Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система моніторингу та аналізу результатів навчання студентів»

Виконала студентка 4-го курсу групи 1СП-21б
спеціальності 123 «Комп'ютерна інженерія»
освітня програма «Системне програмування»
Педосенко М. Ю.

Керівник: к.т.н., доц., доцент каф. ОТ
Снігур А. В.

Рецензент: д.т.н., професор, голова секції УБ
каф. МБІС
Яремчук Ю. Є.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
« » _____ 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
« 21 » березня 2025 року

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

08-54.БКР.011.00.000 ПЗ

Педосенко Марії Юріївні

1 Тема роботи: «Система моніторингу та аналізу результатів навчання студентів», керівник роботи Снігур А. В. к.т.н., доц., доцент кафедри ОТ затверджені наказом вищого навчального закладу від «20» березня 2025 року № 97.

2 Термін подання студентом роботи 13.05.2025 р.

3 Вхідні дані до роботи: функціональні вимоги до застосунку – створення та планування завдань, облік часу навчання за допомогою таймера, візуалізація результатів на діаграмі, календар подій, система облікових записів; середовище розробки – Android Studio; база даних – Room Base.

4 Зміст розрахунково-пояснювальної записки: вступ, огляд існуючих систем моніторингу та аналізу навчальної діяльності студентів, особливості розробки системи моніторингу та аналізу навчання, програмна реалізація системи, висновки, список використаних джерел, додатки.

5 Перелік графічного матеріалу: структурна схема розробленого додатку.

Перелік ілюстративного матеріалу: екран з завданнями, позначення обраних та виділення створених завдань.

6 Консультанти роботи наведені у табл. 1.

Таблиця 1 – Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1 – 3	к.т.н., доц., доцент каф. ОТ Снігур А. В	21.03.2025	13.06.2025
Нормконтроль	ас. каф. ОТ Швець С. І.	14.05.2025	30.05.2025

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план наведений в табл. 2.

Таблиця 2 – Календарний план

№ етапу	Назва етапу	Термін виконання початок	Примітка
1	Аналіз літературних джерел	21.03.2025 – 25.03.2025	вик.
2	Розробка технічного завдання	26.03.2025 – 02.04.2025	вик.
3	Аналіз задачі та архітектури	03.04.2025 – 09.04.2025	вик.
4	Програмна реалізація додатку	10.04.2025 – 24.04.2025	вик.
5	Тестування та оптимізація	25.04.2025 – 01.05.2025	вик.
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	02.05.2025 – 10.05.2025	вик.
7	Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат	10.05.2025 – 13.05.2025	вик.

Студентка _____ Педосенко М. Ю.

Керівник роботи _____ Снігур А. В.

АНОТАЦІЯ

УДК 004.4

Педосенко М.Ю. Система моніторингу та аналізу результатів навчання студентів. Бакалаврська кваліфікаційна робота зі спеціальності 123 – Комп'ютерна інженерія, освітня програма — Системне програмування. Вінниця: ВНТУ, 2025, 74 с.

На укр. мові. Бібліогр. назв: 23; рис.: 18; табл.: 1.

У роботі розроблено мобільний застосунок для планування навчальної діяльності та аналізу продуктивності студентів. Реалізовано функціонал управління завданнями, обліку навчального часу та візуалізації результатів. Система створена на базі Android Studio з використанням Room Database. Застосунок орієнтований на підвищення ефективності навчання та розвиток навичок самоменеджменту.

Ключові слова: навчальна активність, планування, таймер, Room Database, Android Studio.

ABSTRACT

Pedosenko M.Y. Monitoring and Analysis System of Student Learning Outcomes. Bachelor's Qualification Work s in specialty 123 – Computer Engineering, educational program – Systems Programming. Vinnytsia: VNTU, 2025, 74 p.

In Ukrainian. Bibliogr.: titles: 23, figures: 18; tables: 1.

The thesis presents a mobile application for planning and analyzing student learning performance. It implements task management, time tracking, and activity visualization. The system is developed in Android Studio using Room Database. The app is aimed at improving learning efficiency and supporting self-management skills.

Keywords: learning activity, planning, timer, Room Database, Android Studio.

ЗМІСТ

ВСТУП

Актуальність. У сучасному освітньому середовищі стрімкий розвиток цифрових технологій змінює підходи до навчання у закладах вищої освіти. Мобільне навчання стало важливою складовою освітнього процесу, адже забезпечує гнучкість, доступність та персоналізований підхід. Водночас зростає потреба у цифрових інструментах, які дозволяють студентам самостійно організовувати навчання, контролювати час і аналізувати власну ефективність. Існуючі рішення часто не враховують індивідуальні потреби, тому створення мобільного застосунку з функціоналом планування, обліку та візуалізації навчальної активності є актуальним і доцільним [1-3].

Об'єкт дослідження — навчальна діяльність студентів у цифровому освітньому середовищі.

Предмет дослідження — програмні засоби для планування, моніторингу та аналізу навчальної активності студентів на мобільних пристроях.

Мета роботи — створення мобільного додатку для моніторингу навчальної діяльності студентів, який дозволить планувати завдання, відстежувати навчальний час та аналізувати результати навчання за допомогою інтегрованих календаря, таймера та діаграми активності.

Для досягнення поставленої мети необхідно вирішити такі **задачі**:

— провести огляд існуючих систем для моніторингу та аналізу навчальної діяльності студентів;

— визначити вимоги до функціональних модулів системи: календаря, таймера та діаграми активності;

— розробити архітектуру системи та продумати її інтеграцію з Room Database для збереження даних;

— реалізувати основний функціонал додатку: створення завдань, планування дедлайнів, облік навчального часу та побудову діаграми активності;

— провести тестування розробленої системи, виявити можливі помилки та запропонувати шляхи їх усунення.

Методи дослідження:

— аналіз наукових джерел і сучасних рішень;

— проєктування архітектури програмного забезпечення;

— реалізація інтерфейсних і функціональних модулів;

— тестування застосунку на практичних сценаріях;

— апробація результатів у науковому середовищі.

Апробація результатів роботи здійснена у доповіді на LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії, секція «Обчислювальна техніка» (2025).

Публікація Педосенко М.Ю., Снігур А.В. «Підсистема планування навчання та аналізу продуктивності студента». Матеріали конференції «LIV Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025)», Вінниця, 2025. [Електронний ресурс] Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24441/20251

1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ МОНИТОРИНГУ ТА АНАЛІЗУ НАВЧАЛЬНОЇ ДІЯЛЬНОСТІ СТУДЕНТІВ

1.1. Огляд підходів до планування навчального процесу у навчальних закладах

Планування навчального процесу є одним із ключових аспектів організації освітньої діяльності у закладах освіти різного рівня. У середніх навчальних закладах, таких як школи, основними елементами планування є річний та семестровий план. Вони включають визначення основних цілей та завдань на навчальний рік, структурування навчального матеріалу за предметами та встановлення термінів виконання завдань. Важливою складовою є календарний план, що деталізує кожен етап навчального процесу та дозволяє вчасно коригувати навчальні заходи з урахуванням можливих змін у розкладі. Це дає змогу уникати перевантаження учнів завданнями та забезпечує рівномірний розподіл навчального навантаження.

Використання електронних щоденників та платформ для управління завданнями є ефективним інструментом для впорядкування навчальної діяльності учнів у школах. Такі системи, як Google Classroom та EduPage, дозволяють викладачам та учням організовувати завдання, встановлювати нагадування та слідкувати за прогресом виконання завдань. Крім того, ці платформи сприяють інтерактивності навчального процесу, забезпечуючи миттєвий обмін інформацією між учасниками навчального процесу. Наприклад, Google Classroom дає можливість створювати завдання з конкретними дедлайнами, прикріплювати додаткові матеріали та коментувати виконані завдання. Це спрощує взаємодію між учнем та викладачем, дозволяючи обом сторонам залишатися в контексті навчального процесу[4-6].

У старших класах шкіл доцільно використовувати програми для самостійного планування, такі як Todoist або Notion. Вони дозволяють структурувати завдання, встановлювати пріоритети та терміни виконання, а також відстежувати прогрес. Це сприяє розвитку самодисципліни та

відповідальності за виконання завдань. Наприклад, Notion дозволяє створювати індивідуальні календарі, таблиці та списки завдань, що синхронізуються між пристроями. Це особливо корисно для старшокласників, які готуються до ЗНО чи інших екзаменів і потребують чіткого графіку підготовки.

У закладах вищої освіти процес планування навчальної діяльності більш структурований та формалізований. Застосування кредитно-модульної системи (ECTS) дозволяє студентам розподіляти навантаження протягом семестру, контролюючи обсяг виконаних завдань та засвоєного матеріалу. Викладачі використовують платформи управління навчанням (LMS), такі як Moodle та Canvas, які забезпечують можливість планування завдань, тестів та практичних занять із встановленням конкретних дедлайнів. Moodle, наприклад, дозволяє створювати структуровані курси з чітким календарем завдань та тестів, а також вбудованою системою нагадувань. Це спрощує процес підготовки до заліків та екзаменів[7,8].

Важливою складовою планування у ВНЗ є самостійне планування навчального часу студентами. Для цього застосовуються різноманітні трекери часу, календарі завдань та системи візуалізації прогресу. Наприклад, додатки типу MyStudyLife дозволяють студентам організовувати завдання за датами, пріоритетами та типами робіт, що полегшує моніторинг виконання завдань упродовж семестру. Такі інструменти не лише допомагають планувати навчальний процес, але й сприяють формуванню навичок тайм-менеджменту. Додатково, MyStudyLife надає можливість синхронізації календарів, що дозволяє інтегрувати навчальні завдання з іншими особистими справами студента[9-11].

Сучасні підходи до планування навчального процесу також включають інтеграцію адаптивних навчальних систем, які підлаштовують матеріал до рівня знань студента. Такі системи використовують елементи штучного інтелекту для аналізу прогресу студента та надання рекомендацій щодо подальших дій. Наприклад, Coursera та Khan Academy надають персоналізовані рекомендації щодо курсів та завдань на основі даних про попередні результати студента. Це

сприяє більш персоналізованому підходу до навчання та підвищує ефективність планування навчальної діяльності. Аналогічно, Duolingo використовує гейміфіковані методи для навчання іноземних мов, що допомагає структурувати процес вивчення мови та підтримувати мотивацію студента[12-15].

Актуальною є тенденція до поєднання традиційного планування з цифровими інструментами. Використання інтерактивних панелей, аналітичних систем та мобільних додатків дозволяє наочно візуалізувати процес навчання, визначати ключові моменти, які потребують уваги, та коригувати план навчання у режимі реального часу. Важливо зазначити, що поєднання декількох інструментів (календарі, трекери часу, системи нагадувань) дозволяє досягти кращих результатів, оскільки кожен інструмент виконує свою специфічну функцію у загальній системі планування. Наприклад, поєднання Google Calendar для загальних планів, Notion для структуризації завдань та Forest для контролю часу дозволяє комплексно підійти до організації навчального процесу[16-18].

Таким чином, планування навчального процесу у навчальних закладах включає як традиційні методи (річні та семестрові плани), так і сучасні цифрові інструменти, що забезпечують можливість моніторингу завдань, організації навчального часу та аналізу прогресу студента. Використання таких інструментів сприяє підвищенню продуктивності студентів та дозволяє своєчасно коригувати план навчальної діяльності з урахуванням індивідуальних потреб кожного студента. В умовах стрімкого розвитку технологій очікується подальша інтеграція інтелектуальних систем планування, що будуть адаптувати навчальний процес до потреб кожного студента, враховуючи його сильні та слабкі сторони[19-23].

1.2 Огляд систем для планування навчальної діяльності

Для організації навчального процесу студентам важливо використовувати ефективні інструменти для структурування завдань, встановлення дедлайнів та моніторингу прогресу. Серед таких систем виділяються MyStudyLife, Notion та

Todoist, кожна з яких має свої особливості та сфери застосування. Розглянемо їх уважніше.

MyStudyLife є спеціалізованим додатком, орієнтованим на студентів та викладачів, основним призначенням якого є організація навчальних завдань та подій. Цей додаток поєднує у собі функції календаря, розкладу занять, нагадувань та аналітики. Основний акцент зроблено на структуруванні завдань із чіткими дедлайнами та можливістю повторюваних подій, що особливо актуально для студентів, які мають щотижневі лекції чи семінари. Наприклад, студент може створити завдання із зазначенням терміну виконання, а також отримувати сповіщення про наближення дедлайну. Додаток дозволяє синхронізувати дані між пристроями, що забезпечує доступ до розкладу у будь-який момент. Крім того, MyStudyLife формує статистику про виконані та невиконані завдання, що дає змогу користувачу оцінити власну продуктивність протягом тижня або місяця. Це особливо корисно для студентів, які мають багато завдань із різними термінами виконання, адже вони можуть ефективно розподілити навчальний час, враховуючи всі дедлайни. Таким чином, MyStudyLife забезпечує комплексний підхід до планування навчального процесу, поєднуючи функції календаря, нагадувань та аналітики, що робить його зручним інструментом для організації навчальної діяльності. Вигляд головної сторінки додатку зображено на рис. 1.1.

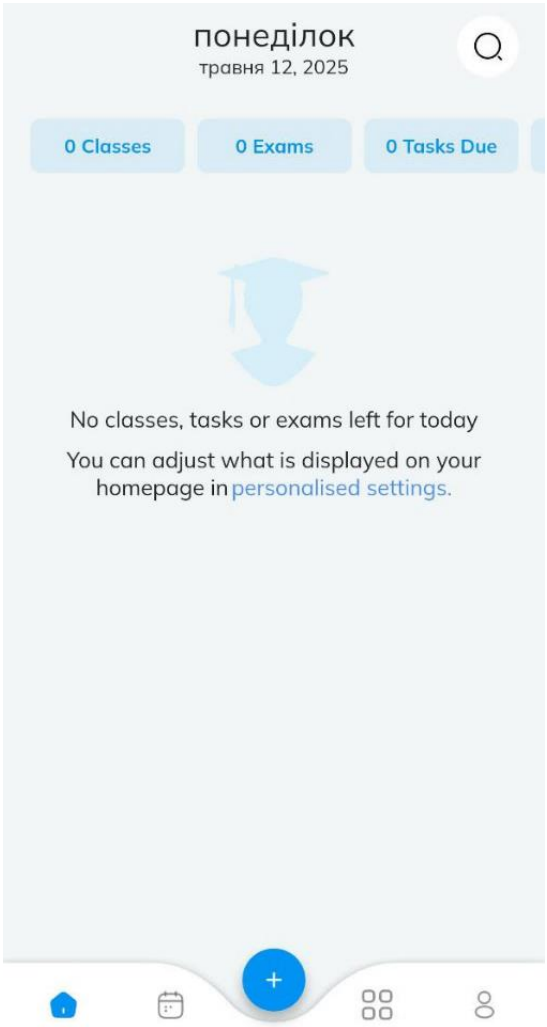

Рисунок 1.1 — Головна сторінка додатку MyStudyLife

Додаток Notion є універсальним органайзером, який надає значно ширший спектр функцій для управління завданнями та навчальними проєктами. Notion дозволяє створювати списки завдань, таблиці, бази даних, а також вести текстові документи з додатковими матеріалами. Його основною перевагою є висока гнучкість, що дозволяє користувачу налаштовувати інтерфейс відповідно до власних потреб. Наприклад, студент може створити окремі сторінки для кожного предмета, на яких відображатимуться завдання, дедлайни, важливі нотатки та корисні посилання. Завдання можна організувати за пріоритетами, датами або етапами виконання, що сприяє структурованому підходу до планування навчання. Крім того, Notion підтримує інтеграцію з іншими інструментами, такими як Google Calendar та Slack, що дозволяє автоматично оновлювати розклад та отримувати нагадування про події. Іншим важливим аспектом є

можливість візуалізації даних у форматі таблиць, канбан-дошок чи календаря. Це дозволяє студенту контролювати завантаженість, а також відстежувати прогрес виконання завдань у реальному часі. Таким чином, Notion не лише організовує завдання, але й допомагає студенту створювати індивідуальні системи планування навчального процесу, що враховують як короткострокові завдання, так і великі проєкти з багатьма етапами. Вигляд головної сторінки додатку зображено на рис. 1.2.

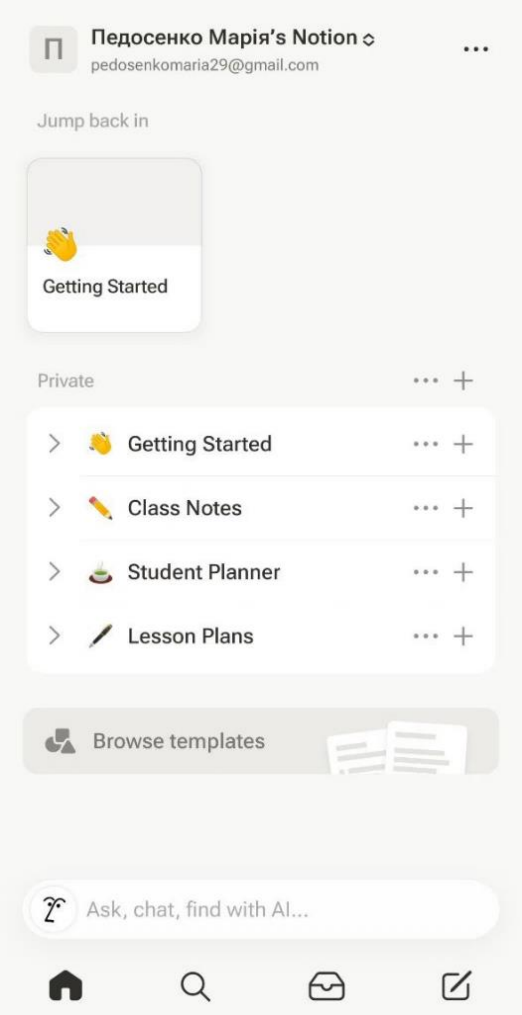

Рисунок 1.2 — Головна сторінка додатку Notion

Ще одним інструментом для планування навчальної діяльності є Todoist, основний фокус якого спрямований на створення списків завдань із можливістю пріоритизації та встановлення нагадувань. Додаток має простий та зрозумілий інтерфейс, що робить його доступним для студентів будь-якого рівня. Todoist дозволяє створювати завдання, розподіляти їх за пріоритетами та встановлювати

конкретні дати та час виконання. Наприклад, студент може створити список завдань на день, який включатиме підготовку до тесту, написання курсової роботи та відвідування лекції. Важливою функцією є інтеграція з Google Calendar, що дозволяє синхронізувати завдання та отримувати сповіщення про дедлайни. Додаток також формує звіти про виконання завдань за певний період, що дозволяє оцінити продуктивність користувача. Такий функціонал сприяє розвитку самодисципліни та дозволяє користувачу краще контролювати власний час. На відміну від Notion, Todoist має обмежений функціонал і орієнтований переважно на прості списки завдань, проте саме це робить його зручним для швидкого управління щоденними справами. Вигляд головної сторінки додатку зображено на рис. 1.3.

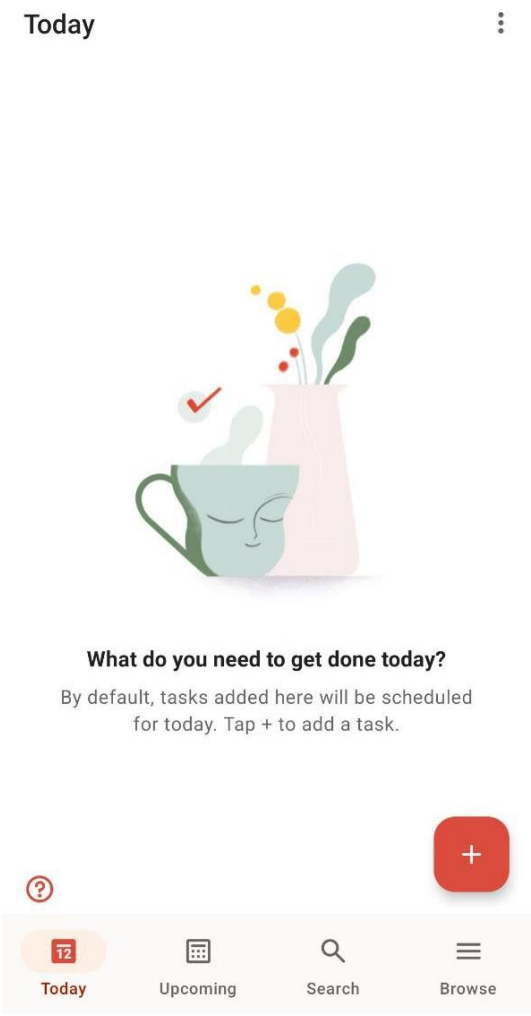

Рисунок 1.3 — Головна сторінка додатку Notion

Усі оглянуті додатки мають різну спрямованість та функціонал, але є корисними інструментами для планування навчальної діяльності. MyStudyLife забезпечує інтеграцію розкладу занять, календаря завдань та нагадувань, що робить його ефективним для організації навчальних процесів у школах та університетах. Notion дозволяє не лише планувати завдання, але й створювати комплексні системи управління проєктами з можливістю структурування матеріалів у таблицях та базах даних. Todoist акцентує увагу на простих списках завдань із можливістю нагадувань та аналізу виконання, що підходить для щоденного планування короткострокових завдань. Вибір інструменту залежить від специфіки навчального процесу, кількості завдань та рівня деталізації, який потребує користувач.

1.3 Огляд систем для моніторингу продуктивності студентів

На відміну від додатків, орієнтованих на планування завдань та структурування навчального процесу, існує також низка інструментів, які фокусуються на відстеженні продуктивності та аналізі витраченого часу. Такі додатки дозволяють студентам оцінити свою активність, виявити періоди найбільшої концентрації та визначити фактори, що відволікають від навчання. Серед них особливої уваги заслуговують Forest, Focus To-Do та RescueTime.

Forest — це додаток, який поєднує у собі тайм-трекер та елементи гейміфікації, створюючи систему мотивації для користувачів. Основна концепція Forest полягає у вирощуванні віртуального дерева протягом визначеного часу, протягом якого користувач має залишатися сфокусованим на завданні. Якщо студент перериває роботу або відкриває інші додатки, дерево гине, що стимулює користувача залишатися зосередженим на поставленій меті. Такий підхід особливо корисний для студентів, які часто відволікаються на соціальні мережі або інші розваги під час навчання.

Forest також формує статистику про завершені сесії, що дозволяє аналізувати витрачений час за день, тиждень або місяць. Користувач може переглянути, скільки часу він присвятив навчальній діяльності, а скільки —

відволіканню. Це дає змогу виявити неефективні періоди та коригувати планування навчання. Крім того, Forest має додаткову функцію — висаджування справжніх дерев. Якщо користувач досягає певної кількості завершених сесій, він може використати накопичені бали для висаджування реального дерева через співпрацю Forest із організацією Trees for the Future. Така ініціатива не лише мотивує до продуктивної роботи, але й створює додаткову цінність для користувача. Головна сторінка додатку зображено на рис. 1.4.

Рисунок 1.4 — Головна сторінка додатку Forest

На відміну від Forest, Focus To-Do базується на техніці Pomodoro — методі управління часом, що передбачає поділ завдань на 25-хвилинні інтервали з короткими перервами. Такий підхід дозволяє зберігати високу концентрацію протягом коротких сесій, не перевантажуючи мозок. Focus To-Do надає можливість налаштовувати тривалість сесій та перерв відповідно до потреб

користувача. Це особливо корисно для студентів, які працюють над великими обсягами завдань і потребують структурованого підходу до їх виконання.

Додаток також генерує звіти про виконану роботу, що дозволяє оцінити ефективність навчальних сесій за певний період. Наприклад, після завершення навчального тижня студент може переглянути загальний обсяг часу, витраченого на виконання завдань, і оцінити, наскільки продуктивно він використовував свій час. Це сприяє розвитку самодисципліни та дозволяє краще планувати подальші навчальні періоди. Окрім того, Focus To-Do підтримує синхронізацію між пристроями, що дозволяє зберігати дані та отримувати доступ до них із будь-якого пристрою, зберігаючи цілісність статистики. Головна сторінка додатку зображено на рис. 1.5.

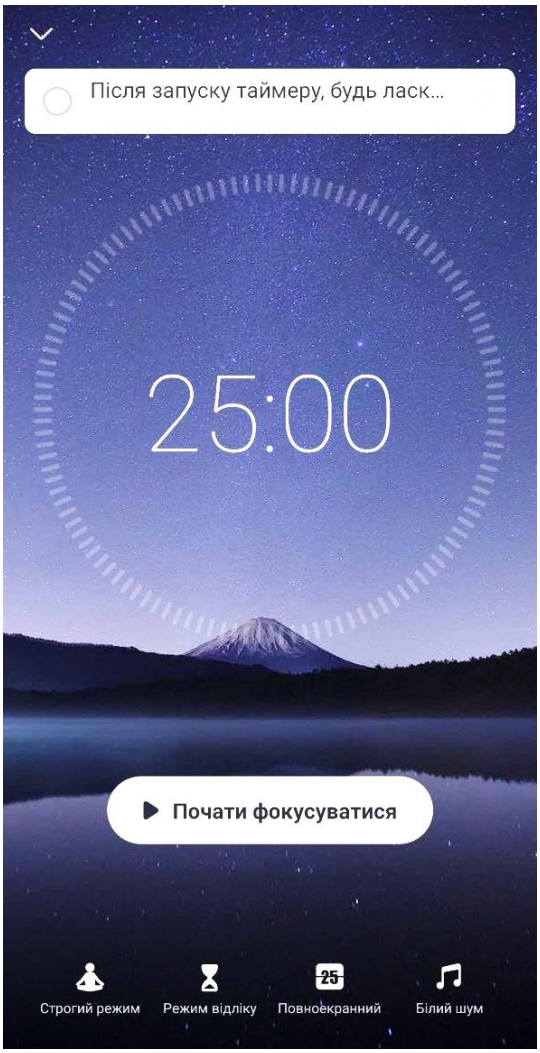

Рисунок 1.5 — Головна сторінка додатку Focus To-Do

RescueTime є одним із найпотужніших інструментів для детального аналізу активності користувача. На відміну від двох попередніх додатків, RescueTime не лише відстежує тривалість сесій, але й аналізує, які саме сайти та додатки були використані протягом робочого часу. Це дозволяє студенту побачити повну картину своєї активності та виявити джерела відволікання.

RescueTime автоматично розподіляє використані ресурси за категоріями, наприклад, «Освіта», «Соціальні мережі», «Розваги» тощо. Це дає змогу побачити, скільки часу студент витратив на корисні завдання та скільки — на непродуктивну діяльність. Користувач також може встановити цілі на день, тиждень або місяць, наприклад, провести не більше 30 хвилин у соціальних мережах або присвятити 3 години підготовці до екзамену. Якщо ці межі перевищуються, RescueTime надсилає попередження, що стимулює користувача коригувати свою активність.

Крім того, RescueTime формує детальні звіти про продуктивність, у яких можна побачити найпродуктивніші та найменш продуктивні дні, а також визначити періоди найвищої концентрації. Це дозволяє студенту краще планувати робочі години, відводячи найбільш важливі завдання на той час, коли він зазвичай найбільш зосереджений та ефективний. Головна сторінка додатку зображено на рис. 1.6.

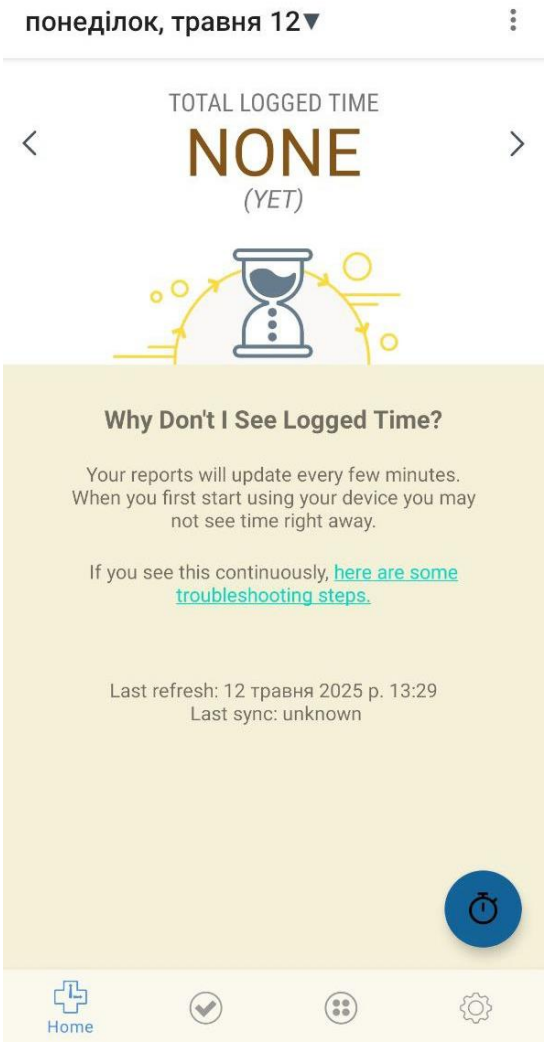

Рисунок 1.6 — Головна сторінка додатку Focus To-Do

Отже, Forest, Focus To-Do та RescueTime орієнтовані на різні аспекти моніторингу продуктивності. Forest використовує гейміфікацію для підтримки концентрації та боротьби з відволіканнями. Focus To-Do забезпечує структурований підхід до управління часом через методику Pomodoro. RescueTime є найбільш аналітичним інструментом, що дозволяє глибоко аналізувати активність користувача та виявляти основні джерела відволікання. Вибір конкретного додатку залежить від потреб користувача: якщо основна проблема — відволікання, Forest буде найкращим вибором; якщо потрібно структурувати навчальний процес, оптимальним варіантом стане Focus To-Do; а якщо необхідно проаналізувати свою активність загалом та виявити слабкі місця, варто використовувати RescueTime.

1.4 Порівняльний аналіз комп'ютерних систем моніторингу та аналізу результатів навчання

Після детального аналізу застосунків для планування навчальної діяльності та моніторингу продуктивності було складено порівняльну таблицю, що відображає основні функції розглянутих додатків у порівнянні з розроблюваним застосунком (табл. 1.1).

Таблиця 1.1 — Порівняльна характеристика розглянутих додатків

Застосунок / Функція	MyStudyLife	Notion	Todoist	Forest	Focus To-Do	RescueTime	Створений додаток
Планування завдань	+	+	+	-	-	-	+
Календар	+	+	+	-	-	-	+
Нагадування	+	+	+	-	-	-	+
Створення списків	-	+	+	-	-	-	+
Візуалізація завдань	+	+	-	-	-	-	+
Таймер/трекер часу	-	-	-	+	+	+	+
Звіти про активність	+	+	-	+	+	+	+
Гейміфікація	-	-	-	+	-	-	-
Синхронізація даних	+	+	+	+	+	+	+
Адаптивні рекомендації	-	-	-	-	-	+	-

Різноманітні додатки для планування навчальної діяльності та моніторингу продуктивності пропонують користувачам різний набір функцій, що дозволяє обирати оптимальний інструмент залежно від потреб. MyStudyLife, Notion та Todoist зосереджені на організації завдань, структуризації розкладу та створенні списків справ. MyStudyLife дозволяє поєднувати календар подій із розкладом занять, проте функції моніторингу часу у ньому відсутні. Notion забезпечує універсальні можливості для структурування даних у форматі таблиць, списків та канбан-дощок, але не має трекера часу. Todoist, орієнтуючись на прості списки справ із нагадуваннями, зручний для короткострокового планування, проте обмежений у функціоналі візуалізації прогресу.

Forest, Focus Booster та RescueTime орієнтовані на моніторинг продуктивності та управління часом. Forest використовує елементи гейміфікації, що стимулює концентрацію через візуалізацію прогресу у вигляді віртуального лісу, але не дозволяє створювати завдання чи структурувати їх за датами. Focus Booster реалізує техніку Pomodoro для коротких навчальних сесій з перервами, формуючи звіти про виконані інтервали часу. RescueTime забезпечує детальний аналіз активності користувача, автоматично фіксуючи витрачений час на сайти та додатки, однак його функціонал обмежується моніторингом без можливості організації завдань.

Мобільний застосунок, що розробляється, має об'єднати простоту планування завдань та інтерактивні методи моніторингу часу, пропонуючи єдину систему для організації навчальних процесів. Він повинен поєднувати календар завдань із можливістю візуалізації прогресу, а також забезпечувати структурований облік витраченого часу та формування звітів. Такий підхід дозволить оптимізувати навчальну діяльність студентів, сприяючи підвищенню їхньої продуктивності.

2 ОСОБЛИВОСТІ РОЗРОБКИ СИСТЕМИ МОНИТОРИНГУ ТА АНАЛІЗУ НАВЧАННЯ

2.1. Цілі, задачі та функціональні вимоги до системи

Метою розробки мобільного додатку є створення інтегрованої системи для планування навчальних завдань та моніторингу продуктивності студентів. Додаток має поєднувати функції управління завданнями, обліку витраченого часу та візуалізації навчальних сесій у вигляді діаграм, що сприятиме організації навчального процесу та підвищенню продуктивності користувачів. Заплановані функції додатку спрямовані на формування комплексної системи, яка забезпечить студентів засобами для планування завдань, контролю виконання та аналізу витраченого часу.

Основними задачами, які має реалізовувати мобільний додаток, є підтримка облікового запису користувача, модуль управління завданнями, інтеграція календаря, реалізація таймера для фіксації навчального часу, візуалізація активності у вигляді діаграм, а також можливість налаштування системи відповідно до потреб користувача.

Передбачається створення системи облікових записів, яка дозволяє користувачу створювати профіль із зазначенням імені, аватара та пароля. Реалізовано можливість входу до системи, зміни паролю та виходу з профілю. Облікові дані зберігаються локально в базі даних, що дозволяє зберігати персональну інформацію після виходу з додатку.

У межах модуля управління завданнями реалізується інтерфейс для створення навчальних завдань із зазначенням назви, дати виконання та статусу. Користувач має можливість позначати завдання як обрані, редагувати, видаляти, а також відмічати їх виконання. Передбачена фільтрація за пріоритетами та датами для швидкої навігації у списку.

Календар виступає центральним елементом для структурування навчальної діяльності. Завдання можуть призначатися на конкретні дати, що дає змогу візуально оцінити навантаження за день, тиждень або місяць. Редагування

завдань можливе безпосередньо з календаря, що забезпечує зручність планування.

Таймер дозволяє користувачам фіксувати час, витрачений на навчання, з можливістю вибору шаблонів тривалості (наприклад, 30 або 45 хвилин). Час зараховується до статистики лише після повного завершення навчальної сесії без переривань, що сприяє формуванню звички до зосередженої роботи.

Для зручного аналізу продуктивності реалізовано екран візуалізації навчальної активності. Діаграма показує загальний час, витрачений на навчання за день, а також кількість завершених сесій. Також передбачено детальну статистику для кожної сесії — її тривалість, статус завершення та дату.

У додатку також реалізовано екран налаштувань, який дозволяє обирати мову інтерфейсу (українську або англійську), а також змінювати пароль безпосередньо з профілю користувача.

Сформульовані цілі та задачі визначають основні вимоги до функціональності додатку. Важливо забезпечити можливість взаємодії між окремими компонентами системи. Так, календар має бути пов’язаний із завданнями, щоб усі створені завдання автоматично відображалися на вибраних датах. Таймер, в свою чергу, має інтегруватися з модулем діаграми активності, передаючи дані про завершені навчальні сесії для їх подальшої візуалізації.

Крім цього, система має підтримувати збереження даних у локальній базі даних Room Database. Це дозволить зберігати дані про обліковий запис, завдання, завершені сесії та часові інтервали навіть після виходу з додатку. Структура бази даних має бути організована таким чином, щоб дані були доступні для подальшого аналізу та візуалізації.

2.2 Аналіз способів обліку навчального часу і планування завдань

Організація навчального процесу студентів часто передбачає не лише планування завдань, але й облік витраченого часу, що дозволяє оцінити продуктивність і раціонально розподіляти навчальні сесії. У сучасних додатках

для моніторингу навчальної активності використовуються різні підходи до обліку часу та планування завдань.

Одним із найпоширеніших способів обліку часу є використання таймера. Такий підхід реалізований у додатках Forest та Focus To-Do, де користувач може встановити певний проміжок часу для навчальної сесії (рис 2.1). У Forest цей процес супроводжується гейміфікацією — користувач вирощує віртуальне дерево, яке зникає у разі переривання сесії. Focus Booster базується на техніці Pomodoro, де навчальний час розподіляється на короткі інтервали з перервами. Такий підхід дозволяє не перевантажувати користувача великими обсягами інформації, натомість акцентуючи увагу на коротких, але ефективних сесіях.

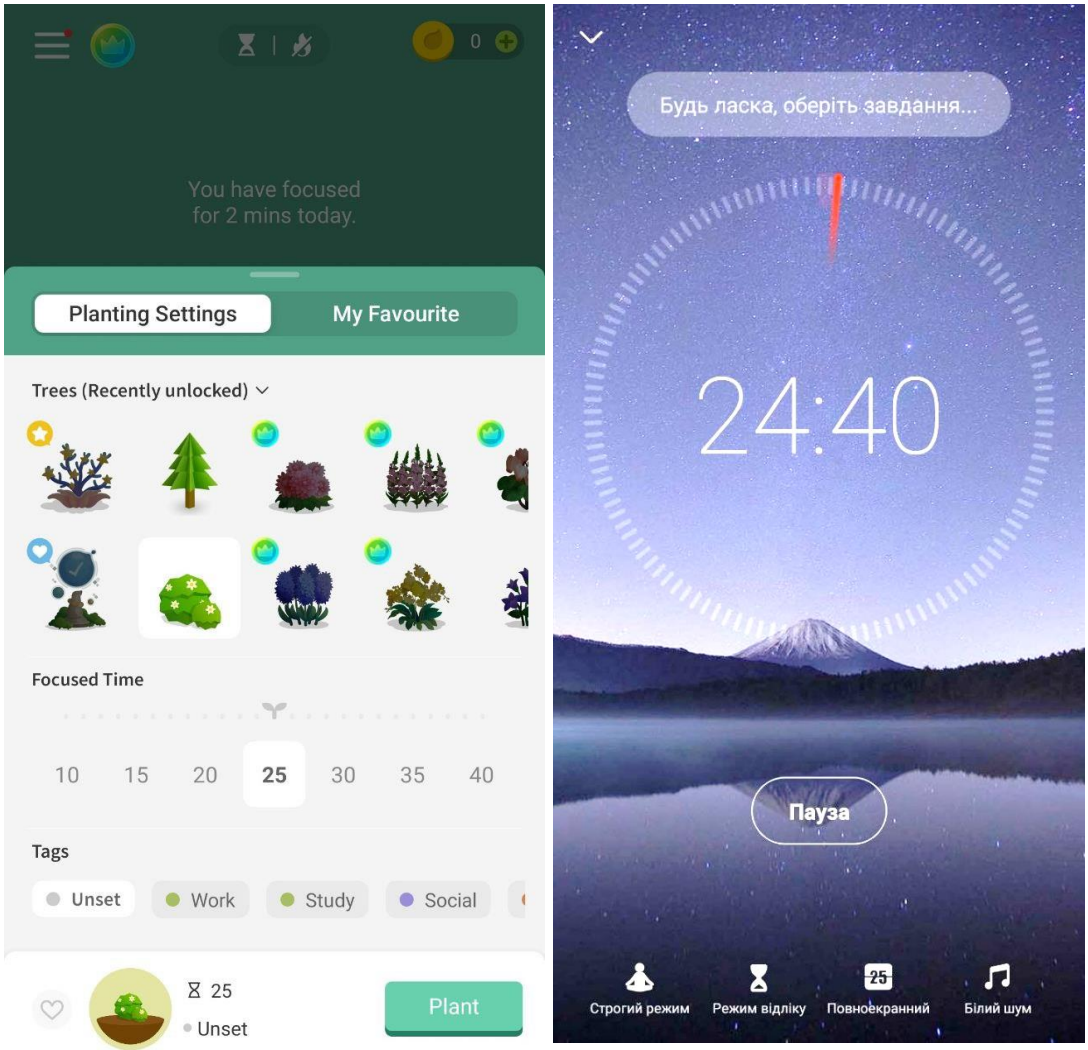

Рисунок 2.1 — Приклад роботи таймерів у додатках Forest та Focus To-Do

Автоматичний трекінг часу є ще одним підходом до обліку навчальної активності. Такий принцип реалізований у RescueTime, який автоматично відстежує, які додатки та сайти використовує користувач, і розподіляє витрачений час за категоріями. Це дозволяє аналізувати, скільки часу було присвячено навчальним завданням, а скільки — соціальним мережам або розвагам. Автоматичний трекінг особливо корисний для тих користувачів, які прагнуть контролювати свою продуктивність без активного втручання у процес відстеження часу.

У розроблюваному додатку реалізовано систему обліку часу на основі таймера. Такий підхід обрано з огляду на простоту реалізації та можливість забезпечення користувачів чіткою візуалізацією виконаних сесій. Користувачу пропонується вибір шаблону часу (15 або 45 хвилин), що дозволяє структурувати навчальний процес у фіксовані інтервали. Після завершення таймера дані про сесію автоматично передаються до блоку статистики, де зберігаються та візуалізуються у вигляді діаграм. Це забезпечує можливість відстеження тривалості кожної сесії та загального часу, витраченого на навчання за день.

Планування завдань також відіграє ключову роль у структурованій організації навчального процесу. Більшість сучасних додатків, таких як Todoist або Notion, використовують список завдань із можливістю встановлення дедлайнів та нагадувань (рис. 2.2). Така структура дозволяє користувачам створювати завдання, позначати їх як виконані або видаляти після завершення. В Notion додатково реалізована можливість структурування завдань у форматі таблиць та канбан-дощок, що забезпечує гнучкість у організації навчального процесу.

Іншим підходом до планування завдань є календарна система, реалізована у MyStudyLife. У ній кожне завдання інтегрується у календар з чітким зазначенням дати виконання, що дозволяє користувачу отримати загальну картину завантаженості за певний період. Такий підхід сприяє кращому розподілу часу між завданнями та вчасному їх виконанню.

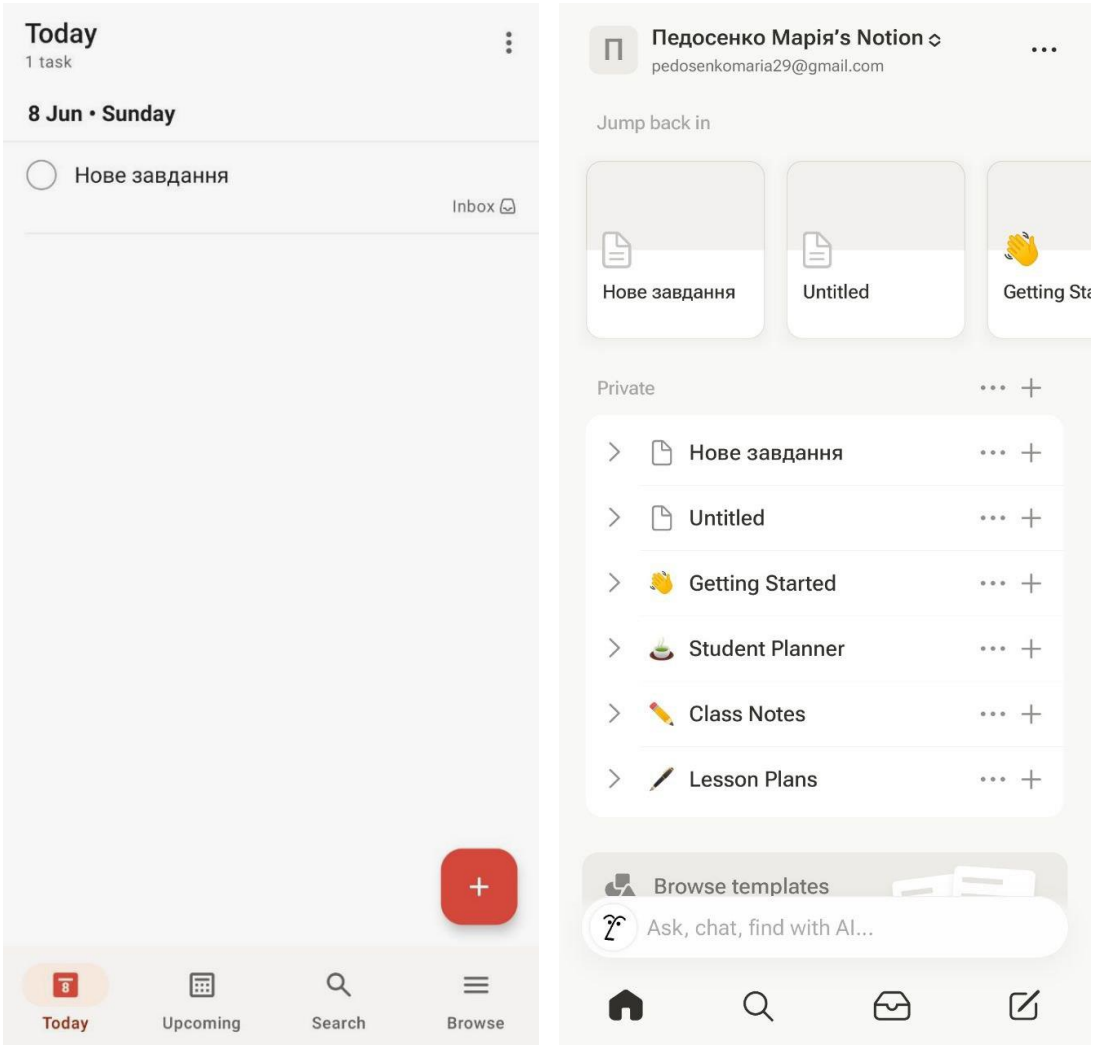

Рисунок 2.2 — Приклад списків завдань у додатках Todoist або Notion

У розроблюваному додатку реалізовано комбіновану систему планування завдань, що поєднуватиме список завдань та календар. Такий підхід дозволяє користувачам створювати завдання у вигляді простого списку, позначати їх як виконані та редагувати за необхідності. Кожне створене завдання можна призначити на конкретний день через календар, що забезпечує зручну візуалізацію навчальної діяльності за певний період. Це рішення спрямоване на те, щоб користувачі могли легко орієнтуватися у своїх завданнях, оцінювати загальний обсяг робіт та уникати перевантаження напередодні дедлайнів.

Аналіз сучасних підходів до обліку часу та планування завдань дозволив визначити найбільш доцільні методи для реалізації у розроблюваному додатку. Використання таймера забезпечить контроль за навчальними сесіями та зберігання даних про виконані завдання у статистиці, тоді як система календаря

дозволить структурувати завдання за днями, забезпечуючи зручну візуалізацію та нагадування про наближення дедлайнів. Така інтеграція сприятиме формуванню системного підходу до організації навчального процесу та підвищенню продуктивності користувачів.

2.3. Архітектура системи: підсистеми таймера, календаря, діаграми активності

Архітектура розроблюваної системи організована таким чином, щоб забезпечити інтеграцію між ключовими компонентами — таймером, календарем та діаграмою активності. Кожен із цих компонентів виконує окрему функцію, але водночас вони взаємопов’язані, що дозволяє формувати цілісну систему для планування та аналізу навчального процесу.

Архітектура додатку побудована за модульним принципом, що передбачає виділення окремих підсистем: таймер, календар, діаграма активності, обліковий запис, налаштування та база даних. Взаємодія між компонентами реалізується через Room Database, яка зберігає дані про завдання, навчальні сесії, параметри облікового запису та статистику продуктивності. Центральним елементом архітектури виступає база даних, що об’єднує всі компоненти у єдину систему, забезпечуючи синхронізацію даних між ними.

Підсистема таймера виконує функцію обліку навчального часу та зберігання даних про завершені сесії. Таймер реалізовано у вигляді окремого екрану, де користувач може обрати один із шаблонів тривалості (30 хвилин, 40 хвилин). Після запуску таймера розпочинається навчальна сесія, і користувачеві рекомендується не відволікатися до завершення часу. Якщо сесію було завершено успішно, її дані — тривалість, дата, статус — автоматично передаються до діаграми активності. У разі переривання сесії до завершення часу дані не зберігаються, що мотивує користувача зосереджено працювати до завершення таймера.

Структура таймера передбачає збереження інформації у таблиці Sessions, яка містить поля session_id, start_time, end_time, duration, status. При завершенні

таймера система створює новий запис у таблиці із зазначенням дати та тривалості сесії. Ці дані надалі використовуються для формування діаграми активності, яка відображає загальний час навчання за день.

Підсистема календаря відповідає за організацію завдань у часовому форматі, що дозволяє візуалізувати навчальні події за днями. Календар реалізовано у вигляді інтерактивного елементу, на якому користувач може обрати конкретний день та призначити на нього завдання. Завдання створюються у відповідному розділі, після чого користувач може додати їх до календаря із зазначенням дати виконання.

Календар синхронізується із таблицею Tasks, яка містить інформацію про завдання: task_id, task_name, date, status, is_favorite. Кожне завдання, додане до календаря, автоматично зберігається у цій таблиці з прив'язкою до конкретної дати. Якщо користувач позначає завдання як виконане, його статус змінюється, а візуалізація у календарі оновлюється відповідно до нового статусу. Це дозволяє контролювати навчальні події та своєчасно реагувати на дедлайни.

Підсистема діаграми активності забезпечує візуалізацію даних про завершені навчальні сесії у вигляді кругової або лінійної діаграми. Основною функцією діаграми є формування статистики про загальний час, витрачений на навчання за день. У разі натискання на діаграму користувач переходить до деталізованої статистики, де можна переглянути інформацію про кожну окрему сесію: дату, тривалість та статус.

Діаграма активності базується на даних із таблиці Sessions, яка містить інформацію про кожну навчальну сесію. Під час побудови діаграми система аналізує всі завершені сесії за день та сумує їх тривалість, відображаючи її як загальний обсяг витраченого часу. Користувач також може переглянути дані за попередні дні, що дозволяє оцінити динаміку навчальної активності за певний період.

Загальна архітектура системи забезпечує взаємозв'язок між усіма компонентами. Завдання, створені у відповідному розділі, автоматично інтегруються у календар, де їх можна редагувати або позначати як виконані.

Таймер передає дані про завершені сесії до діаграми активності, яка візуалізує інформацію про навчальні сесії за певний день. Водночас усі дані зберігаються у локальній базі даних, що забезпечує їх доступність навіть після виходу з додатку.

Таким чином, запропонована архітектура дозволяє створити інтегровану систему для планування завдань, обліку часу та візуалізації результатів навчальної діяльності. Взаємодія між компонентами забезпечує зручність у користуванні, а збереження даних у Room Database гарантує їх цілісність та доступність для подальшого аналізу.

2.4. Особливості UX/UI-дизайну для ефективного освітнього моніторингу

При проєктуванні мобільного додатку для планування навчальної діяльності та моніторингу продуктивності важливим аспектом є розробка інтуїтивного та зрозумілого інтерфейсу, який сприятиме зручній навігації між основними екранами та забезпечить користувача всією необхідною інформацією у доступній формі. Дизайн системи має відповідати принципам мінімалізму, з акцентом на простоту та чіткість відображення даних.

У додатку було використано кольорову гаму, що поєднує відтінки синього та блакитного з обмеженим застосуванням рожевого для акцентних елементів. (рис. 2.3). Основним кольором додатку є блакитний, який використовується для фонових панелей, кнопок навігації та більшості інтерактивних елементів. Блакитний створює спокійний візуальний фон та полегшує сприйняття контенту, не перевантажуючи екран.

Синій колір використовується для текстових елементів, зокрема заголовків, назв кнопок та основних індикаторів. Крім того, синій відтінок застосовується у діаграмі активності для відображення завершених навчальних сесій, що дозволяє користувачу одразу оцінити обсяг витраченого часу.

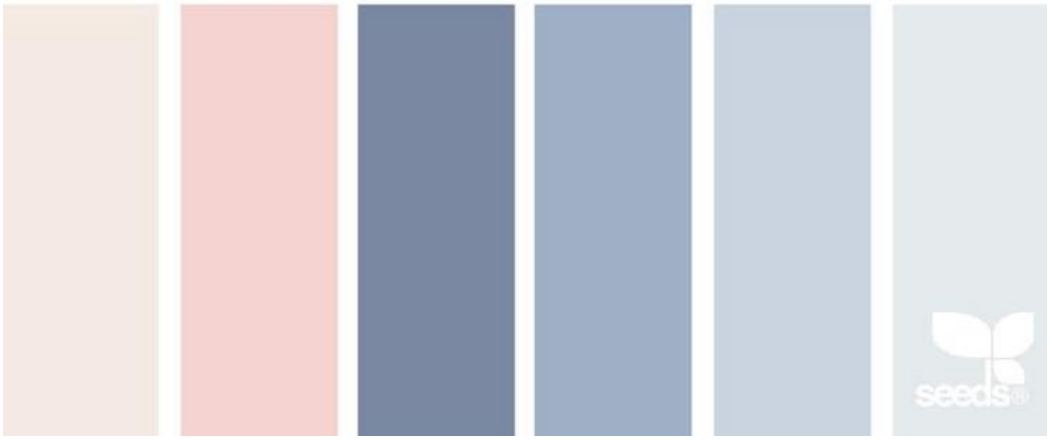

Рисунок 2.3 — Колірна гама застосунку

Рожевий колір у дизайні використовується лише для акцентування певних дій або елементів, які потребують додаткової уваги. Зокрема, рожевим кольором позначається текст «Вийти з облікового запису» (рис. 2.4), щоб привернути увагу до цієї дії та уникнути випадкових натискань. У календарі рожевим кольором буде підсвічено дати, на які призначено завдання. У такому випадку цифра дати змінюватиме колір із синього на рожевий, що дозволить користувачу швидко орієнтуватися у розкладі.

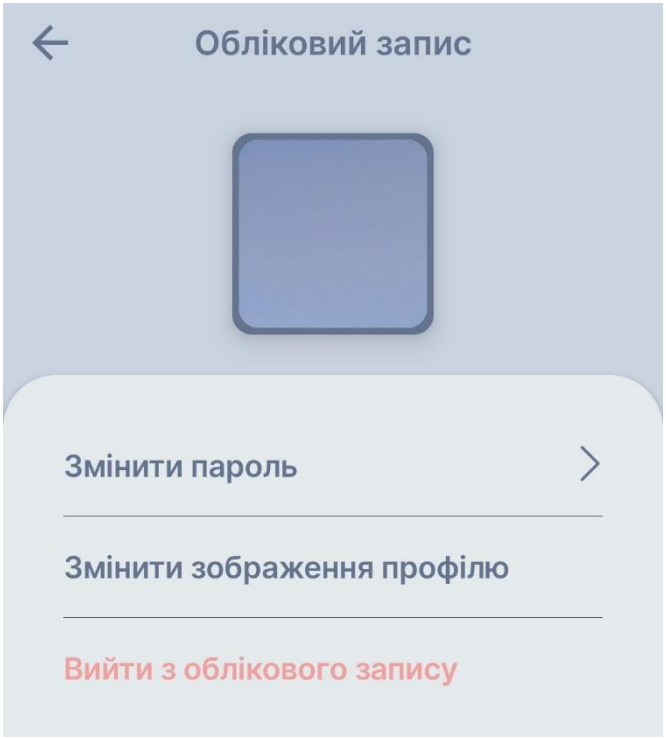

Рисунок 2.4 – Застосування рожевого кольору у додатку

Додаток має назву SmartStudent, що відповідає його призначенню — допомагати студентам у плануванні та контролі навчального процесу. Піктограма застосунку містить дві літери S, які є скороченням від назви (рис. 2.5).

Рисунок 2.5 — Піктограма додатку

Основний екран додатку реалізовано у вигляді панелі з основними елементами: діаграмою активності, кнопками для переходу до завдань, календаря та таймера. Діаграма активності має форму кругової діаграми, де сині сегменти відображатимуть завершені навчальні сесії за поточний день. Важливо, що діаграма не деталізує кожну окрему сесію, а лише відображатиме загальний час навчання. Детальна інформація про кожну сесію доступна після натискання на діаграму — користувач перейде до екрану зі списком сесій, де вказано тривалість, дату та статус кожної з них.

Екран завдань реалізовано як список, де кожне завдання відображається у вигляді картки із зазначенням назви та дати. Завдання у цьому списку не можна позначити як виконане. Єдина взаємодія з ними — можливість видалити завдання або позначити його як обране, що буде відображено через іконку зірочки. Обрані завдання візуально не виділяються кольором, а лише мають відповідний індикатор.

Відмічення завдань як виконаних реалізовано через календар. Після натискання на певну дату у календарі користувач потрапляє до списку завдань, призначених на цей день. Вибране завдання відкривається через меню з трьома крапками, де є можливість позначити його як виконане. Виконані завдання

візуально виділені темно-блакитним кольором, тоді як незавершені залишаються у звичайному блакитному відтінку. Такий підхід дозволяє зосередити увагу користувача саме на календарі, що сприяє формуванню звички до планування навчальних завдань за датами.

Календар у додатку реалізовано у вигляді стандартного календарного сіткового формату з відображенням чисел місяця. Дні, на які призначено завдання, виділяються рожевим кольором, що дозволяє користувачу швидко орієнтуватися у датах із запланованими завданнями. Натискання на певну дату відкриває список завдань за цей день, де користувач може позначити їх як виконані. Це забезпечує не лише візуальну організацію завдань, але й інтеграцію календаря як основного засобу для контролю дедлайнів.

Екран таймера реалізовано у вигляді окремого модуля з трьома основними кнопками. Ліва кнопка дозволяє обрати мелодію для таймера, центральна запускає або зупиняє таймер, а права — відкриває екран для вибору часу сесії. Вибір часу реалізовано через три окремі секції: години, хвилини та секунди, які користувач може налаштовувати за допомогою свайпу, подібно до стандартних таймерів у мобільних пристроях. Цей підхід дозволяє швидко обрати потрібний проміжок часу без додаткових налаштувань.

Після завершення таймера користувач отримує повідомлення «Час вийшов» із звуковим сигналом. Таймер не прив’язаний до конкретного завдання, а фіксує загальний час навчальної сесії. Дані про завершені сесії автоматично передаються до діаграми активності, де користувач може побачити загальний обсяг витраченого часу за день.

У процесі створення дизайну додатку було забезпечено простоту взаємодії, використано контрастні кольори для акценту на важливих елементах та створено зручну візуальну структуру. Основними кольорами інтерфейсу є блакитний та синій, які сприяють зосередженню уваги та створюють спокійне візуальне середовище для виконання навчальних завдань. Рожевий колір використовується мінімально, виключно для акцентування важливих дій або

індикаторів. Такий підхід дозволяє уникнути перевантаження кольорами та забезпечує візуальну узгодженість між основними компонентами системи.

2.5. Вибір інструментів та технологій для реалізації системи

Проєктування та розробка мобільного застосунку для моніторингу та аналізу результатів навчання студентів потребує обґрунтованого вибору технологічного стеку. Від правильності цього вибору залежить не лише стабільність роботи системи, але й можливість її масштабування, підтримки та подальшого вдосконалення. У цьому проєкті було використано набір сучасних засобів і підходів, які забезпечують ефективну розробку клієнтської частини та зручну організацію локального зберігання даних.

У якості основної мови розробки було обрано Kotlin — сучасну статично типізовану мову програмування, яка офіційно підтримується Google як пріоритетна для розробки під Android. Основними перевагами Kotlin є його лаконічний синтаксис, висока читабельність коду, безпечність щодо нульових значень (null safety) та підтримка функціональних конструкцій, які спрощують реалізацію логіки додатку. Застосування Kotlin дозволяє скоротити обсяг коду на 20–30% у порівнянні з Java, що позитивно впливає на швидкість розробки та спрощує налагодження. Крім того, Kotlin має повну сумісність із Java, що відкриває доступ до великої кількості бібліотек і сторонніх компонентів, які можуть бути задіяні у майбутніх версіях системи.

Для реалізації програмного продукту використано Android Studio — офіційне середовище розробки від Google для створення Android-застосунків. Середовище підтримує розширення, що автоматично підключають необхідні залежності, інтегрується з системою керування версіями (наприклад, Git), має вбудований емулятор для тестування додатків, а також забезпечує можливість профілювання продуктивності та моніторингу використання пам’яті. Однією з ключових переваг Android Studio є гнучка інтеграція з Android Jetpack — набором бібліотек, які значно полегшують розробку сучасних додатків, зокрема, через автоматизацію роботи з базами даних, навігацією між екранами,

управлінням життєвим циклом тощо. Наявність вбудованих інструментів для візуального дизайну інтерфейсу дозволила спростити реалізацію інтуїтивно зрозумілого UX/UI, адаптованого до екранів різних розмірів.

Для локального збереження інформації обрано Room — бібліотеку для роботи з базами даних, що входить до складу Android Jetpack і є надбудовою над SQLite. На відміну від прямої роботи з SQL-запитами, Room дозволяє визначати структуру бази даних за допомогою анотацій у коді та використовувати об’єктно-орієнтовану модель взаємодії з даними. Це значно знижує імовірність логічних помилок, забезпечує автоматичну перевірку типів і спрощує підтримку коду. Крім того, Room підтримує використання корутин та LiveData, що дозволяє створити реактивний інтерфейс, який автоматично оновлюється при зміні даних у базі. У межах розробленого застосунку Room використовується для збереження інформації про облікові записи користувачів, навчальні завдання, завершені сесії таймера та дані для побудови діаграм активності.

Попри переважно офлайн-призначення застосунку, у проєкті також реалізована базова підтримка хмарних сервісів. Зокрема, для обробки реєстрації та входу користувача використано Firebase Authentication — надійний сервіс для керування автентифікацією, який дозволяє забезпечити захист облікових записів без необхідності створювати власний сервер. Firebase автоматично обробляє запити на створення профілів, шифрує паролі та генерує унікальні токени для кожного користувача. Такий підхід дозволяє централізовано керувати сесіями та гарантує високий рівень безпеки, навіть у разі масштабування системи у майбутньому.

Firebase також розглядається як основа для розширення функціоналу додатку у наступних версіях — зокрема, для реалізації синхронізації даних між пристроями, автоматичного резервного копіювання інформації, зберігання журналів активності та аналітики. У разі необхідності зберігання статистики у хмарі або створення веб-версії системи можна буде інтегрувати Firebase Firestore як основну базу даних, не змінюючи основної архітектури клієнтської частини.

Обраний набір технологій дозволив досягти кількох ключових цілей:

— спростити розробку за рахунок зручного середовища, високорівневих бібліотек і сучасної мови програмування;

— підвищити стабільність системи, забезпечивши правильну організацію взаємодії з базою даних та чітке управління станами;

— забезпечити можливість розширення — додаток побудований з урахуванням майбутнього масштабування, зокрема, з переходом на хмарну інфраструктуру або мультиплатформність;

— оптимізувати роботу в офлайн-режимі — Room забезпечує швидкий доступ до локальних даних навіть за відсутності інтернет-з’єднання.

Таким чином, використані інструменти не лише відповідають завданням, поставленим у межах даного проєкту, а й створюють міцну основу для подальшого розвитку мобільного застосунку.

2.6. Забезпечення безпеки користувацьких даних

З огляду на те, що мобільний додаток працює з персональними даними користувачів (облікові записи, навчальні завдання, інформація про сесії), одним із пріоритетних завдань під час розробки стало впровадження механізмів захисту та конфіденційності даних. Безпека інформації є критично важливою як з точки зору захисту особистої інформації користувачів, так і для забезпечення довготривалої стабільності системи.

Для автентифікації користувачів використовується сервіс Firebase Authentication, який забезпечує захищену реєстрацію та вхід через логін і пароль. Дані облікових записів не зберігаються локально у відкритому вигляді — після автентифікації Firebase видає унікальний маркер доступу (token), який використовується для підтвердження ідентичності користувача. Паролі зберігаються у зашифрованому вигляді у хмарній інфраструктурі Google, що відповідає сучасним стандартам безпеки (наприклад, використання алгоритму bcrypt для хешування).

Користувач може змінити пароль безпосередньо з інтерфейсу додатку. Усі запити на зміну чи відновлення пароля обробляються на стороні Firebase, що виключає можливість доступу до пароля зі сторони клієнтського застосунку.

Дані про завдання, навчальні сесії та календарні події зберігаються у Room Database, яка працює поверх SQLite. Щоб забезпечити конфіденційність інформації на пристрої, база даних може бути зашифрована за допомогою сторонньої бібліотеки (наприклад, SQLCipher), що дає змогу запобігти несанкціонованому доступу до даних у разі втрати або крадіжки пристрою. У даній версії додатку реалізовано захист через обмежений доступ до каталогу з даними додатку в Android-середовищі, а також можливість очищення локальної інформації під час виходу з облікового запису.

Інтерфейс додатку передбачає обов’язкову валідацію користувацького вводу. Наприклад, при створенні завдання перевіряється, чи не є поле з назвою пустим, чи дата дедлайну має коректний формат. Це дозволяє уникнути помилок зберігання, збоїв при побудові діаграм і некоректного відображення інформації у календарі.

Усі основні функції додатку доступні лише після авторизації. Невідомий або неавторизований користувач не може переглядати чи змінювати дані, що зберігаються у системі. Якщо маркер авторизації недійсний або його термін дії минув, користувача буде автоматично перенаправлено на екран входу.

У разі використання Firebase для передачі чи синхронізації інформації (наприклад, під час розширення функціоналу), всі запити здійснюються через захищені канали з використанням протоколу HTTPS. Це унеможливлює перехоплення або зміну даних у процесі їхньої передачі мережею.

Можливі шляхи посилення безпеки в майбутньому:

— використання двофакторної аутентифікації (2FA) для додаткового захисту облікових записів;

— автоматичне шифрування всієї Room-бази за допомогою SQLCipher;

— впровадження механізму блокування додатку за допомогою PIN-коду або біометричних даних (Face ID, Touch ID);

— визначення політик збереження сесій — наприклад, автоматичний вихід після тривалого простою.

У межах роботи реалізовано базові та надійні заходи для забезпечення безпеки персональних даних користувача. Архітектура системи дозволяє у подальшому легко масштабувати рівень захисту без потреби повного перепроєктування. Це забезпечує не лише відповідність сучасним стандартам, а й створює довіру з боку користувачів до мобільного застосунку як до безпечного інструменту самоменеджменту в навчанні.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Налаштування середовища розробки та використання Room Database

Для розробки мобільного додатку було обрано середовище розробки Android Studio. Це інтегроване середовище забезпечує всі необхідні інструменти для створення Android-застосунків, включаючи редактор коду, емулятор, систему контролю версій та можливість інтеграції з базами даних. Android Studio підтримує Kotlin — мову, що є основною для розробки Android-додатків, а також надає засоби для інтеграції з Room Database.

Вибір Android Studio зумовлений кількома ключовими факторами:

— Room Database є частиною Android Jetpack, тому Android Studio автоматично підключає необхідні бібліотеки та забезпечує підтримку ORM, що дозволяє взаємодіяти з SQLite через об’єкти Kotlin;

— Android Studio має вбудований емулятор, який дозволяє перевіряти роботу додатку на різних версіях Android та різних розмірах екранів, це дає змогу протестувати коректність відображення даних у базі та роботу з LiveData;

Для того, щоб створити проєкт у Android Studio, потрібно у вкладці «File» вибрати «New Project», після чого обрати шаблон «Empty Project» та вказати ім’я проєкту, його розташування та мінімальний SDK (рис. 3.1).

Для того, щоб використовувати бібліотеку Room у проєкті, необхідно спочатку додати відповідні залежності та підключити плагін для обробки анотацій. Це забезпечить коректну роботу анотацій @Entity, @Dao, @Database та дозволить використовувати Kotlin-розширення і корутини (див. лістинг 3.1). Ці рядки підключають основну бібліотеку Room, а також компілятор анотацій, необхідний для генерації коду під час компіляції.

Лістинг 3.1 – Додавання залежностей Room у файл build.gradle

```
implementation "androidx.room:room-ktx:2.4.3"
kapt "androidx.room:room-compiler:2.4.3"
```

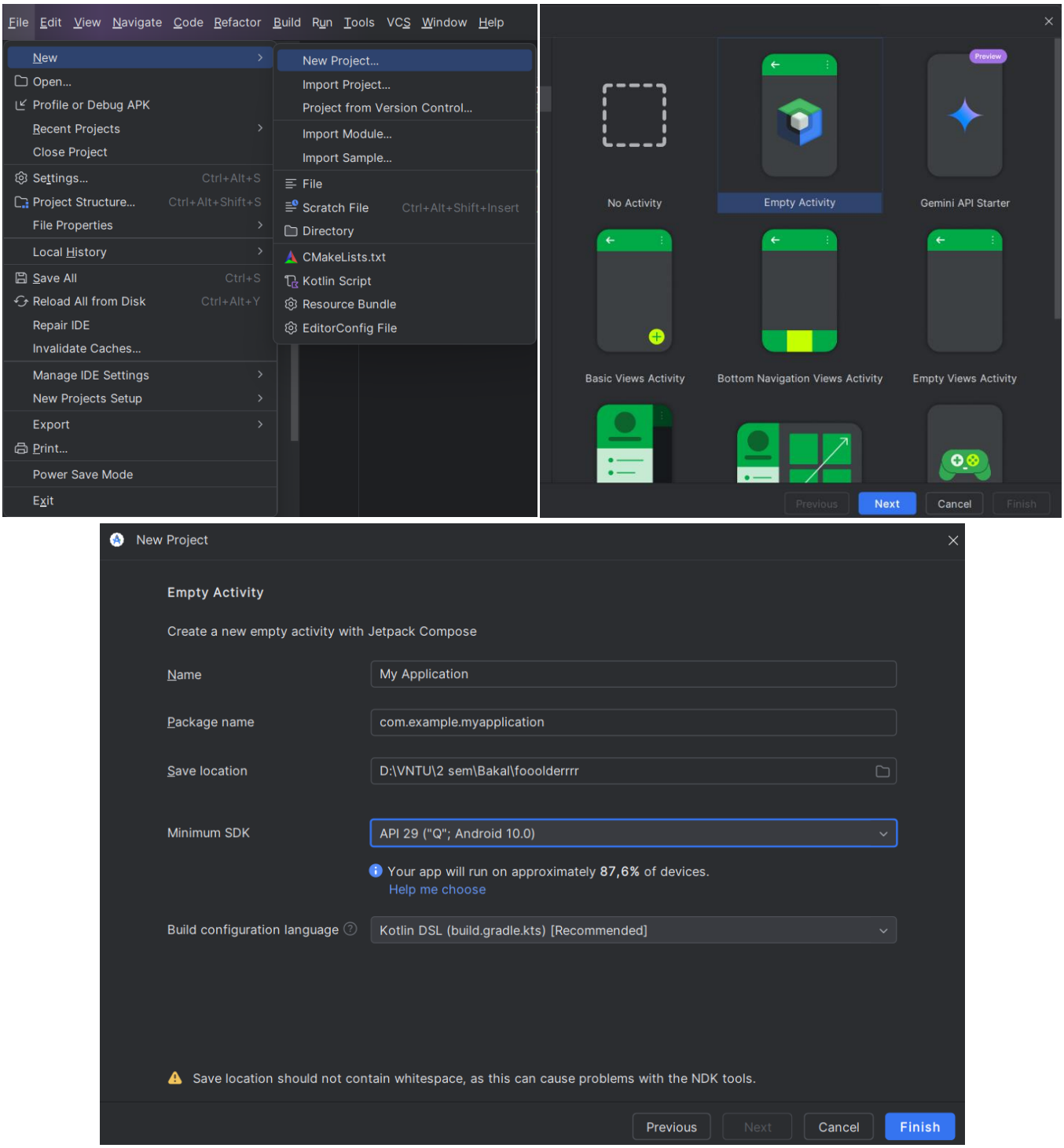

Рисунок 3.1 — Створення нового проєкту в Android Studio

Плагін для обробки анотацій («apply plugin: 'kotlin-kapt'») дозволяє Room обробляти анотації та автоматично генерувати необхідний код (наприклад, реалізації інтерфейсів DAO) під час збирання проєкту.

3.2 Серверна частина

Серверна частина мобільного додатку реалізована з використанням хмарного сервісу Firebase, який виконує функції централізованого зберігання облікових записів, авторизації користувачів, а також може використовуватись для зберігання та синхронізації даних між пристроями. Firebase надає набір готових інструментів для реалізації бекенду без необхідності самостійного розгортання серверів. Це дозволяє зосередитися на розробці функціональності клієнтської частини, спираючись на надійний і масштабований хмарний сервіс.

У межах роботи основна роль Firebase полягає в обробці запитів на авторизацію та зберіганні облікових записів користувачів. Користувачі можуть зареєструватися або увійти в додаток за допомогою логіна та пароля, які обробляються через Firebase Authentication. Після успішної авторизації сервер повертає унікальний токен доступу, який зберігається локально на пристрої. Усі подальші запити від користувача супроводжуються цим токеном, що дозволяє забезпечити захист даних та контроль доступу (лістинг 3.2).

Лістинг 3.2 — Реалізація екрана входу користувача

```
@Composable
fun SignInScreen(
    onAuthNavigationEvent: (NavigationEvent) -> Unit,
    viewModel: SignInViewModel = koinViewModel()
) {
    val context = LocalContext.current
    LaunchedEffect(Unit) {
        viewModel.events.collect {
            when (it) {
                SignInEvent.SignInFailed -> {
                    Toast.makeText(context, "Sign in failed",
Toast.LENGTH_SHORT).show() } } } }
    val uiState by viewModel.uiState.collectAsStateWithLifecycle()
    if (uiState.isLoading) {
        ContentLoader()
    } else {      Content(
            screenMode = uiState.screenMode,
            email = uiState.email,
            password = uiState.password,
            isNextButtonEnabled = uiState.isNextButtonEnabled,
```

```
onEvent = viewModel::onEvent,
onAuthNavigationEvent = onAuthNavigationEvent) }}
```

Крім цього, Firebase підтримує можливість зберігання резервних копій даних у Firestore або Realtime Database, що може бути використано для синхронізації даних між різними пристроями користувача або для відновлення інформації після перевстановлення додатку.

Для локального зберігання даних у додатку використовується Room Database, яка працює як клієнтська частина сховища і забезпечує швидкий доступ до важливої інформації без необхідності постійного підключення до інтернету. Room виступає обгорткою над SQLite та дозволяє зручно працювати з базою даних за допомогою анотацій та об'єктів Kotlin. У цьому проєкті Room використовується для збереження:

— створених користувачем завдань;

— записів таймера, які надалі використовуються для побудови діаграм активності.

Room і Firebase у додатку мають чітко розмежовані функції: Firebase відповідає за обробку облікових даних та керування сесіями користувачів, тоді як Room забезпечує зберігання локальної інформації про завдання та часові записи, необхідні для побудови діаграм та роботи додатку в офлайн-режимі.

3.3 Клієнтська частина: інтерфейс користувача та функціональні компоненти

Клієнтська частина програмної системи виконує функцію інтерактивної взаємодії кінцевого користувача з функціональними можливостями додатку. Вона слугує візуальним представленням логіки роботи застосунку та забезпечує доступ до основних його сервісів. Реалізація клієнтської частини передбачає розробку інтерфейсів для відображення даних, обробки подій, навігації між екранами та взаємодії з локальними і віддаленими ресурсами.

У контексті розробленої системи було реалізовано кілька ключових екранів, кожен з яких відповідає за окремий сегмент функціональності:

керування навчальними завданнями, календарне планування, запуск і завершення навчальних сесій, перегляд статистики, а також зміну користувацьких налаштувань. У подальших розділах наведено опис реалізації основних елементів клієнтського інтерфейсу та їх функціональних компонентів.

Екран завдань призначений для відображення списку завдань, управління обраними завданнями та їх видалення. Він складається з двох основних компонентів — списку завдань (LazyColumn) та панелі інструментів (TopBar). Користувач може переглядати всі завдання, обирати обрані (із зірочкою) та видаляти їх через спливаюче вікно підтвердження.

На початку екран отримує дані про завдання з viewModel, зокрема список завдань (tasks) та поточний режим екрану (screenMode). Залежно від режиму, екран може перебувати у двох станах:

— default — стандартний режим перегляду завдань;

— delete — режим видалення завдань, коли користувач обирає завдання для видалення.

Вміст екрану формується у функції Content(). Тут відображається TopBar, яка змінює свій вміст залежно від поточного режиму. У режимі Default панель містить кнопку для повернення на головний екран та кнопку для створення нового завдання. У режимі Delete замість цих кнопок відображається кількість обраних для видалення завдань та кнопка для підтвердження видалення (лістинг 3.3).

Лістинг 3.3 — Формування набору іконок для TopBar

```
val topBarIcons by remember(screenMode) {
    val icons = when (screenMode) {
        AllTasksScreenMode.Default -> setOf(
            TopBarIcon.Back { navigateTo(NavigationEvent.Back) },
            TopBarIcon.Add { navigateTo(NavigationEvent.NewTask) }
        )
        is AllTasksScreenMode.Delete -> setOf(
            TopBarIcon.Cancel{
onEvent(AllTasksScreenEvent.ClearSelectedTasks) },
            TopBarIcon.Delete { showDeleteBottomSheet() }
        )
```

```
    }
    mutableStateOf(icons)
}
```

Список завдань реалізовано через LazyColumn, що дозволяє відображати завдання динамічно. Кожне завдання представлене у вигляді TaskCard, яка може мати різний режим:

— favorite — завдання позначене як обране;

— delete — завдання обране для видалення.

Користувач може натиснути на завдання для його перегляду або утримувати, щоб обрати для видалення. Якщо у поточному режимі є обрані завдання, відкривається спливаюче вікно DeleteConfirmationBottomSheet, у якому користувач може підтвердити або скасувати видалення.

Екран календаря дозволяє користувачу переглядати завдання за днями, обирати дати та переходити до списку завдань для обраного дня. Основний компонент цього екрану — це календар, який візуально відображає місяць із позначеними датами, на які призначено завдання.

На початку екран перевіряє стан завантаження даних. Якщо дані ще не завантажені, відображається ContentLoader(). Після завантаження виконується відображення основного контенту через функцію Content() (лістинг 3.4).

Лістинг 3.4 – Реалізація екрану календаря з урахуванням стану даних

```
if (state.isLoading) {
    ContentLoader()
} else {
    Content(
        appLanguage = state.appLanguage,
        calendarData = state.calendarData,
        selectedDay = state.selectedDay,
        dataList = state.dataList,
        showYearMonthPickerDialog = { showYearMonthPicker = true },
        onEvent = viewModel::onEvent,
        navigateTo = onNavigationEvent)
}
```

Календар дозволяє користувачу вибрати день. Якщо на обрану дату призначено завдання, у нижній частині екрана з'являється список завдань за цей день. Список реалізовано через LazyColumn, що дозволяє динамічно відображати завдання із можливістю прокручування. Якщо на обрану дату завдань немає, замість списку відображається EmptyTaskCard() з повідомленням про відсутність завдань.

Користувач також може змінити місяць, натиснувши на назву поточного місяця. Це відкриває діалогове вікно YearMonthPickerDialog, у якому можна обрати інший місяць і рік. Вибір місяця зберігається та застосовується до календаря.

Панель інструментів (TopBar) включає кнопку для повернення на головний екран та кнопку для редагування завдань. Якщо обрано конкретний день, кнопка редагування стає активною і дозволяє перейти до детальної інформації про завдання на цю дату.

Екран таймера реалізує функціонал для запуску навчальної сесії, налаштування часу, вибору звуку сповіщення та перегляду залишкового часу. Основний елемент екрану — таймер, який відображає час у форматі 00:00. Користувач може вибрати час вручну або скористатися готовими шаблонами, наприклад, 15 чи 45 хвилин.

На початку екран перевіряє поточний стан таймера: чи є активна сесія, чи таймер завершився або ще не розпочався Це відображено у трьох основних станах: setup — налаштування таймера, ongoing — активний таймер із кнопками для паузи та скасування, finished — завершений таймер із повідомленням про закінчення часу (лістинг 3.5).

Лістинг 3.5 – Визначення станів екрану таймера та відображення відповідного контенту

```
when (state) {
    is TimerInfoScreenState.Finished -> FinishedContent(
        totalTime = state.totalTime,
        onEvent = viewModel::onEvent)
    is TimerInfoScreenState.Setup -> SetupContent(
```

```
            mode = state.mode,
            timerSound = state.currentTimerSound,
            onEvent = viewModel::onEvent)
        is TimerInfoScreenState.Ongoing -> OngoingContent(
            timerDisplayData = state.timerDisplayData,
            onEvent = viewModel::onEvent)}
```

При запуску таймера користувач може обрати звук сповіщення. Для цього на екрані є три кнопки: ліва кнопка відкриває список доступних звуків, центральна кнопка запускає таймер, права кнопка відкриває список шаблонів часу.

Під час активного таймера користувач бачить кругову шкалу з візуалізацією часу, що залишився, та кнопки для паузи або скасування. Таймер працює у фоні навіть після виходу з екрану — за це відповідає TimerService, який активується при ON_PAUSE.

Після завершення часу відкривається спливаюче вікно із загальною тривалістю сесії та кнопкою для вимкнення звуку. У цей момент відтворюється обраний звук, який можна зупинити або дочекатися його автоматичного завершення. Таймер не пов’язаний із завданнями — він просто фіксує тривалість сесії, яка зберігається для подальшого аналізу активності.

Екран налаштувань відповідає за надання користувачу можливості змінити налаштування додатку. У даному випадку, користувач може обрати мову інтерфейсу зі списку доступних мов. Вибір мови реалізовано за допомогою компонента OptionsBottomSheet, який викликається з елемента списку налаштувань.

Цей екран побудований за принципами Jetpack Compose і використовує архітектуру MVVM, де SettingsViewModel містить логіку обробки подій і зберігає поточний стан екрану (лістинг 3.6).

Лістинг 3.6 – Формування даних для вибору мови

```
@Composable
fun SettingsScreen(
    onNavigationEvent: (NavigationEvent) -> Unit,
    viewModel: SettingsViewModel = koinViewModel()
) {
```

```
    val currentLanguage by
viewModel.currentLanguage.collectAsStateWithLifecycle()
    var selectLanguageVisible by remember { mutableStateOf(false) }
    val selectLanguageData = AppLanguage.values().map { language ->
        OptionsBottomSheetButtonData(
            titleRes = language.titleRes,
            onClick = {
                selectLanguageVisible = false
viewModel.onEvent(SettingsScreenEvent.OnLanguageSelected(language))
            },
            isSelected = currentLanguage == language)
    }
```

У лістингу 3.7 наведено реалізацію вмісту екрана профілю користувача у Jetpack Compose — сучасному підході до створення інтерфейсів в Android. У цьому компоненті Content використовується декларативний підхід до побудови інтерфейсу, при якому структура та логіка відображення UI залежать від переданих параметрів і стану програми.

Головний контейнер Column розміщує всі елементи вертикально, центруючи їх по горизонталі. Оформлення задається за допомогою модифікаторів background, fillMaxSize та padding, що дозволяє легко адаптувати макет до різних екранів. Зовнішній вигляд враховує кольорову схему теми (MaterialTheme.colorScheme.surface), що гарантує візуальну узгодженість з іншими частинами застосунку.

У верхній частині розміщено компонент TopBar, який містить іконку для навігації назад. Він є частиною загального підходу до побудови екранів застосунку, де кожен з них має власну верхню панель з відповідними діями.

Під панеллю відображається аватар користувача (ProfilePicture) з тінню для візуального виділення, що додає глибини інтерфейсу. За ним розміщено блок кнопок, які реалізовані у вигляді компонентів ProfileButton. Кожна кнопка відповідає окремій функції: зміна пароля, зміна фото профілю, вихід з облікового запису.

Між кнопками застосовано візуальні роздільники (Box з фоновим кольором), які надають інтерфейсу чіткої структури. Колір і текст кнопок можна

змінювати, що видно у прикладі з кнопкою "Вийти", де текст має червоний відтінок (Wewak) для підкреслення ризику дії.

Усі взаємодії реалізовані через передані в Content функції (navigateTo, showSignOutDialog, showChangeProfilePictureBottomSheet), що забезпечує гнучкість, перевикористання компонента та тестованість. Такий підхід відповідає принципам розділення відповідальностей та односпрямованого потоку даних (unidirectional data flow), які є основою архітектури в Jetpack Compose.

Лістинг 3.7 — Вміст екрану профілю користувача

```
@Composable
private fun Content(
   profilePicture: ProfilePictureData,
   showSignOutDialog: () -> Unit = {},
   showChangeProfilePictureBottomSheet: () -> Unit = {},
   navigateTo: (NavigationEvent) -> Unit = {})
{    Column(
       modifier = Modifier
          .background(MaterialTheme.colorScheme.surface)
          .fillMaxSize(),
       horizontalAlignment = Alignment.CenterHorizontally)
{  TopBar(
          data = TopBarData.Profile,
          icons = setOf(
             TopBarIcon.Back { navigateTo(NavigationEvent.Back) }  )   )
       Spacer(Modifier.height(24.dp))
       ProfilePicture(
          data = profilePicture,
          modifier = Modifier.shadow(
             elevation = 12.dp,
             shape = MaterialTheme.shapes.small ))
```

При завантаженні фото для профілю може виникнути проблема з його правильним відображенням — зображення може бути надто великим або мати некоректну орієнтацію. Це пов’язано з тим, що багато камер додають інформацію про орієнтацію зображення у метадані (EXIF), але не змінюють саме зображення. Для вирішення цієї проблеми у застосунку реалізовано спеціальний

клас ImageResizeManager, який відповідає за попередню обробку зображень профілю.

Основна функція класу resizeImage виконує кілька ключових операцій:

— зчитує зображення за наданим URI;

— аналізує метадані EXIF для визначення потрібного кута повороту;

— обертає зображення у правильну орієнтацію;

— змінює розмір зображення до заданого обмеження (наприклад, 512 пікселів по ширині або висоті), зберігаючи пропорції;

— стискає його до формату JPEG із середньою якістю (85%) та повертає у вигляді байтового масиву.

Окремий метод rotateBitmapIfRequired використовує клас Matrix для повороту зображення відповідно до коду орієнтації, а метод bitmapToByteArray перетворює оброблене зображення у формат, зручний для збереження або передавання.

Всі операції виконуються у фоновому потоці (Dispatchers.IO), щоб уникнути блокування головного інтерфейсу під час роботи з великими файлами. Такий підхід відповідає кращим практикам розробки Android-додатків та забезпечує плавну роботу застосунку навіть на слабких пристроях.

У результаті клас ImageResizeManager гарантує, що фотографії профілю будуть відображені коректно, мати оптимальний розмір і не призводитимуть до помилок або надмірного використання ресурсів системи (лісттинг Д.3).

3.4 База даних: структура, моделі даних та взаємозв’язки

База даних у запропонованій системі відіграє ключову роль у забезпеченні збереження та обробки користувацьких даних, навчальних завдань, сесій таймера, а також інформації про їх виконання. Для реалізації сховища даних було обрано Room Database — сучасну обгортку над SQLite, яка входить до складу архітектурних компонентів Jetpack. Її використання забезпечує зручну інтеграцію з життєвим циклом компонентів Android-застосунку, підтримку

реактивного підходу до зчитування даних та безпечну типізацію SQL-запитів за допомогою анотацій.

У структурі бази даних передбачено декілька основних сутностей, кожна з яких відповідає окремому аспекту функціональності системи. Зокрема, сутність User призначена для зберігання облікової інформації користувача — ідентифікатора, імені, паролю, обраної мови інтерфейсу та зображення профілю. Сутність Task представляє навчальні завдання, що створюються користувачем. Вона містить такі поля, як унікальний ідентифікатор завдання, його назву, позначку «обране», а також дату створення.

Для прив’язки завдань до конкретної дати в календарі використовується окрема сутність CalendarEntry, яка зберігає ідентифікатор завдання, обрану дату та статус виконання завдання (виконано/не виконано). Такий підхід дозволяє досягти гнучкості у плануванні, оскільки одне завдання може бути закріплене за кількома датами. Окрім того, у базі передбачено сутність StudySession, яка фіксує кожен завершений таймер — із збереженням дати, тривалості, часу початку й завершення сесії.

Між вказаними сутностями існують логічні взаємозв’язки. Так, між таблицями Task та CalendarEntry реалізовано відношення типу «один до багатьох»: кожне завдання може мати кілька записів у календарі. Подібно, кожен запис StudySession може бути опосередковано пов’язаний з користувачем, хоча без прямого зовнішнього ключа (якщо обліковий запис один). Усі зв’язки задаються через поля зовнішніх ключів або обробляються у вигляді вкладених запитів через @Relation у Room.

Для доступу до даних у системі використовується набір DAO (Data Access Object) інтерфейсів, які визначають базові операції над таблицями: додавання нових записів (@Insert), оновлення (@Update), видалення (@Delete) та запити з вибіркою (@Query). Кожна DAO-структура забезпечує інкапсульований доступ до відповідної таблиці, дозволяючи здійснювати як прості, так і складні операції, зокрема фільтрацію за датою, статусом виконання чи ідентифікатором користувача.

Усі запити реалізовані з урахуванням принципів реактивного програмування. Для потоків даних застосовується механізм Flow, що дозволяє автоматично оновлювати інтерфейс при зміні інформації в базі. Це особливо важливо для компонентів, які мають динамічний характер — таких як список завдань або діаграма навчальної активності, що повинні миттєво реагувати на будь-які зміни у даних.

3.5 Тестування та оптимізація підсистеми

Тестування функціональності мобільного застосунку здійснювалося вручну, з метою перевірки стабільності роботи клієнтської частини, коректної взаємодії з базою даних (Room Database), а також належної роботи окремих підсистем: управління завданнями, таймера, календаря та облікового запису користувача. Основний акцент було зроблено на перевірку логіки обробки подій, збереження даних та реактивного оновлення інтерфейсу.

Тестування проводилося без використання автоматизованих тестових фреймворків, оскільки на етапі дослідної реалізації було достатньо емпіричного підходу, орієнтованого на виявлення потенційних помилок у сценаріях взаємодії користувача з інтерфейсом.

Для перевірки працездатності додатку використовувався фізичний мобільний пристрій Redmi Note 7, що працює під керуванням операційної системи Android 10. Вибір даного пристрою був зумовлений тим, що він належить до сегменту масового користування, а отже дозволяє змоделювати реальні умови експлуатації застосунку. Крім того, його апаратні характеристики є достатніми для перевірки продуктивності інтерфейсів та відгуку на користувацькі дії без впливу надлишкової потужності пристрою (що могло б приховати проблеми з оптимізацією).

В ході тестування перевірялося:

— коректне створення, редагування та видалення навчальних завдань;

— правильна прив’язка завдань до дати у календарі, а також зміна статусу їх виконання;

— стабільна робота таймера з обраним звуковим сигналом, правильне зарахування часу навчальної сесії лише у разі її завершення;

— виведення статистики навчання у вигляді діаграми, з правильним групуванням сесій за датами;

— збереження даних між сесіями додатку та після перезапуску;

— зміна мови інтерфейсу та оновлення відображення після перемикання;

— оновлення аватарки профілю та обробка зміни пароля.

Усі функціональні компоненти показали стабільну роботу під час взаємодії з користувачем. За результатами тестування не було зафіксовано критичних помилок, однак було виявлено кілька дрібних недоліків, які були усунені (наприклад, некоректне скидання вибраного часу після скасування таймера, а також потреба в уточненні стану при відображенні завершених сесій).

На головному екрані додатку знаходиться діаграма, яка відображає загальний обсяг витраченого на навчання часу. У ході тестування перевірялась актуальність даних, оновлення після завершення кожної нової сесії, а також правильна агрегація часу по датах. Було підтверджено, що додаток фіксує лише повністю завершені таймери, як і передбачено вимогами (рис. 3.2).

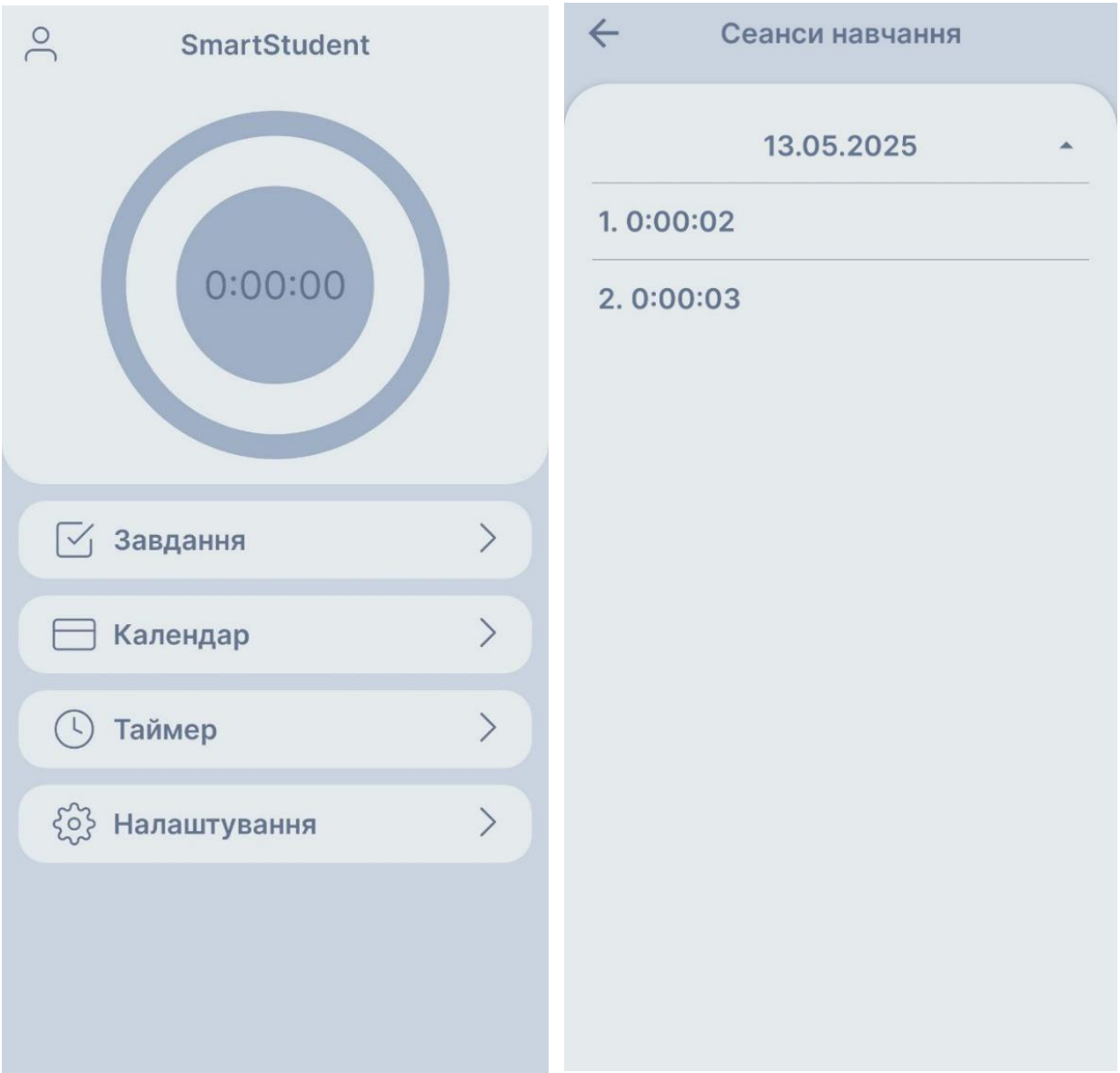

Рис. 3.2 – Головний екран додатку та екран зі списком всіх сесій

На рисунку Г.1 зображено екран, що відповідає за створення та відображення навчальних завдань. У ході тестування було перевірено роботу кнопки створення нового завдання, а також функціонал позначення обраного (зірочка) та перемикання в режим видалення. Усі дії виконувалися без затримок,

а зміни стану завдань відображалися у реальному часі, завдяки використанню зв’язку з Room Database.

На рис. 3.3 зображено екран з календарем. При призначенні завдання на певний день, він відображається іншим, рожевим, кольором. Натиснувши на дату, можна побачити список завдань, що були призначені на цей день. Ці завдання можна позначати як виконані.

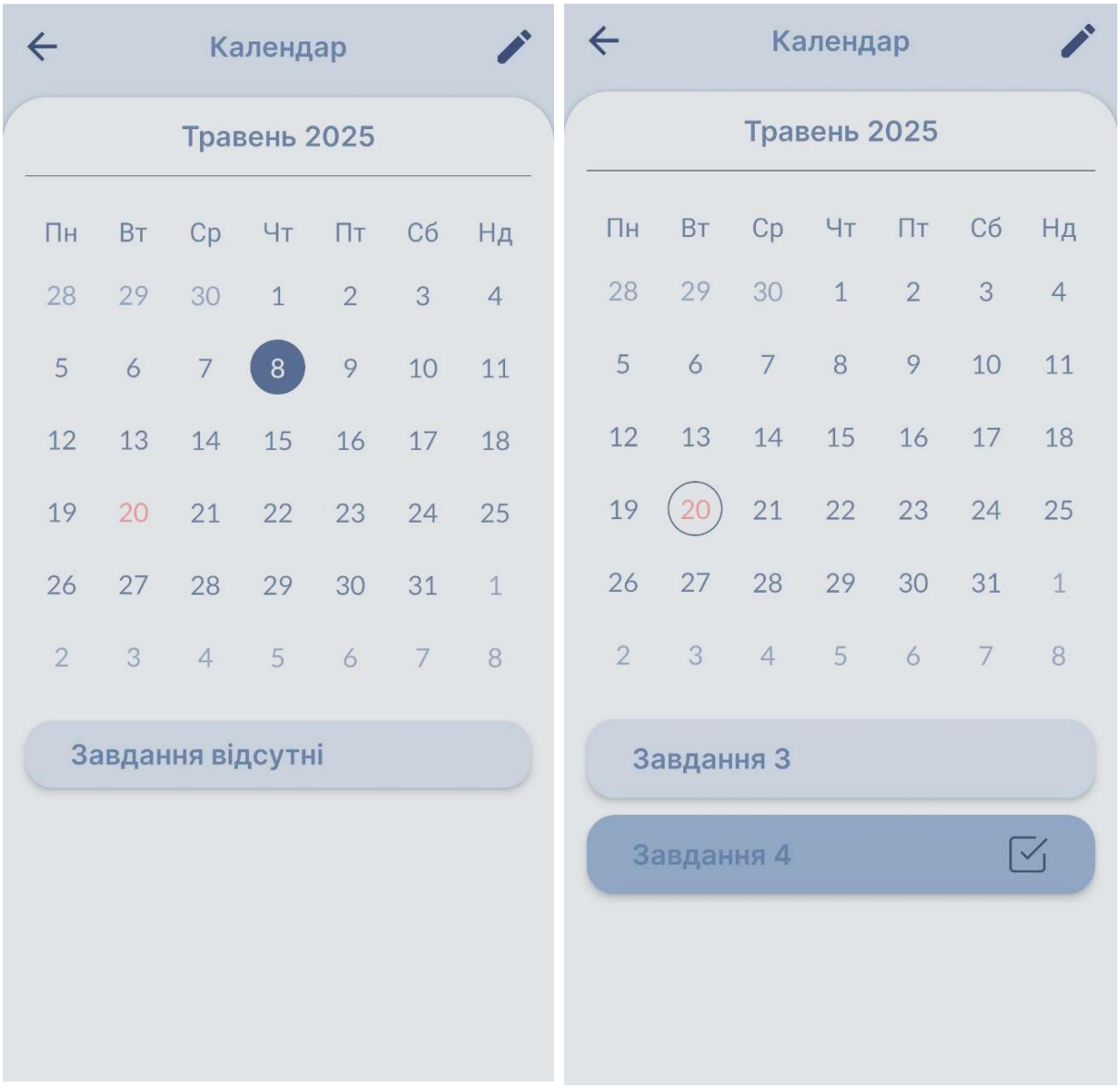

Рис. 3.3 – Вибір дати в календарі та список завдань, прив’язаних до цієї дати

На рисунку 3.4 зображено процес запуску таймера: користувач обирає тривалість сесії вручну або через шаблон, обирає мелодію сповіщення, після чого запускає відлік часу. Було перевірено запуск, паузу, відновлення та завершення

таймера, а також роботу сервісу у фоновому режимі. Система правильно зберігає інформацію про завершену сесію лише в тому випадку, якщо таймер завершено коректно.

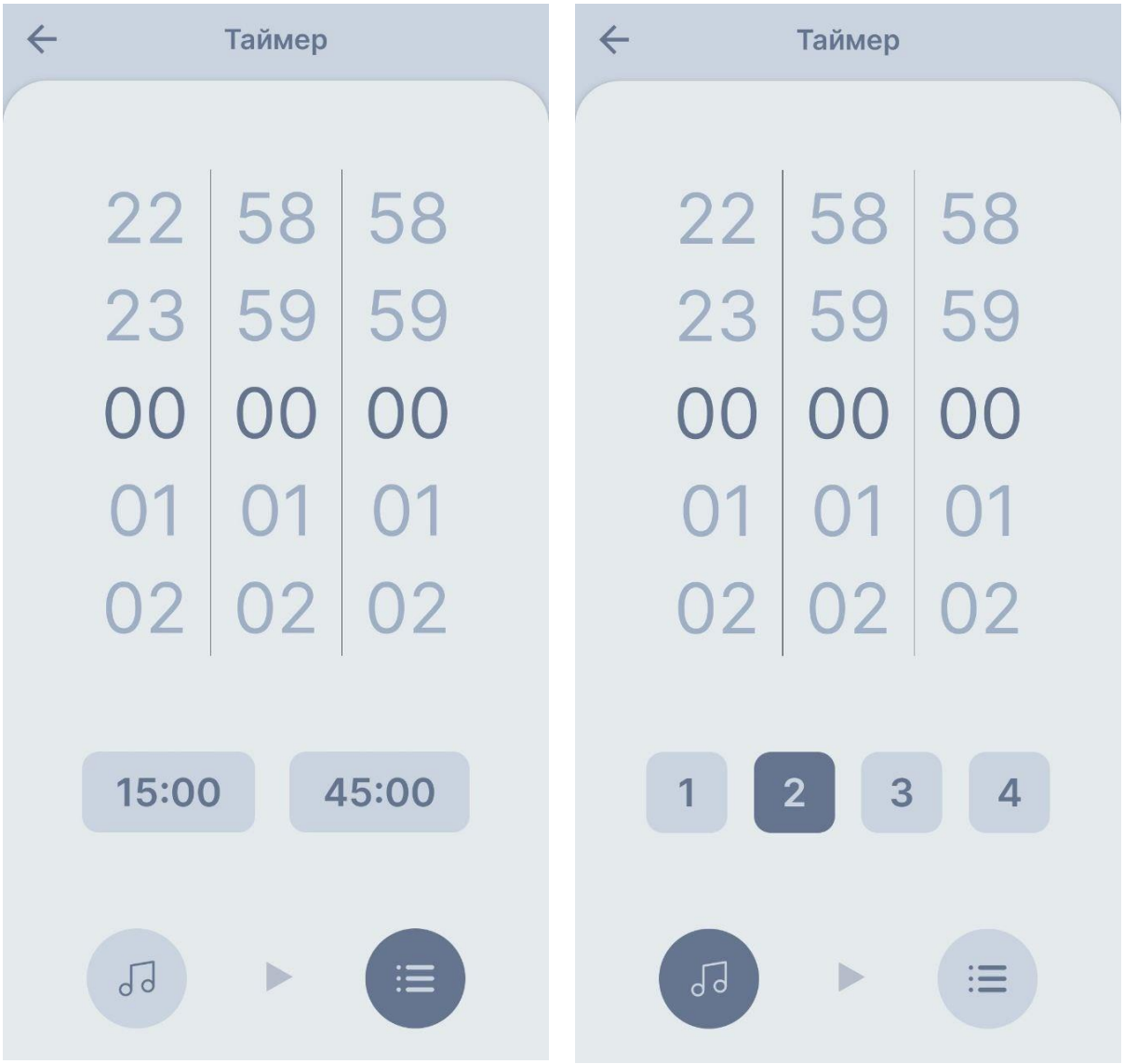

Рисунок 3.4 — Налаштування таймеру, шаблони та налаштування мелодії

У додатку можна змінювати мову інтерфейсу через екран налаштувань. Була протестована робота OptionsBottomSheet із двома доступними мовами — українською та англійською. Крім того, перевірено зміну аватарки, оновлення паролю та вихід з облікового запису (рис. 3.5). Усі операції відбуваються з відповідною реакцією інтерфейсу без зависань або помилок.

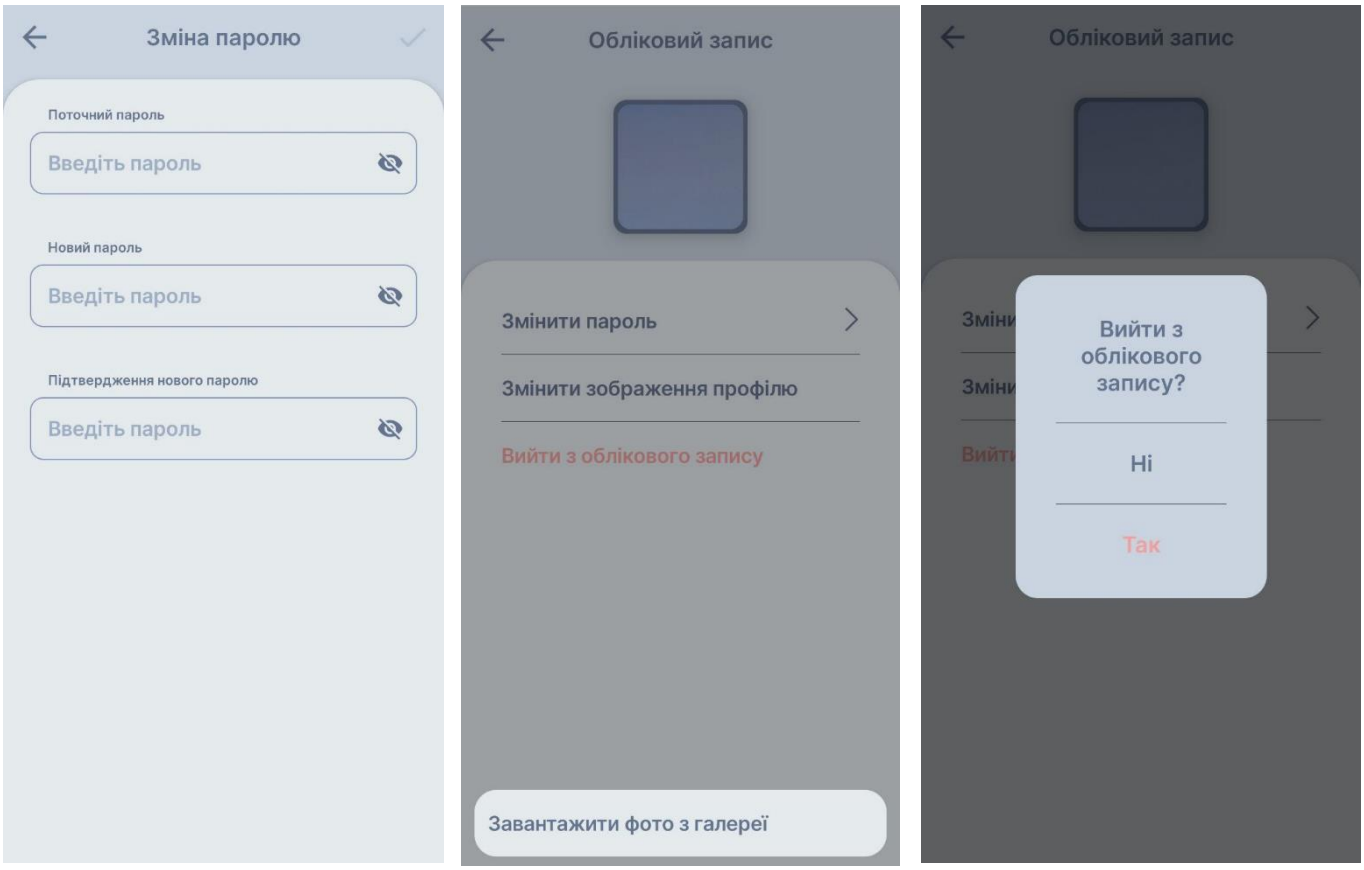

Рисунок 3.5 – Налаштування облікового запису: зміна паролю, завантаження фото та вихід з облікового запису

Проведене тестування на фізичному пристрої засвідчило стабільну роботу всіх основних компонентів клієнтської частини застосунку. Інтерфейс реагує на дії користувача без затримок, а база даних коректно обробляє збереження та оновлення інформації. Завдяки використанню реактивних потоків (Flow), усі зміни миттєво відображаються у відповідних елементах інтерфейсу. Результати тестування підтверджують функціональну готовність додатку до використання в реальних умовах та подальшого вдосконалення.

ВИСНОВКИ

У бакалаврській роботі було розроблено мобільний додаток, який допомагає студентам планувати навчання, контролювати витрачений час і аналізувати власну продуктивність. Ідея створення такого додатку виникла на основі аналізу існуючих сервісів, які або мають занадто вузький функціонал, або орієнтовані на інші цілі. Було вирішено поєднати календар завдань, таймер для навчання та систему збору статистики в одному зручному інтерфейсі.

Перед початком розробки були визначені ключові функції, які мав би мати додаток. Це можливість створювати завдання, прив’язувати їх до конкретної дати, відзначати виконання, запускати таймер сесій і переглядати результати у вигляді діаграми. Усе це реалізовано у вигляді окремих екранів, між якими легко перемикатися з головного меню. Інтерфейс зроблений простим, щоб ним було зручно користуватись без зайвих пояснень.

Для зберігання даних використовується Room Database. У ній зберігаються всі завдання, сесії таймера, облікові записи та інші дані. Додаток не залежить від підключення до інтернету, що дозволяє користуватись ним навіть офлайн. Серверна частина реалізована з використанням Firebase — вона відповідає за реєстрацію, авторизацію і збереження облікових даних.

Робота додатку перевірялась на телефоні Redmi Note 7 з Android 10. Під час тестування були перевірені всі основні функції: створення завдань, редагування, запуск таймера, завершення сесій, перегляд статистики та перемикання мови. Усі екрани працювали стабільно, збереження даних відбувалося без помилок. Якщо користувач завершує сесію правильно, дані потрапляють у діаграму, якщо перериває — ні, як і задумано. Виправлення дрібних багів під час перевірки дозволили покращити загальну стабільність додатку.

У результаті було реалізовано повноцінний мобільний застосунок, який відповідає початковим вимогам. Він дозволяє студентам краще організовувати своє навчання, бачити прогрес і використовувати час ефективніше. Додаток

можна використовувати як у рамках особистого тайм-менеджменту, так і як частину більшого освітнього інструменту. У майбутньому його можна розширити — додати синхронізацію з іншими сервісами, нові типи статистики або систему нагадувань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сущенко Л. П. Організація освітнього процесу у закладах загальної середньої освіти. Київ: ЦУЛ, 2020.

2. О.В. Шевчук. Інформаційні технології в освіті: навч. посіб. Київ: КНТ, 2020.

3. D. Randy Garrison. E-Learning in the 21st Century: A Framework for Research and Practice. Routledge, 2017.

4. Биков В. Ю., Шишкіна М. П. Хмаро орієнтоване навчальне середовище: методологія та досвід проєктування. К.: Педагогічна думка, 2018.

5. Сергієнко О. М. Планування навчального процесу в умовах цифрової трансформації освіти. ІТЗН, 2021, №6.

6. Теплицький І. О. Інформаційні технології в навчальному процесі: організація, ефективність, проблеми. Харків: Основа, 2020.

7. Dabbish, L., Mark, G., & González, V. M. (2011). Why do I keep interrupting myself? Environment, habit and self-interruption.

8. Коваль Т. В. Мобільне навчання в системі вищої освіти: переваги, ризики, перспективи розвитку. Інформаційні технології і засоби навчання, 2021, №1.

9.Siemens, G. (2005). Connectivism: A Learning Theory for the Digital Age. International Journal of Instructional Technology and Distance Learning.

10. А.М. Коломоєць. Цифрові технології в освіті: теорія і практика використання Харків: НТУ «ХПІ», 2021.

11. M. S. Raisinghani et al. Mobile Learning: Challenges, Opportunities, and Trends. IGI Global, 2019.

12. Bennett, M. Best Productivity Apps for Students in 2022.RescueTime Blog. 2022.

13. Гуменюк І. М. Цифрові інструменти організації навчання: з досвіду вчителів та студентів. Освітній простір України, 2022, №48.

14. Chen, B. X. Apps to Keep You Off Your Phone, So You Can Learn or Work.The New York Times. 2020.

15. Traxler, J. Defining, Discussing, and Evaluating Mobile Learning: The moving finger writes and having writ... The International Review of Research in Open and Distributed Learning, 2007.

16. Kukulska-Hulme, A. Mobile Learning and Student Autonomy in Higher Education. Research in Learning Technology, 2012.

17. Meier, A., Reinecke, L., & Meltzer, C. E. "Facebocrastination"? Predictors of using Facebook for procrastination and its effects on students' well-being. Computers in Human Behavior, 2016, p. 65–76.

18. Sharples, M. et al. A Theory of Learning for the Mobile Age // In R. Andrews & C. Haythornthwaite (Eds.), The SAGE Handbook of E-learning Research.

19. Bernard, R. M., Borokhovski, E., Tamim, R. M. et al. A meta-analysis of blended learning and technology use in higher education: From the general to the applied. Computers & Education, 2014.

20. Anderson, T., & Dron, J. Three Generations of Distance Education Pedagogy. The International Review of Research in Open and Distributed Learning, 2011.

21. Bower, M. (2017). Design of Technology-Enhanced Learning: Integrating Research and Practice. Emerald Publishing.

22. Siemens, G. (2005). Connectivism: A Learning Theory for the Digital Age. International Journal of Instructional Technology and Distance Learning.

23. Martin, F., & Sunley, R. (2021). Student Learning Analytics: An Evidence-Based Approach to Learning Monitoring. Educational Technology Research and Development.

ДОДАТОК А

Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
_____ 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
«Система моніторингу та аналізу результатів навчання студентів»
08-54.БКР.011.00.000 ТЗ

Науковий керівник: к.т.н., доцент кафедри ОТ
_____ Снігур А.В.

Студентка групи 1СП-21б
_____ Педосенко М.Ю.

ВНТУ 2025

1 Підстава для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Широке впровадження мобільних освітніх додатків і зростання популярності самостійного навчання обумовлює необхідність створення систем, що поєднують планування навчальної діяльності та моніторинг результатів. Студенти потребують інструментів, які допомагають структурувати завдання, обліковувати навчальний час і візуалізувати продуктивність.

1.2 Наказ про затвердження теми БКР від 11 березня 2025 року №82.

2 Мета БКР і призначення розробки

2.1 Мета роботи — розробка мобільного додатку для Android, який дозволяє студентам планувати завдання, відстежувати навчальний час та аналізувати результати у вигляді візуалізацій.

2.2 Призначення розробки — забезпечити студентів інструментом для організації навчального процесу з можливістю гнучкого планування, самоменеджменту та аналізу активності.

3 Вихідні дані для виконання БКР

3.1 Аналіз існуючих систем планування навчання та моніторингу продуктивності.

3.2 Формування функціональних вимог до додатку.

3.3 Проєктування архітектури підсистем (таймер, календар, діаграма активності).

3.4 Розробка інтерфейсу користувача та реалізація функціональності в Android Studio з Room Database.

3.5 Тестування додатку та усунення виявлених помилок.

4 Вимоги до виконання БКР

Додаток має бути зручним, стабільним, не втрачати дані після перезапуску, дозволяти керування завданнями, облік навчального часу та візуалізацію статистики. Інтерфейс має бути інтуїтивним і відповідати принципам мобільного UX-дизайну.

5 Етапи БКР та очікувані результати

Етапи роботи БКР та очікувані результати наведено у табл. А.1.

Таблиця А.1 — Етапи БКР

№	Назва етапу	Термін виконання	Очікуваний результат
1	Аналіз літературних джерел	21.03.2025 – 25.03.2025	Розділ 1
2	Розробка технічного завдання	26.03.2025 – 02.04.2025	Сформульоване ТЗ
3	Аналіз задачі та архітектури	03.04.2025 – 09.04.2025	Розділ 2
4	Програмна реалізація додатку	10.04.2025 – 24.04.2025	Розділ 3
5	Тестування та оптимізація додатку	25.04.2025 – 01.05.2025	Розділ 3
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	02.05.2025 – 10.05.2025	ПЗ, граф. матеріали та презентація
7	Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат	10.05.2025 – 13.05.2025	Готові документи

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні матеріали (скріншоти інтерфейсу, діаграми), презентація, відгук керівника, анотація, протокол попереднього захисту, довідка про перевірку на плагіат.

7 Порядок контролю виконання та захисту БКР

Контроль виконання здійснюється керівником згідно з календарним планом. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення БКР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп'ютерна інженерія» (освітня програма «Системне програмування»). Кафедра обчислювальної техніки ВНТУ 2023;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:
СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ РЕЗУЛЬТАТІВ НАВЧАННЯ СТУДЕНТІВ

Тип роботи: бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 99,5% Схожість 0,5%

Аналіз звіту подібності (відмітити потрібне):

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

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

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

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

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

Автор роботи _____ Педосенко М. Ю.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Снігур А.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічна частина

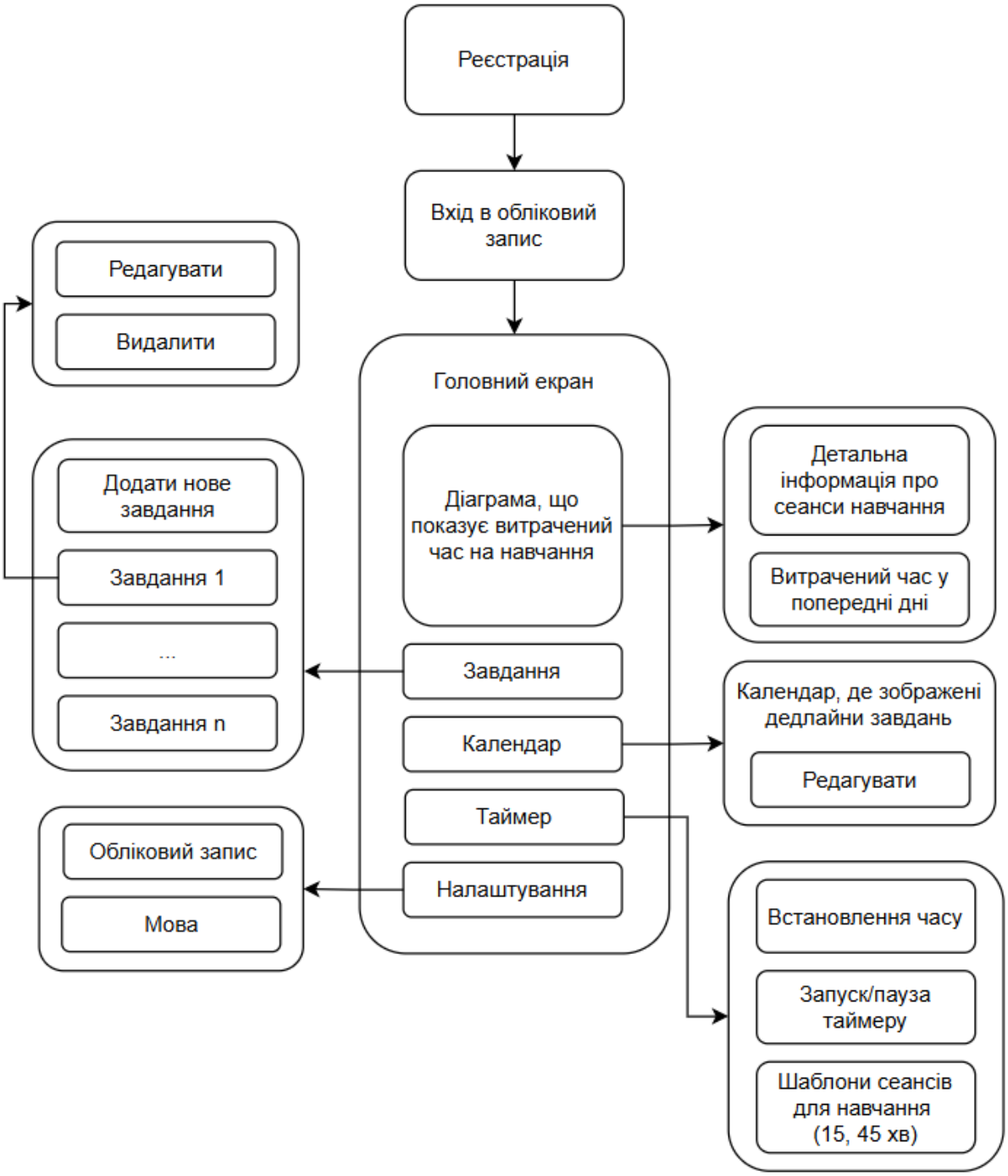

Рисунок В.1 — Структурна схема розробленого додатку

ДОДАТОК Г

Ілюстративна частина

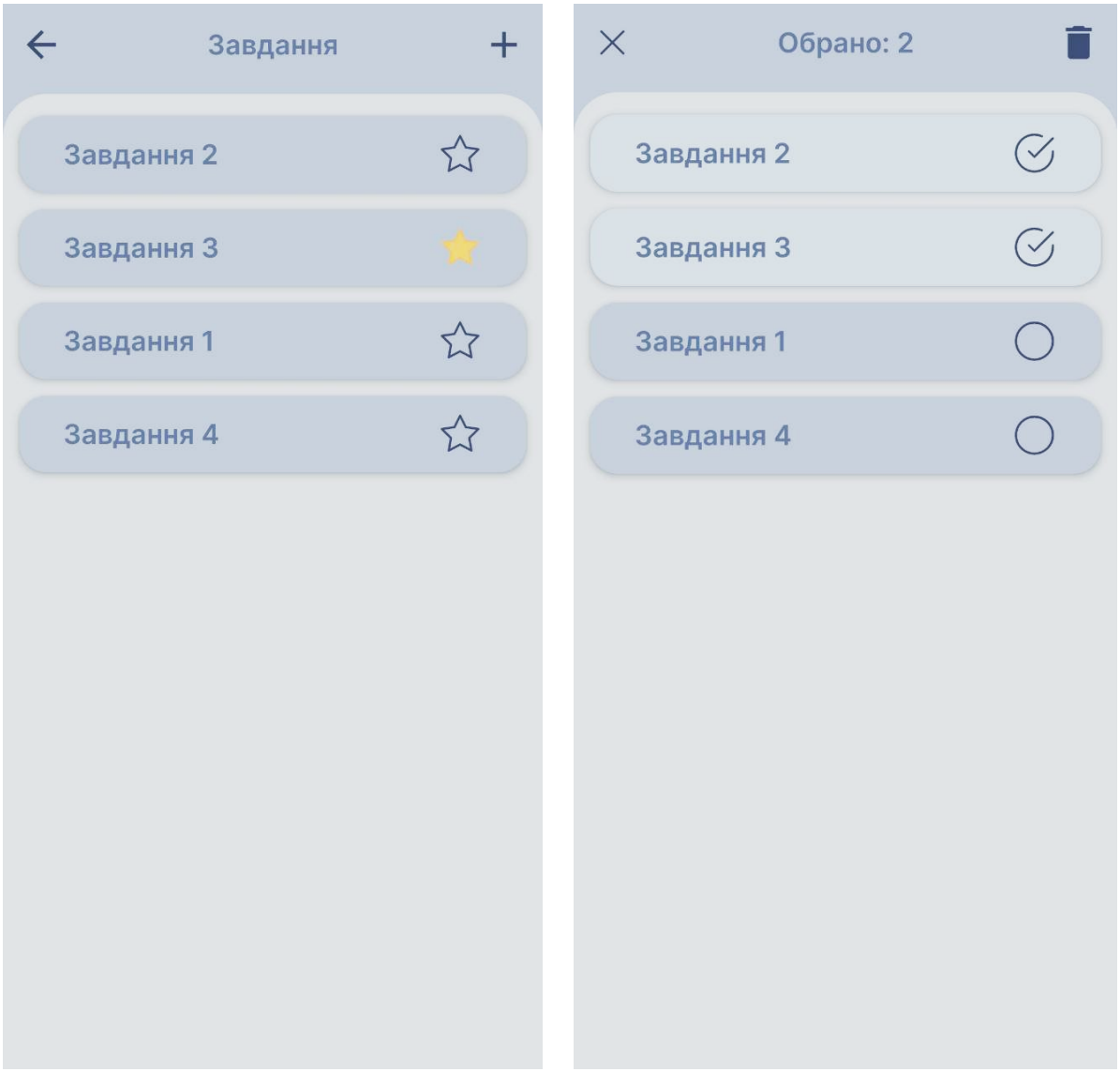

Рисунок Г.1 — Екран з завданнями, позначення обраних та виділення створених завдань

ДОДАТОК Д

Лістинги застосунку

Лістинг Д.1 — Екран з календарем

```
@Composable
fun CalendarScreen(
   viewModel: CalendarViewModel,
   onNavigationEvent: (NavigationEvent) -> Unit) {
   var showYearMonthPicker by remember { mutableStateOf(false) }
   val state by viewModel.uiState.collectAsStateWithLifecycle()
   if (state.isLoading) {       ContentLoader()    } else {
      Content(
         appLanguage = state.appLanguage,
         calendarData = state.calendarData,
         selectedDay = state.selectedDay,
         dataList = state.dataList,
         showYearMonthPickerDialog = { showYearMonthPicker = true },
         onEvent = viewModel::onEvent,
         navigateTo = onNavigationEvent)}
   if (showYearMonthPicker) {
      YearMonthPickerDialog(
         currentYearMonth = state.calendarData.month,
         appLanguage = state.appLanguage,
         onDismissRequest = {
            showYearMonthPicker = false
viewModel.onEvent(CalendarScreenEvent.OnYearMonthChange(it))})}}
@Composable
private fun Content(
   appLanguage: AppLanguage,
   calendarData: CalendarData,
   selectedDay: SelectedDayState,
   dataList: List<TaskCardData>,
   showYearMonthPickerDialog: () -> Unit = {},
   onEvent: (CalendarScreenEvent) -> Unit = {},
   navigateTo: (NavigationEvent) -> Unit = {},) {
   Column(
      modifier = Modifier
         .background(MaterialTheme.colorScheme.surface)
         .fillMaxSize()    ) {
      TopBar(
         data = TopBarData.Calendar,
         icons = setOf(
            TopBarIcon.Back { navigateTo(NavigationEvent.Back) },
```

```
                TopBarIcon.Edit(selectedDay is SelectedDayState.Selected) {
                    when (selectedDay) {
                        SelectedDayState.None -> {}
                        is SelectedDayState.Selected -> {
                            navigateTo(
avigationEvent.CalendarDayDetails(selectedDay.date.toMillis()))}}}))
            Spacer(Modifier.height(6.dp))
            ContentContainer {
                Column(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(16.dp),
                    horizontalAlignment = Alignment.CenterHorizontally ) {
                    Text(
                        text = calendarData.month.formatYearMonth(appLanguage),
                        modifier = Modifier.clickable { showYearMonthPickerDialog() },
                        color = MaterialTheme.colorScheme.onPrimary,
                        style = MaterialTheme.typography.titleMedium )
                    Spacer(Modifier.height(16.dp))
                    Box(
                        modifier = Modifier
                            .fillMaxWidth()
                            .height(0.5.dp)
                            .background(BrightGray))
                    Spacer(Modifier.height(20.dp))
                    Calendar(
                        appLanguage = appLanguage,
                        data = calendarData,
                        selectedDay = (selectedDay as? SelectedDayState.Selected)?.date,
                        onMonthChange = {
onEvent(CalendarScreenEvent.OnYearMonthChange(it)) },
                        onDayChange = {
onEvent(CalendarScreenEvent.OnDayChange(it)) } )
                    Spacer(Modifier.height(20.dp))
                    if (selectedDay is SelectedDayState.Selected) {
                        if (dataList.isEmpty()) {
                            EmptyTaskCard()
                        } else {
                            LazyColumn(
                                modifier = Modifier.fillMaxSize(),
                                verticalArrangement = Arrangement.spacedBy(12.dp)
                            ) {
                                items(dataList) { item ->
                                    TaskCard(
```

```
                                data = item,
                                onClick = {
navigateTo(NavigationEvent.CalendarTaskInfo(item.id)) }  ) }} }}}}}}
        @Composable
        private fun YearMonthPickerDialog(
            currentYearMonth: YearMonth,
            appLanguage: AppLanguage,
            onDismissRequest: (YearMonth) -> Unit = {},
        ) {
            val state = rememberYearMonthPickerState(
                initialYearMonth = currentYearMonth,
                appLanguage = appLanguage
            )
            Dialog(
                onDismissRequest = { onDismissRequest(state.selectedYearMonth) },
                content = {
                    YearMonthPicker(
                        modifier = Modifier.padding(horizontal = 16.dp),
                        state = state ) })}
        @Preview
        @Composable
        private fun CalendarScreenPreview() {
            SmartStudentTheme {
                val state = CalendarScreenState(
                    calendarData = CalendarData(
                        incompleteTaskDeadlines = setOf(
                            LocalDate.now(),
                            LocalDate.now().plusDays(15))),
                    dataList = listOf(
                        TaskCardData(
                            id = 1,
                            title = "Task 1",
                            mode = TaskCardMode.Completion(false)
                        ),
                        TaskCardData(
                            id = 2,
                            title = "Task 2",
                            mode = TaskCardMode.Completion(true)) ))
                Content(
                    appLanguage = state.appLanguage,
                    calendarData = state.calendarData,
                    selectedDay = state.selectedDay,
                    dataList = state.dataList )}}
```

Лістинг Д.2 — Екран з таймером

```
private val TimerServiceIntent: (Context) -> Intent = {
   Intent(it, TimerService::class.java)}
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TimerInfoScreen(
   viewModel: TimerInfoViewModel,
   onNavigationEvent: (NavigationEvent) -> Unit) {
   var mediaPlayer by remember { mutableStateOf<MediaPlayer?>(null) }
   val uiState by viewModel.uiState.collectAsStateWithLifecycle()
   LaunchedEffect(Unit) {
      viewModel.event.collect {
         when (it) {
            TimerInfoViewModelEvent.StopPlayingSound -> {
               mediaPlayer?.stopPlaying()
               mediaPlayer = null            }          }       }    }
   val context = LocalContext.current
   val lifecycleOwner = LocalLifecycleOwner.current
   LifecycleEventEffect(
      event = Lifecycle.Event.ON_RESUME,
      lifecycleOwner = lifecycleOwner    ) {
      context.stopService(TimerServiceIntent(context))    }
   LifecycleEventEffect(
      event = Lifecycle.Event.ON_PAUSE,
      lifecycleOwner = lifecycleOwner    ) {
      viewModel.onEvent(TimerInfoScreenEvent.SaveOrFinishTimer)
      if (uiState is TimerInfoScreenState.Ongoing) {
         ContextCompat.startForegroundService(
            context,
            TimerServiceIntent(context)          )       }
      if (uiState is TimerInfoScreenState.Finished) {
         viewModel.onEvent(TimerInfoScreenEvent.FinishTimer)}}
   Content(
      state = uiState,
      onEvent = viewModel::onEvent,
      onBackPressed = { onNavigationEvent(NavigationEvent.Back) }    )
   val finishBottomSheetState = rememberModalBottomSheetState(
      skipPartiallyExpanded = true    )
   if (uiState is TimerInfoScreenState.Finished) {
      val timerId = (uiState as TimerInfoScreenState.Finished).timerId
      LaunchedEffect(timerId) {
         val sound = (uiState as
TimerInfoScreenState.Finished).currentTimerSound?.resourceId
            ?: return@LaunchedEffect
```

```
                mediaPlayer?.stopPlaying()
                val newPlayer = MediaPlayer.create(context, sound).apply {
                    isLooping = true
                    setOnCompletionListener {
                        it.release()
                        mediaPlayer = null}}
                mediaPlayer = newPlayer
                newPlayer.start()}
            ModalBottomSheet(
                onDismissRequest                                =                {
viewModel.onEvent(TimerInfoScreenEvent.FinishTimer) },
                sheetState = finishBottomSheetState,
                dragHandle = null        ) {
                FinishedContent(
                    totalTime = (uiState as TimerInfoScreenState.Finished).totalTime,
                    onEvent = viewModel::onEvent) }  }}
        @Composable
        private fun Content(
            state: TimerInfoScreenState,
            onEvent: (TimerInfoScreenEvent) -> Unit = {},
            onBackPressed: () -> Unit = {}) {
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .background(MaterialTheme.colorScheme.surface)    ) {
                TopBar(
                    data = TopBarData.Timer,
                    icons = setOf(TopBarIcon.Back(onBackPressed)))
                ContentContainer {
                    when (state) {
                        is TimerInfoScreenState.Finished -> {}
                        is TimerInfoScreenState.Setup -> SetupContent(
                            mode = state.mode,
                            timerSound = state.currentTimerSound,
                            onEvent = onEvent)
                        is TimerInfoScreenState.Ongoing -> OngoingContent(
                            timerDisplayData = state.timerDisplayData,
                            onEvent = onEvent )} }}}
        @Composable
        private fun SetupContent(    mode: TimerInfoSetupMode,
            timerSound: TimerSound?,
            onEvent: (TimerInfoScreenEvent) -> Unit,) {
            val coroutineScope = rememberCoroutineScope()
            val timePickerState = rememberTimePickerState()
```

```
        Column(        modifier = Modifier
            .fillMaxSize()
            .padding(horizontal = 32.dp)
            .padding(top = 64.dp, bottom = 32.dp),
          horizontalAlignment = Alignment.CenterHorizontally,    ) {
          TimePicker(
            state = timePickerState        )
          Spacer(Modifier.weight(1f))
          when (mode) {
            TimerInfoSetupMode.Idle -> {}
            TimerInfoSetupMode.Sound -> SetupSoundsContent(
              currentTimerSound = timerSound,
              onSoundClick                                =                                {
onEvent(TimerInfoScreenEvent.SelectNewTimerSound(it)) }           )
            TimerInfoSetupMode.Templates -> SetupTemplatesContent(
              templates = listOf(              15 * 60000,              45 * 60000,           ),
              onTemplateClick = {
                coroutineScope.launch {
                  timePickerState.setTime(it)} } ) }
          Spacer(Modifier.weight(1f))
          Row(           modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly       ) {
            TimerIconButton(
              icon = R.drawable.ic_music,
              onClick                                =                                {
onEvent(TimerInfoScreenEvent.SelectMode(TimerInfoSetupMode.Sound)) },
              iconModifier = Modifier.size(32.dp),
              isSelected = mode == TimerInfoSetupMode.Sound)
            TimerIconButton(
              icon = R.drawable.ic_play,
              onClick                                =                                {
onEvent(TimerInfoScreenEvent.StartNewTimer(timePickerState.totalMillis)) },
              iconModifier = Modifier
                .padding(start = 5.dp)
                .size(24.dp),
              enabled = timePickerState.totalMillis > 0)
            TimerIconButton(              icon = R.drawable.ic_list,
              onClick                                =                                {
onEvent(TimerInfoScreenEvent.SelectMode(TimerInfoSetupMode.Templates)) },
              iconModifier = Modifier.size(32.dp),
              isSelected = mode == TimerInfoSetupMode.Templates)}}}
      @Composable
      private fun SetupTemplatesContent(
        templates: List<Long>,
```

```
            onTemplateClick: (Long) -> Unit,) {
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceEvenly    ) {
                templates.forEach {
                    TimerTextButton(
                        text = it.formatDurationTimer(),
                        onClick = { onTemplateClick(it) },
                        contentPadding = PaddingValues(
                            horizontal = 24.dp,
                            vertical = 12.dp) )}}}
        @Composable
        private fun SetupSoundsContent(
            currentTimerSound: TimerSound?,
            onSoundClick: (TimerSound) -> Unit,
            timerSounds: List<TimerSound> = TimerSound.values(),
        ) {    Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceEvenly    ) {
                timerSounds.forEachIndexed { index, item ->
                    TimerTextButton(
                        text = (index + 1).toString(),
                        onClick = {
                            onSoundClick(item)              },
                        isSelected = item == currentTimerSound,
                        contentPadding = PaddingValues(
                            horizontal = 12.dp,
                            vertical = 12.dp) )  } }}
        @Composable
        private fun OngoingContent(
            timerDisplayData: TimerDisplayData,
            onEvent: (TimerInfoScreenEvent) -> Unit = {}) {
            val density = LocalDensity.current
            var textWidthPx by remember { mutableFloatStateOf(0f) }
            val animatedProgress by animateFloatAsState(
                targetValue = timerDisplayData.remainingMillis.toFloat() /
timerDisplayData.totalMillis,
                animationSpec = tween(durationMillis = 500),
                label = "timerProgress"    )
            Column(        modifier = Modifier
                    .fillMaxSize()
                    .padding(top = 64.dp, bottom = 32.dp),
                horizontalAlignment = Alignment.CenterHorizontally    ) {
                Text(
```

```
            text = timerDisplayData.totalFormatted,
            modifier = Modifier.onGloballyPositioned { coordinates ->
                textWidthPx = coordinates.size.width.toFloat()          },
            color = MaterialTheme.colorScheme.onPrimary,
            fontSize = 32.sp,
            fontWeight = FontWeight.Normal,
            fontFamily = interFamily       )
        Spacer(Modifier.height(8.dp))
        Box(          modifier = Modifier
                .width(with(density) { (textWidthPx.toDp()) + 24.dp })
                .height(0.5.dp)
                .background(Color.Black)       )
        Spacer(Modifier.height(24.dp))
        CircularCountdownTimer(
            totalTime = timerDisplayData.totalMillis,
            remainingTime = (animatedProgress * timerDisplayData.totalMillis).toLong(),
            timeText = timerDisplayData.remainingFormatted,
            progressColor = MaterialTheme.colorScheme.primary,
            modifier = Modifier
                .size(300.dp)
                .padding(16.dp)       )
        Spacer(Modifier.weight(1f))
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly       ) {
            TimerIconButton(
                icon = R.drawable.ic_stop,
                onClick = { onEvent(TimerInfoScreenEvent.CancelTimer) },
                iconModifier = Modifier.size(24.dp),          )
            AnimatedContent(timerDisplayData.isRunning) {
                if (it) {
                    TimerIconButton(
                        icon = R.drawable.ic_pause,
                        onClick = { onEvent(TimerInfoScreenEvent.PauseTimer) },
                        iconModifier = Modifier.size(24.dp)
                    )
                } else {
                    TimerIconButton(
                        icon = R.drawable.ic_play,
                        onClick = { onEvent(TimerInfoScreenEvent.ResumeTimer) },
                        iconModifier = Modifier
                            .padding(start = 5.dp)
                            .size(24.dp)              )          }          }      }  }}
```

```
@Composable
private fun FinishedContent(
    totalTime: String,
    onEvent: (TimerInfoScreenEvent) -> Unit) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .fillMaxHeight(0.4f)
            .padding(36.dp),
        horizontalAlignment = Alignment.CenterHorizontally    ) {
        Text(
            text = stringResource(R.string.time_is_up),
            modifier = Modifier,
            color = MaterialTheme.colorScheme.onPrimary,
            style = MaterialTheme.typography.titleMedium.copy(
                fontSize = 32.sp           )       )
        Spacer(Modifier.height(24.dp))
        Icon(
            painter = painterResource(R.drawable.ic_bell),
            contentDescription = null,
            modifier = Modifier.size(64.dp),
            tint = MaterialTheme.colorScheme.onPrimary       )
        Spacer(Modifier.height(18.dp))
        Text(          text = totalTime,
            color = MaterialTheme.colorScheme.onPrimary,
            fontSize = 20.sp,
            fontWeight = FontWeight.Normal,
            fontFamily = interFamily       )
        Spacer(Modifier.weight(1f))
        TimerTextButton(
            text = stringResource(R.string.turn_off),
            onClick = { onEvent(TimerInfoScreenEvent.FinishTimer) },
            modifier = Modifier.fillMaxWidth()       )    }}
private fun MediaPlayer.stopPlaying() {
    try {       if (isPlaying) {          stop()       }    } catch (e: IllegalStateException)
{         e.printStackTrace()     } finally {          try {              release()          } catch (e:
Exception) {          e.printStackTrace()       }    }}
@Preview
@Composable
private fun TimerInfoScreenPreview() {
    SmartStudentTheme {       FinishedContent(totalTime = "15:00") { }    }}
```

Лістинг Д.3 — Клас, що відповідає за стиснення та перевертання фото для профілю користувача

```
class ImageResizeManager(
    private val context: Context
) {
    suspend fun resizeImage(uri: Uri, maxSize: Int = 512): ByteArray? {
        return try {
            withContext(Dispatchers.IO) {
                val inputStream =
                    context.contentResolver.openInputStream(uri)                    ?:
return@withContext null
                val exif = ExifInterface(inputStream)
                val orientation = exif.getAttributeInt(
                    ExifInterface.TAG_ORIENTATION,
                    ExifInterface.ORIENTATION_NORMAL            )
                val imageStream =
                    context.contentResolver.openInputStream(uri)                    ?:
return@withContext null
                val originalBitmap = BitmapFactory.decodeStream(imageStream)
                val    rotatedBitmap    =    rotateBitmapIfRequired(originalBitmap,
orientation)
                val    aspectRatio    =    rotatedBitmap.width.toFloat()    /
rotatedBitmap.height.toFloat()
                val width: Int
                val height: Int
                if (aspectRatio > 1) {
                    width = maxSize
                    height = (maxSize / aspectRatio).toInt()
                } else {
                    height = maxSize
                    width = (maxSize * aspectRatio).toInt()          }
                val   bitmap   =   Bitmap.createScaledBitmap(originalBitmap,   width,
height, true)
                bitmapToByteArray(bitmap)          }
            } catch (e: Exception) {
                e.printStackTrace()
                null       }    }
        private suspend fun bitmapToByteArray(bitmap: Bitmap): ByteArray {
            return withContext(Dispatchers.IO) {
                val stream = ByteArrayOutputStream()
                bitmap.compress(Bitmap.CompressFormat.JPEG, 85, stream)
                stream.toByteArray()       }    }
        private fun rotateBitmapIfRequired(bitmap: Bitmap, orientation: Int): Bitmap
matrix = Matrix()
```

```
            when (orientation) {
                ExifInterface.ORIENTATION_ROTATE_90 -> matrix.setRotate(90f)
                ExifInterface.ORIENTATION_ROTATE_180 -> matrix.setRotate(180f)
                ExifInterface.ORIENTATION_ROTATE_270 -> matrix.setRotate(-90f)
                else -> return bitmap        }
            return Bitmap.createBitmap(bitmap, 0, 0, bitmap.width, bitmap.height, matrix, true)    }}
```