Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка мобільного застосунку: WorkTime Manager – менеджер робочого часу»

Виконав: студент 4 курсу
групи 5ПІ-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Сливка Р. М.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Мельник О. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф ОТ Кожем'яко А.В.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сливці Ростиславу Миколайовичу

1. Тема роботи – Розробка мобільного застосунку: WorkTime Manager – менеджер робочого часу.

Керівник роботи: Мельник Олександр Васильович, к. т. н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: середовище розробки Android Studio, мова розробки Kotlin, Windows 11, алгоритми перегляду даних пройдених робочих сесій.

4. Зміст текстової частини: вступ; аналіз розвитку мобільних систем для менеджменту робочого часу та постановка задач розробки, розробка моделей та алгоритмів програмного застосунку, розробка програмних модулів реалізації мобільного застосунку, тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: приклади аналогів; модель роботи застосунку; структурна схема застосунку; блок-схеми алгоритмів роботи застосунку; графічний інтерфейс застосунку; тестування застосунку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Мельник О. В., к.т.н, доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задачі	25.03.25 – 06.04.25	
2	Розробка моделей, структури та алгоритмів	07.04.25 – 20.04.25	
3	Розробка програмного забезпечення	20.04.25 – 07.05.25	
4	Тестування застосунку	07.05.25 – 20.05.25	
5	Оформлення матеріалів до захисту БДР	20.05.25– 30.05.25	

Студент _____ Р. М. Сливка
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ О.В. Мельник
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Сливка Р. М. Розробка мобільного застосунку: WorkTime Manager – менеджер робочого часу: бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – Іінженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 88 с. На укр. мові. Бібліогр.: 20 назв; рис.: 46; табл. 2.

У бакалаврській кваліфікаційній роботі проведено аналіз стану мобільних систем для менеджменту робочого часу користувача, доведено доцільність розробки власної програмної реалізації, зважаючи на порівняльний аналіз аналогів.

У ході виконання роботи розроблено мобільний застосунок: WorkTime Manager – менеджер робочого часу, що включає в себе модулі для відслідковування часу, блокування сповіщень та модуль перегляду результатів. Застосунок призначений для організації та контролю робочого часу користувача за допомогою вбудованого таймера. Під час активного таймера застосунок блокує сповіщення, що дозволяє зосередитися на виконанні завдань без зайвих відволікань.

Застосунок розроблено з використанням мови програмування Kotlin та середовища Android Studio. Для реалізації графічного інтерфейсу використано фреймворк Jetpack Compose, а для збереження даних – бібліотеку Room.

Розроблені програмні модулі можуть бути використані розробниками для створення мобільних систем.

Ключові слова: менеджер робочого часу, мобільна система, фреймворк.

ANNOTATION

Slyvka R. M. Development of a mobile application to support the activities of charitable organisations: bachelor's thesis in speciality 121 Software Engineering, educational programme - Software Engineering. Vinnytsia: VNTU, 2025. 88 p. In Ukrainian. Bibliography: 20 titles; Figures: 46; Table 2.

The bachelor's thesis analyses the state of mobile systems for managing user's working time, proves the feasibility of developing its own software implementation, taking into account the comparative analysis of analogues.

In the course of the work, a mobile application was developed: WorkTime Manager - a working time manager that includes modules for time tracking, blocking notifications and a module for viewing results. The application is designed to organise and control the user's working time using a built-in timer. When the timer is active, the application blocks notifications, which allows you to focus on performing tasks without unnecessary distractions.

The application was developed using the Kotlin programming language and the Android Studio environment. The Jetpack Compose framework was used to implement the graphical interface, and the Room library was used to store data.

The developed software modules can be used by developers to create mobile systems.

Keywords: working time manager, mobile system, framework.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний ритм життя потребує високого рівня самоорганізації, особливо в умовах зростаючих вимог до продуктивності та ефективного використання часу. Зокрема, для студентів, фахівців у різних галузях та осіб, які працюють віддалено, актуальним стає питання раціонального планування робочого часу та уникнення зовнішніх відволікань. Саме тому мобільні застосунки, що спрямовані на тайм-менеджмент, набувають все більшого значення у повсякденному житті.

Попри наявність великої кількості застосунків, орієнтованих на планування завдань або відстеження активності, більшість із них не забезпечує комплексного підходу до зосередженої роботи. Часто їм бракує функцій блокування сповіщень, гнучкого аналізу відпрацьованого часу або простого та інтуїтивного інтерфейсу для редагування історії сесій. Це створює потребу у розробці спеціалізованого мобільного застосунку, що поєднуватиме функції таймера, блокування сповіщень та ведення статистики продуктивності.

Розробка такого застосунку сприятиме підвищенню концентрації під час роботи, зниженню рівня цифрових відволікань та формуванню корисних звичок у сфері тайм-менеджменту. Він надасть користувачам простий у використанні інструмент для організації сесій зосередженої роботи, ведення обліку витраченого часу та його подальшого аналізу. Таким чином, створення мобільного застосунку: WorkTime Manager – менеджер робочого часу є актуальним і важливим завданням, що відповідає сучасним викликам цифрової епохи та сприяє підвищенню особистої ефективності.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення управління робочим часом користувачів через розробку мобільного застосунку, який сприятиме ефективному використанню часу, зменшуючи відволікання та

підвищуючи концентрацію. Застосунок забезпечить блокування сповіщень під час активних сесій, дозволяючи користувачам зосередитися на роботі.

Основними задачами дослідження є:

- розробка механізму для аналізу витраченого часу, що дозволить користувачам переглядати статистику, базуючись на їх діяльності;
- розробка інтерфейсу користувача, що забезпечить зручну взаємодію з застосунком та легкий доступ до необхідної інформації;
- розробка модуля відліку часу для відслідковування часу, витраченого на різні завдання та додаткові функції;
- розробка модуля для блокування сповіщень, що забезпечуватиме зосередженість користувачів під час виконання завдань;
- розробка модуля для збереження результатів відліку часу попередніх робочих сесій.

Об’єкт дослідження - процес розробки мобільного застосунку WorkTime Manager – менеджер робочого часу.

Предмет дослідження - методи та програмні засоби розробки мобільного застосунку WorkTime Manager – менеджер робочого часу.

Методи дослідження. У процесі досліджень використовувались: методи розробки мобільного застосунку для операційної системи Android; методи побудови структури застосунку з підтримкою масштабованості; методи розробки графічного інтерфейсу застосунку за допомогою фреймворку JetpackCompose задля створення зручного та адаптивного графічного інтерфейсу користувача; методи тестування задля забезпечення коректності роботи застосунку.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод автоматичного керування режимами сповіщень смартфона, який активує режим «Не турбувати» під час запуску таймера робочого часу, оптимізуючи фокус користувача та забезпечуючи миттєве усунення відволікаючих факторів.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби мобільного застосунку WorkTime Manager – менеджер робочого часу.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: обґрунтовано доцільність розробки мобільного застосунку для менеджменту робочого часу [1]; обґрунтовано інтеграцію Jetpack Compose, Room та Kotlin Coroutines у створенні мобільного застосунку [2].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення бакалаврської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних: 2-га Міжнародна науково-практична конференція «Сучасні наукові дослідження: Теоретичні та практичні аспекти» (2025); 2-га Міжнародна науково-практична конференція «Досягнення науки та прикладних досліджень» (2025).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях у матеріалах конференцій.

1 АНАЛІЗ РОЗВИТКУ МОБІЛЬНИХ СИСТЕМ ДЛЯ МЕНЕДЖМЕНТУ РОБОЧОГО ЧАСУ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз стану мобільних систем створених для менеджменту робочого часу

В останні роки мобільні застосунки для управління робочим часом користувачів здобули значну популярність, що відображає зростаючу потребу у вдосконаленні продуктивності та ефективності праці. Ці тенденції показують, що користувачі шукають прості та доступні інструменти, які дозволяють організувати робочий процес та мінімізувати відволікання. Мобільні застосунки [3], що дозволяють стежити за витраченим часом і блокувати сповіщення, стають важливими помічниками для тих, хто прагне підвищити свою продуктивність.

Наразі існують кілька аналогів застосунків для управління часом, проте більшість з них мають обмежену функціональність, складний інтерфейс або не забезпечують повноцінної інтеграції з іншими інструментами для організації роботи. Багато застосунків фокусуються лише на базовому відстежуванні часу, не надаючи можливості аналізувати продуктивність або налаштовувати індивідуальні параметри. Крім того, деякі з них не здатні блокувати сповіщення належним чином, що знижує їх функціональність.

З огляду на ці недоліки, важливо розробити більш функціональний і зручний мобільний застосунок, який забезпечить користувачам повний контроль над своїм робочим часом. Такий застосунок повинен не лише блокувати сповіщення, але й зберігати історію сесій, надавати можливість редагувати записані дані, а також забезпечувати аналіз витраченого часу для оптимізації робочого процесу.

Таким чином, розробка такого мобільного застосунку стане важливим кроком у напрямку покращення ефективності користувачів, дозволяючи їм краще організувати свій робочий час та підвищити продуктивність.

1.2 Порівняльний аналіз аналогів

Під час проведення досліджень аналогів до розроблюваного програмного застосунку було виявлено, що подібних додатків наразі існує досить багато. Це включає як популярні рішення для відстеження робочого часу, так і менш відомі застосунки, що пропонують різні функції для продуктивності. Однак, незважаючи на велику кількість доступних варіантів, більшість із цих застосунків не забезпечують належний функціонал.

В якості аналогів було обрано веб-сайти «Toggl Track», «Clockify» та «RescueTime».

Toggl Track – це кросплатформний мобільний застосунок для відстеження робочого часу, який широко використовується як окремими користувачами, так і командами [4]. Він допомагає користувачам ефективно контролювати, скільки часу витрачається на ті чи інші завдання, що особливо корисно для фрілансерів та організацій.

Основними особливостями Toggl Track є простий інтерфейс із можливістю швидкого запуску та зупинки таймера, а також розширені функції генерації звітів і аналітики. Крім того, застосунок підтримує синхронізацію між пристроями та інтегрується з популярними сервісами, такими як Trello, Asana, Slack та Google Calendar.

Мобільний застосунок «Toggl Track» представлено на рисунку 1.1 .

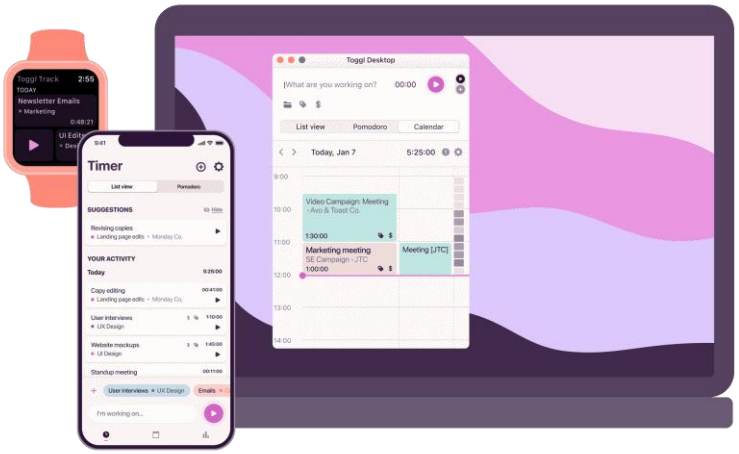


Рисунок 1.1 – Мобільний застосунок «Toggl Track»

Clockify – це безкоштовний кросплатформний мобільний застосунок для відстеження робочого часу, який підходить як для індивідуального використання, так і для команд [5]. Він дозволяє користувачам вести облік часу, що витрачається на різні завдання та проєкти, і зручно керувати своєю продуктивністю.

Мобільний застосунок «Clockify» представлено на рисунку 1.2.

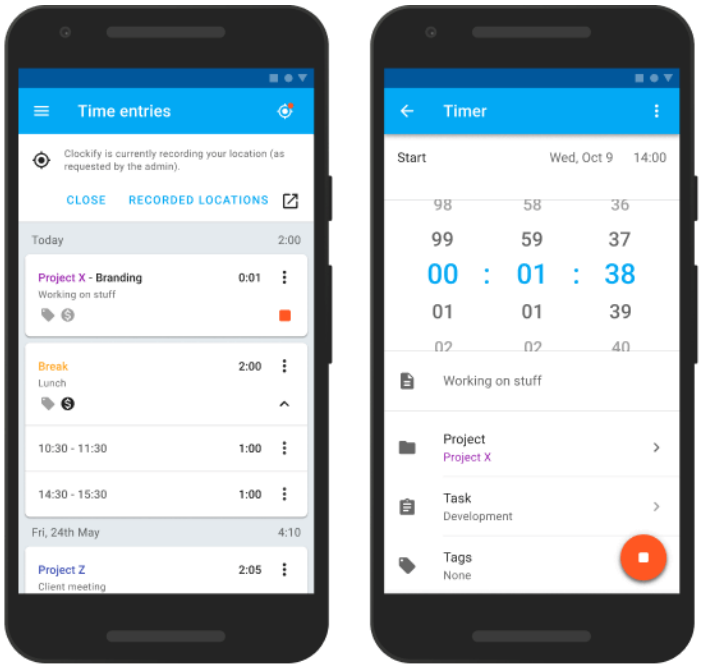


Рисунок 1.2 – Мобільний застосунок «Clockify»

Застосунок підтримує як ручне, так і автоматичне введення записів, а також надає гнучкий таймер для реального часу. Clockify дозволяє створювати детальні звіти про витрачений час, розподіляти години за проєктами, клієнтами чи завданнями. Він також містить функції для визначення оплачуваних і неоплачуваних годин, що корисно для фрілансерів або команд із погодинною оплатою.

RescueTime – це застосунок для автоматичного відстеження часу, який фокусується на аналізі цифрової поведінки користувача та допомагає підвищити продуктивність [6]. Він працює у фоновому режимі й збирає дані про

використання програм, вебсайтів та активностей без необхідності ручного введення.

Застосунок «RescueTime» представлено на рисунку 1.3.

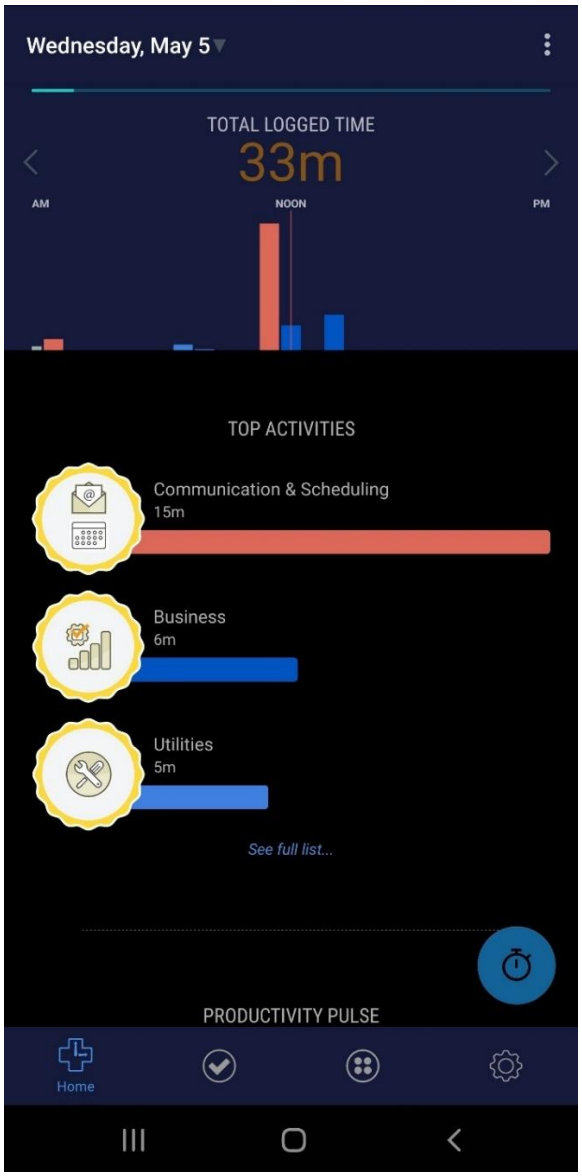


Рисунок 1.3 – Застосунок «RescueTime»

Однією з ключових функцій RescueTime є автоматична класифікація активностей за рівнем продуктивності – наприклад, робочі інструменти вважаються продуктивними, а соціальні мережі – ні. Застосунок надає щоденні та щотижневі звіти, які показують, скільки часу користувач провів у певних

додатках чи на сайтах. Також є функція "Focus Sessions", яка дозволяє блокувати відволікаючі сайти на заданий час.

Щоб підсумувати порівняння аналогів приведено таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Toggl Track	Clockify	RescueTime	Власна розробка
Ручний запуск таймера	+	+	-	+
Автоматичне блокування сповіщень	-	-	+	+
Збереження історії таймер-сесій	+	+	+	+
Редагування збережених сесій	+	+	-	+
Мінімалістичний інтерфейс	-	-	+	+
Офлайн-режим роботи	-	+	+	+

Відповідно до проведеного порівняльного аналізу, розробка власного мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача є обґрунтованою та актуальною. Запропонований продукт дозволяє реалізувати функціональні можливості, які відсутні або обмежено представлені в існуючих рішеннях, зокрема, блокування сповіщень, гнучке редагування сесій та роботу в офлайн-режимі.

В результаті, створення власного мобільного застосунку відкриває широкі можливості для покращення особистої ефективності користувачів. Застосунок сприятиме концентрації уваги, кращій організації часу та продуктивному виконанню повсякденних завдань за рахунок зручного та адаптованого функціоналу.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів мобільного застосунку: WorkTime Manager – менеджер робочого часу потребує всебічного аналізу та чіткого планування етапів реалізації. Успішне створення такого застосунку вимагає не лише технічних знань, а й розуміння поведінкових аспектів користувачів, які прагнуть підвищити власну продуктивність.

Цей процес можна поділити на кілька етапів:

Першим етапом є вивчення наявних аналогів на ринку – таймерів продуктивності, трекерів активності та фокус-застосунків. Аналізуються їхні сильні та слабкі сторони, функціональні обмеження, інтерфейси та зручність використання. Також досліджується досвід користувачів, відгуки в онлайн-сервісах та форумах, що дозволяє краще зрозуміти поточні потреби цільової аудиторії.

Наступним кроком є формування вимог до застосунку. Це несе у собі визначення ключових сценаріїв використання: запуск таймера для зосередженої роботи, автоматичне блокування сповіщень, збереження та редагування сесій. На цьому етапі створюється базовий концепт інтерфейсу та окреслюються функціональні модулі системи.

Після цього обираються технології для реалізації: мова програмування, фреймворки для мобільної розробки, локальне або хмарне збереження даних. Важливою є сумісність з популярними мобільними платформами, зокрема Android, а також забезпечення стабільної роботи в офлайн-режимі.

Далі проектується архітектура застосунку – модульна структура, з чітким розмежуванням між інтерфейсною частиною, логікою обробки даних та механізмом збереження інформації. Особлива увага приділяється мінімалістичному та інтуїтивному дизайну, оскільки цільова аудиторія очікує простоти й ефективності.

На завершальному етапі відбувається реалізація функціоналу, тестування застосунку в різних сценаріях та виправлення помилок. Після завершення

технічної частини можливе пілотне розгортання серед обмеженої групи користувачів для збирання зворотного зв’язку та подальших покращень.

В результаті, розглянувши кожен метод, переваги та недоліки, було прийнято рішення створити мобільний застосунок, який не лише дозволяє зосереджено працювати без відволікань, але й надає можливості для перегляду, аналізу та управління власним робочим часом у зручному та сучасному форматі.

1.4 Постановка задачі

Для успішної розробки мобільного застосунку для ефективного управління робочим часом необхідно вирішити ряд завдань:

- Провести аналіз потреб користувачів, які прагнуть покращити свою продуктивність та організацію часу.
- Виявити ключові функції, які повинні бути включені в застосунок для забезпечення зручного трекінгу часу та блокування відволікаючих сповіщень.
- Сформулювати основні функції, такі як старт/стоп таймерів, автоматичне блокування сповіщень та збереження історії сесій.
- Встановити пріоритети функцій на основі важливості для користувачів, таких як простота використання та доступність.
- Обрати платформу для розробки мобільного застосунку (наприклад, Android, iOS або кросплатформену розробку).
- Розробити структуру застосунку, визначити основні модулі та компоненти.
- Забезпечити оптимальну функціональність, що дозволяє користувачам зручно управляти своїм робочим часом.
- Врахувати потреби цільової аудиторії для забезпечення оптимальної сумісності та доступності застосунку.
- Розробити програмний код, що відповідає визначеним вимогам та функціональності, обробка даних та збереження інформації.
- Створити інтуїтивно зрозумілий та зручний інтерфейс для взаємодії користувачів з додатком.

- Виконати тестування застосунку на різних пристроях для перевірки його стабільності та ефективності.
- Виявити та виправити будь-які помилки або недоліки, щоб забезпечити безперебійну роботу програми.

Виконання цих завдань дозволить створити мобільний застосунок, який ефективно допоможе користувачам управляти своїм часом, покращуючи їхню продуктивність і фокус, а також знижуючи рівень відволікань.

1.5 Висновки

У першому розділі розглянуто стан програмних засобів для розробки мобільного застосунку для управління робочим часом користувачів. Проаналізовано існуючі програмні засоби, такі як Toggl Track, Clockify та RescueTime. Кожен з цих застосуків має свої переваги і недоліки, які були детально проаналізовані.

Проаналізувавши сильні та слабкі сторони існуючих реалізацій, сформульовано завдання для подальшої розробки власного програмного забезпечення мобільної системи, яка відповідає потребам ефективного управління робочим часом користувачів. Таким чином, доведена доцільність розробки власного рішення. На основі отриманих результатів було сформовано перелік завдань, які необхідно виконати для створення власних програмних засобів.

2 РОЗРОБКА МОДЕЛЕЙ ТА ІНТЕРФЕЙСУ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи для менеджменту робочого часу

Аналіз вхідних даних системи є ключовим етапом у розробці мобільного застосунку WorkTime Manager – менеджер робочого часу. Цей процес полягає у ретельному розгляді та розумінні різних типів інформації, яка може надходити до системи через взаємодію користувачів із застосунком.

Одним із важливих типів вхідних даних є налаштування таймера. Користувач може налаштовувати таймер для відліку часу на конкретні завдання, вибираючи параметри початку і кінця сесії, а також встановлювати інтервали для нагадувань або сповіщень. Такі дані дозволяють системі коректно вести облік часу і забезпечувати ефективну організацію робочого процесу.

Іншим важливим типом вхідних даних є інформація сесій та їх редагування. Користувач може змінювати або коригувати вже збережені записи, наприклад, редагувати тривалість сесії або змінювати категорію завдання. Це дозволяє підтримувати точність обліку часу та дає користувачам можливість коригувати свої записи в разі необхідності.

В результаті, аналіз вхідних даних допомагає забезпечити правильне функціонування застосунку, оптимізувати використання ресурсу та покращити його якість. Це важливий етап у розробці, оскільки від правильного аналізу даних залежить успішність та працездатність програмного забезпечення.

2.2 Розробка моделі системи для менеджменту робочого часу

Для більш детального розуміння функціонування модулів програмного забезпечення мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача було вирішено створити модель системи з використанням UML діаграм [7].

Діаграма діяльності є візуальним відображенням послідовності операцій або процесів, що відбуваються в системі. Вона показує, як елементи взаємодіють

між собою під час виконання конкретних завдань чи процесів. Кожен етап діяльності позначається прямокутником, що містить опис операції або функції, що виконується.

Основною перевагою діаграм діяльності є їхня простота та наочність, що дозволяє швидко оцінити хід подій [8]. Такі діаграми активно використовуються в аналізі бізнес-процесів, системному аналізі та управлінні проєктами. Вони також допомагають виявляти проблеми в процесах та здійснювати їх оптимізацію.

Діаграму діяльності програмних модулів мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача відображено на рисунку 2.1.

Рисунок 2.1 – Діаграма діяльності програмних модулів

Згідно створеної діаграми, користувач на початку взаємодії із програмним застосунком потрапляє одразу ж на головний екран застосунку.

На головному екрані застосунку перед користувачем відкриється навігаційне меню для кращого розуміння основних екранів застосунку.

Після цього користувач матиме змогу перейти до потрібного йому екрана: «Фокус» або «Результати».

Наступним кроком буде перенесення користувача на екран результатів застосунку.

Після успішної взаємодії із функціями застосунку, користувач зможе закінчити роботу із застосунком.

Розробка діаграми діяльності була виконана у програмному забезпеченні Microsoft Visio [9].

2.3 Розробка алгоритму та структури системи для менеджменту робочого часу

Перед розробкою мобільного застосунку була розроблена його архітектура поділена загальні модулі, які визначатимуть послідовність дій користувача в застосунку.

Згідно цього модулю, перш за все, користувач бачить завантажувальний екран, після цього, користувач потрапляє на головний екран застосунку.

Також, перед користувачем візуалізовано меню навігації, що складається з двох екранів: «Фокус» та «Результати». У випадку, якщо користувач залишиться на головному екрані, він бачитиме навігаційне меню.

При переході на екран «Результати» користувач побачить усі фокус-сесії, їх дату та час. Таким чином, запускається модуль перегляду результатів.

Також, було створено навігаційну діаграму програмних модулів мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача.

Створення діаграми навігації мобільного застосунку включає в себе визначення структури додатка та послідовності переходів між екранами [10].

Діаграма навігації мобільного застосунку відображена на рисунку 2.2.

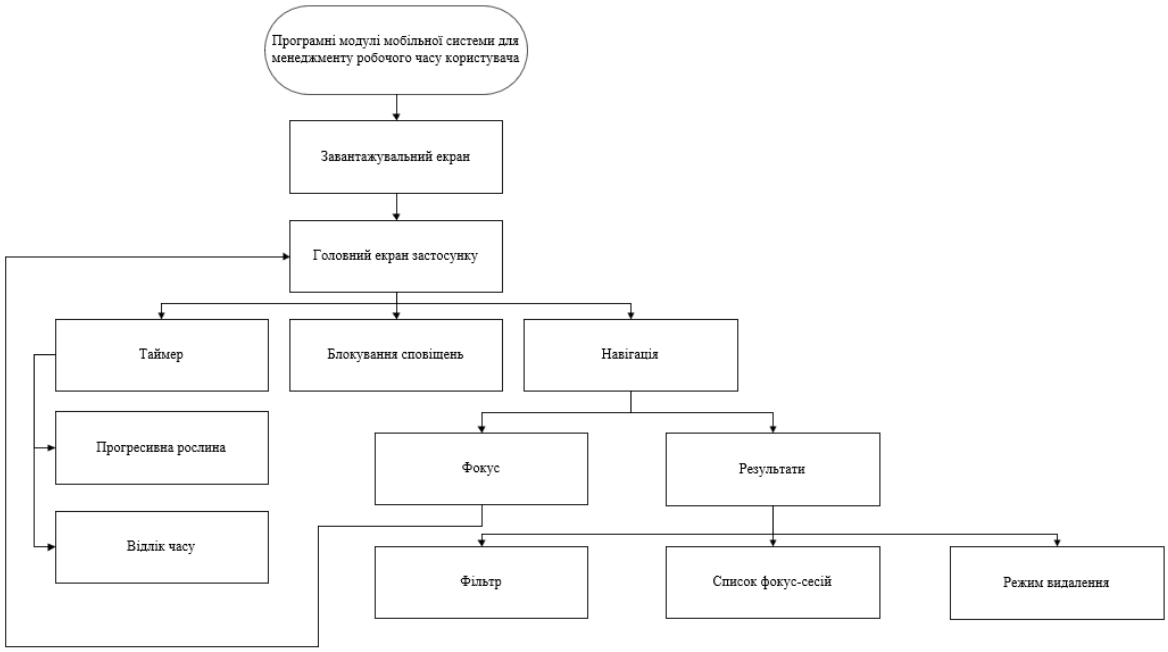


Рисунок 2.2 – Діаграма навігації мобільного застосунку

Також було розроблено блок-схему алгоритму роботи застосунку.

Алгоритм роботи застосунку WorkTime Manager розпочинається з запуску програми, після чого виконується ініціалізація необхідних систем. На першому етапі користувач обирає опцію запуску таймера, що призводить до автоматичного вмикання режиму «Не турбувати» на смартфоні для мінімізації відволікань. Далі система перевіряє, чи введено таймер або чи обрано автоматичний режим, і залежно від цього або продовжує відлік часу, або пропонує користувачу ввести потрібні дані.

Після запуску таймера алгоритм переходить до циклу відліку, де періодично перевіряється, чи не було перервано процес користувачем. Якщо перерва відсутня, таймер продовжує роботу, а по завершенні відображає статистику використаної робочої години. У разі переривання чи завершення таймера система повертається до початкового стану, дозволяючи користувачу знову запустити цикл або завершити роботу застосунку, що завершується зумуванням таймера.

Блок-схема алгоритму роботи застосунку відображена на рисунку 2.3.

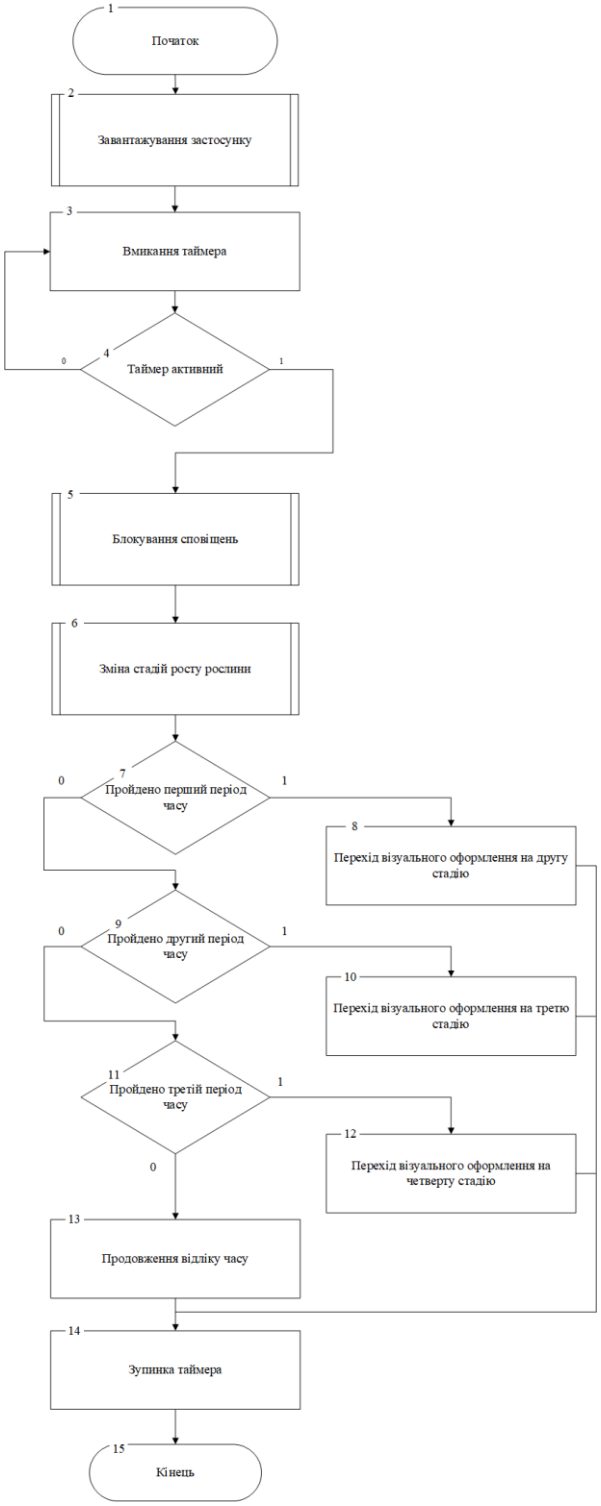


Рисунок 2.3 – Блок-схема алгоритму роботи застосунку

В результаті, розробка архітектури системи є важливою складовою процесу розробки, що дозволяє забезпечити зручну навігацію для користувачів.

2.4 Розробка інтерфейсу програмного застосунку для менеджменту робочого часу

При розробці інтерфейсу програмного застосунку було приділено особливу увагу його зрозумілості та зручності для користувача.

Основними кольорами інтерфейсу обрано білий та зелений, оскільки ці кольори є досить актуальними відносно теми та легким для сприйняття користувачами. Також допоміжними кольорами обрано оранжевий та фіолетовий.

При відкритті мобільного застосунку, користувача зустрічає його початковий екран, що відображено на рисунку 2.4.

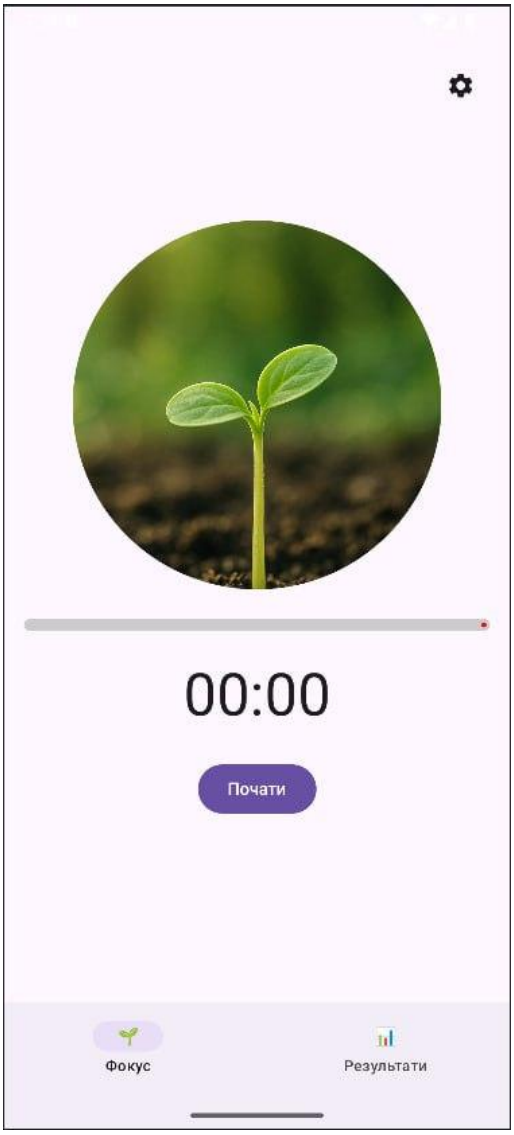


Рисунок 2.4 – Початковий екран

При запуску застосунку користувача зустрічає початковий екран, де перед ним відобразиться основний екран застосунку, у якому відображено сам таймер, кнопка його запуску, та меню.

Якщо користувач натисне кнопку «Почати», таймер почне свою дію. Відповідно до пройденого часу, користувача буде зустрічати рослина, яка з часом буде переходити на нову стадію росту та позначатиме пройдений зфокусований час.

Друга стадія відліку часу відображена на рисунку 2.5.

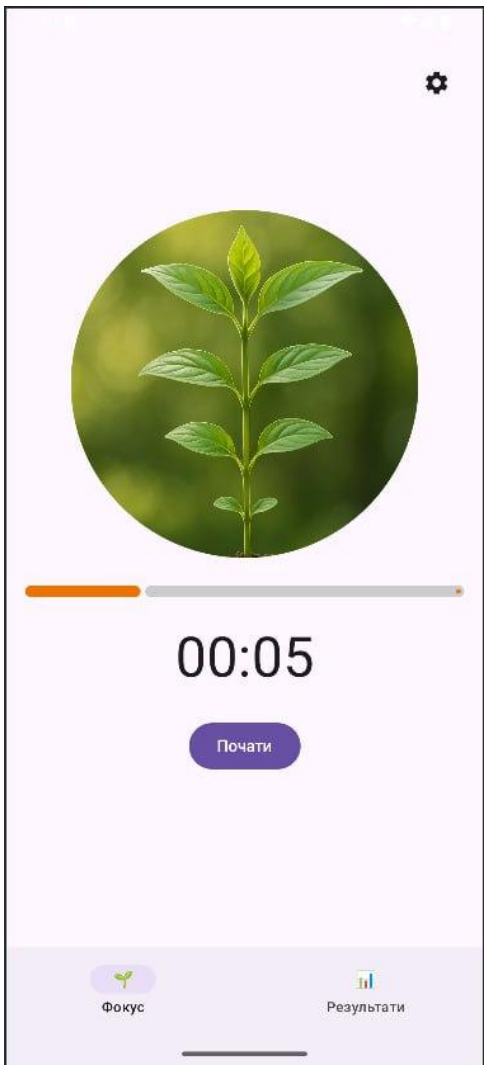


Рисунок 2.5 – Друга стадія відліку часу

Після того, як проходить певний час, стадії відліку часу змінюються, та відповідно, змінюється і прогрес росту рослини.

Третю стадію відліку часу відображено на рисунку 2.6.

Рисунок 2.6 – Третя стадія відліку часу

Четверту стадію відліку часу відображено на рисунку 2.7.

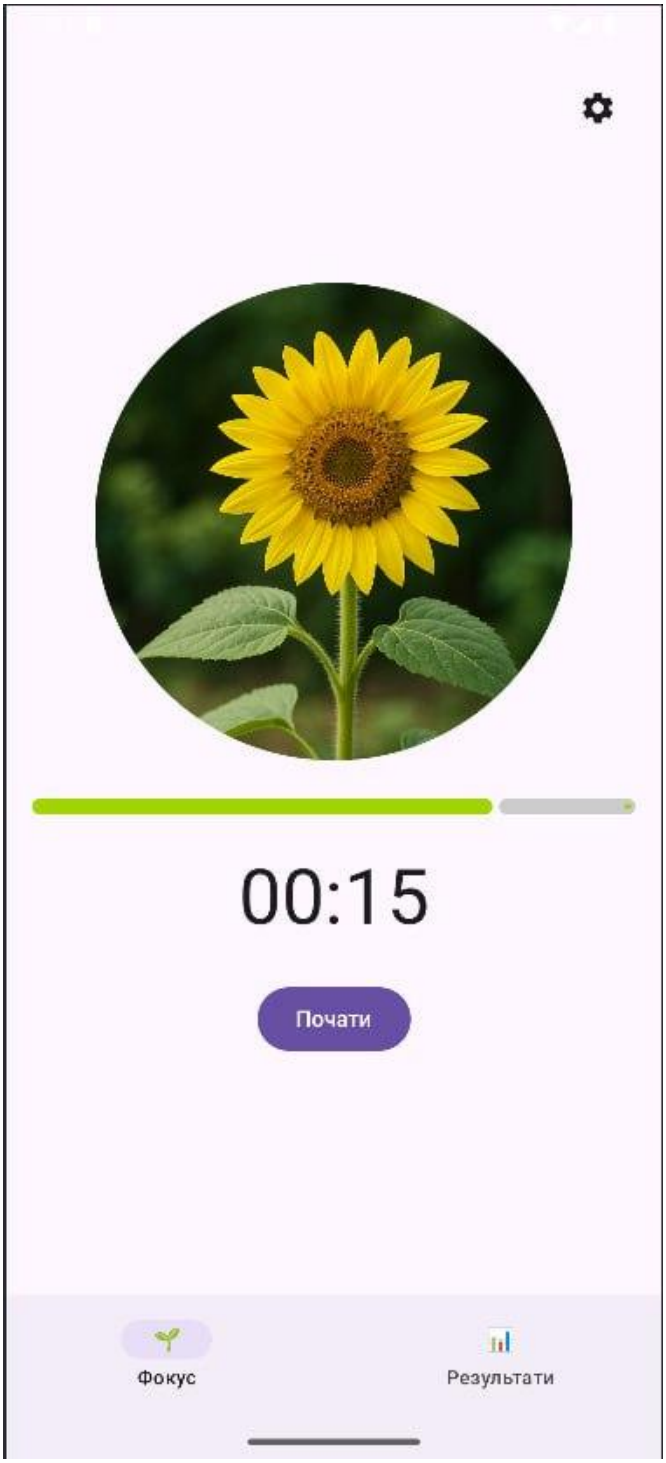


Рисунок 2.7 – Четверта стадія відліку часу

Користувач має можливість відслідковувати сесії таймера шляхом переходу на екран «Результати», де відображено вирощені рослини, що показують записаний пройдений час кожного фокусу та його дату.

Екран «Результати» відображено на рисунку 2.8.

Рисунок 2.8 – Екран «Результати»

Після перегляду результатів користувач матиме змогу видалити сесію, натиснувши кнопку «Режим видалення» для усунення застарілих сесій, що відображено на рисунку 2.9.

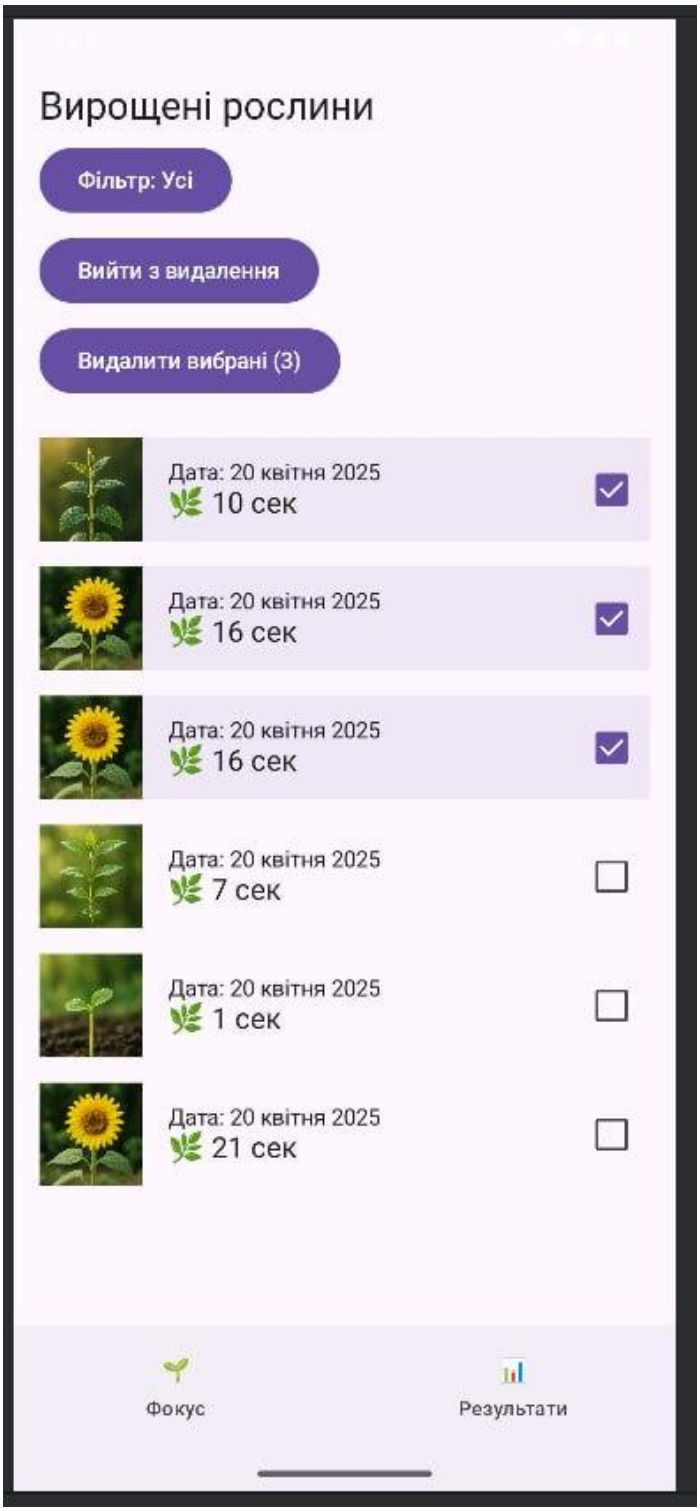


Рисунок 2.9 – Перехід користувачем в «Режим видалення»

Також користувач має змогу вимкнути сповіщення, натиснувши кнопку «Почати», тим самим, застосунок автоматично вимикає сповіщення.

Показ вимкнення сповіщень пристрою відображено на рисунку 2.10.

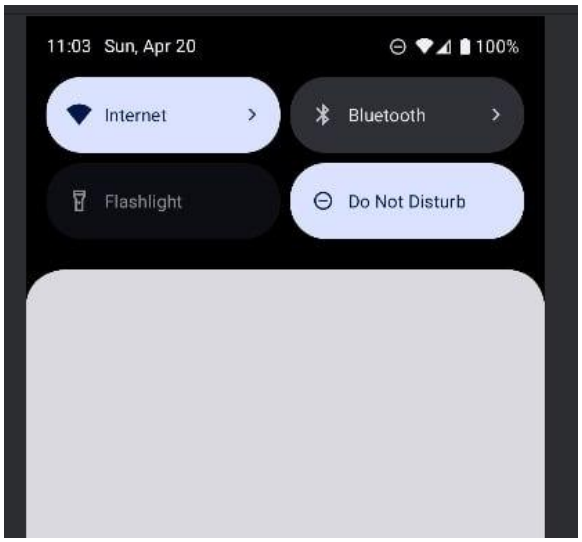


Рисунок 2.10 – Вимкнення сповіщень пристрою

Закінчується екран результатів кнопкою «Фільтр», що дає змогу відсортувати усі фокус-сесії за датою та часом.

Сортування фокус-сесій застосунку за допомогою кнопки «Фільтр» відображено на рисунку 2.11.

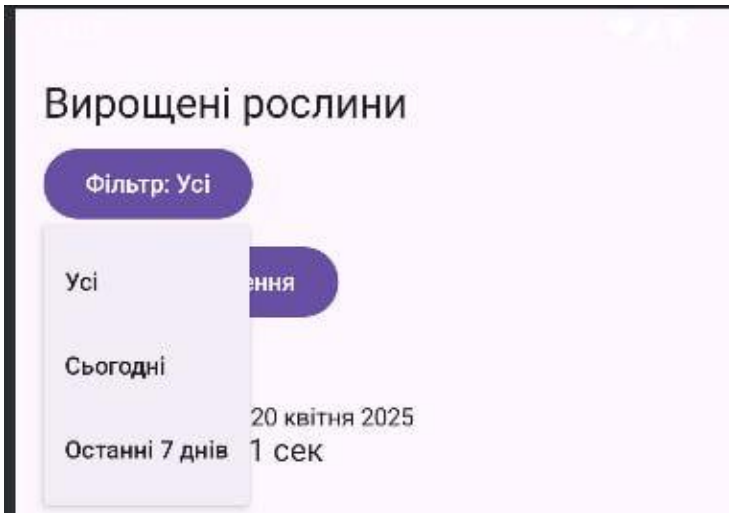
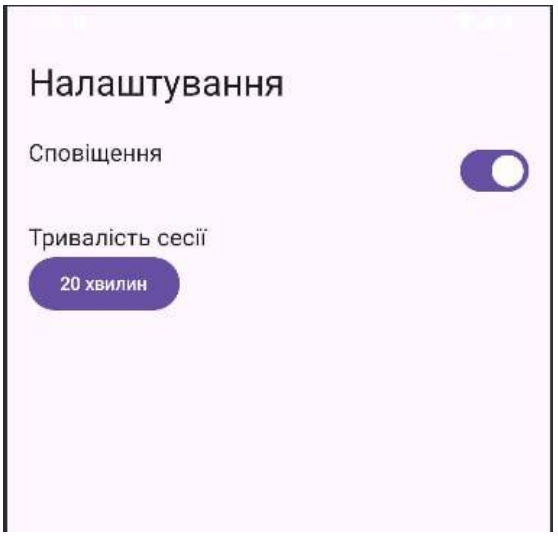


Рисунок 2.11 – Сортування фокус-сесій застосунку за допомогою кнопки «Фільтр»

Також користувач має змогу перейти до налаштувань застосунку, натиснувши на шестерню, що відображено на рисунку 2.12.



Розробка графічного інтерфейсу програмного застосунку була проведена у середовищі розробки Android Studio з використанням мови програмування Kotlin та її технології JetpackCompose.

2.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних програмних засобів. Розроблено основний алгоритм роботи мобільного застосунку та архітектуру системи. Також було розроблену модель роботи мобільного застосунку за допомогою UML діаграм. Також, розглянуто процес створення графічного інтерфейсу програмного застосунку.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Варіантний аналіз і обгрунтування вибору засобів для реалізації програмного забезпечення

При розробці програмного забезпечення мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача важливо обрати найбільш оптимальні технології для розробки кожного модуля програмних засобів.

Kotlin – це сучасна, статично типізована мова програмування, яка розроблена компанією JetBrains. Вона була представлена у 2011 році, а в 2017 році Google оголосила Kotlin офіційною мовою для розробки Android-додатків, що підвищило її популярність серед розробників мобільних застосунків. Kotlin є інтерпретованою мовою, що підтримує об’єктно-орієнтоване, функціональне та імперативне програмування, і її синтаксис спрощує розробку в порівнянні з іншими мовами, такими як Java [11].

Основною метою Kotlin є забезпечення зручності та безпеки при розробці додатків, з одночасною сумісністю з існуючими бібліотеками та фреймворками на Java. Kotlin не тільки замінив Java як основну мову для Android-розробки, а й став важливим інструментом для створення серверних застосунків, десктопних застосунків і навіть веб-додатків за допомогою фреймворків, таких як Ktor [12].

Основними особливостями Kotlin є:

1. Kotlin забезпечує перевірку типів на етапі компіляції, що допомагає уникнути багатьох помилок під час виконання програми.

2. Kotlin пропонує зручні синтаксичні конструкції, які дозволяють писати менше коду при збереженні високої продуктивності. Наприклад, функції в Kotlin можна записати коротше, ніж в Java, завдяки використанню лямбда-виразів, розширень функцій та інших інноваційних можливостей.

3. Однією з найпотужніших особливостей Kotlin є підтримка нульових типів на рівні мови. Це дозволяє запобігти NullPointerException, яка є однією з

найпоширеніших помилок в Java. У Kotlin є чітке розмежування між типами, які можуть бути null, і тими, що не можуть, що підвищує безпеку коду.

4. Kotlin повністю сумісний з Java, що дає змогу використовувати наявні бібліотеки та фреймворки. Це забезпечує зручний перехід для розробників, які хочуть покращити свою продуктивність, використовуючи Kotlin, не відмовляючись від існуючого коду на Java.

5. Kotlin підтримує функціональний стиль програмування, що дає змогу використовувати концепції, такі як вищі функції, лямбда-вирази та невизначені функції, що дозволяє створювати чистий і елегантний код.

6. Kotlin дозволяє створювати додатки для різних платформ, таких як Android, iOS, веб та сервери, використовуючи одну спільну кодову базу. Це можливо завдяки підтримці Kotlin Multiplatform, що дозволяє писати код, який працює як на Android, так і на iOS без дублювання.

Для розробки ефективних та зручних мобільних застосунків, які потребують гнучкої роботи з інтерфейсами та базами даних, використовуються сучасні технології Android, такі як Jetpack Compose для створення декларативних інтерфейсів і Room для ефективного управління даними в базах даних. Ці інструменти дозволяють спростити процес розробки та забезпечити високу продуктивність і масштабованість застосунків.

Jetpack Compose – це декларативний фреймворк для створення користувацьких інтерфейсів в Android-додатках, який було представлено Google як частину Android Jetpack [13]. Він дозволяє спростити розробку інтерфейсів завдяки використанню мови Kotlin і надає можливість писати більш компактний, зрозумілий і менш схильний до помилок код.

Jetpack Compose дозволяє описувати UI за допомогою функцій, де кожен елемент інтерфейсу визначається як функція. Це відрізняється від традиційного імперативного підходу до розробки інтерфейсів, який використовує XML в Android [14]. Використання декларативного підходу в Compose дозволяє автоматично оновлювати інтерфейс, коли дані змінюються, без необхідності вручну керувати змінами стану UI.

Основними перевагами Jetpack Compose є:

1. Розробники описують, як інтерфейс повинен виглядати в залежності від даних, а не як оновлювати його вручну. Це дозволяє зменшити кількість коду та зробити додаток більш гнучким.

2. Jetpack Compose дозволяє розробникам створювати кастомізовані UI компоненти, які інтегруються з іншими частинами системи без необхідності застосовувати складні бібліотеки.

3. оскільки Jetpack Compose побудований на основі Kotlin, розробники можуть використовувати всю потужність цієї мови для створення чистого та легкого в підтримці коду. Це дозволяє інтегрувати функціональність, таку як лямбда-вирази, розширення функцій та інші конструкції мови Kotlin.

4. Jetpack Compose працює в реактивному стилі, що означає, що UI автоматично оновлюється, коли змінюються дані або стан програми. Це дає можливість зменшити складність коду, адже не потрібно вручну керувати змінами стану елементів.

5. Jetpack Compose допомагає позбутися великої кількості шаблонного коду, який традиційно використовувався для визначення елементів інтерфейсу, що полегшує процес розробки та підтримки додатків.

Room – це бібліотека для роботи з базами даних в Android, яка надає абстракцію над SQLite і спрощує збереження та отримання даних у додатках [15].

Вона є частиною Android Jetpack і дозволяє зберігати об’єкти Java або Kotlin у базі даних за допомогою простого API, що знижує складність роботи з базами даних порівняно з використанням безпосередньо SQLite.

Room працює за принципом об’єктно-реляційного відображення (ORM), що дозволяє мапити об’єкти на таблиці бази даних без необхідності писати складні SQL-запити вручну.

SQLite – це легка, вбудована реляційна база даних, яка широко використовується в мобільних застосунках, зокрема для Android [16].

Вона дозволяє зберігати дані локально на пристрої, що є важливим для застосунків, які не завжди мають доступ до Інтернету або потребують швидкого

доступу до інформації. SQLite є безкоштовною та відкритою, а також підтримується на всіх мобільних платформах, включаючи Android.

У Android-застосунках SQLite використовується для збереження великих обсягів даних, таких як користувацькі налаштування, історія, списки, а також для кешування даних. Інтерфейс SQLite в Android реалізується через SQLiteDatabase і SQLiteOpenHelper, що дозволяє ефективно керувати базами даних.

Оскільки SQLite є вбудованою базою даних, вона не вимагає встановлення або конфігурації серверної частини, що робить її ідеальним рішенням для мобільних пристроїв, де ресурси обмежені.

Вона також має низьке споживання пам'яті та високу швидкість роботи, що дозволяє швидко обробляти запити до бази даних навіть у великих застосунках.

Враховуючи потребу в збереженні даних на пристрої без необхідності підключення до серверів.

Room є сучасним абстракційним рівнем над SQLite, що забезпечує легшу інтеграцію та обробку бази даних за допомогою об'єктно-орієнтованого підходу.

Основними перевагами Room є:

1. Room пропонує зручний API для створення і використання бази даних, що робить роботу з даними менш схильною до помилок і більш зрозумілою.

2. оскільки Room тісно інтегрується з Kotlin, можна використовувати переваги мови для написання типобезпечного коду, що автоматично генерується компілятором.

3. Room автоматично генерує SQL-код для всіх операцій, таких як вставка, оновлення та вибірка даних. Це відчутно спрощує роботу з базами даних.

4. Room інтегрується з LiveData, що дозволяє створювати реактивні додатки, які автоматично оновлюють UI при зміні даних у базі даних. Також вона підтримує Kotlin Coroutines, що дозволяє асинхронно працювати з даними без блокування основного потоку.

5. Завдяки використанню Room, розробники можуть написати чистий код, що легко тестується, та використовувати інструменти для перевірки коректності роботи з базою даних.

Kotlin є офіційною мовою програмування для розробки Android-застосунків і активно використовується в середовищі Android Studio. Ця мова була впроваджена Google як основний інструмент для створення мобільних застосунків через її сучасність, простоту та сумісність з Java.

Android Studio – це інтегроване середовище розробки (IDE) від Google, яке забезпечує всі необхідні інструменти для ефективної роботи з Kotlin, Java та іншими технологіями для розробки Android-застосунків [17].

Вона включає потужні функції для налагодження, тестування, та візуального дизайну інтерфейсів, що полегшує процес створення додатків.

Android Studio активно підтримує як Kotlin, так і всі інші компоненти екосистеми Android, зокрема такі технології, як Jetpack Compose для інтерфейсів та Room для роботи з базами даних.

Отже, вибір технологій для розробки мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача, зокрема таких як Kotlin, Jetpack Compose, Room та SQLite, є досить важливим для створення ефективного і надійного мобільного застосунку.

Kotlin є потужною мовою програмування, яка дозволяє розробляти швидкі, надійні та масштабовані додатки для Android. Використання Jetpack Compose дозволяє створювати сучасні, адаптивні та інтуїтивно зрозумілі інтерфейси користувача, що сприяє полегшенню взаємодії з додатком для керування робочим часом.

Зберігання даних та їх обробка через Room та SQLite забезпечує ефективне управління інформацією про робочі сесії, дозволяючи зберігати історію роботи та здійснювати швидкий доступ до необхідних даних навіть без підключення до Інтернету.

3.2 Розробка модуля відслідковування часу

При розробці модуля відслідковування часу для мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача особливу увагу було приділено зручності використання та надійності функціоналу. Основною метою цього модуля є забезпечення користувачів інструментом для точного відстеження витраченого часу на різні завдання чи проєкти. Це дозволяє користувачам легко фіксувати час, що витрачається на робочі сесії, а також оптимізувати їх розподіл для більш ефективної організації робочого дня.

При розробці модуля відслідковування часу в мобільному застосунку: WorkTime Manager – менеджер робочого часу користувача було реалізовано простий і водночас ефективний механізм таймера. Основна логіка відліку часу побудована на основі корутин Kotlin та використання LaunchedEffect з Jetpack Compose, що дозволяє реалізовувати асинхронну поведінку без перевантаження головного потоку [18].

Фрагмент коду із застосуванням механізму таймера відображено на рисунку 3.1.

```
// Таймер
LaunchedEffect(isTimerRunning) {
    while (isTimerRunning) {
        delay(1000)
        timeInSeconds++
    }
}
```

Рисунок 3.1 – Фрагмент коду із застосуванням механізму таймера

Одним із ключових елементів взаємодії з користувачем є кнопка, яка перемикає стан таймера – запускає або зупиняє його.

Фрагмент коду для реалізації логіки кнопки перемикання таймера відображено на рисунку 3.2.

```
Button(onClick = {
    if (isTimerRunning) {
        savePlantResult(timeInSeconds, context, scope)
        disableDoNotDisturb(context)
    } else {
        requestDoNotDisturbPermission()
        enableDoNotDisturb(context)
    }
    isTimerRunning = !isTimerRunning
}) {
    Text(if (isTimerRunning) "Зупинити" else "Почати")
}
```

Рисунок 3.2 – Фрагмент коду для реалізації логіки кнопки перемикання таймера

Крім функціональної частини, реалізовано також візуальний елемент – динамічне зображення стадії росту рослини, яке змінюється залежно від тривалості сесії. Таким чином, користувач може не лише бачити відлік часу, але й візуально спостерігати прогрес, що сприяє підвищенню залученості та створює ефект досягнення мети.

Фрагмент коду для відображення стадії росту продемонстровано на рисунку 3.3.

```
fun getGrowthStageRes(seconds: Int): Int {
    return when {
        seconds < 5 -> R.drawable.stage1
        seconds < 10 -> R.drawable.stage2
        seconds < 15 -> R.drawable.stage3
        else -> R.drawable.stage4
    }
}
```

Рисунок 3.3 – Фрагмент коду для відображення стадії росту

Ця функція повертає ресурс зображення відповідно до кількості секунд, що минули від початку сесії. На початку відображається початкова стадія (паросток), яка згодом змінюється на більш розвинені етапи росту рослини, до повністю сформованої. Кожен етап відображає поступовий прогрес, що символізує розвиток зосередженості користувача протягом робочого часу.

Такий підхід не лише функціональний, а й емоційно мотивуючий, адже користувач бачить «плід» своєї концентрації у вигляді вирощеної віртуальної рослини. Це робить застосунок не просто інструментом для відслідковування часу, а також засобом підтримки дисципліни та внутрішньої мотивації.

Отже, модуль відслідковування часу функціонує автономно, забезпечуючи безперервний облік тривалості робочої сесії без необхідності постійної взаємодії з користувачем. Простота запуску й зупинки таймера, а також візуалізація прогресу у вигляді росту рослини, створюють інтуїтивно зрозумілий інтерфейс.

3.3 Розробка модуля блокування сповіщень

При розробці модуля блокування сповіщень основною метою є мінімізація відволікаючих факторів під час активної робочої сесії користувача, що дозволяє зосередитися на виконанні завдань та підвищити продуктивність. Подальшого розвитку отримав метод автоматичного керування режимами сповіщень смартфона, який активує режим «Не турбувати» під час запуску таймера робочого часу, оптимізуючи фокус користувача та забезпечуючи миттєве усунення відволікаючих факторів.

При розробці модуля блокування сповіщень основною метою є мінімізація відволікаючих факторів під час активної робочої сесії користувача, що дозволяє зосередитися на виконанні завдань та підвищити продуктивність. У межах даного мобільного застосунку цей функціонал реалізовано через інтеграцію з системним режимом "Не турбувати" (Do Not Disturb), який тимчасово вимикає сповіщення від інших програм.

Для реалізації такого підходу необхідно виконати декілька послідовних кроків. Насамперед, додаток запитує у користувача дозвіл на зміну політики

сповіщень через системні налаштування. Це здійснюється за допомогою виклику Settings.ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS, який відкриває відповідне меню в Android.

Фрагмент коду для запиту дозволу відображено на рисунку 3.4.

```
// Запит дозволу на "не турбувати"
fun requestDoNotDisturbPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        if (!notificationManager.isNotificationPolicyAccessGranted) {
            val intent = Intent(Settings.ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS)
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
            context.startActivity(intent)
        }
    }
}
```

Рисунок 3.4 – Фрагмент коду для запиту дозволу

Після отримання дозволу застосунок може самостійно вмикати або вимикати режим "Не турбувати" в залежності від стану таймера. Якщо користувач розпочинає робочу сесію, сповіщення автоматично вимикаються, і навпаки – після завершення сесії режим скасовується, і користувач знову отримує всі повідомлення.

Реалізація увімкнення блокування сповіщень відображена на рисунку 3.5.

```
// Увімкнення режиму "не турбувати"
fun enableDoNotDisturb(context: Context) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        if (notificationManager.isNotificationPolicyAccessGranted) {
            notificationManager.setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_NONE)
        }
    }
}
```

Рисунок 3.5 – Реалізація блокування сповіщень

Завдяки такій реалізації, користувач отримує повністю автоматизований процес блокування сторонніх повідомлень, що дозволяє створити оптимальні

умови для зосередженої роботи. Це підвищує продуктивність таймера та позитивно впливає на досвід користування мобільним застосунком загалом.

Реалізація вимкнення блокування сповіщень відображена на рисунку 3.6.

```
// Вимкнення режиму "не турбувати"
fun disableDoNotDisturb(context: Context) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        if (notificationManager.isNotificationPolicyAccessGranted) {
            notificationManager.setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_ALL)
        }
    }
}
```

Рисунок 3.6 – Реалізація вимкнення блокування сповіщень

Завдяки такій реалізації, користувач отримує повністю автоматизований процес блокування сторонніх повідомлень, що дозволяє створити оптимальні умови для зосередженої роботи. Це підвищує продуктивність таймера та позитивно впливає на досвід користування мобільним застосунком загалом.

Отже, було створено модуль блокування сповіщень, що інтегрується з системним режимом "Не турбувати" та автоматично активується під час роботи таймера. Такий підхід дозволяє мінімізувати відволікання користувача під час робочої сесії та підвищити загальну продуктивність взаємодії з застосунком.

3.4 Висновки

У третьому розділі обґрунтовано вибір технологій та інструментів, що використовувались для розробки програмних модулів мобільного застосунку: WorkTime Manager – менеджер робочого часу користувача. Після аналізу поставлених задач і особливостей застосунку було обрано мову програмування Kotlin, середовище Android Studio, а також бібліотеки Jetpack Compose для побудови інтерфейсу та Room для збереження локальних даних. У цьому розділі також розглянуто реалізацію ключових модулів, таких як модуль таймера з візуалізацією стадій росту рослини та модуль автоматичного блокування сповіщень

4 ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Тестування системи

Після завершення розробки мобільного застосунку було проведено тестування для перевірки його працездатності, надійності та відповідності функціональним вимогам. Тестування включало перевірку функціоналу додатку.

Метою тестування було забезпечити якість програмного продукту шляхом виявлення та усунення помилок (дефектів), що можуть виникнути в процесі розробки. Тестування дозволяє переконатися, що система працює згідно з технічними вимогами, очікуваннями користувача та забезпечує необхідний рівень стабільності, надійності, зручності використання.

Тестування проводиться для досягнення наступних цілей: перевірити відповідність реалізованих функцій вимогам користувача, виявити помилки, які могли бути допущені під час розробки, переконатися у стабільній роботі додатку за різних умов, оцінити зручність взаємодії користувача з інтерфейсом, забезпечити відсутність критичних збоїв при нестандартному використанні, підвищити якість продукту перед його передачею користувачеві [19].

У межах розробки мобільного застосунку, тестування мало за мету:

1. Перевірити коректність функціоналу, таймера фокусування запуск, зупинка, збереження результатів та оновлення візуального зображення (анімації рослини).
2. Переконатися у надійності збереження даних, правильна робота бази даних Room та вивід усіх сесій у вкладці результатів.
3. Перевірити поведінку системи в особливих випадках, відсутність дозволу на режим «Не турбувати» та примусове закриття застосунку.
4. Переконатися у зручності використання інтерфейсу, читабельність текстів, коректна робота кнопок.
5. Перевірити інші супутні функції, візуальний прогрес-бар і фільтрація та видалення результатів.

Таким чином, мета тестування полягала в гарантуванні надійної та передбачуваної роботи усіх компонентів застосунку на різних етапах використання, а також підвищенні загального рівня якості програмного забезпечення перед передачею користувачам.

Функціональне тестування проводилося з метою перевірити правильність реалізації основної логіки мобільного застосунку відповідно до визначених функціональних вимог. Кожна функція перевірялася вручну на емуляторі та реальному пристрої з Android 14. Опис кожного етапу тестування наведено нижче.

Android Virtual Device (AVD) – це програмне забезпечення, яке виконує функцію емулятора мобільного пристрою. Воно працює на персональному комп’ютері та імітує конфігурацію пристрою Android певного типу. Це може бути будь-який телефон, планшет, телевізор, годинник або пристрій Android Auto. Ви будете використовувати AVD для запуску вашого додатка [20].

Першим етапом тестування є перевірка запуску і зупинки таймера. Мета даної перевірки, перевірити, чи запускається та зупиняється таймер при натисканні кнопки.

Кроки перевірки:

1. Запустити додаток.
2. На головному екрані натиснути кнопку «Почати».
3. Спостерігати за оновленням текстового поля таймера та прогрес бару.
4. Почекати декілька секунд, після чого натиснути кнопку «Зупинити».
5. Переконатися, що таймер зупинився, а час більше не змінюється.

Очікуваний результат: Таймер починає рахувати з 00:00, змінюється щосекунди, після зупинки зберігає останнє значення. Прогрес бар починає рух з ліва на право змінюючи свій колір.

Відображення тестування таймера та прогрес бару відображено на рисунку 4.1.

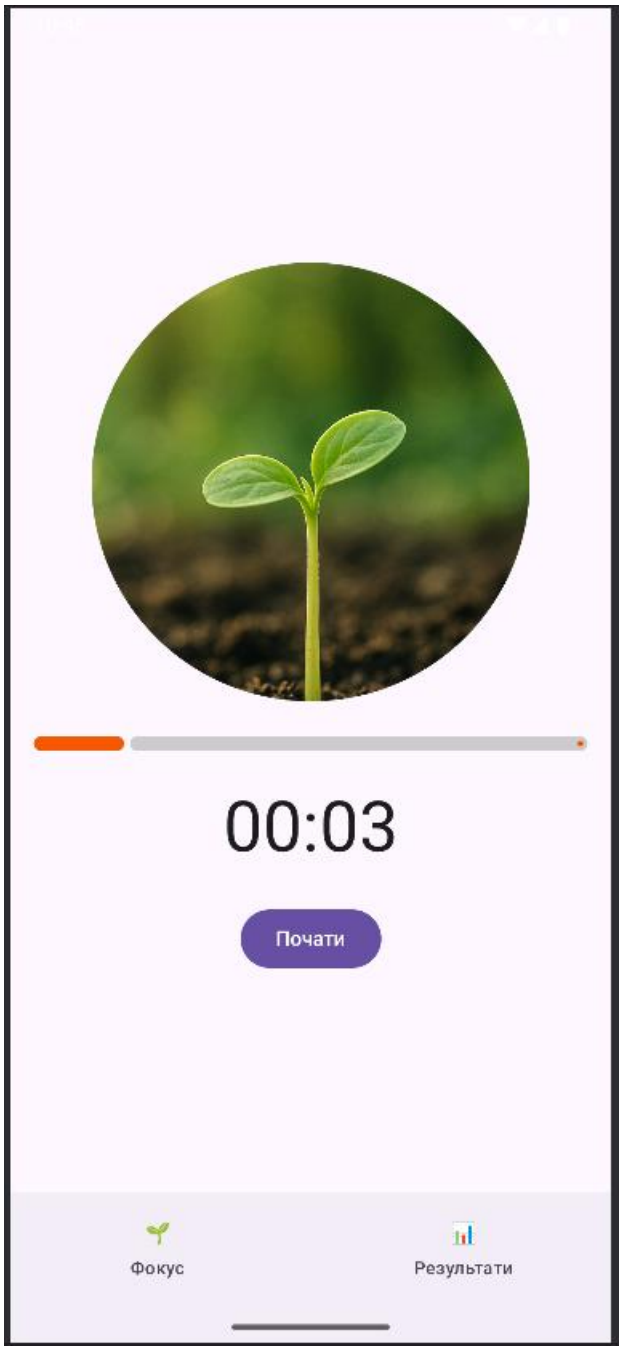


Рисунок 4.1 – Зображення тестування таймера та прогрес бару

Під час натискання кнопки «Почати» відбувається старт таймера та рух прогрес бару, після натиснення кнопки «Зупинити» відбулась зупинка таймера та прогрес бару.

Під час старту таймер та прогрес бар працювали коректно та без збоїв і по натиску кнопки зупинились.

Відображення тестування таймера та прогрес бару відображено на рисунку 4.2.

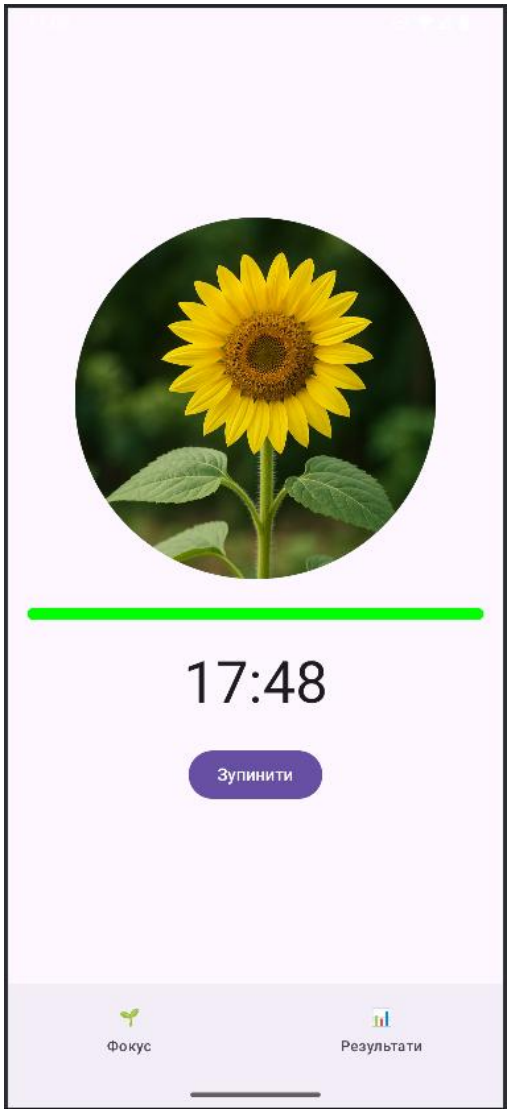


Рисунок 4.2 – Зображення тестування таймера та прогрес бару

Другим етапом тестування була перевірка збереження результатів в базу даних. Мета перевірки – перевірити, чи результат зберігається в Room після завершення таймера.

Кроки перевірки:

1. Запустити фокус-сесію на декілька секунд.

2. Натиснути «Зупинити», після чого результат автоматично має зберегтись.

3. Перейти до вкладки «Результати».

4. Перевірити наявність нового запису з відповідною тривалістю фокус-сесії.

Очікуваний результат: Результат з’являється в базі з полями: часу, дати та зображенням росту рослини (прогресу).

Вигляд бази даних до тестування представлено на рисунку 4.3.

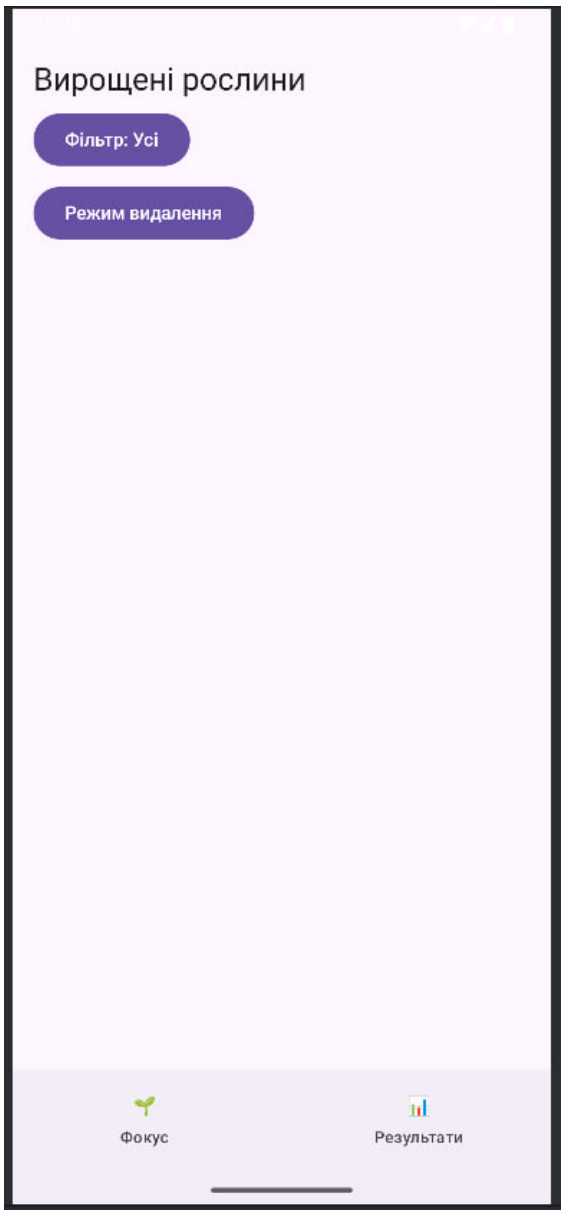


Рисунок 4.3 – Зображення екрану «Результати» до тестування

Для тесту було запущено таймер та зупинено через декілька секунд, це було потрібно для перевірки збереження даних, таких тестових запусків було декілька для перевірки збереження результатів.

Після переходу на вкладку результати, зафіксовано, що результати зберігаються та відображаються в вкладці результати (див. рисунок 4.4).

Рисунок 4.4 – Зображення екрану «Результати» після тестування

Третім етапом тестування стало тестування зміни стадії росту рослини. Для даного тестування було виставлено тестовий час для кожного етапу росту, а саме, 5 секунд, 10, 15 та 20 секунд. Метою цього тестування була перевірка, чи відображається правильне зображення росту рослини відповідно до тривалості сесії.

Кроки перевірки:

1. Запустити фокус-сесію.

2. Спостерігати за зображенням до 5 секунд — початкова стадія, до 10 секунд — друга стадія, до 15 секунд — третя стадія, понад 15 секунд — фінальна стадія.

Очікуваний результат**:** Картинка змінюється відповідно до заданих меж часу.

Для тестування було запущено фокус-сесію, під час якої велось спостереження за зміною етапу росту рослини. Час стадій для тестування був заздалегідь скорочений, щоб можна було відслідкувати зміну етапу і не очікувати робочий час який був прописаний при створенні додатку. Тестування пройшло вдало, так як етапи росту змінювались по проходженні тестового часу.

Перший етап росту рослини зображено на рисунку 4.5.

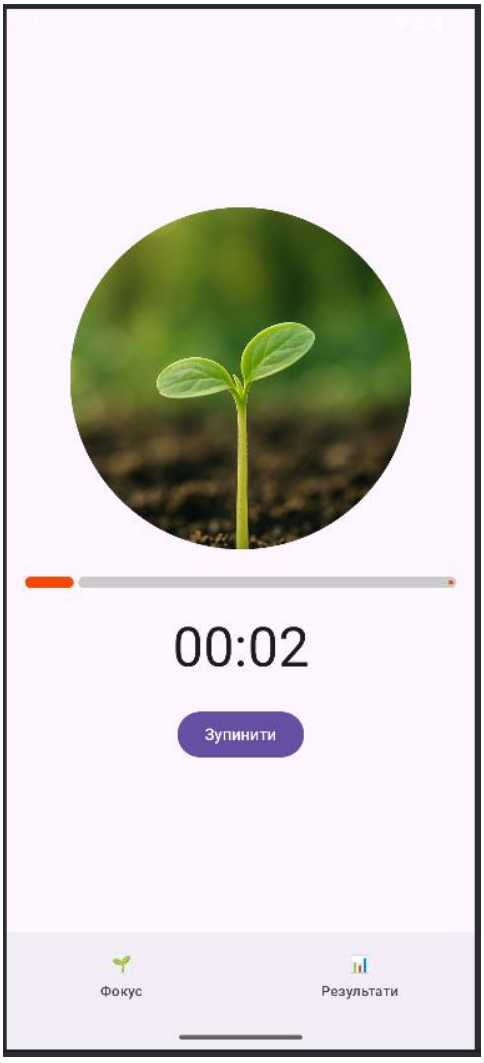


Рисунок 4.5 – Перший етап росту

Другий етап росту наступав по проходжені 5 секунд таймера, результат тесту зображено на рисунку 4.5.

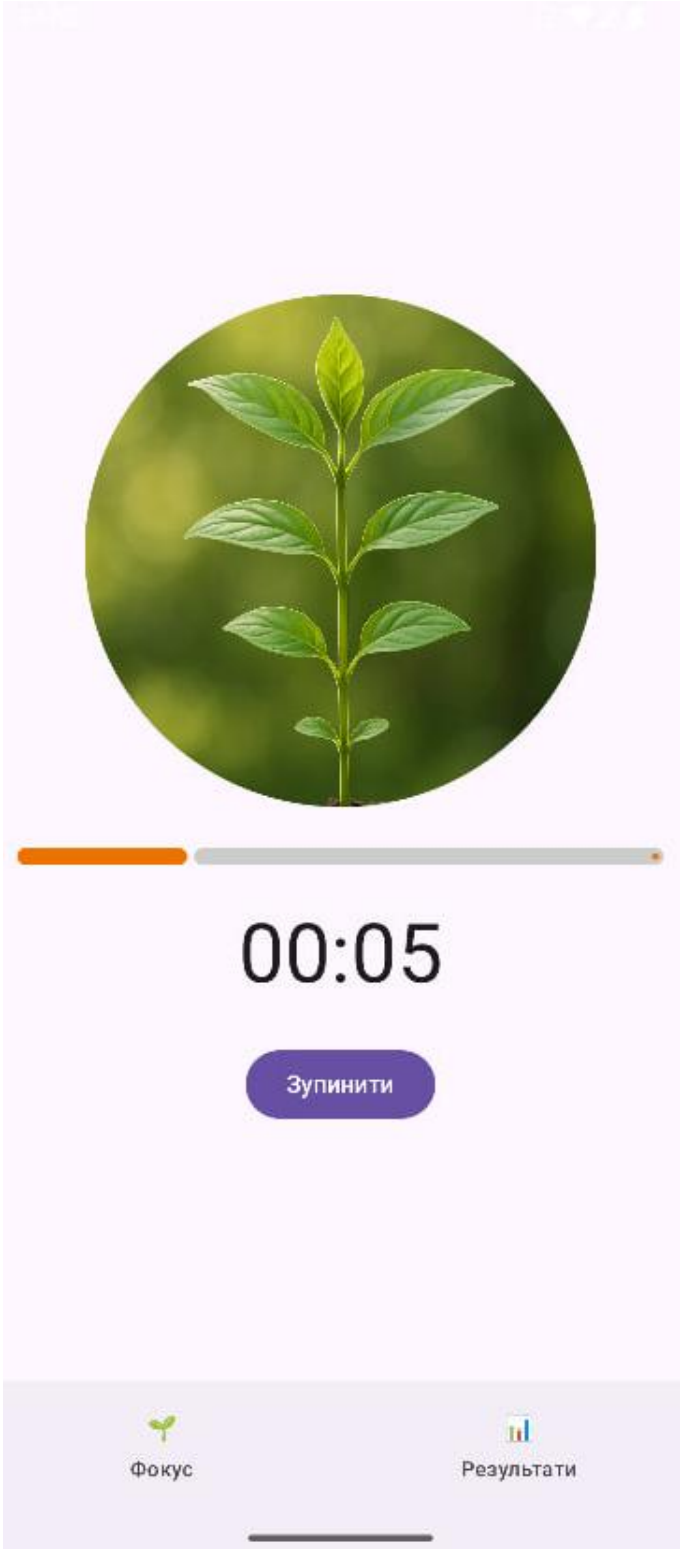


Рисунок 4.6 – Другий етап росту

Третій етап росту наступав по проходженні 10 тестових секунд, результат тестування зображено на рисунку 4.7.

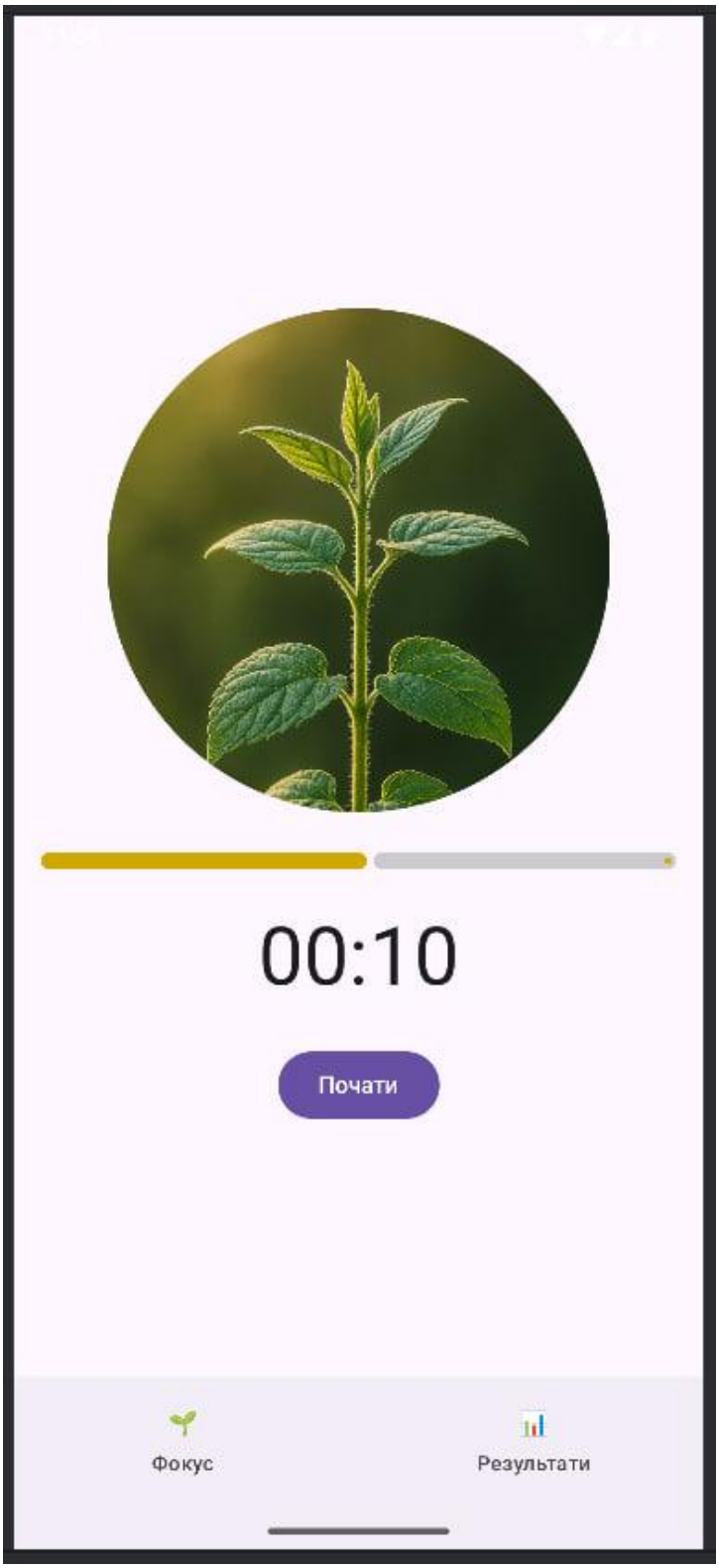


Рисунок 4.7 – Третій етап росту

Фінальний етап росту наступав по проходженні 15 секунд і діяв до 20 секунд, результат зображено на рисунку 4.8.

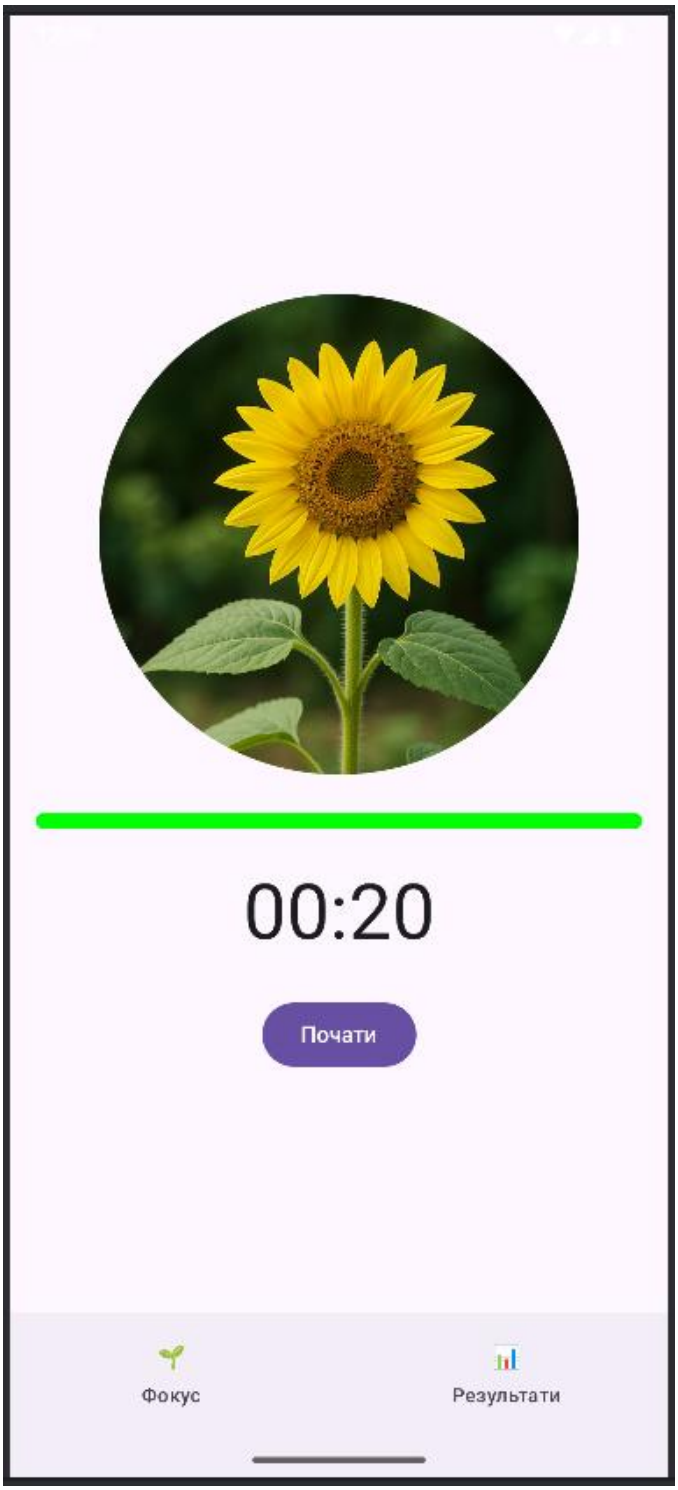


Рисунок 4.8 – Фінальний етап росту

Отже третій етап тестування додаток пройшов успішно.

Четвертий етап тестування – перевірка фільтрації результатів. Метою даного тестування є первірка, чи працює фільтрація: «Усі», «Сьогодні», «Останні 7 днів».

Кроки перевірки:

1. Перейти до вкладки «Результати».
2. Натиснути на кнопку фільтру.
3. Обрати опцію «Сьогодні» — відображаються лише сьогоднішні сесії.
4. Обрати «Останні 7 днів» — відображаються записи останнього тижня.
5. Обрати «Усі» — показуються всі записи.

Очікуваний результат: Фільтр змінює список результатів відповідно до обраного критерію.

Тестування почалось з переходу на вкладку результати та натисканням кнопки «Фільтр». З випадного меню (див. зображення 4.9), було обрано опцію одну з трьох опцій, «Сьогодні», так як по стандарту у фільтрі відображаються усі результати, потім було обрано опцію «Останні 7 днів».

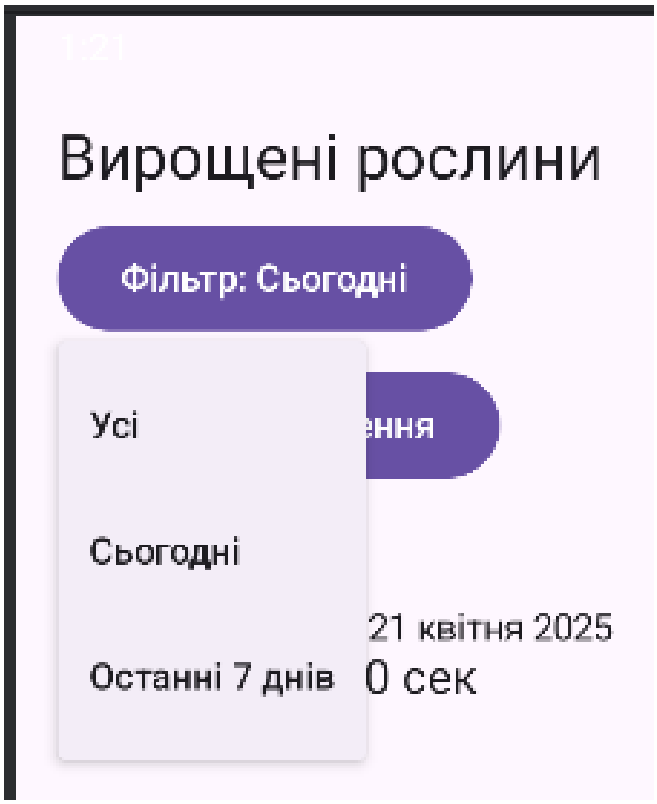


Рисунок 4.9 – Зображення випадного меню

Зображення фільтрації за опцією «Сьогодні» зображенна на рисунку 4.10.

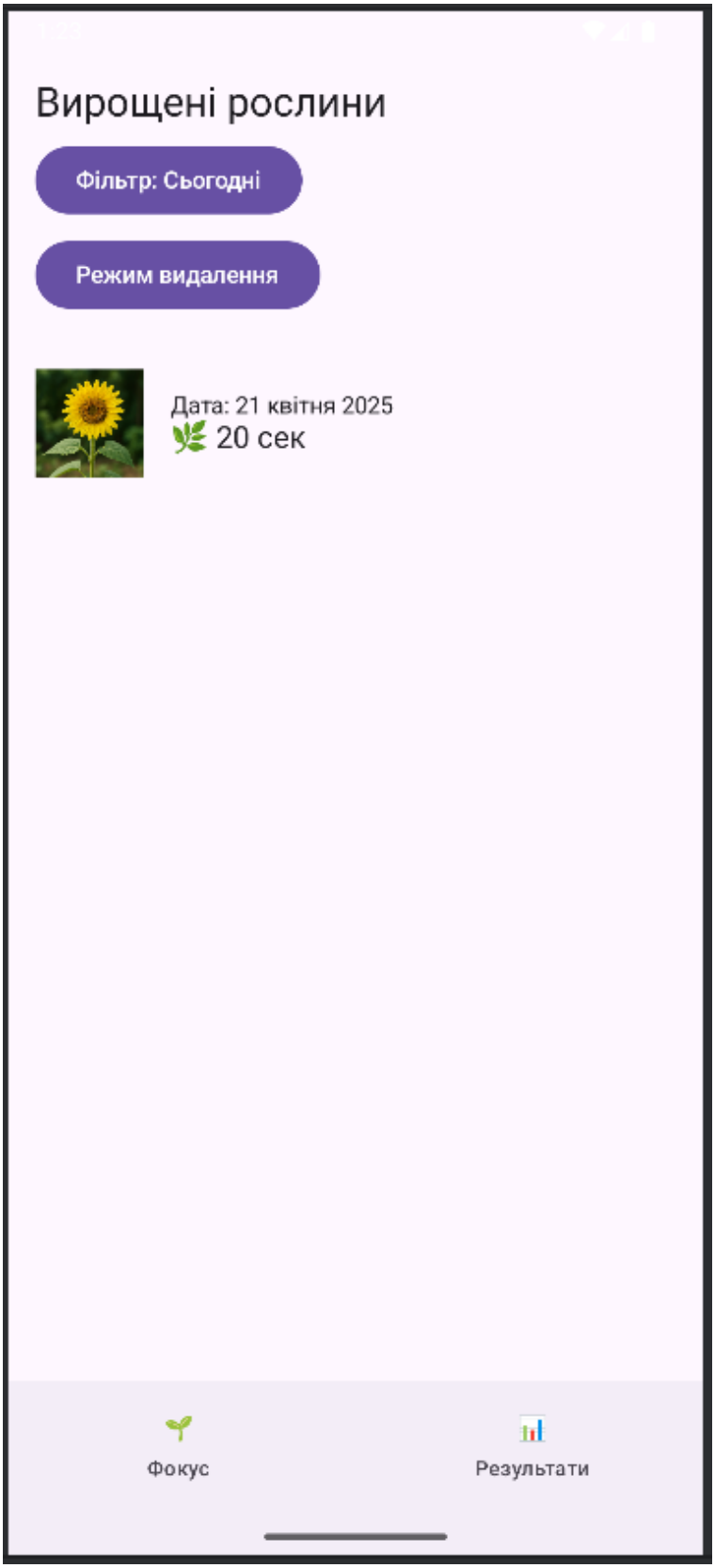


Рисунок 4.10 – Зображення роботи фільтра

Зображення фільтрації за опцією «Останні 7 днів» було зображено на рисунку 4.11.

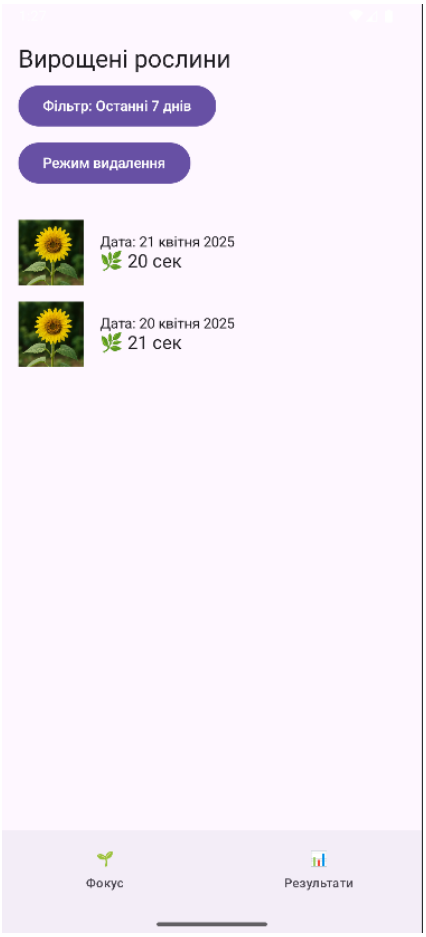


Рисунок 4.11 – Зображення роботи опції фільтра

Отже очікувані результати тестування отримано, функціонал фільтру тестування виконано успішно.

Фінальним етапом перевірки була перевірка видалення результатів. Поставлена мета тестування, це переконатися, що видалення вибраних результатів працює коректно.

Кроки перевірки:

1. Натиснути кнопку «Режим видалення».
2. Вибрати чек-бокс біля одного або кількох записів.
3. Натиснути «Видалити вибрані».
4. Переконатися, що записи зникають зі списку.

Очікуваний результат**:** Виділені записи зникають і не зберігаються в базі даних.

Тестування відбувалось в вкладці тестування та після натискання кнопки «Режим видалення» (див. рисунок. 4.12).

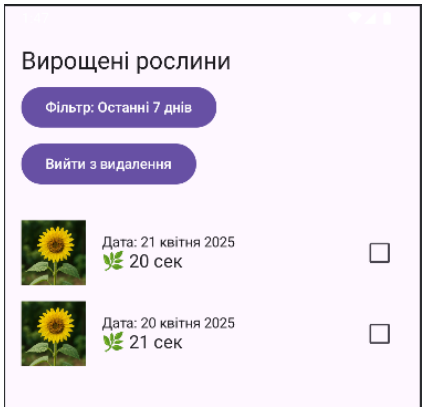


Рисунок 4.12 – Зображення тестування режиму видалення

Після натискання кнопки видалення, з правої сторони з’являлись комірки біля збережених результатів, при обранні якогось результату пуста комірка змінювалась на галочку (див. рисунок 4.13), яка підтверджувала вибір результату для видалення.

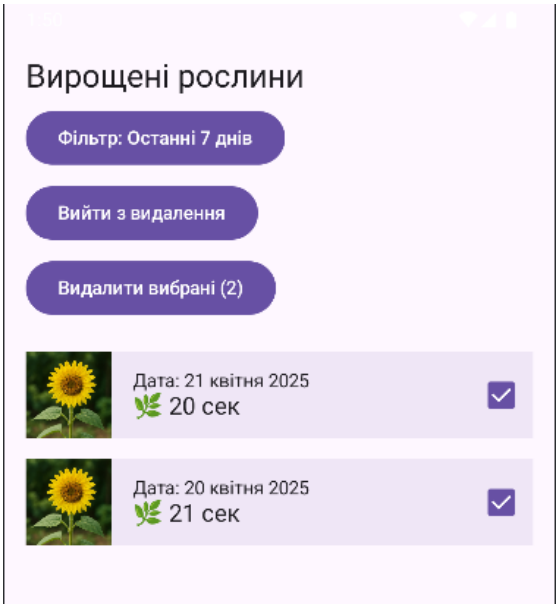


Рисунок 4.13 – Зображення тестування режиму видалення

Після натискання кнопки «Видалити вибрані» яка з’являється після натискання на результат відбувався процес видалення (див. рисунок 4.14).

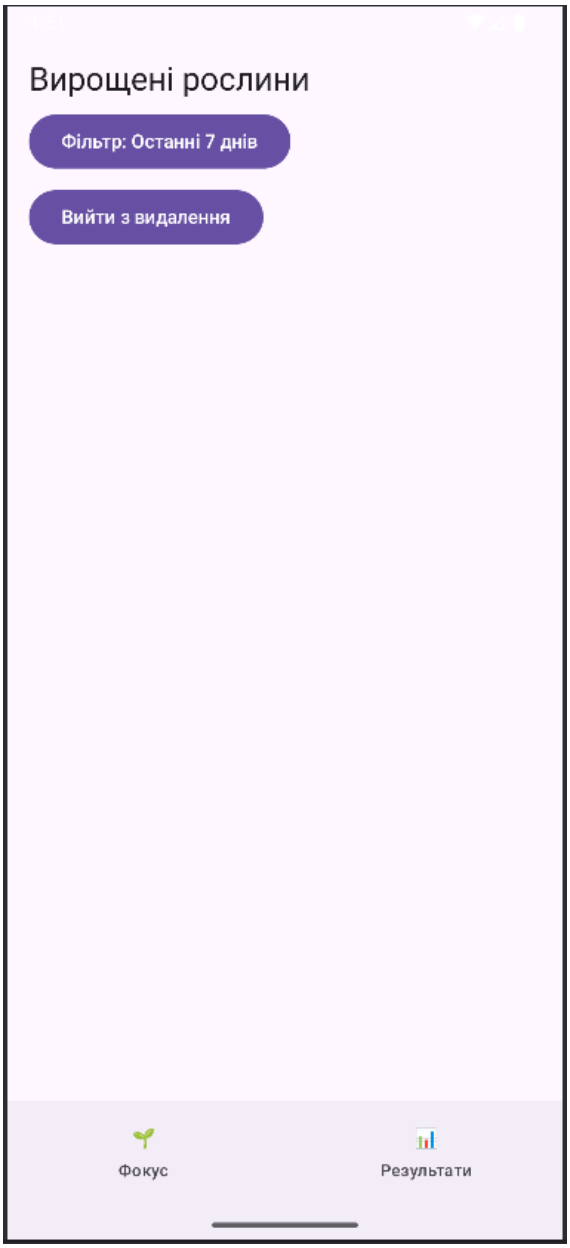


Рисунок 4.14 – Зображення тестування режиму видалення

Отже режим видалення пройшов тестування на відмінно.

Таким чином, мета тестування полягала в гарантуванні надійної та передбачуваної роботи усіх компонентів застосунку на різних етапах використання, а також підвищенні загального рівня якості програмного забезпечення перед передачею користувачам.

4.2 Розробка інструкції користувача

Мобільний застосунок розроблений для підвищення концентрації користувача під час виконання робочих або навчальних завдань. Він реалізує техніку фокус-сесій із вбудованим таймером, блокуванням сповіщень через режим «Не турбувати» та з візуальним зображенням результату у вигляді росту рослини. Завдяки цьому користувач не тільки стежить за часом, а й бачить візуальний прогрес, що підсилює мотивацію.

Після встановлення програми її можна відкрити на смартфоні з операційною системою Android. Основний інтерфейс складається з фокус-екрану з таймером, кнопки запуску, зображення рослини та індикатора прогресу. Для початку сесії користувач натискає кнопку «Почати», після чого запускається таймер. У той самий момент активується режим «Не турбувати», щоб уникнути відволікань. Якщо раніше дозвіл на цей режим не було надано, система перенаправляє користувача до відповідних налаштувань, де його можна активувати вручну.

Протягом сесії на екрані з’являється рослина, яка росте у відповідності до тривалості фокусування. Залежно від кількості часу, що минули, відображається певна стадія росту. Паралельно з цим в центрі екрану розташовується прогрес-бар, який поступово заповнюється. Його колір змінюється — на початку індикатор червоний, а в кінці, при досягненні цілі, стає зеленим. Це візуально демонструє досягнення мети. Коли користувач натискає кнопку «Зупинити», таймер зупиняється, режим «Не турбувати» вимикається, а отриманий результат автоматично зберігається в базу даних разом із часом та датою завершення сесії.

Окрема вкладка програми призначена для перегляду збережених результатів. Там відображаються всі попередні сесії фокусування у вигляді списку. Кожен результат включає дату, тривалість, а також візуальне зображення вирощеної рослини відповідно до досягнутого часу. Для зручності реалізована можливість фільтрувати результати за періодами: усі записи, лише за сьогодні або за останні сім днів. Крім того, можна активувати режим видалення результатів.

Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Мінімальні	Рекомендовані
Процесор	З тактовою частотою 1,5-2ГГц	З тактовою частотою 2-2,5 ГГц
Оперативна пам'ять	2 ГБ RAM	4 ГБ RAM
Операційна система	Android 8.0	Android 14
Дисплей	Роздільна здатність 720p, діагональ 5-6 дюймів	Роздільна здатність 1080p, діагональ 5,5-6,5 дюймів

Загалом, застосунок пропонує простий, але ефективний підхід до організації часу, підтримуючи користувача не лише функціонально, але й візуально. Застосунок може стати гарним помічником для тих, хто прагне покращити свою продуктивність, мінімізувати відволікання та зберігати історію своєї фокусної активності.

4.3 Висновки

У четвертому розділі проведено тестування програмних модулів мобільного застосунку: WorkTime Manager – менеджер робочого часу.

Перевірялась стабільність роботи застосунку, реакція на помилкові ситуації та коректність обробки різних типів вхідних даних. Особливу увагу було приділено роботі таймера, зміні стадій візуалізації, а також функціонуванню модуля блокування сповіщень. Крім того, було підготовлено інструкцію користувача для початкового ознайомлення та ефективного використання функціональних можливостей програмного продукту.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розроблено модулі мобільного застосунку: WorkTime Manager – менеджер робочого часу. Для розробки використовувалося середовище програмування Android Studio.

У процесі виконання бакалаврської кваліфікаційної роботи здійснено аналіз актуального стану проблеми. Розглянуто існуючі аналоги програмного забезпечення, визначено їхні недоліки та проведено порівняння з розроблюваним у рамках роботи продуктом. Виходячи з отриманих результатів, сформовано ключові завдання, що лягли в основу реалізації бакалаврської кваліфікаційної роботи.

Під час аналізу технологій розробки було обґрунтовано вибір мов програмування Kotlin, її технологі Jetpack Compose та Room.

У бакалаврській кваліфікаційній роботі було зроблено наступне:

- розробка механізму для аналізу витраченого часу, що дозволить користувачам переглядати статистику, базуючись на їх діяльності;
- розробка інтерфейсу користувача, що забезпечить зручну взаємодію з застосунком та легкий доступ до необхідної інформації;
- розробка модуля відліку часу для відслідковування часу, витраченого на різні завдання та додаткові функції;
- розробка модуля для блокування сповіщень, що забезпечуватиме зосередженість користувачів під час виконання завдань;
- розробка модуля для збереження результатів відліку часу попередніх робочих сесій.

Розроблено схеми навігації програмного застосунку. Також, розроблено модель системи з використанням UML діаграм.

Тестування застосунку підтвердило його повну працездатність та відповідність вимогам технічного завдання. Також, розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сливка Р.М. Порівняльний аналіз мобільних застосунків для фокус-сесій: виявлення функціональних прогалин / О.В. Мельник, Р.М. Сливка // Матеріали конференції «2-га Міжнародна науково-практична конференція «Сучасні наукові дослідження: Теоретичні та практичні аспекти (2025)», Рига, Латвія, 2025. [Електронний ресурс] – Режим доступу до ресурсу: https://surl.lu/adubdl .

2. Сливка Р.М. Інтеграція Jetpack Compose, Room та Kotlin Coroutines у створенні мобільного застосунку / О.В. Мельник, Р.М. Сливка // Матеріали конференції «2-га Міжнародна науково-практична конференція «Досягнення науки та прикладних досліджень (2025)», Дублін, Ірландія, 2025. [Електронний ресурс] – Режим доступу до ресурсу: https://surl.lu/cqyjwp .

3. Мобільні застосунки [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/hrdiwa

4. Toggl Track [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/iwtqfg.

5. Clockify [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/fvybeh.

6. RescueTime [Електронний ресурс] – Режим доступу до ресурсу: https://surli.cc/rwbyeg.

7. UML діаграми [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/fkkwzh.

8. Діаграма діяльності [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/gxvmub.

9. Microsoft Visio [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/xhdwjt.

10. Діаграма навігації [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/psytst.

11. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/vqhncn.

12. Ktor [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/vtuced.

13. Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/mkiqtr.

14. XML [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/elneqb.

15. Room [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/jpttmk.

16. SQLite [Електронний ресурс] – Режим доступу до ресурсу: https://surl.lu/fgjckm.

17. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/epmfbx.

18. LaunchedEffect [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/adbkcg.

19. Тестування [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/okgfuz.

20. AVD [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/gxobqo.

ДОДАТОК А

(обов'язковий)

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 року

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка мобільного застосунку: WorkTime Manager – менеджер робочого часу»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Мельник О.В.
«24» 03 2025 року.

Виконав:
студент гр. 5ПІ-21б Сливка Р.М.
«24» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка мобільного застосунку: WorkTime Manager – менеджер робочого часу».

Галузь застосування – мобільні технології.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 18 від 18 лютого 2025 р.

3. Мета та призначення розробки.

Метою роботи є покращення управління робочим часом користувачів через розробку мобільного застосунку, який сприятиме ефективному використанню часу, зменшуючи відволікання та підвищуючи концентрацію. Застосунок забезпечить блокування сповіщень під час активних сесій, дозволяючи користувачам зосередитися на роботі.

Основними задачами дослідження є:

- розробка механізму для аналізу витраченого часу, що дозволить користувачам переглядати статистику, базуючись на їх діяльності;
- розробка інтерфейсу користувача, що забезпечить зручну взаємодію з застосунком та легкий доступ до необхідної інформації;
- розробка модуля відліку часу для відслідковування часу, витраченого на різні завдання та додаткові функції;
- розробка модуля для блокування сповіщень, що забезпечуватиме зосередженість користувачів під час виконання завдань;
- розробка модуля для збереження результатів відліку часу попередніх робочих сесій.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/vqhncn.
2. Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/mkiqtr.
3. Room [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/jpttmk.
4. SQLite [Електронний ресурс] – Режим доступу до ресурсу: https://surl.lu/fgjckm.
5. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/epmfbx.
6. LaunchedEffect [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/adbkcg.

5. Технічні вимоги

- середовище розробки – Android Studio;
- мова програмування – Kotlin;
- вхідні дані: налаштування таймера, інформація про пройдені сесії фокусу;
- вихідні дані: застосунок із графічним інтерфейсом користувача для менеджменту робочого часу.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та постановка задачі	25.03.25 – 05.04.25
2	Розробка моделей, структури та алгоритмів	06.04.25 – 18.04.25
3	Розробка програмного забезпечення	22.04.25 – 17.05.25
4	Тестування застосунку	18.05.25 – 25.05.25
5	Оформлення матеріалів до захисту БДР	26.05.25 – 30.05.25

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка мобільного застосунку: WorkTime Manager – менеджер робочого часу

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота

Підрозділ кафедра ПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 7,3 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія: _____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Мельник О. В., к. т. н., доцент кафедри ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Сливка Р.М.
(підпис) (прізвище, ініціали)

ДОДАТОК В

(довідниковий)

Лістинг програмного коду мобільного застосунку

Модуль MainActivity.kt

```
package com.example.worktimefocus

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.navigation.NavHostController
import androidx.navigation.compose.*
import kotlinx.coroutines.delay

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            WorkTimeFocusApp()
        }
    }
```

```
}

@Composable
fun WorkTimeFocusApp() {
    val navController = rememberNavController()

    Scaffold(
        bottomBar = {
            BottomNavigationBar(navController)
        }
    ) { innerPadding ->
        NavHost(
            navController = navController,
            startDestination = "focus",
            modifier = Modifier.padding(innerPadding)
        ) {
            composable("focus") { FocusScreen() }
            composable("results") { ResultsScreen() }
        }
    }
}

@Composable
fun BottomNavigationBar(navController: NavHostController) {
    NavigationBar {
        NavigationBarItem(
            selected = navController.currentDestination?.route == "focus",
            onClick = { navController.navigate("focus") },
            label = { Text("Фокус") },
```

```
            icon = { Text("🍸") }
        )
        NavigationBarItem(
            selected = navController.currentDestination?.route == "results",
            onClick = { navController.navigate("results") },
            label = { Text("Резултати") },
            icon = { Text("📊") }
        )
    }
}
```

Модуль FocusScreen.kt

```
package com.example.worktimefocus

import android.annotation.SuppressLint
import android.content.Context
import android.os.Build
import android.provider.Settings
import android.app.NotificationManager
import android.content.Intent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
```

```
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.example.worktimefocus.data.DatabaseProvider
import com.example.worktimefocus.data.PlantResult
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.ui.graphics.lerp

@SuppressLint("UnrememberedMutableState")
@Composable
fun FocusScreen() {
    var isTimerRunning by remember { mutableStateOf(false) }
    var timeInSeconds by remember { mutableStateOf(0) }
    val context = LocalContext.current
    val scope = rememberCoroutineScope()

    val sessionDuration = 20f // можна змінити на інше значення (в секундах)

    // Прогрес: нормалізований і анімований
    val rawProgress by derivedStateOf {
        timeInSeconds.coerceAtMost(sessionDuration.toInt()).toFloat() / sessionDuration
    }
    val animatedProgress by animateFloatAsState(targetValue = rawProgress, label = "Progress")
```

```
        // Кольорова індикація: від червоного до зеленого
        val progressColor = lerp(Color.Red, Color.Green, animatedProgress)

        // Таймер
        LaunchedEffect(isTimerRunning) {
            while (isTimerRunning) {
                delay(1000)
                timeInSeconds++
            }
        }

        // Запит дозволу на "не турбувати"
        fun requestDoNotDisturbPermission() {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
                if (!notificationManager.isNotificationPolicyAccessGranted) {
                    val intent = Intent(Settings.ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS)
                    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
                    context.startActivity(intent)
                }
            }
        }

        // Увімкнення режиму "не турбувати"
        fun enableDoNotDisturb(context: Context) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
```

```
            val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            if (notificationManager.isNotificationPolicyAccessGranted) {

notificationManager.setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_NONE)
            }
        }
    }

    // Вимкнення режиму "не турбувати"
    fun disableDoNotDisturb(context: Context) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            if (notificationManager.isNotificationPolicyAccessGranted) {

notificationManager.setInterruptionFilter(NotificationManager.INTERRUPTION_FILTER_ALL)
            }
        }
    }

    fun getGrowthStageRes(seconds: Int): Int {
        return when {
            seconds < 5 -> R.drawable.stage1
            seconds < 10 -> R.drawable.stage2
            seconds < 15 -> R.drawable.stage3
```

```
            else -> R.drawable.stage4
        }
    }

    fun formatTime(totalSeconds: Int): String {
        val minutes = totalSeconds / 60
        val seconds = totalSeconds % 60
        return "%02d:%02d".format(minutes, seconds)
    }

    fun savePlantResult(durationInSeconds: Int, context: Context, scope: CoroutineScope) {
        scope.launch {
            val db = DatabaseProvider.getDatabase(context)
            val newResult = PlantResult(durationInSeconds = durationInSeconds)
            db.plantResultDao().insertPlantResult(newResult)
        }
    }

    val growthStageRes = getGrowthStageRes(timeInSeconds)

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Box(
            contentAlignment = Alignment.Center,
```

```
    modifier = Modifier
        .size(300.dp)
        .clip(CircleShape)
        .background(MaterialTheme.colorScheme.surfaceVariant)
) {
    Image(
        painter = painterResource(id = growthStageRes),
        contentDescription = null,
        modifier = Modifier
            .size(300.dp)
            .clip(CircleShape),
        contentScale = ContentScale.Crop
    )
}

Spacer(modifier = Modifier.height(24.dp))

// Прогрес-бар
LinearProgressIndicator(
    progress = animatedProgress,
    modifier = Modifier
        .fillMaxWidth()
        .height(10.dp),
    color = progressColor,
    trackColor = Color.LightGray
)

Spacer(modifier = Modifier.height(24.dp))

Text(
```

```
            text = formatTime(timeInSeconds),
            fontSize = 48.sp
        )

        Spacer(modifier = Modifier.height(24.dp))

        Button(onClick = {
            if (isTimerRunning) {
                savePlantResult(timeInSeconds, context, scope)
                disableDoNotDisturb(context)
            } else {
                requestDoNotDisturbPermission()
                enableDoNotDisturb(context)
            }
            isTimerRunning = !isTimerRunning
        }) {
            Text(if (isTimerRunning) "Зупинити" else "Почати")
        }
    }
}
```

Модуль ResultsScreen.kt

```
package com.example.worktimefocus

import com.example.worktimefocus.data.DatabaseProvider
import com.example.worktimefocus.data.PlantResult

import androidx.compose.runtime.*
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
```

```
import androidx.compose.ui.unit.sp
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.material3.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.Alignment
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.*

@Composable
fun ResultsScreen() {
    val context = LocalContext.current
    val db = remember { DatabaseProvider.getDatabase(context) }
    var plantResults by remember { mutableStateOf<List<PlantResult>>(emptyList()) }

    var isDeleteMode by remember { mutableStateOf(false) }
    var selectedItems by remember { mutableStateOf(setOf<Int>()) }

    var expanded by remember { mutableStateOf(false) }
    var selectedFilter by remember { mutableStateOf("Усі") }

    val coroutineScope = rememberCoroutineScope()
    val filterOptions = listOf("Усі", "Сьогодні", "Останні 7 днів")

    suspend fun loadResults() {
```

```
    val allResults = db.plantResultDao().getAllResults()
    val filtered = when (selectedFilter) {
        "Сьогодні" -> {
            val today = Calendar.getInstance().apply {
                set(Calendar.HOUR_OF_DAY, 0)
                set(Calendar.MINUTE, 0)
                set(Calendar.SECOND, 0)
                set(Calendar.MILLISECOND, 0)
            }.timeInMillis
            allResults.filter { it.timestamp >= today }
        }
        "Останні 7 днів" -> {
            val weekAgo = System.currentTimeMillis() - 7 * 24 * 60 * 60 * 1000L
            allResults.filter { it.timestamp >= weekAgo }
        }
        else -> allResults
    }
    plantResults = filtered
}

LaunchedEffect(selectedFilter) {
    loadResults()
}

fun formatTime(seconds: Int): String {
    val hours = seconds / 3600
    val minutes = (seconds % 3600) / 60
    val secs = seconds % 60
    return when {
        hours > 0 -> "$hours год ${minutes} хв"
```

```
        minutes > 0 -> "$minutes хв"
        else -> "$secs сек"
    }
}

fun formatDate(timestamp: Long): String {
    val date = Date(timestamp)
    val format = SimpleDateFormat("dd MMMM yyyy", Locale("uk"))
    return format.format(date)
}

fun getGrowthStageRes(seconds: Int): Int {
    return when {
        seconds < 5 -> R.drawable.stage1
        seconds < 10 -> R.drawable.stage2
        seconds < 15 -> R.drawable.stage3
        else -> R.drawable.stage4
    }
}

Column(modifier = Modifier
    .fillMaxSize()
    .padding(16.dp)) {

    Text("Вирощені рослини", fontSize = 24.sp)
    Spacer(modifier = Modifier.height(8.dp))

    // Dropdown фільтр
    Box {
        Button(onClick = { expanded = true }) {
```

```
            Text("Фільтр: $selectedFilter")
        }
        DropdownMenu(expanded = expanded, onDismissRequest = { expanded
= false }) {
            filterOptions.forEach { option ->
                DropdownMenuItem(
                    text = { Text(option) },
                    onClick = {
                        selectedFilter = option
                        expanded = false
                    }
                )
            }
        }
    }

    Spacer(modifier = Modifier.height(8.dp))

    // Кнопка перемикання режиму
    Button(onClick = {
        isDeleteMode = !isDeleteMode
        selectedItems = emptySet()
    }) {
        Text(if (isDeleteMode) "Вийти з видалення" else "Режим видалення")
    }

    Spacer(modifier = Modifier.height(8.dp))

    // Кнопка для видалення вибраних
    if (isDeleteMode && selectedItems.isNotEmpty()) {
```

```
            Button(onClick = {
                coroutineScope.launch {
                    selectedItems.forEach { id ->
                        db.plantResultDao().deleteById(id)
                    }
                    loadResults()
                    selectedItems = emptySet()
                }
            }) {
                Text("Видалити вибрані (${selectedItems.size})")
            }
        }

        Spacer(modifier = Modifier.height(16.dp))

        plantResults.forEach { result ->
            val timeFormatted = formatTime(result.durationInSeconds)
            val dateFormatted = formatDate(result.timestamp)
            val imageRes = getGrowthStageRes(result.durationInSeconds)
            val isSelected = result.id in selectedItems

            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(vertical = 8.dp)
                    .background(if (isSelected)
MaterialTheme.colorScheme.primary.copy(alpha = 0.1f) else
MaterialTheme.colorScheme.background),
                verticalAlignment = Alignment.CenterVertically
            ) {
```

```
                Image(
                    painter = painterResource(id = imageRes),
                    contentDescription = null,
                    modifier = Modifier.size(64.dp),
                    contentScale = ContentScale.Crop
                )
                Spacer(modifier = Modifier.width(16.dp))
                Column(modifier = Modifier.weight(1f)) {
                    Text(text = "Дата: $dateFormatted", fontSize = 14.sp)
                    Text(text = " 🌿 $timeFormatted", fontSize = 18.sp)
                }

                if (isDeleteMode) {
                    Checkbox(
                        checked = isSelected,
                        onCheckedChange = { checked ->
                            selectedItems = if (checked) {
                                selectedItems + result.id
                            } else {
                                selectedItems - result.id
                            }
                        }
                    )
                }
            }
        }
    }
}
```

Модуль NotificationHelper.kt

```
package com.example.worktimefocus

import android.app.NotificationManager
import android.content.Context
import android.content.Intent
import android.provider.Settings
import android.widget.Toast

object NotificationHelper {
    fun requestDoNotDisturbPermission(context: Context) {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        if (!notificationManager.isNotificationPolicyAccessGranted) {
            val intent = Intent(Settings.ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS)
            intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
            context.startActivity(intent)

            Toast.makeText(context, "Надай доступ до \"Не турбувати\"", Toast.LENGTH_LONG).show()
        }
    }

    fun enableDoNotDisturb(context: Context) {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
```

```
            if (notificationManager.isNotificationPolicyAccessGranted) {

notificationManager.setInterruptionFilter(NotificationManager.INTERRUPTION_FI
LTER_NONE)
            }
        }

        fun disableDoNotDisturb(context: Context) {
            val notificationManager =
context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

            if (notificationManager.isNotificationPolicyAccessGranted) {

notificationManager.setInterruptionFilter(NotificationManager.INTERRUPTION_FI
LTER_ALL)
            }
        }
    }
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ: WORKTIME MANAGER – МЕНЕДЖЕР РОБОЧОГО ЧАСУ

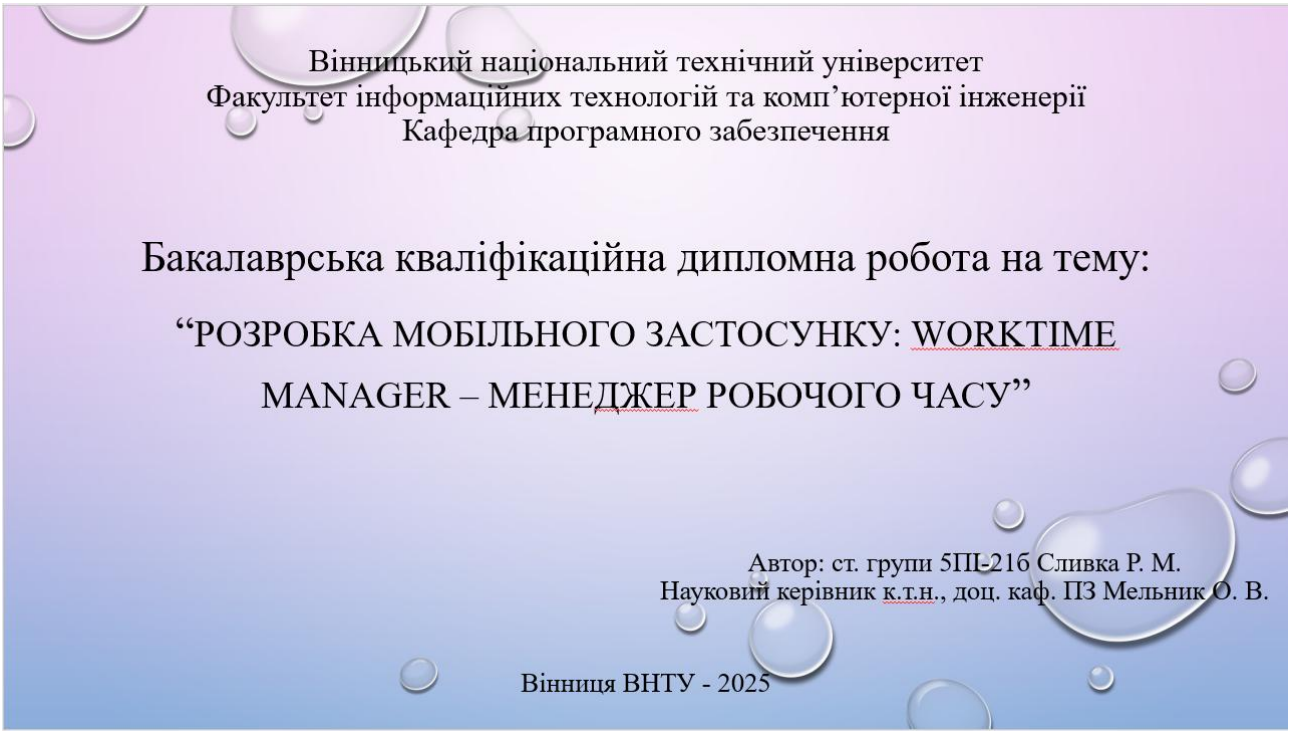


Рисунок Г.1 – Титульний слайд

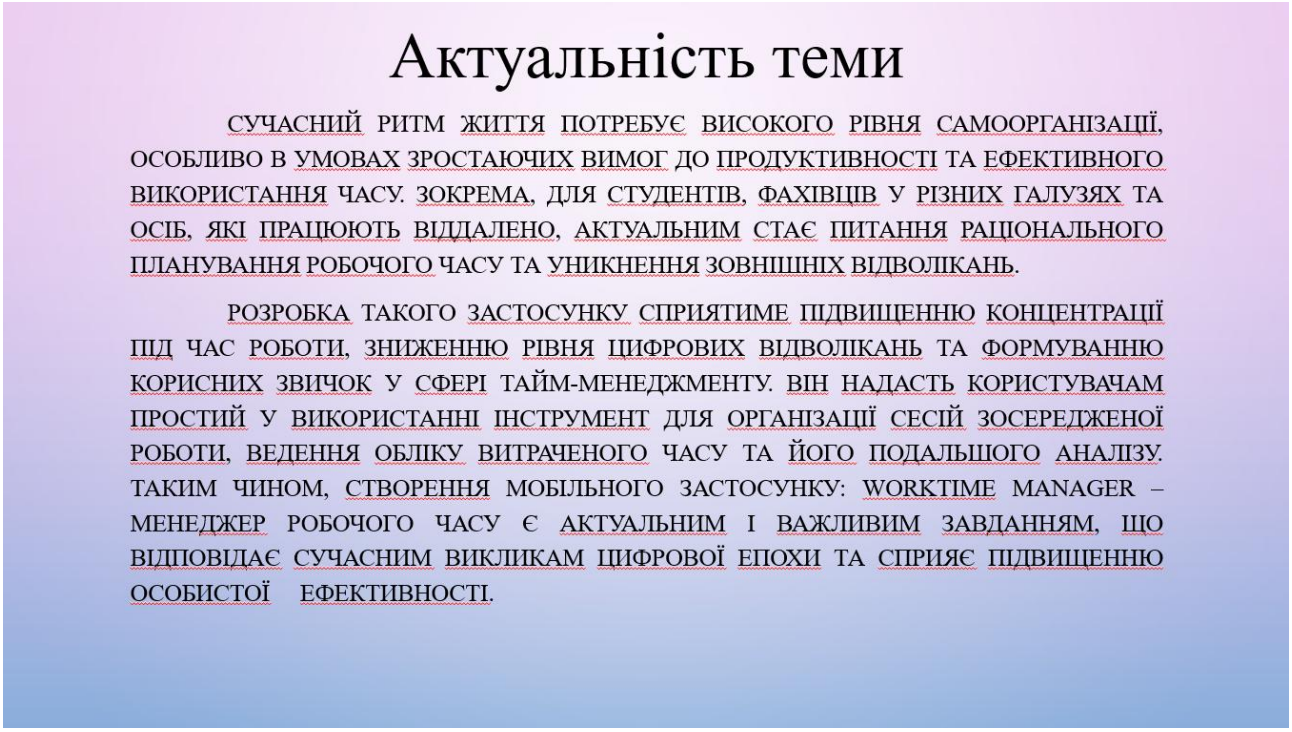


Рисунок Г.2 – Актуальність теми

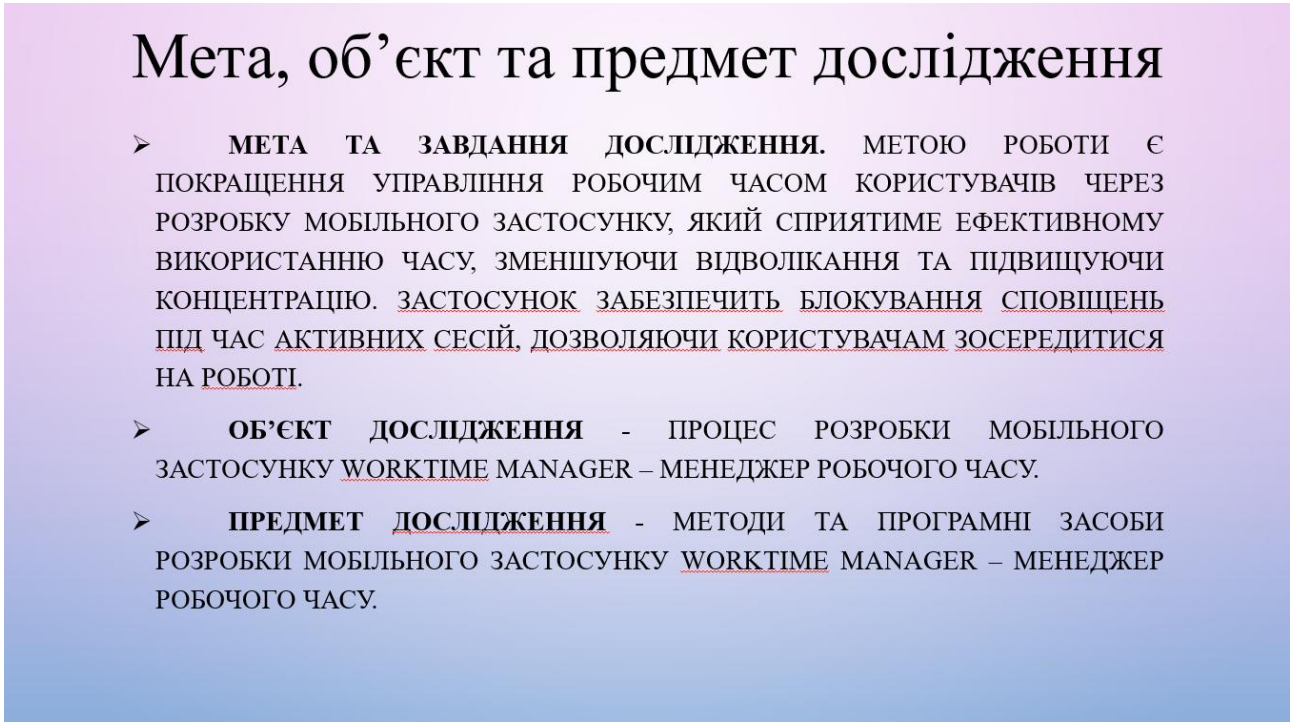


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

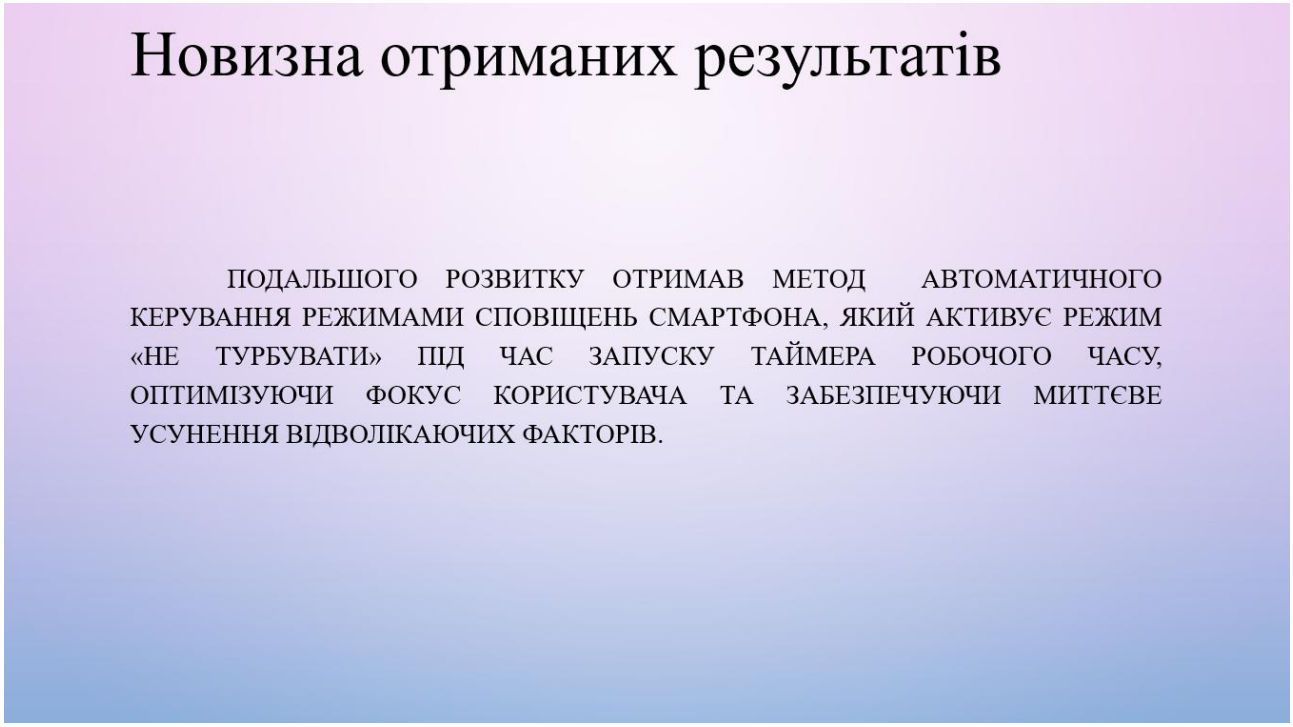


Рисунок Г.4 – Новизна отриманих результатів

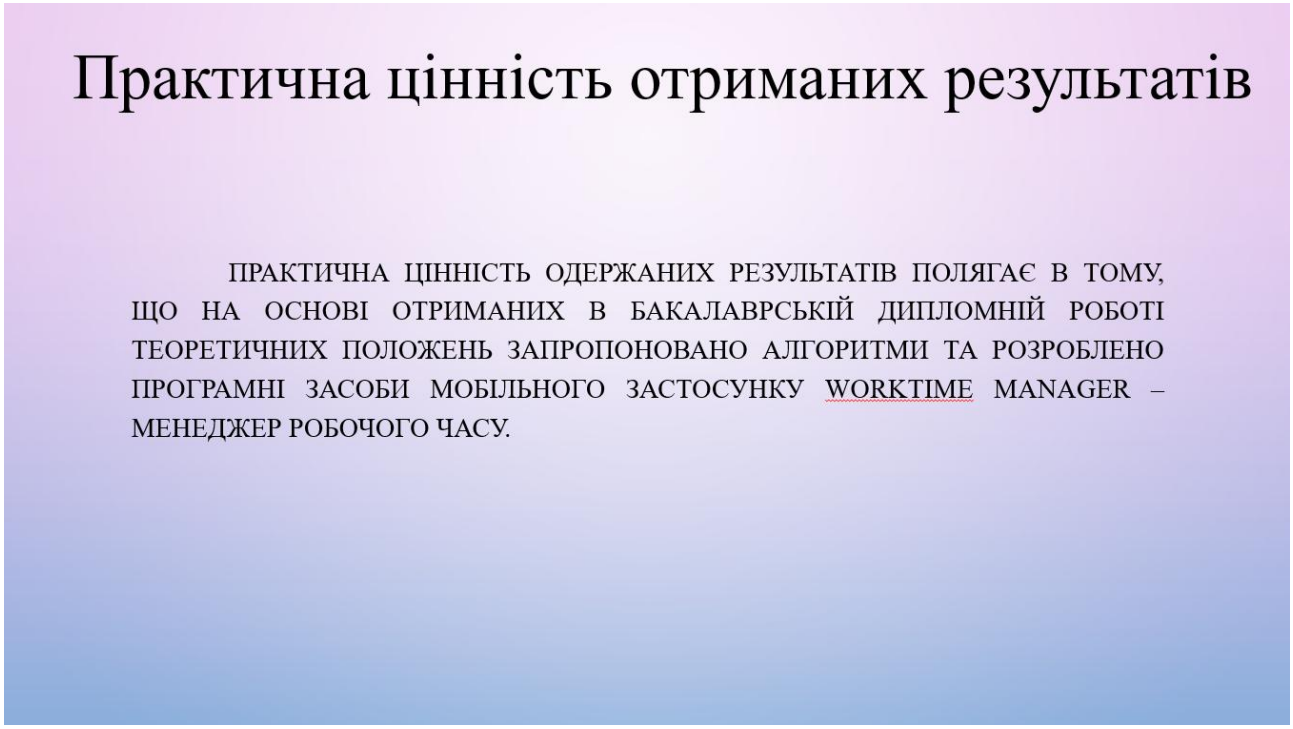


Рисунок Г.5 – Практична цінність отриманих результатів

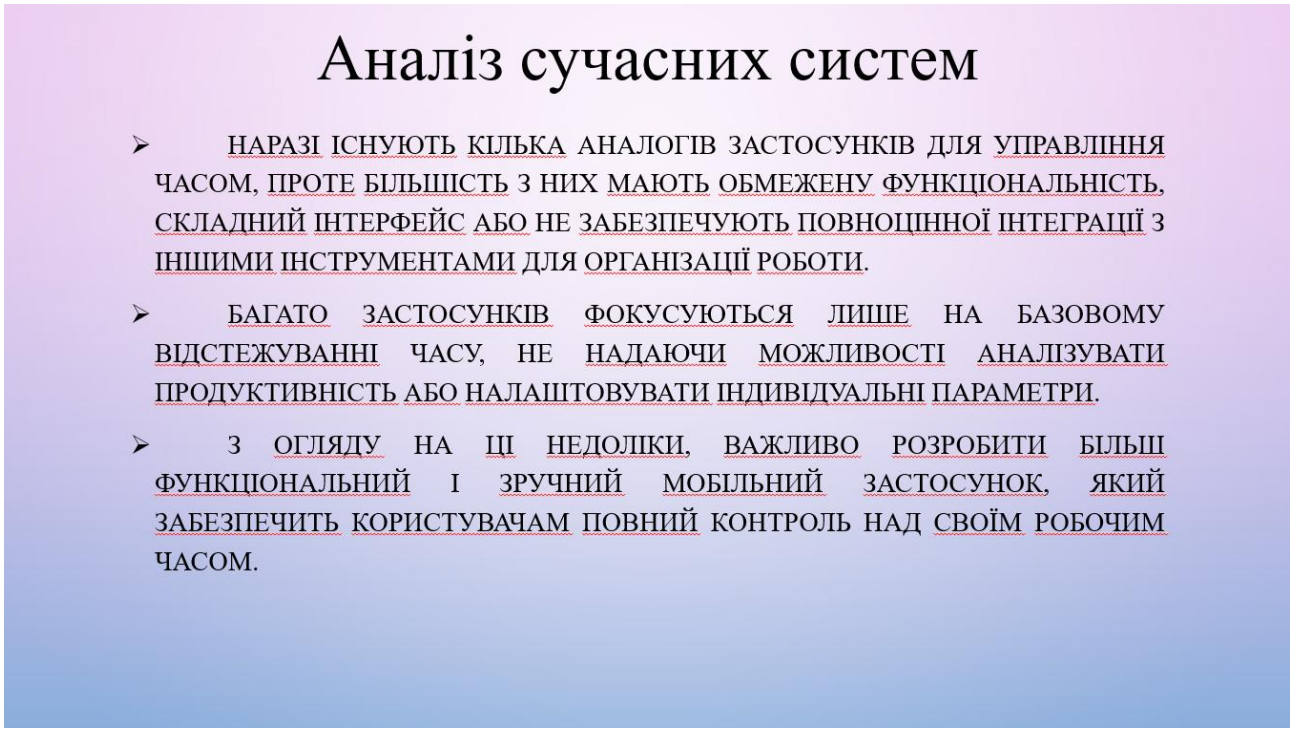


Рисунок Г.6 – Аналіз сучасних систем

Структура мобільної системи

ДЛЯ РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ ПОВИННА БУТИ РОЗРОБЛЕНА ЙОГО СТРУКТУРА, ПОДІЛЕНА НА ЗАГАЛЬНІ МОДУЛІ, ЯКІ ВИЗНАЧАТИМУТЬ ПОСЛІДОВНІСТЬ ДІЙ КОРИСТУВАЧА В ЗАСТОСУНКУ. ДЛЯ КРАЩОГО РОЗУМІННЯ СТРУКТУРИ МОБІЛЬНОЇ СИСТЕМИ БУЛО РОЗРОБЛЕНО ДІАГРАМУ ДІЯЛЬНОСТІ.

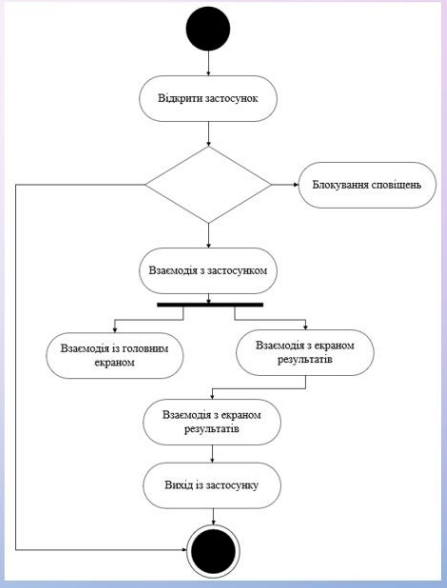


Рисунок Г.7 – Структура мобільної системи

Розробка графічного інтерфейсу

ПРИ РОЗРОБЦІ ІНТЕРФЕЙСУ ПРОГРАМНОГО ЗАСТОСУНКУ БУЛО ПРИДІЛЕНО ОСОБЛИВУ УВАГУ ЙОГО ЗРОЗУМІЛОСТІ ТА ЗРУЧНОСТІ ДЛЯ КОРИСТУВАЧА.

ОСНОВНІ ЕКРАНИ ЗАСТОСУНКУ

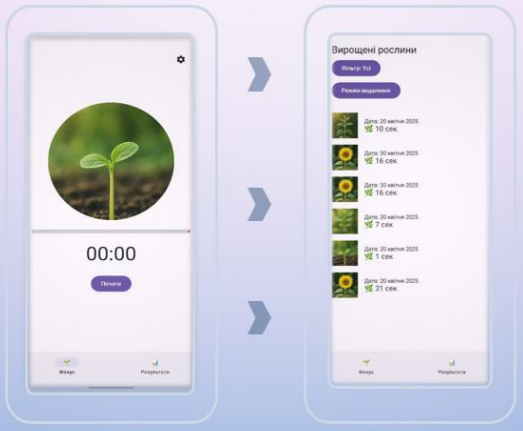


Рисунок Г.8 – Розробка графічного інтерфейсу

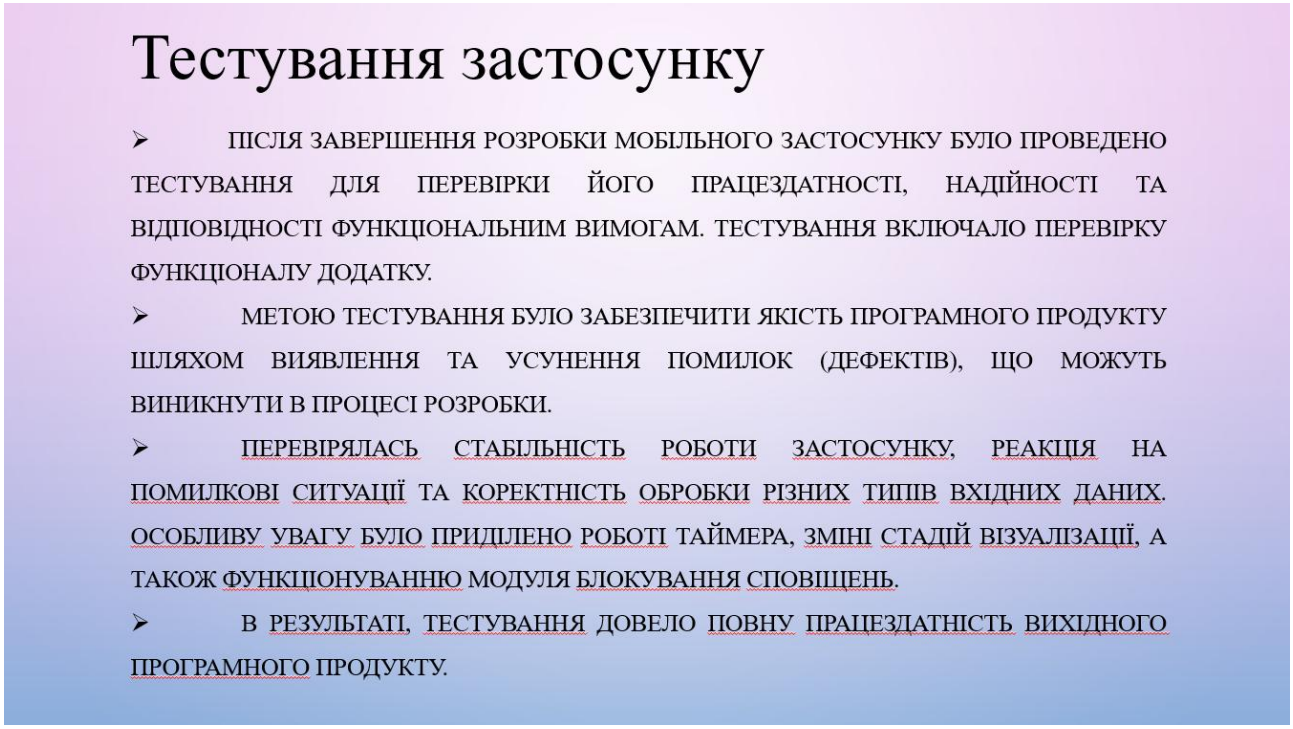


Рисунок Г.9 – Тестування застосунку

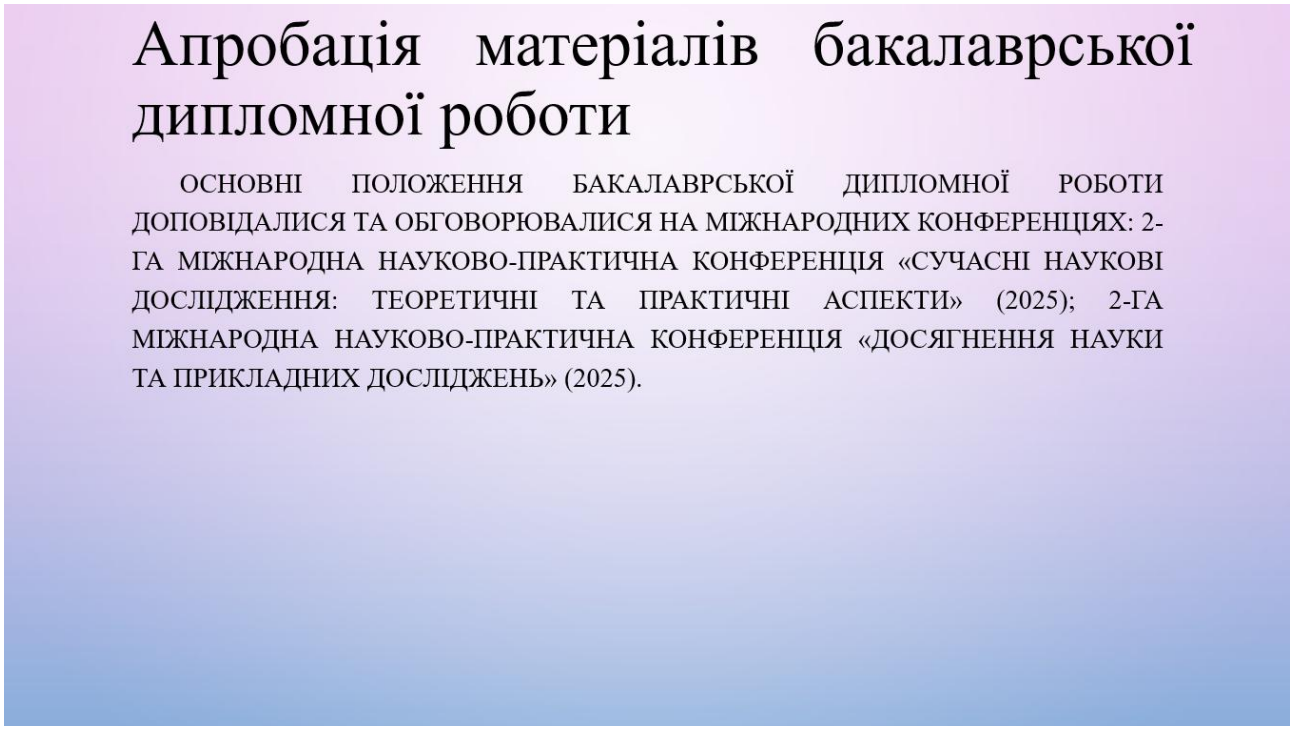


Рисунок Г.10 – Апробація матеріалів бакалаврської кваліфікаційної роботи

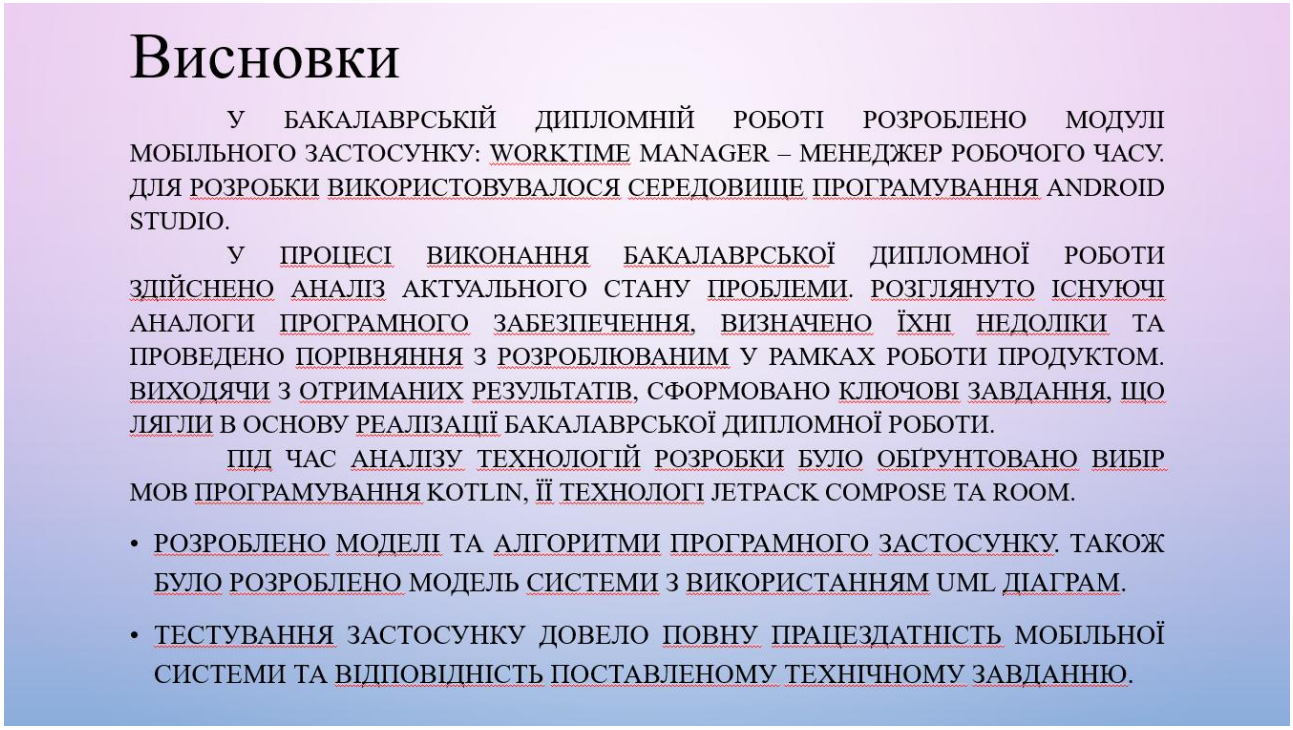


Рисунок Г.11 – Висновки