залежать від інтервалу поливу.

Модуль головного екрана виконує роль основного центру взаємодії користувача із застосунком. На ньому користувач бачить стан рослини, маскота, прогрес, рівень, повідомлення та швидкі дії. Головний екран повинен швидко відповідати на питання користувача: як почувається рослина, чи потрібно щось зробити, яка наступна дія, чи є ознаки проблеми. Саме цей екран формує основну щоденну взаємодію з продуктом.

Модуль check-in реалізує щоденну взаємодію користувача з рослиною. Користувач може вказати, що з рослиною все добре, що щось не так або що він не впевнений у її стані. Залежно від вибору користувача застосунок переходить до відповідного сценарію: позитивного check-in, уточнювальних питань або rescue-режиму. Цей модуль є важливим для формування звички, оскільки перетворює догляд за рослиною на короткий регулярний ритуал.

Модуль rescue-режиму призначений для ситуацій, коли користувач помічає проблему з рослиною. У такому випадку застосунок допомагає користувачу описати симптоми, перевірити стан ґрунту або перейти до плану дій. Rescue-сценарій дозволяє не залишати користувача сам на сам із

проблемою, а провести його через послідовність простих кроків. У серверній частині для цього використовуються rescue-сесії, diagnosis service та пов’язані репозиторії.

Модуль історії догляду реалізовано у вигляді timeline. Він дозволяє користувачу переглядати минулі дії, фото, нотатки та події, пов’язані з рослиною. Користувач може додавати моменти, переглядати деталі подій, бачити фотографії та фіксувати зміни стану рослини з часом. Такий модуль має практичну цінність, оскільки дозволяє не тільки виконувати дії догляду, а й аналізувати історію розвитку рослини.

Модуль сповіщень відповідає за нагадування користувачу про важливі дії. Сповіщення можуть бути пов’язані з поливом, поверненням до застосунку, rescue-follow-up або іншими сценаріями. У застосунку передбачено як локальні сповіщення, так і push-сповіщення. Це дозволяє підтримувати користувача навіть тоді, коли застосунок не відкритий.

Модуль гейміфікації включає маскота, прогрес, рівні, XP, streak-механіку, milestone. Його задача полягає не в тому, щоб перетворити застосунок на гру, а в тому, щоб зробити регулярний догляд більш мотивуючим. Маскот виступає візуальним і комунікаційним посередником між системою та користувачем. Він може відображати стан рослини, реагувати на дії користувача та створювати більш емоційний інтерфейс.

Модуль профілю користувача містить персональні налаштування, статистику, рівень, інформацію про кількість днів взаємодії з рослиною, можливість редагування профілю та видалення акаунту. Профіль дозволяє користувачу бачити власний прогрес і керувати частиною персональних даних.

Окремим технічним модулем є модуль оброблення підключення до інтернету. За допомогою ConnectivityProvider застосунок може визначати втрату мережі та перенаправляти користувача на екран відсутності інтернету. Це покращує користувацький досвід, оскільки користувач не отримує

незрозумілих технічних помилок, а бачить зрозуміле повідомлення про проблему з підключенням.

Серверна частина також має модульну структуру. Основними її частинами є handlers, services, repositories, models та middleware. Handlers приймають HTTP-запити та формують відповіді. Services містять бізнес-логіку. Repositories відповідають за роботу з базою даних. Models описують структури даних. Middleware виконує допоміжні функції: авторизацію, логування, CORS, recovery, rate limiting та інші технічні перевірки. Такий поділ дозволяє підтримувати чисту структуру backend і спрощує подальше розширення функціональності.

Загальну структуру взаємодії компонентів MoriMori можна описати так: користувач взаємодіє з Flutter-застосунком; застосунок через ApiClient надсилає запит до REST API; backend обробляє запит через handler, service та repository; дані зберігаються або отримуються з бази даних; після цього backend повертає відповідь мобільному застосунку, а інтерфейс оновлюється через відповідний Provider. На рисунку 3.2 та на рисунку 3.3 наведено загальну структуру мобільного застосунку MoriMori.


Рисунок 3.2 – Загальна структура мобільного застосунку MoriMori

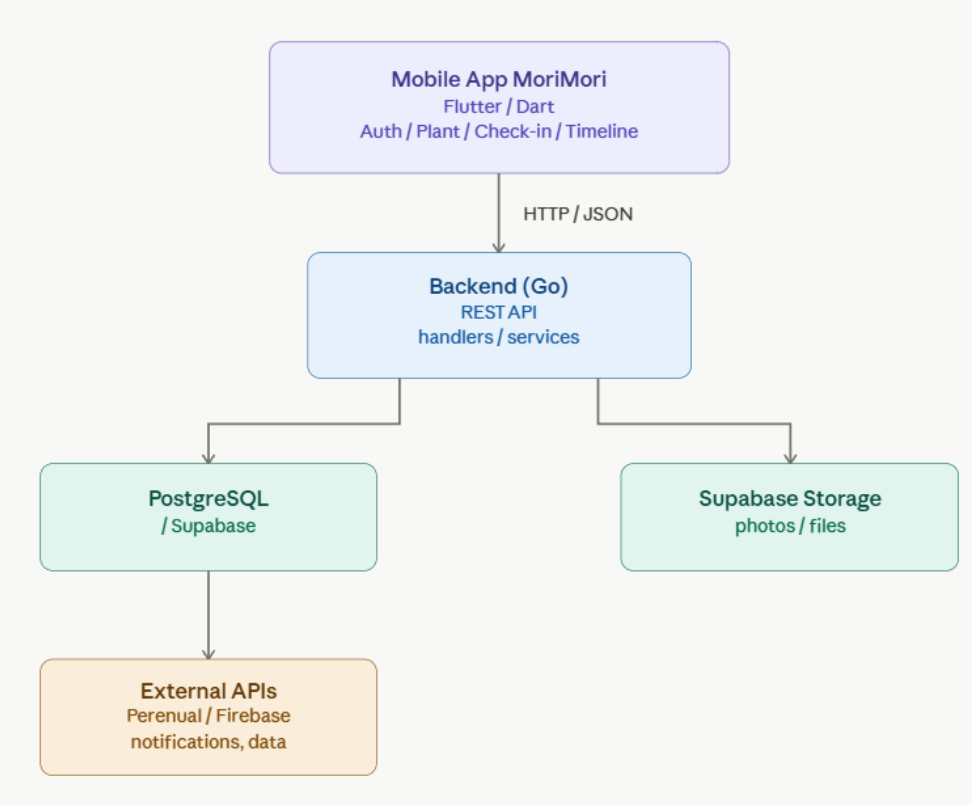


Рисунок 3.3 – Схема взаємодії мобільного застосунку з серверною частиною

Таким чином, структура MoriMori побудована за принципом розділення відповідальності між клієнтською частиною, серверною частиною та зовнішніми сервісами. Такий підхід дозволяє реалізувати мобільний застосунок як масштабовану систему, у якій кожен модуль виконує окрему функцію, але всі модулі разом підтримують основну мету застосунку – допомогу користувачу в регулярному догляді за кімнатною рослиною через зручний інтерфейс та ігрову модель взаємодії.

## 3.3 Реалізація основних функціональних модулів

Мобільний застосунок MoriMori реалізовано як набір функціональних модулів, кожен із яких відповідає за окремий сценарій взаємодії користувача із системою. Такий підхід дозволяє розділити логіку застосунку на зрозумілі частини, спростити підтримку коду та забезпечити можливість подальшого

розширення функціональності. Основними функціональними модулями застосунку є модуль авторизації, модуль створення профілю рослини, модуль щоденного догляду, модуль історії, модуль rescue-режиму, модуль розпізнавання рослини, модуль гейміфікації, та модуль сповіщень. На рисунку 3.4 наведено загальну структуру основних функціональних модулів застосунку MoriMori.

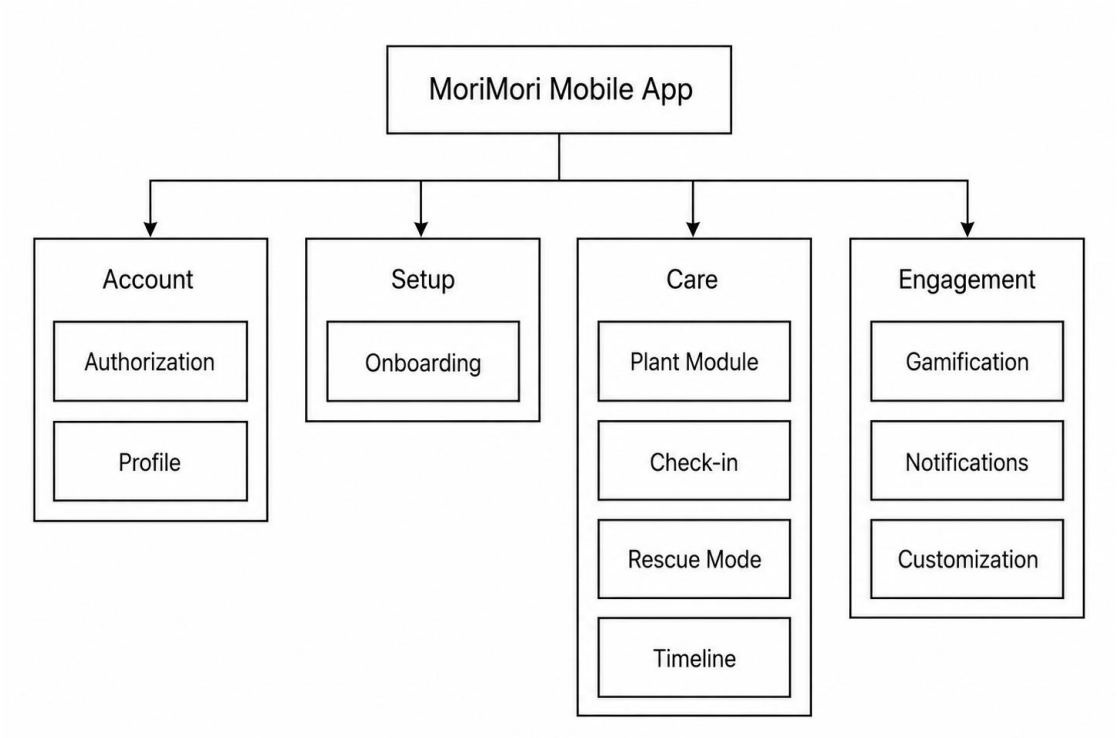


Рисунок 3.4 – Основні функціональні модулі мобільного застосунку MoriMori

Одним із перших сценаріїв взаємодії користувача із застосунком є авторизація. У застосунку передбачено можливість реєстрації, входу в акаунт, збереження токенів доступу та відновлення сесії після повторного відкриття застосунку. На клієнтській частині за цей процес відповідає AuthProvider, який працює з ApiClient та забезпечує передачу даних авторизації до серверної частини. Після успішної авторизації користувач отримує доступ до персоналізованих функцій застосунку: створення рослини, перегляду профілю, історії догляду, та інших модулів.

Для збереження токенів використовується захищене сховище мобільного пристрою. Це дозволяє не виконувати повторний вхід щоразу після запуску застосунку та водночас не зберігати дані авторизації у відкритому вигляді. Якщо під час виконання запиту сервер повертає помилку авторизації, ApiClient може виконати оновлення токенів через refresh token. Такий підхід підвищує стабільність роботи застосунку та покращує користувацький досвід.

Після авторизації користувач переходить до onboarding-сценарію. Його задача полягає не лише в технічному створенні профілю рослини, а й у формуванні первинного емоційного зв’язку між користувачем і рослиною. У межах onboarding користувач може обрати тип рослини, ввести її ім’я, обрати аватар, додати перше фото та задати базові параметри догляду. До таких параметрів належать місце розташування рослини, відстань до вікна, напрям сонячного світла, дата останнього поливу, пересаджування або підживлення.

Модуль створення рослини реалізовано через PlantProvider. На етапі onboarding дані частково зберігаються локально у стані провайдера, після чого передаються на сервер через запит створення рослини. До тіла запиту передаються назва рослини, тип, індекс аватара, дата знайомства з рослиною, посилання на фото та додаткові параметри догляду. Якщо рослина для користувача вже існує, застосунок може обробити конфлікт і оновити наявні дані замість створення дубліката. Лістинг 3.1 демонструє фрагмент формування даних для створення рослини у мобільному застосунку.

Лістинг 3.1 Формування даних для створення рослини:

```
Future<bool> createPlant() async {
  final body = <String, dynamic>{
    'name': _plantName,
    'plant_type': _plantType,
    'avatar_index': _avatarIndex,
    'meet_date': _meetDate.toUtc().toIso8601String(),
```

```
    };

    if (_photoUrl != null) body['photo_url'] = _photoUrl;
    if (_room != null) body['room'] = _room;
    if (_windowDist != null) body['window_dist'] = _windowDist;
    if (_sunDir != null) body['sun_dir'] = _sunDir;

    final data = await _api.post('/plants', body: body);
    _applyPlantData(data);
    notifyListeners();
    return true;
  }
```

Після створення профілю рослини користувач переходить до головного екрана застосунку. Головний екран є основним центром взаємодії користувача із застосунком. На ньому відображається стан рослини, маскот, рівень, прогрес, інформація про наступний полив та швидкі дії. Завдання цього екрана полягає в тому, щоб користувач одразу розумів поточний стан рослини та міг швидко виконати потрібну дію.

Окремим елементом головного екрана є модель стану рослини. У застосунку стан може відображатися через настрій маскота, рівень води, дату наступного поливу, стан “стабільно”, “потребує уваги” або інші візуальні сигнали. Такий підхід дозволяє перетворити технічні дані в більш зрозумілий для користувача інтерфейс. Наприклад, замість сухого повідомлення про необхідність поливу користувач бачить стан маскота й отримує більш м’який сигнал про потребу рослини.

Модуль щоденного check-in є одним із центральних модулів застосунку. Його задача полягає у формуванні регулярної взаємодії користувача з рослиною. Користувач може вказати один із варіантів стану: з рослиною все добре, щось не так або користувач не впевнений у стані

рослини. Залежно від вибору застосунок або фіксує позитивний check-in, або запускає уточнювальний сценарій, або переводить користувача до rescue-режиму.

Логіка check-in реалізована у CheckInProvider. Провайдер формує тіло запиту, передає на сервер mood, додаткові відповіді щодо ґрунту та листя, а також, за наявності, фото рослини. Після цього сервер повертає результат check-in: номер дня, новий стан рослини, пораду, ознаку необхідності переходу до rescue-режиму та можливу інформацію про milestone. Це дозволяє одразу оновити інтерфейс користувача без зайвого повторного запиту. На рисунку 3.5 наведено сценарій виконання щоденного check-in у застосунку.


Рисунок 3.5 – Сценарій виконання щоденного check-in у застосунку MoriMori

Лістинг 3.2 Фрагмент реалізації щоденного check-in:

```
Future<CheckInResult?> doCheckIn({
  required String mood,
  bool? soilDry,
  bool? leavesHealthy,
  String? photoUrl,
  String? plantId,
}) async {
  final body = <String, dynamic>{'mood': mood};

  if (soilDry != null) body['soil_dry'] = soilDry;
  if (leavesHealthy != null) body['leaves_healthy'] = leavesHealthy;
  if (photoUrl != null) body['photo_url'] = photoUrl;
  if (plantId != null) body['plant_id'] = plantId;

  final data = await _api.post('/check-ins', body: body);
  _lastResult = CheckInResult.fromJson(data);
  notifyListeners();
  return _lastResult;
}
```

Якщо користувач вказує, що з рослиною щось не так, застосунок може запустити rescue-режим. Rescue-режим призначений для ситуацій, коли користувач бачить проблему, але не знає, як її правильно інтерпретувати. У межах цього сценарію користувач може обрати симптоми, додати фото, пройти перевірку ґрунту та отримати план дій. Такий підхід дозволяє не залишати користувача самостійно шукати рішення, а провести його через послідовність зрозумілих кроків.

У застосунку передбачено окремі екрани для rescue-сценарію: екран вибору симптомів, екран додавання фото, екран перевірки ґрунту, екран

плану порятунку та екран успішного завершення. На рівні стану за цей модуль відповідає RescueProvider, який зберігає поточну rescue-сесію, список кроків, статус завантаження, помилки та результат завершення сценарію. Серверна частина формує план дій на основі переданих симптомів і повертає його мобільному застосунку.

Модуль історії догляду реалізовано у вигляді timeline. Його задача полягає у збереженні подій, пов’язаних із рослиною: фото, нотаток, моментів росту, подій догляду, rescue-сесій або інших змін. Користувач може переглядати минулі події, додавати нові моменти, відкривати деталі запису, редагувати або видаляти події. Такий функціонал має практичну цінність, оскільки користувач може бачити розвиток рослини з часом і краще розуміти, як виконані дії впливають на її стан.

Для роботи з історією використовується TimelineProvider. Він відповідає за завантаження списку подій із сервера, пагінацію, фільтр подій із фотографіями, створення нових записів, оновлення та видалення подій. Додавання нового моменту може включати назву, опис, дату, нотатку та посилання на фото. Фото попередньо завантажується через upload-модуль, після чого отримане посилання зберігається в timeline-події.

Модуль розпізнавання рослини реалізовано як окремий сценарій роботи з фото. Користувач може обрати або зробити фотографію рослини, після чого зображення передається на оброблення. У застосунку за цей процес відповідає UploadProvider, який працює з image_picker, формує файл і передає його на відповідний endpoint серверної частини. Сервер повертає результат ідентифікації: назву рослини, ймовірність, поширені назви, посилання на додаткову інформацію, рекомендації щодо поливу, освітлення та догляду.

У межах кваліфікаційної роботи не ставиться задача навчання власної складної моделі машинного навчання з нуля [9, 18, 27, 32]. Основний акцент зроблено на інтеграції функції розпізнавання в загальний сценарій користувача. Такий підхід є доцільним, оскільки задача роботи полягає у

проєктуванні та реалізації мобільного застосунку, а не в дослідженні нової моделі комп'ютерного зору. Розпізнавання використовується як прикладний модуль, який зменшує вхідний бар'єр для користувача під час додавання рослини.

Для роботи з фото також використовується модуль завантаження. Він забезпечує додавання фото рослини, аватара, фото для timeline та фото для check-in. Після вибору файлу застосунок надсилає його на сервер як multipart/form-data, а сервер повертає URL збереженого зображення. Отримане посилання далі використовується в інших модулях застосунку: у профілі рослини, історії догляду або check-in.

Модуль нагадувань реалізовано через поєднання локальних і push-сповіщень. Нагадування потрібні для підтримки регулярного догляду: користувач може отримати повідомлення перед поливом, у день поливу, після пропуску догляду або для повернення до застосунку. Важливо, що нагадування не мають бути надмірними, інакше користувач може почати їх ігнорувати. Тому в MoriMori сповіщення прив'язуються до параметрів рослини, зокрема до інтервалу поливу та останньої взаємодії.

Логіка нагадувань використовує дані з PlantProvider: дату останнього check-in, інтервал поливу, дату наступного поливу, локаль користувача та стан рослини. Після завантаження або оновлення рослини застосунок може перепланувати нагадування. Такий підхід дозволяє зробити сповіщення контекстними, а не фіксованими для всіх користувачів.

Окремим модулем є ігрова модель взаємодії. Вона включає маскота, рівень, XP, streak, milestone, і кастомізацію. Маскот використовується як візуальний посередник між користувачем і системою. Він допомагає зробити взаємодію менш механічною та підтримує емоційний характер застосунку. Замість того щоб показувати лише список задач, застосунок показує "живий" стан рослини через персонажа, кольори, повідомлення та реакції на дії користувача.

Рівні, XP і milestone використовуються для візуалізації прогресу [7, 14, 15]. Користувач отримує відчуття поступового розвитку, коли виконує регулярні дії з догляду. Це дозволяє зробити повторювані дії менш нудними. При цьому гейміфікація не замінює основну задачу застосунку, а лише підтримує її. Основна мета залишається практичною: допомогти користувачу краще доглядати за кімнатною рослиною.

Профіль користувача реалізовано як окремий модуль. Він містить інформацію про користувача, рівень, статистику, кількість check-in, кількість rescue-сесій, події timeline, налаштування нагадувань і статус підписки. Через профіль користувач може переглядати власний прогрес, редагувати дані, змінювати налаштування або видалити акаунт.

Таким чином, реалізовані функціональні модулі формують повний цикл використання MoriMori: користувач реєструється, додає рослину, отримує інформацію, виконує щоденний check-in, переглядає історію, отримує нагадування, за потреби переходить у rescue-режим і взаємодіє з маскотом. Це дозволяє розглядати застосунок не як простий довідник або нагадувач, а як систему підтримки регулярного догляду за кімнатною рослиною.

## 3.4 Реалізація взаємодії з серверною частиною

Взаємодія мобільного застосунку MoriMori із серверною частиною реалізована через REST API. Такий підхід дозволяє розділити клієнтську логіку, яка відповідає за інтерфейс і взаємодію з користувачем, та серверну логіку, яка відповідає за збереження даних, авторизацію, оброблення запитів, формування відповідей і роботу з базою даних. Мобільний застосунок надсилає HTTP-запити до серверної частини, а сервер повертає відповіді у форматі JSON.

На рисунку 3.6 наведено загальну схему взаємодії мобільного застосунку із серверною частиною.

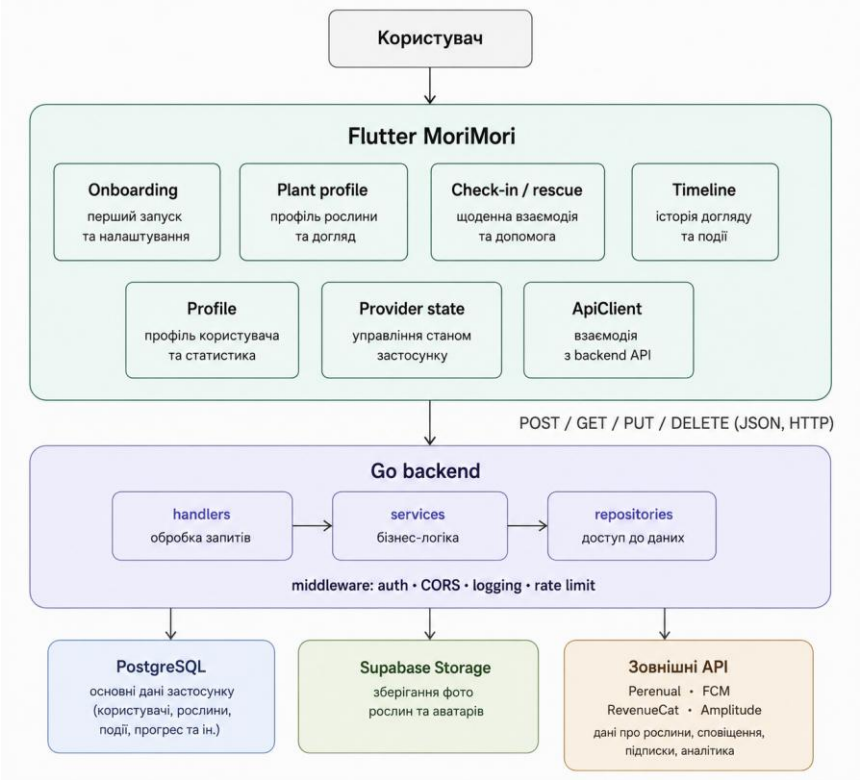


Рисунок 3.6 – Схема взаємодії мобільного застосунку MoriMori із серверною частиною

На клієнтській частині центральним компонентом для роботи з API є ApiClient. Він відповідає за формування HTTP-запитів, додавання заголовків авторизації, серіалізацію тіла запиту у JSON, оброблення відповідей, оброблення помилок і автоматичне оновлення токенів. Завдяки цьому інші провайдери не працюють напряму з HTTP-клієнтом, а використовують єдиний сервіс для взаємодії з backend.

Основними типами запитів, які використовуються у застосунку, є GET, POST, PATCH та DELETE. GET-запити застосовуються для отримання даних, наприклад профілю користувача, рослини, історії check-in, timeline. POST-запити використовуються для створення нових сутностей: реєстрації, створення рослини, виконання check-in, запуску rescue-сесії або завантаження фото. PATCH-запити використовуються для оновлення

існуючих даних, наприклад профілю, рослини або timeline-події. DELETE-запити застосовуються для видалення акаунту або окремих подій.

Для доступу до захищених endpoint використовується JWT-авторизація. Після входу користувача access token та refresh token зберігаються у захищеному сховищі мобільного пристрою. Під час виконання запиту ApiClient додає access token у заголовок Authorization. Якщо сервер повертає помилку 401, застосунок виконує спробу оновлення токена через refresh token. Якщо оновлення успішне, запит повторюється. Якщо refresh token також невалідний, користувач переводиться до стану неавторизованої сесії. Фрагмент реалізації авторизованого запиту наведено в лістингу 3.3.

Лістинг 3.3 Фрагмент реалізації авторизованого HTTP-запиту:

```
Future<Map<String, dynamic>> _authedRequest(
  String method,
  String path, {
  Map<String, dynamic>? body,
  Map<String, String>? queryParams,
}) async {
  var response = await _sendAuthed(
    method,
    path,
    body: body,
    queryParams: queryParams,
  );

  if (response.statusCode == 401) {
    if (await refreshTokens()) {
      response = await _sendAuthed(
        method,
        path,
```

```
      body: body,
      queryParams: queryParams,
    );
   } else {
    forceSessionExpired();
    throw ApiException(401, 'UNAUTHORIZED', 'Session expired');
   }
  }

  return _handleResponse(response);
 }
```

Для оброблення помилок у застосунку використовується окремий клас ApiException. Він містить HTTP-статус, код помилки та повідомлення. Це дозволяє провайдерам показувати користувачу зрозуміле повідомлення або виконувати спеціальну дію залежно від типу помилки. Наприклад, помилка 404 під час отримання рослини означає, що користувач ще не створив рослину і його потрібно перенаправити до onboarding. Помилка 409 під час створення рослини означає, що рослина вже існує, тому застосунок може оновити наявний запис.

Серверна частина реалізована мовою Go та має модульну структуру. Основними її частинами є handlers, services, repositories, models та middleware. Handlers приймають HTTP-запити, зчитують параметри, перевіряють тіло запиту та формують відповідь. Services містять бізнес-логіку застосунку. Repositories відповідають за роботу з базою даних. Models описують структури даних і DTO, які використовуються для обміну інформацією між клієнтом і сервером. Middleware виконує проміжні технічні задачі: авторизацію, CORS, логування, rate limiting, recovery та безпекові перевірки.

Така структура серверної частини дозволяє розділити відповідальність між різними шарами. Handler не містить складної бізнес-логіки, а лише приймає запит і передає його до service. Service виконує основну логіку: перевіряє правила, формує результат, працює з кількома repository або зовнішніми сервісами. Repository відповідає за виконання SQL-запитів або взаємодію з базою даних. Завдяки такому розділенню код backend легше підтримувати, тестувати та розширювати.

Основні API-маршрути застосунку можна згрупувати за функціональними модулями. До модуля авторизації належать маршрути реєстрації, входу та оновлення токена. До модуля рослини належать маршрути створення рослини, отримання поточної рослини та оновлення її параметрів. До модуля check-in належать маршрути створення check-in, отримання історії, поливу, пересаджування, підживлення та завершення мікрочілу. До модуля rescue належать маршрути створення rescue-сесії, отримання активної сесії та завершення плану дій. До модуля timeline належать маршрути отримання, створення, редагування та видалення подій.

У таблиці 3.1 наведено основні групи API-маршрутів, які використовуються мобільним застосунком.

Таблиця 3.1 – Основні групи API-маршрутів застосунку MoriMori

| **Група API** | **Призначення** |
|---|---|
| **1** | **2** |
| /auth | Реєстрація, вхід, оновлення токена |
| /plants | Створення, отримання та оновлення рослини |
| /check-ins | Щоденний check-in та історія взаємодій |
| /rescue-sessions | Запуск і завершення rescue-сценаріїв |
| /timeline | Історія догляду та події розвитку рослини |
| /profile | Профіль користувача, статистика, налаштування |
| /uploads | Завантаження фото рослини, аватара, timeline,check-in |

Продовження таблиці 3.1

| 1 | 2 |
|---|---|
| /identify | Розпізнавання рослини за фотографією |
| /iap | Каталог покупок, перевірка покупок і підписки |

Для створення рослини мобільний застосунок надсилає POST-запит до endpoint /plants. У тілі запиту передаються основні дані, зібрані під час onboarding: ім’я рослини, тип, аватар, дата знайомства, фото та додаткові параметри. Сервер перевіряє вхідні дані, створює запис у базі даних і повертає об’єкт рослини. Після отримання відповіді PlantProvider оновлює локальний стан, а інтерфейс переходить до головного екрану.

Для щоденного check-in використовується endpoint /check-ins. Клієнт передає mood і, за потреби, додаткові параметри: стан ґрунту, стан листя або посилання на фото. Сервер обробляє ці дані, оновлює стан рослини, розраховує прогрес, streak, можливі нагороди та повертає відповідь. Якщо відповідь містить redirect_to_rescue, мобільний застосунок відкриває rescue-сценарій.

Для історії догляду використовується група маршрутів /timeline. Мобільний застосунок може отримати список подій, створити новий запис, відкрити деталі події, оновити або видалити її. Така взаємодія дозволяє зберігати не лише технічні записи про полив, а й суб’єктивні моменти користувача: фото, нотатки, описи змін або важливі події розвитку рослини.

Для завантаження фото використовується multipart/form-data. Це відрізняється від звичайних JSON-запитів, оскільки файл передається окремою частиною multipart-запиту. На клієнтській частині ApiClient має окремий метод uploadFile, який додає файл до запиту, визначає тип зображення та надсилає його на сервер. Після успішного завантаження сервер повертає URL, який потім використовується в інших модулях застосунку. Фрагмент реалізації завантаження файлу наведено в лістингу 3.4.

Лістинг 3.4 Фрагмент реалізації завантаження фото на сервер:

```
Future<Map<String, dynamic>> uploadFile(
  String path,
  String fieldName,
  File file,
) async {
  final uri = Uri.parse('$_baseUrl$path');
  final request = http.MultipartRequest('POST', uri)
    ..headers['Authorization'] = 'Bearer $_accessToken'
    ..files.add(await http.MultipartFile.fromPath(
      fieldName,
      file.path,
    ));

  final streamed = await _http.send(request);
  final response = await http.Response.fromStream(streamed);

  return _handleResponse(response);
}
```

Endpoint /identify використовується для розпізнавання рослини за фотографією. Клієнт надсилає зображення як multipart/form-data, а сервер повертає результат ідентифікації. У відповіді можуть міститися назва рослини, ймовірність, поширені назви, посилання на додаткову інформацію та рекомендації щодо догляду. Якщо параметр auto_update активний і результат має достатню впевненість, сервер може автоматично оновити дані поточної рослини.

У застосунку також передбачено взаємодію з API профілю. Через endpoint /profile/me клієнт отримує інформацію про користувача, статистику, кількість check-in, rescue-сесій, timeline-подій і стан нагадувань. Через

PATCH-запит користувач може змінити частину налаштувань. Через DELETE-запит може бути виконано видалення акаунту та пов’язаних даних.

Важливою частиною взаємодії з backend є локалізація. ApiClient може додавати до запитів заголовок Accept-Language, якщо в застосунку задана поточна мова. Це дозволяє серверу повертати частину повідомлень або порад мовою користувача. Для застосунку, який орієнтований на українську та англійську локалізацію, це є важливим елементом користувацького досвіду.

Також у застосунку враховано сценарій втрати інтернет-з’єднання. ConnectivityProvider відстежує стан мережі та дозволяє перенаправляти користувача на окремий екран відсутності інтернету. Це дає змогу уникнути ситуації, коли користувач бачить незрозумілу технічну помилку під час виконання запиту. Замість цього інтерфейс повідомляє про проблему підключення у зрозумілому вигляді.

Оброблення відповідей сервера побудоване так, щоб позитивні відповіді одразу оновлювали локальний стан відповідного провайдера. Наприклад, після створення рослини PlantProvider оновлює назву, тип, аватар, дату, стан, рівень, XP і параметри догляду. Після check-in CheckInProvider зберігає результат, а PlantProvider може застосувати plant snapshot, повернутий сервером. Після створення timeline-події TimelineProvider оновлює список подій. Такий підхід дозволяє підтримувати інтерфейс актуальним без повного перезавантаження застосунку.

Загалом взаємодія з серверною частиною у MoriMori побудована за принципом розділення відповідальності. Мобільний застосунок відповідає за інтерфейс, локальний стан і сценарії користувача. Серверна частина відповідає за збереження даних, бізнес-логіку, авторизацію, формування результатів, роботу з файлами та інтеграції. Це дозволяє підтримувати застосунок у більш структурованому вигляді та створює основу для подальшого розвитку: додавання кількох рослин, розширення rescue-сценаріїв, покращення розпізнавання, персоналізація рекомендацій і розвиток ігрової моделі взаємодії.

## 3.5 Реалізація користувацького інтерфейсу

Користувацький інтерфейс мобільного застосунку MoriMori реалізовано з урахуванням основної ідеї продукту: допомогти користувачу регулярно доглядати за кімнатною рослиною та підтримувати взаємодію через ігрову модель. Інтерфейс повинен бути не лише функціональним, а й зрозумілим для користувача, оскільки застосунок орієнтований не тільки на досвідчених власників рослин, а й на новачків.

Під час розроблення інтерфейсу було враховано кілька основних вимог. По-перше, користувач повинен швидко розуміти поточний стан рослини. По-друге, основні дії мають бути доступні без складної навігації. По-третє, ігрові елементи не повинні заважати базовому догляду, а мають підсилювати мотивацію користувача. По-четверте, інтерфейс має бути візуально цілісним, щоб застосунок сприймався як єдиний продукт, а не набір окремих екранів.

Візуальна концепція застосунку побудована навколо маскота, карток, контрастних елементів, великих кнопок та зрозумілих станів. Маскот використовується не лише як декоративний елемент, а як частина інтерфейсу, що допомагає передавати емоційний стан рослини, реакцію на дії користувача та загальний прогрес догляду [7, 14, 15]. Завдяки цьому взаємодія із застосунком стає менш формальною та більш емоційною.

Основним екраном застосунку є головний екран. Він виконує роль центру взаємодії користувача з рослиною. На головному екрані відображається маскот, стан рослини, рівень користувача, прогрес, інформація про воду або догляд, а також швидкі дії. Такий підхід дозволяє користувачу одразу побачити, чи потребує рослина уваги, і швидко перейти до потрібної дії.

На рисунку 3.7 наведено приклад головного екрана мобільного застосунку MoriMori.


Рисунок 3.7 – Головний екран мобільного застосунку MoriMori

Екран додавання рослини в мобільному застосунку MoriMori наведено на рисунку 3.8.


Рисунок 3.8 – Екран додавання рослини у мобільному застосунку MoriMori

Екран додавання рослини є важливим, оскільки саме з нього починається персоналізація застосунку. Користувач не просто створює запис, а формує профіль конкретної рослини. У профілі можуть зберігатися назва, тип рослини, фото, дата знайомства, параметри поливу, місце розташування та інші характеристики. Надалі ці дані використовуються для відображення інформації, нагадувань, історії догляду та персоналізації взаємодії.

Для щоденної взаємодії використовується check-in сценарій. Його інтерфейс має бути максимально простим, оскільки користувач може виконувати цю дію регулярно. Замість складної форми застосунок пропонує

користувачу коротко оцінити стан рослини: все добре, щось не так або користувач не впевнений. Після вибору стану застосунок або фіксує подію, або переходить до уточнювальних питань, або запускає rescue-режим.

Перевага такого підходу полягає в тому, що користувач не повинен витрачати багато часу на взаємодію із застосунком. Основна дія займає кілька секунд, але водночас створює регулярну історію догляду. Це важливо для формування звички, оскільки складні або довгі сценарії можуть знижувати частоту використання застосунку.

Якщо користувач вказує, що з рослиною щось не так, застосунок може перейти до rescue-режиму. Інтерфейс rescue-режиму побудований як послідовність простих екранів. Користувач поступово вказує симптоми, може додати фото, перевірити стан ґрунту та отримати короткий план дій. Такий формат зменшує когнітивне навантаження, оскільки користувач не бачить одразу велику форму або довгий список можливих проблем. На рисунку 3.9 наведено приклад екрана rescue-режиму.

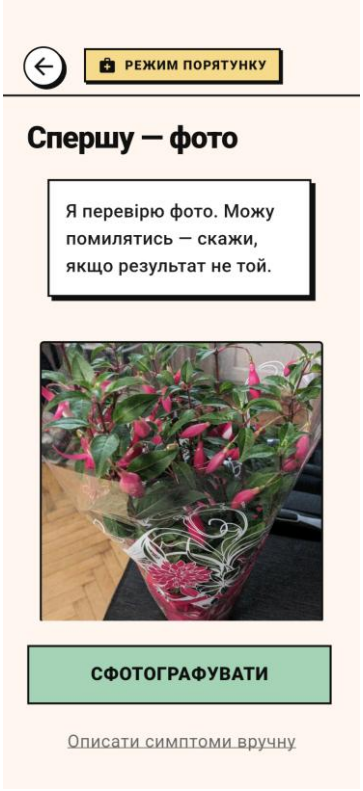


Рисунок 3.9 – Екран rescue-режиму у мобільному застосунку MoriMori

Rescue-режим має практичну цінність, оскільки користувачі не завжди можуть самостійно визначити причину погіршення стану рослини. Послідовний сценарій дозволяє зібрати базову інформацію та запропонувати прості дії. У межах поточної версії застосунку rescue-режим може бути реалізований як набір правил і сценаріїв, без необхідності створення складної власної моделі машинного навчання.

Екран історії догляду реалізовано у форматі timeline. Він дозволяє користувачу переглядати події, пов’язані з рослиною: полив, check-in, додані фото, нотатки, rescue-сесії або інші зміни. Такий підхід допомагає користувачу бачити розвиток рослини з часом і аналізувати, які дії були виконані раніше. На рисунку 3.10 наведено приклад екрана історії догляду.

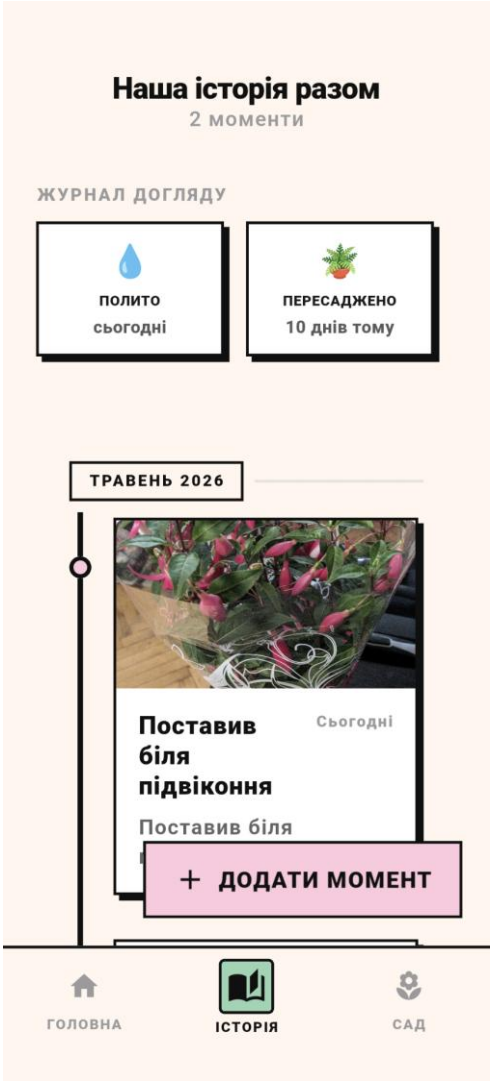


Рисунок 3.10 – Екран історії догляду за рослиною

Timeline є важливим елементом не лише для зберігання даних, а й для підсилення емоційної цінності застосунку. Користувач може бачити не просто список технічних дій, а історію взаємодії з рослиною. Якщо в історії зберігаються фотографії, користувач може візуально відстежувати зміни стану рослини.

Окремим елементом інтерфейсу є профіль користувача. Він містить інформацію про користувача, рівень, статистику взаємодій, налаштування та, за потреби, статус підписки. Через профіль користувач може редагувати власні дані, переглядати прогрес або перейти до налаштувань застосунку. Профіль також може використовуватися для відображення загальної статистики: кількості check-in, кількості днів догляду, рівня або прогресу. На рисунку 3.11 наведено приклад екрана профілю користувача.

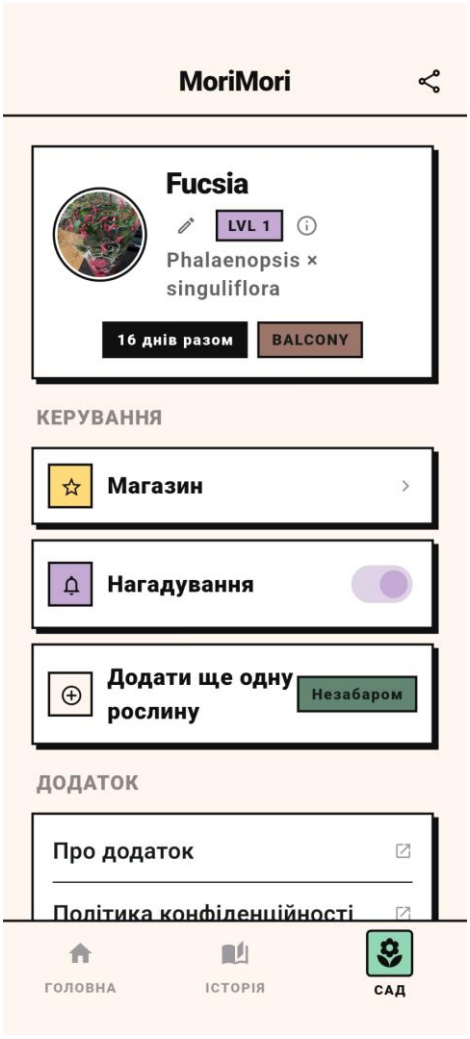


Рисунок 3.11 – Екран профілю користувача

Також у застосунку передбачено екран короткої спокійної взаємодії з рослиною. Цей сценарій може бути поданий як таймер або «мікрочіл». Його задача полягає у створенні короткого ритуалу взаємодії користувача з рослиною. Такий екран не є основною функцією догляду, але допомагає підсилити емоційну складову продукту. На рисунку 3.12 наведено приклад екрана таймера короткої взаємодії з рослиною.

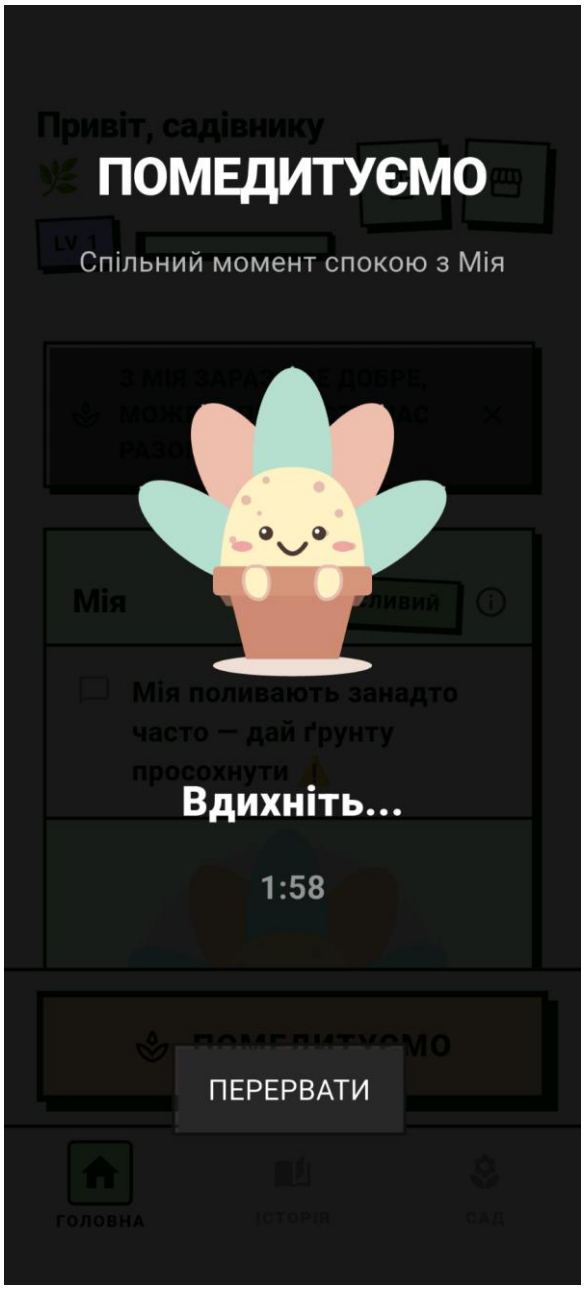


Рисунок 3.12 – Екран короткої взаємодії з рослиною

Навігація в застосунку побудована так, щоб користувач міг швидко переходити між основними сценаріями. Основними напрямами навігації є головний екран, профіль рослини, історія, rescue-сценарій і профіль користувача. Для цього використовується маршрутизація через GoRouter, а стан екранів оновлюється через відповідні Provider-класи. Такий підхід дозволяє підтримувати зрозумілу структуру інтерфейсу й уникати надмірної складності в навігації. Окремо враховано сценарій відсутності інтернет-з'єднання. Якщо застосунок не може звернутися до серверної частини, користувачу відображається зрозумілий екран із повідомленням про проблему підключення. Це краще, ніж показувати технічну помилку або залишати користувача без пояснення. Такий підхід покращує стабільність користувацького досвіду.

Загалом інтерфейс MoriMori побудований навколо поєднання практичних дій і емоційної взаємодії. Практичні елементи допомагають користувачу доглядати за рослиною, а ігрові елементи підтримують мотивацію та регулярність. Така структура відповідає основній меті застосунку − зробити догляд за кімнатними рослинами більш зрозумілим, регулярним і персоналізованим.

## 3.6 Тестування роботи застосунку

Після реалізації основних функціональних модулів було проведено тестування мобільного застосунку MoriMori. Метою тестування була перевірка працездатності основних сценаріїв користувача, коректності взаємодії клієнтської та серверної частин, правильності збереження даних, стабільності навігації та відповідності інтерфейсу заявленій логіці роботи застосунку.

Тестування виконувалося для основних сценаріїв, критичних для роботи мобільного застосунку MoriMori: реєстрації та входу користувача,

створення профілю рослини, відкриття головного екрана, виконання щоденного check-in, запуску rescue-режиму, додавання події до історії догляду, завантаження фото, роботи профілю користувача та оброблення втрати інтернет-з’єднання. Перелік функціональних сценаріїв і умови їх виконання наведено в таблиці 3.2.

У межах тестування перевірялося не лише виконання окремої дії, а й відповідність фактичного результату очікуваному. Наприклад, після створення рослини користувач повинен перейти на головний екран, а дані рослини мають бути збережені. Після check-in має оновитися стан застосунку, а подія повинна бути доступна для подальшого перегляду. Результати перевірки функціональних сценаріїв подано в таблиці 3.3.

Таблиця 3.2 – Результати тестування основних функцій застосунку MoriMori

| № | Функціональний сценарій | Умови виконання |
|---|---|---|
| **1** | **2** | **3** |
| 1 | Реєстрація нового користувача | Введено коректні дані для створення облікового запису |
| 2 | Авторизація користувача | Введено дійсні облікові дані |
| 3 | Відновлення сесії після повторного запуску | У сховищі наявні дійсні токени доступу |
| 4 | Створення профілю рослини | Користувач заповнив дані рослини під час onboarding |
| 5 | Відображення головного екрана | Після створення рослини користувач відкриває головний екран |
| 6 | Щоденний check-in | Користувач обирає стан рослини та надсилає чекін |

Продовження таблиці 3.2

| 1 | 2 | 3 |
|---|---|---|
| 7 | Запуск rescue-сценарію | Користувач вказує, що з рослиною є проблема |
| 8 | Додавання події до історії догляду | Користувач додає нову подію або фото |
| 9 | Завантаження фото рослини | Користувач обирає фото з камери або галереї |
| 10 | Перевірка push та локальних сповіщень | Для рослини задано нагадування |
| 11 | Оброблення втрати інтернет-з’єднання | Мережеве з’єднання відсутнє |
| 12 | Відображення профілю користувача | Користувач відкриває екран профілю |
| 13 | Перехід між основними екранами | Користувач перемикається між розділами застосунку |

Таблиця 3.3 – Результати виконання функціональних сценаріїв застосунку

| № | Очікуваний результат | Фактичний результат | Статус |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | Користувач успішно створює акаунт і переходить до початкового налаштування застосунку | Акаунт створено, користувача перенаправлено до onboarding-сценарію | Виконано |
| 2 | Користувач входить у застосунок і отримує доступ до персональних функцій | Авторизація виконана успішно, токени збережено | Виконано |
| 3 | Застосунок автоматично відновлює сесію | Сесію відновлено автоматично | Виконано |

Продовження таблиці 3.3

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 4 | У системі створюється новий профіль рослини | Профіль рослини створено та збережено | Виконано |
| 5 | Відображаються стан рослини, маскот, прогрес та швидкі дії | Дані відображаються коректно | Виконано |
| 6 | Система зберігає подію та оновлює стан рослини | Check-in збережено, стан оновлено | Виконано |
| 7 | Відкривається rescue-режим із подальшими кроками | Rescue-сценарій відкрито коректно | Виконано |
| 8 | Подія зберігається у timeline та відображається в історії | Подію додано та відображено в історії | Виконано |
| 9 | Фото завантажується на сервер і повертається URL | Фото завантажено успішно | Виконано |
| 10 | Користувач отримує нагадування у визначений час | Сповіщення надходять коректно | Виконано |
| 11 | Відображається зрозумілий екран без технічної помилки | Екран відсутності інтернету працює коректно | Виконано |
| 12 | Відображаються статистика, рівень та налаштування | Профіль відкривається коректно | Виконано |
| 13 | Навігація виконується без помилок і зависань | Навігація працює стабільно | Виконано |

Окремо було перевірено роботу onboarding-сценарію. Для цього було виконано проходження всіх кроків створення першої рослини: введення імені, вибір типу, вибір аватара, додавання фото та збереження даних. У результаті застосунок коректно передав дані на сервер, створив профіль

рослини та відкрив головний екран. Це підтверджує працездатність первинного сценарію користувача.

Також було перевірено сценарій щоденної взаємодії з рослиною. Під час тестування користувач обирав різні варіанти стану рослини. Якщо обиралося, що з рослиною все добре, система фіксувала позитивний check-in. Якщо обиралося, що з рослиною є проблема, застосунок переходив до rescue-режиму. Це підтверджує коректність зв’язку між check-in та rescue-сценарієм.

Під час тестування модуля історії догляду було перевірено створення нової події, її відображення у списку, відкриття деталей події та роботу з фото. Події коректно зберігалися та відображалися в timeline. Це підтверджує, що модуль історії може використовуватися для фіксації дій користувача і спостереження за змінами стану рослини.

Модуль завантаження фото перевірявся через додавання фото під час створення рослини, check-in і timeline. Після вибору зображення застосунок формував multipart-запит до серверної частини, а сервер повертав посилання на завантажений файл. У результаті отримане посилання могло використовуватися в інших частинах застосунку.

Окремо було перевірено роботу інтерфейсу на типових сценаріях використання. Було протестовано відкриття головного екрана, переходи між основними екранами, відкриття профілю, історії догляду та rescue-сценарію. Навігація працювала коректно, а основні елементи інтерфейсу відображалися відповідно до очікуваної логіки.

Під час тестування було виявлено, що застосунок має залежність від стабільного інтернет-з’єднання, оскільки значна частина даних отримується із серверної частини. Для зменшення негативного впливу цієї проблеми було передбачено екран відсутності інтернет-з’єднання. У майбутньому застосунок можна покращити за допомогою розширеного локального кешування даних.

Також було враховано обмеження модуля розпізнавання рослин. Якість визначення рослини залежить від якості фотографії, освітлення, ракурсу та

наявності зайвих об’єктів у кадрі. Тому в застосунку доцільно залишати можливість ручного уточнення або редагування результату. Такий підхід дозволяє уникнути повної залежності від автоматичного розпізнавання.

Результати тестування показали, що основні сценарії застосунку працюють відповідно до очікуваної логіки. Користувач може створити профіль рослини, переглянути головний екран, виконати check-in, перейти до rescue-режиму, додати фото, переглянути історію догляду, відкрити профіль. Це підтверджує працездатність основних функціональних модулів MoriMori.

Водночас застосунок має напрями для подальшого удосконалення. До них належать покращення точності розпізнавання рослин, розширення бази рекомендацій, підтримка кількох рослин, поглиблення rescue-сценаріїв, покращення локального кешування, подальша оптимізація системи нагадувань. Також перспективним є використання аналітики для вивчення поведінки користувачів і покращення сценаріїв утримання.

Таким чином, проведене тестування підтвердило практичну цінність розробленого мобільного застосунку. MoriMori реалізує основні функції підтримки догляду за рослиною та доповнює їх ігровою моделлю взаємодії. Це дозволяє використовувати застосунок не лише як довідник або нагадувач, а як персональний мобільний інструмент для регулярного догляду за кімнатними рослинами.

# ВИСНОВКИ

У рамках кваліфікаційної роботи розроблено та реалізовано мобільний застосунок MoriMori для підтримки користувача під час догляду за кімнатними рослинами з ігровою моделлю взаємодії. У процесі виконання роботи було повністю досягнуто поставлену мету та вирішено такі завдання:

– проаналізовано предметну область домашнього рослинництва й досліджено існуючі аналогічні мобільні рішення, що дозволило виявити їхні функціональні недоліки та обґрунтувати актуальність проєкту;

– розглянуто сучасні методи розпізнавання рослин за зображенням та визначено легкі оптимізовані архітектури моделей комп'ютерного зору як найбільш придатні для інтеграції в мобільні системи;

– визначено функціональні вимоги до системи, спроєктовано структуру застосунку, побудовано модель даних та описано алгоритми ключових сценаріїв взаємодії (створення профілю, планування нагадувань, трекінг дій);

– реалізовано клієнтську частину застосунку (12 основних екранів) на базі фреймворку Flutter та серверну частину з використанням REST API;

– інтегровано модуль розпізнавання рослин за фото, розроблено логіку формування нагадувань про полив та ведення історії догляду, а також впроваджено елементи гейміфікації (інтерактивний маскот, рівні прогресу та check-in сценарії);

– проведено комплексне тестування розробленого рішення, яке підтвердило стабільність клієнт-серверного обміну даними та відповідність системи всім технічним і функціональним вимогам.

Результати роботи апробовано у вигляді 2 тез доповідей: під час XXX Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [31]; під час XVI Міжнародної науково-технічної конференції «Інформаційні технології та комп'ютерне моделювання» (ІТММ-2025) [32].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Trubchaninova, S., & Suprun, O. (2026). Architecture of mobile AR applications using Unity and ARCore for gamified environments. *Матеріали конференції. Харківський національний університет радіоелектроніки*.

2. Andreiev, A. S., & Sotnik, S. V. (2024). Computer games and web design. *Матеріали конференції. Харківський національний університет радіоелектроніки*.

3. Гороховатський, В. О., & Творошенко, І. С. (2025). Оцінювання значущості ознак для підвищення продуктивності структурних методів класифікації зображень. *Проблеми інформатики та моделювання (ПІМ-2025): Тези двадцять п'ятої міжнародної науково-технічної конференції* (25–28 вересня 2025 року). Харків: НТУ «ХПІ», С. 38–43.

4. Ikwunne, T., Hederman, L., & Wall, P. J. (2021). *Designing mobile health for user engagement: The importance of socio-technical approach*. arXiv preprint arXiv:2108.09786.

5. Fan, S., Jain, R. C., & Kankanhalli, M. S. (2023). *A comprehensive picture of factors affecting user willingness to use mobile health applications*. arXiv preprint arXiv:2305.05962.

6. Ben Yehuda, C., Gilad-Bachrach, R., & Udi, Y. (2024). *Improving engagement and efficacy of mHealth micro-interventions for stress coping: An in-the-wild study*. arXiv preprint arXiv:2407.11612.

7. Tsay, C. H.-H., Kofinas, A. K., Trivedi, S. K., & Yang, Y. (2020). Overcoming the novelty effect in online gamified learning systems: An empirical evaluation of student engagement and performance. *Journal of Computer Assisted Learning*, *36*(2), 128–142.

8. Han, K.-T., Ruan, L.-W., & Liao, L.-S. (2022). Effects of indoor plants on human functions: A systematic review with meta-analyses. *International Journal of Environmental Research and Public Health*, *19*(12), 7453–7471.

9. Wäldchen, J., Mäder, P., Rzanny, M., & Seeland, M. (2018). Automated plant species identification—Trends and future directions. *PLoS Computational Biology, 14*(4), 1–19.

10. Rzanny, M., Seeland, M., Wäldchen, J., & Mäder, P. (2017). Acquiring and preprocessing leaf images for automated plant identification: Understanding the tradeoff between effort and information gain. *Plant Methods*, *13*(1), 97–109.

11. Wei, Y., et al. (2020). Design features for improving mobile health intervention user engagement: Systematic review and thematic synthesis. *Journal of Medical Internet Research, 22*(12), 1–21.

12. Eaton, C., et al. (2024). User engagement with mHealth interventions to promote health behaviors: Systematic review. *Digital Health, 10,* 1–18.

13. Milne-Ives, M., et al. (2020). Mobile apps for health behavior change in physical activity, diet, drug and alcohol use, and mental health: Systematic review. *JMIR mHealth and uHealth*, *8*(3), 1–18.

14. Tran, S., et al. (2022). The use of gamification and incentives in mobile health apps for medication adherence: Scoping review. *JMIR Serious Games*, *10*(2), 1–15.

15. Cheng, V. W. S., et al. (2019). Gamification in apps and technologies for improving mental health and well-being: Systematic review. *JMIR Mental Health*, *6*(6), 1–16.

16. Ma, J. (2022). Interaction with nature indoor: Psychological impacts of houseplants care behaviour on mental well-being and mindfulness in Chinese adults. *Journal of Environmental Psychology, 82*, 1–10.

17. Lee, M., et al. (2015). Interaction with indoor plants may reduce psychological and physiological stress by suppressing autonomic nervous system activity in young adults: A randomized crossover study. *Journal of Physiological Anthropology*, *34*(21), 1–6.

18. Sun, Y., Liu, Y., Wang, G., & Zhang, H. (2017). Deep learning for plant identification in natural environment. *Computational Intelligence and Neuroscience*, *2017*, 1–6.

19. *Flutter documentation: Build apps for mobile, web, and desktop from a single codebase*. URL: https://flutter.dev (дата звернення 12.05.2026).

20. *The Go programming language specification and language documentation*. URL: https://go.dev/ref/spec (дата звернення 12.05.2026).

21. *Provider – a wrapper around InheritedWidget to make state management easier and more scalable in Flutter applications*. URL: https://pub.dev/packages/provider (дата звернення 12.05.2026).

22. *Go_router – declarative routing package for Flutter supporting deep links and nested navigation*. URL: https://pub.dev/packages/go_router (дата звернення 12.05.2026).

23. *Supabase documentation: open source Firebase alternative with PostgreSQL, authentication, storage and APIs*. URL: https://supabase.com/docs (дата звернення 12.05.2026).

24. *Firebase Cloud Messaging documentation: cross-platform messaging solution for Android, iOS and web applications*. URL: https://firebase.google.com/docs/cloud-mesaging (дата звернення 12.05.2026).

25. *Flutter_secure_storage – secure storage for sensitive data such as JWT tokens and credentials in Flutter applications*. URL: https://pub.dev/packages/flutter_secure_storage (дата звернення 12.05.2026).

26. *Flutter_local_notifications – Flutter plugin for displaying and scheduling local notifications on Android and iOS*. URL: https://pub.dev/packages/flutter_local_notifications (дата звернення 12.05.2026).

27. *Perenual Plant API documentation: plant species database, watering schedules, diseases and identification API*. URL: https://perenual.com/docs/api (дата звернення 12.05.2026).

28. *RevenueCat documentation: subscription and in-app purchase infrastructure for mobile applications*. URL: https://www.revenuecat.com/docs (дата звернення 12.05.2026).

29. Lin, Z., Althoff, T., & Leskovec, J. (2018). *I'll be back: On the multiple lives of users of a mobile activity tracking application*. arXiv preprint arXiv:1802.08972.

30. Lapkovskis, A., Nefedova, N., & Beikmohammadi, A. (2024). *Automatic fused multimodal deep learning for plant identification*. arXiv preprint arXiv:2406.01455.

31. Шевченко, О. О. (2026). Порівняльний аналіз моделей комп'ютерного зору для розпізнавання рослин. *МРФ 2026*. Конференція 7 ІТМ Інф, ПМ, ВМ, С. 184–185.

32. Шевченко, О. О. (2025). Інструменти візуалізації даних як засіб підтримки прийняття рішень. *ITMM 2025*, С. 258–260.