Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації

Виконав: студент 4 курсу
групи ЗПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Кучеренко Максим Володимирович
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Ракитянська Г.Б.
(прізвище та ініціали)
Рецензент:. к.т.н., доц. каф. ЗІ Баришев Ю.В..
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____ О. Н. Романюк
«10» 06 2024 р.

ВНТУ – 2024

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«12» березня 2024 р.

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

Кучеренку Максиму Володимировичу

1. Тема роботи – Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації.

Керівник роботи: Ракитянська Ганна Борисівна, канд. техн. наук, доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: середовище розробки – Android studio, мова розробки – Kotlin, операційна система – Windows 10.

4. Зміст текстової частини: вступ, аналіз предметної області, порівняльний аналіз аналогів, постановка задач, розробка інтерфейсу застосунку, розробка алгоритму роботи застосунку, розробка модулів застосунку, тестування застосунку, висновки, список використаних джерел, додатки.

5. Перелік ілюстративного матеріалу: титульний слайд, актуальність теми, мета предмет та об'єкт дослідження, наукова новизна, задачі дослідження, аналіз аналогів, розробка макету та дизайну інтерфейсу, розробка алгоритму застосунку, тестування застосунку, апробація та публікація результатів роботи, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Ракитянська Г. Б., к.т.н, доц. каф. ПЗ	13.03.2024	03.06.2024

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану мобільних систем з використанням технологій геймифікації	14.03.2024 – 28.03.2024	Вик.
2	Розробка інтерфейсу та алгоритмів програмного продукту	29.03.2024 – 10.04.2024	Вик.
3	Розробка модулів програмного продукту	11.04.2024 – 15.05.2024	Вик.
4	Тестування програми	16.05.2024 – 26.05.2024	Вик.
5	Оформлення матеріалів до захисту БДР	26.05.2024– 30.05.2024	Вик.

Студент _____ **М. В. Кучеренко**
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ **Г. Б. Ракитянська**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 59 сторінок формату А4, на яких є 32 рисунка, 3 таблиці, список використаних джерел містить 22 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз методів і засобів геймифікації навчального процесу учнів переважно молодших класів. Сформульовано мету досліджень – підвищення мотивації користувачів переважно шкільного віку до виконання завдань за рахунок використання технологій геймифікації у вигляді моделі віртуального персонажа.

Запропоновано метод геймифікації навчального процесу за допомогою віртуального персонажа, за яким віртуальний персонаж отримує певні покращення та віртуальну валюту у залежності від виконаних користувачем завдань та отриманих досягнень.

У бакалаврській дипломній роботі розроблено програмні засоби мобільної системи геймифікації навчального процесу учнів молодших класів. Застосунок призначений для мобільної системи, яка буде здатна надавати користувачам мотивації до виконання завдань шляхом створення моделі віртуального персонажа та його прокачки виконанням завдань. Було розроблено такі програмні модулі як: Модуль завдань, модуль віртуального персонажа та модуль винагород. Програмні модулі, розроблені в проєкті, можуть бути використані як приклад та основа застосування технологій геймифікації.

Застосунок розроблено з використанням мови Kotlin та середовища Android studio.

Отримані в бакалаврській дипломній роботі результати можна використати для побудови системи трекінгу та управління успішністю у більшості навчальних закладів.

Ключові слова: геймифікація, мобільна система, навчальний процес, віртуальний персонаж.

Annotation

The bachelor thesis consists of 59 pages of A4 format, on which there are 32 figures, 3 tables, the list of used sources contains 22 items.

In the bachelor thesis, a detailed analysis of methods and means of gamification of the educational process of students, mainly in younger grades, was carried out. The goal of the research is formulated - to increase the motivation of mostly school-age users to perform tasks through the use of gamification technologies in the form of a virtual character model.

A method of gamification of the educational process using a virtual character is proposed, according to which the virtual character receives certain improvements and virtual currency depending on the tasks performed by the user and the achievements obtained.

In the bachelor's diploma work, the software tools of the mobile system of gamification of the educational process of junior school students were developed. The application is designed for a mobile system that will be able to provide motivation to users to complete tasks by creating a model of a virtual character and pumping it up by completing tasks. The following software modules were developed: Task module, virtual character module and rewards module. The software modules developed in the project can be used as an example and basis for the application of gamification technologies.

The application is developed using the Kotlin language and the Android studio environment.

The results obtained in the bachelor's thesis can be used to build a system of tracking and managing success in most educational institutions.

Keywords: gamification, mobile system, educational process, virtual character.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний світ відомий своєю швидкістю та постійним розвитком технологій, що надає нам необмежені можливості впливати на різні аспекти нашого повсякденного життя. Однією з ключових інновацій, яка активно використовується для покращення досвіду користувачів у різних сферах є геймифікація.

Геймифікація, вона ж застосування ігрових елементів у неігрових контекстах, здійснює значний вплив на мотивацію та залучення користувачів у повсякденних застосунках [1]. Ця технологія виявляється особливо ефективною в управлінні навчанням, здоров'ям, фітнесом, фінансами та іншими сферами, де стимулювання участі та досягнень відіграє ключову роль.

При розгляді даної теми важливо дослідити не лише самі механізми геймифікації, але й її вплив на психологічний аспект користувачів, та відповідно, на формування їхньої мотивації, інтересу до програмних продуктів та стимулювання активності.

Аналогічних мобільних систем з використанням технологій геймифікації досить мало, а наявні не надають достатньо потужного та корисного функціоналу. Використання технологій геймифікації у повсякденних справах хоча і є обговорюваною темою, проте застосунків з використанням даних технологій не так багато, тому розробка власної мобільної системи з використанням технологій гейміфікації є актуальною.

Проєкт передбачає розробку різних пакетів, які будуть об’єднані в мобільну систему. Ці модулі включають модуль управління даними, модуль для роботи з базою даних, модуль графічного інтерфейсу. Крім того буде реалізований модуль віртуального персонажа.

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

Мета та завдання дослідження. Метою дослідження є удосконалення наявних мобільних інструментів мотивації до виконання завдань за допомогою використання технологій геміфікації у вигляді віртуального персонажа що асоціюється з користувачем.

Відповідно до поставленої мети в бакалаврській дипломній роботі необхідно виконати такі завдання:

– розробка бази даних, сумісної з вимогами мобільної системи;

– розробка модуля для роботи з віртуальним персонажем, який буде надсилати у базу даних дані про персонажа та відображати ці дані користувачу;

– розробка модуля графічного інтерфейсу, який може ефективно представляти користувачеві завдання;

– розробка модуля контролю виконання завдань для спільного використання;

– розробка модуля роботи з завданнями для відстеження, завершення та зміни завдань.

Об'єкт дослідження – процес розробки програмних модулів для реалізації мобільної системи, що використовує технології геймифікації у вигляді моделі віртуального персонажа.

Предмет дослідження – алгоритми і засоби реалізації мобільного застосунку з використанням технологій геймифікації.

Методи дослідження. У процесі досліджень використовувались: модульний, об’єктно-орієнтований та рекурсивний підходи для проєктування програмного забезпечення, методи доповненої реальності для створення віртуального навчального середовища; методи імітаційного моделювання для розробки моделі віртуального персонажа; методи статистичної обробки інформації для аналізу успішності виконання завдань.

Новизна отриманих результатів.

1. Здобула подальшого розвитку технологія геймифікації, яка, на відміну від більшості існуючих аналогів, відображає загальний розвиток віртуального

персонажа в залежності від виконаних завдань, що дозволяє підвищити мотивацію користувачів.

2. Удосконалено метод адміністрування завдань, особливість якого полягає у залучені іншої людини до процесу виконання завдань, що дозволяє об’єднуватись у групи людей задля сумісного виконання і розподіленої перевірки завдань.

Практична цінність отриманих результатів. Практична цінність полягає в розробленій мобільній системі, яка здатна надавати користувачам мотивації до виконання завдань шляхом застосування технологій геймифікації. Програмні модулі, розроблені в проєкті, можуть бути використані розробниками як приклад та основа для створення систем на різних платформах, які використовують технології геймифікації в різні додатки, включаючи повсякденні, освітні і багато інших.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. В ході виконання даної бакалаврської дипломної роботи, здобувач особисто провів дослідження, аналізував літературні джерела, вивчав існуючі аналоги та їх функціональність, проектував та реалізував програмне забезпечення платформи, а також проводив тестування та оцінку його працездатності.

Апробація та публікація результатів дипломної роботи. Результати роботи доповідалися на конференції LIII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024) і опубліковані в тезах доповіді цієї конференції [2].

1 АНАЛІЗ РОЗВИТКУ МОБІЛЬНИХ СИСТЕМ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ГЕЙМІФІКАЦІЇ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану мобільних систем з використанням технологій геймифікації

З розвитком сучасних технологій, в галузі освіти стає все більш очевидною необхідність застосування інноваційних підходів для залучення учнів до навчального процесу. Одним із найефективніших методів є геймифікація, тобто використання елементів гри для підвищення мотивації та інтересу учнів до вивчення навчального матеріалу. Через це розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації є актуальним завданням.

Однією з переваг такої системи є можливість індивідуалізації навчального процесу, де кожен учень може працювати у власному темпі та з урахуванням своїх особистих якостей та можливість відстеження свого прогресу у чисельному еквіваленті. Це сприяє підвищенню ефективності навчання та збереженню мотивації до досягнення навчальних цілей [3].

Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації вимагає комплексного підходу та співпраці між педагогами, програмістами та психологами. Лише зіставляючи педагогічний досвід з технічними можливостями сучасних мобільних технологій можна створити ефективні та інноваційні засоби, спрямовані на підвищення якості навчання у молодших школярів.

Крім того розроблювана система може бути використана у повсякденних справах, а також як основа для створення більш просунутих систем з використанням технологій геймифікації. Дані системи мають дуже великий потенціал для майбутніх розробок.

Отже, Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації є потенційно

дуже важливою задачею в сучасному цифровому середовищі, а також основою для майбутніх розробок. Використання досвіду спеціалізованих компаній у розробці програмного забезпечення дозволяє створювати інноваційні рішення, що покращують користувацький досвід та надають конкурентні переваги. Ураховуючи швидкий розвиток технологій мобільних платформ, необхідно постійно оновлювати знання та слідкувати за останніми тенденціями у цій галузі, щоб залишатися ефективними та актуальними.

1.2 Порівняльний аналіз аналогів

З впровадженням концепції геймифікації в навчальний процес з'являється потреба в розробці спеціалізованих програмних засобів, які б сприяли активізації навчальної діяльності та підвищенню мотивації учнів. Розробка мобільної системи геймифікації навчального процесу для учнів молодших класів у такому випадку виявляється актуальною та перспективною. Такі програмні засоби можуть об'єднувати в собі функціонал листа завдань для відстеження завдань та геймифіковані елементи, що стимулюють учнів до активної участі у навчанні. До аналогів розроблюваної системи можна віднести:

- Classcraft [4];
- Do It Now: RPG To Do List [5];
- Kahoot! [6];
- Daily habits [7].

Одним із прикладів використання технологій геймифікації є сервіс Classcraft (див. рисунок 1.1). Цей сервіс дозволяє студентам створити свого власного персонажа та прокачувати його у відповідності з прогресом у навчанні, він надає багато функцій та можливостей для налаштування. Даний сервіс добре допомагає вчителям у мотивуванні студентів та частково автоматизує учбовий процес [4].

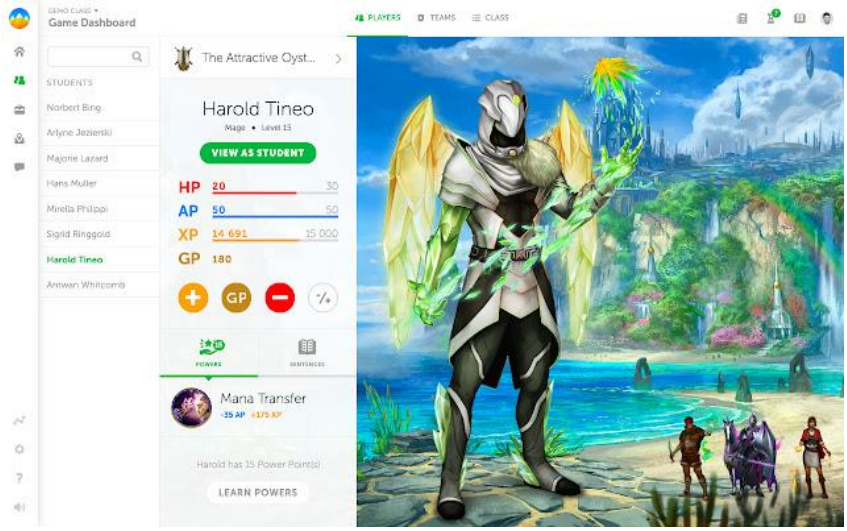

Рисунок 1.1 – Приклад роботи сервісу Classcraft

Інший приклад – застосунок Do it now: RPG To-Do List (див. рисунок 1.2), який додає певні ігрові елементи у виконання повсякденних справ, крім того він також має деякі додаткові функції як календар, динамічні завдання та організація завдань, та інші. Це дозволяє ефективно розподіляти час для виконання будь-яких завдань та відстежувати свій прогрес [5].

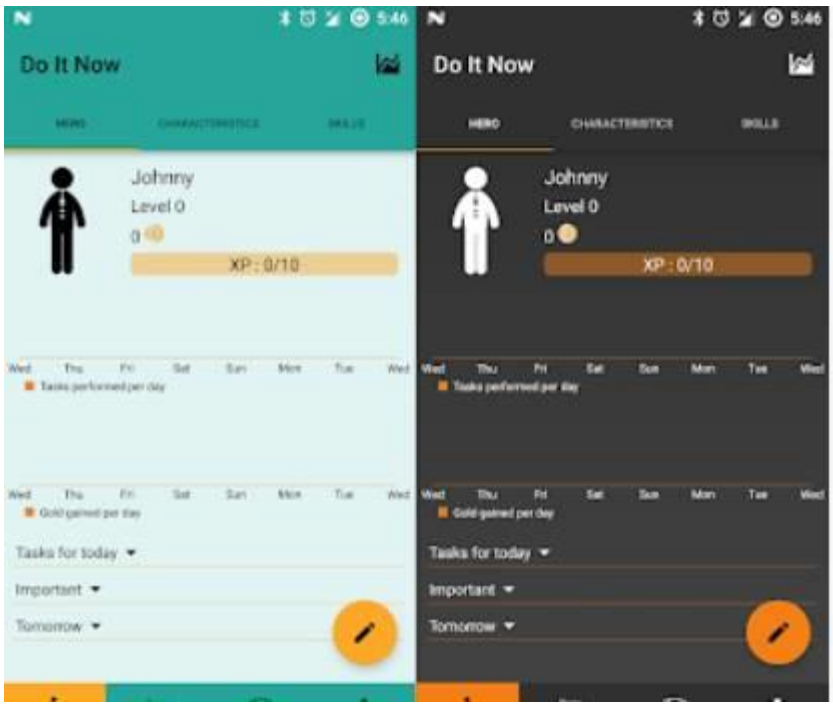

Рисунок 1.2 – Приклад роботи застосунку Do it now: RPG To-Do List

Третім аналогом є сервіс “Kahoot!”. Kahoot! – це популярна платформа для проведення інтерактивних онлайн-квізів та ігор, яка використовується у навчальних закладах по всьому світу. За допомогою Kahoot! вчителі можуть створювати свої власні квізи та ігри або користуватися уже готовими шаблонами, а учні можуть брати участь у них за допомогою своїх мобільних пристроїв. Ця платформа створює захопливу атмосферу у класі, стимулює активну участь учнів та допомагає у вивченні матеріалу через ігровий підхід до навчання [6].

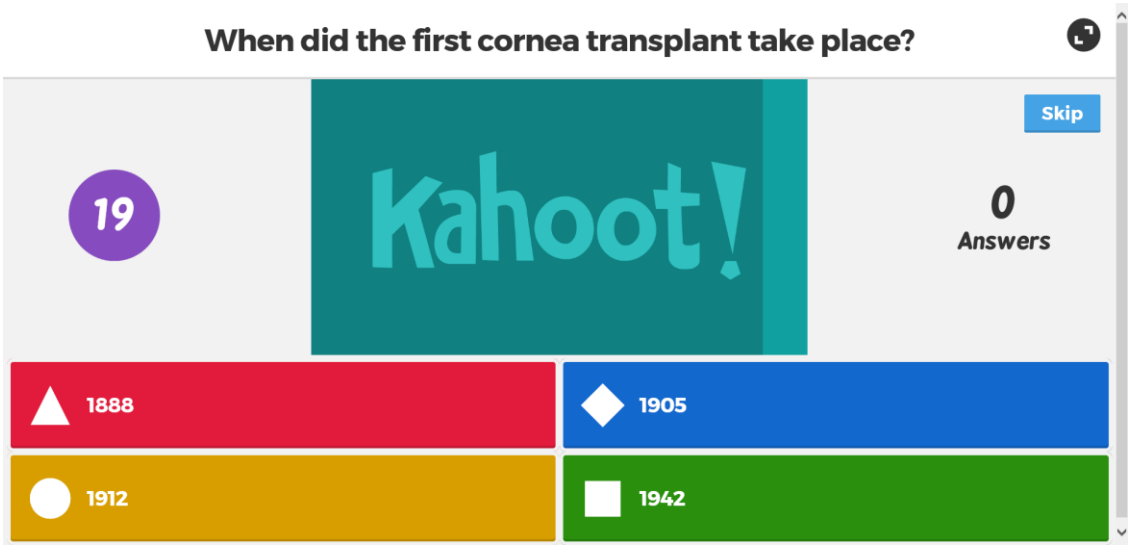

Рисунок 1.3 – Приклад роботи сервісу Kahoot!

Четвертим аналогом є застосунок DailyHabits. DailyHabits – це застосунок типу "habit tracker", який допомагає користувачам створювати й відстежувати їхні щоденні звички та цілі. Завдяки його інтуїтивному інтерфейсу, користувачі можуть легко створювати нові звички. Застосунок також надає можливість встановлювати нагадування, що допомагає дотримуватися цілі. Крім того, DailyHabits забезпечує зручну статистику та аналітику, що дозволяє користувачам оцінювати свій прогрес та розвиток у формуванні нових корисних звичок. Завдяки цьому застосунку користувачі можуть ефективно керувати своїм часом та зосереджуватися на досягненні своїх цілей [7].

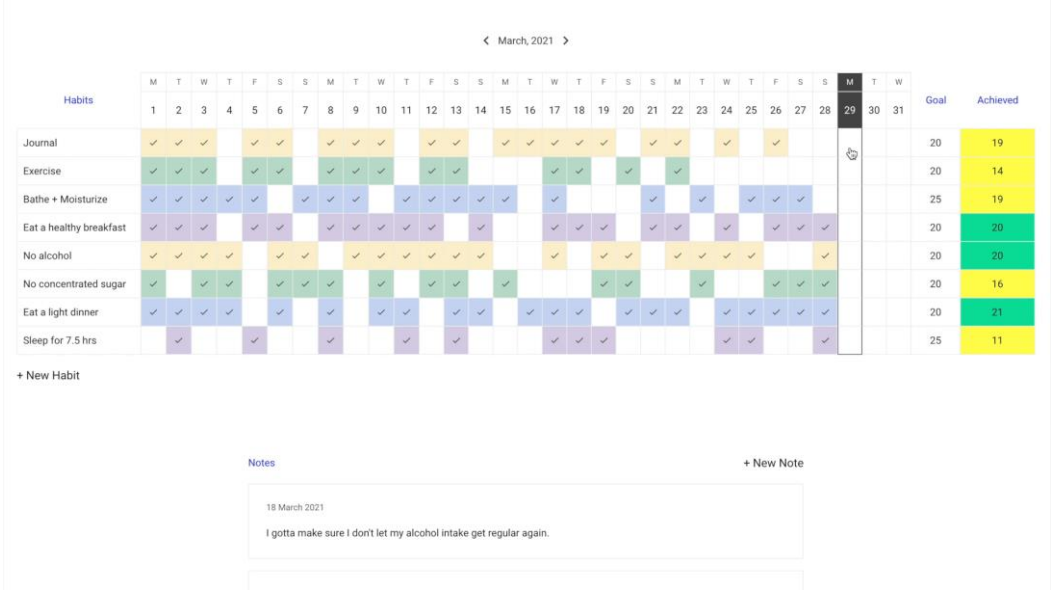

Рисунок 1.4 – Приклад роботи додатку Daily habits

Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації вимагає комплексного підходу до розробки. Це означає, що необхідно мати досвід у галузі розробки програмного забезпечення, а також розуміння основ технологій геймифікації та дизайну користувацького досвіду. Покладається значний акцент на розробку інтерфейсу, який буде інтуїтивно зрозумілий для учнів молодших класів.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Власна система	Classcraft	Do it Now	Kahoot!	Daily habits
Геймифікований інтерфейс	+	+	-	+	-
Віртуальний персонаж	+	+	+-	-	-
Широкий спектр можливих завдань	+	-	+	-	+
Можливість використання поза навчального процесу	+	-	+	-	+
Загальна оцінка	100%	50%	63%	25%	50%

Відповідно до таблиці порівняльних характеристик розробка власної мобільної системи геймифікації навчального процесу учнів молодших класів є доцільною. Отриманий продукт зможе вирішити недоліки існуючих рішень та забезпечити новий досвід та можливості застосування технологій геймифікації.

Отже, розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації відкриває перед освітніми установами та батьками широкі можливості для поліпшення навчального процесу, створення цікавого та інноваційного навчального середовища, а також підвищення мотивації та інтересу учнів до навчання. Розуміючи потенційні переваги застосування технологій геймифікації та вимоги до розробки програмного забезпечення, компанії можуть створювати інноваційні та ефективні мобільні системи для освітніх закладів для покращення навчального процесу та досягнення успіху у навчанні.

1.3 Аналіз методів розв’язання задачі

Існує кілька методів реалізації мобільної системи, включаючи мобільний застосунок, десктопний застосунок і вебзастосунок. Кожен з цих методів має свої переваги та недоліки, які варто врахувати при виборі найбільш підходящого рішення:

1. Мобільний застосунок має такі переваги як зручний для користувачів, оскільки дозволяє отримати доступ до системи з будь-якого місця і у будь-який час через смартфон або планшет, також він може використовувати функціональні можливості пристрою, такі як GPS для визначення місцезнаходження, проте він має і недоліки, такі як вимога встановлення на пристрої користувача, що може бути не зручно для деяких користувачів. Розробка та підтримка мобільного застосунку може бути витратною та складною.

2. Десктопний застосунок має такі переваги як більший функціонал та масштабування у порівнянні з мобільним застосунком, а також є зручним для великих екранів і робочих станцій, проте він має і недоліки – зазвичай вимагає

встановлення на комп'ютері, що може бути не зручно для користувачів, які працюють з великою кількістю мобільних пристроїв, а також може бути складніше у використанні для менш досвідчених користувачів.

3. Вебзастосунок має такі переваги як легка доступність через браузер на будь-якому пристрої з інтернет-підключенням та не вимагає встановлення на пристрої користувача, проте має такі недоліки як буття менш зручним для використання на мобільних пристроях порівняно з мобільним додатком, а також деякі функції, такі як доступ до GPS, можуть бути обмежені.

Створення десктопного застосунку або веб-застосунку може бути виправданим у певних випадках, але у цьому випадку мобільний застосунок має перевагу над ними через доступність будь-де будь-коли при наявності смартфону.

Отже, у випадку застосунку для підвищення мотивації учнів молодших класів з використанням технологій геймифікації розробка у якості мобільного застосунку буде кращим вибором через свою доступність та невимогливість.

1.4 Постановка задач проєкту

Проаналізувавши переваги та недоліки існуючих аналогів з використанням технологій геймифікації та організаційних застосунків було визначено наступні завдання, які необхідно виконати для розробки мобільного застосунку:

– визначити та розробити загальний алгоритм застосунку;

– розробити модель віртуального персонажа для застосування в якості технологій геймифікації;

– розробити базу даних для збереження інформації про персонажа та нагороди;

– розробити зручний та адаптивний графічний інтерфейс мобільного застосунку, що може бути використаний учнями молодших класів;

– реалізувати мобільний застосунок з використанням технологій геймифікації;

– здійснити тестування мобільного застосунку згідно поставлених задач.

1.5 Висновки

У першому розділі було розглянуто стан мобільних застосунків з використанням технологій геймифікації. Було проведено порівняння схожих відомих платформ для покращення навчального процесу з використанням технологій геймифікації або зі схожою орієнтованістю, таких як: Classcraft, Do it Now: RPG To-Do list, Kahoot! та Daily habits.

У результаті порівняння було виявлено, що досліджувані платформи набули значної популярності серед застосунків для тайм-менеджменту та покращення навчального процесу завдяки своїй багатофункціональності, вдалому застосуванню технологій геймифікації або простоті. Проаналізувавши переваги та недоліки відомих платформ, було виявлено, що вони не надають простого для розуміння учнями молодших класів інтерфейсу або ж вдалих технологій геймифікації. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного застосунку.

2 РОЗРОБКА ІНТЕРФЕЙСУ ТА АЛГОРИТМІВ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Аналіз вхідних та вихідних даних застосунку

Вхідні дані для розробки програмного застосунку геймифікації навчального процесу учнів молодших класів представлені у формі завдань, де користувач самостійно вводить такі дані, як:

– заголовок – коротка назва завдання, яка дозволяє швидко зрозуміти його суть. Заголовок має бути чітким та інформативним, щоб користувачі могли легко орієнтуватися серед багатьох завдань;

– опис завдань – тут надається детальна інформація про завдання. Опис має включати чіткі інструкції щодо виконання завдання, мету, яку треба досягти, і будь-які інші важливі деталі, що допоможуть учням виконати завдання правильно;

– складність – кожне завдання має рівень складності, який вказує на те, наскільки важким та винагороджуючим воно буде для учнів. Це позначено словами, такими як "легкий", "середній", "важкий". Рівень складності допомагає користувачам обирати завдання відповідно до їхніх можливостей і рівня підготовки;

– кінцеву дату завдання - це дата, до якої завдання має бути виконане. Вказання кінцевої дати допомагає учням планувати свій час та не пропускати дедлайни. Це також сприяє розвитку навичок управління часом.

Розробка програмних модулів для реалізації мобільного застосунку з використанням технологій геймифікації передбачає кілька етапів. По-перше, потрібно створити модуль, який буде керувати базою даних завдань та віртуального персонажа. Цей модуль буде відповідати за внесення даних у базу даних, їх зберігання, отримання та оновлення. Далі буде розроблений модуль для обробки отриманих даних. Цей модуль буде використовувати приклад комп'ютерних ігор для відображення даних про персонажа у ігровій формі.

Розробка цих програмних модулів вимагає розуміння логіки системи прокачки персонажів у комп’ютерних іграх. Для реалізації цих модулів будуть використані різноманітні інструменти та бібліотеки програмування. Програмні модулі будуть протестовані та доопрацьовані, щоб забезпечити надійну та зручну роботу з застосунком для користувачів.

Вихідними даними застосунку є відображення усіх наявних завдань у якості списку та віртуальний персонаж у якості відображення кількості виконаних завдань та самого користувача.

Отже, розробка програмних модулів для реалізації мобільного застосунку з використанням технологій геймифікації є комплексним завданням, яке включає кілька етапів. Модулі повинні бути ретельно спроєктовані, реалізовані та протестовані, щоб забезпечити якісний досвід роботи з застосунком. Однак, маючи правильні інструменти та досвід, можна створити мобільний застосунок, який може значно підвищити ефективність навчання і загальну продуктивність.

2.2 Розробка інтерфейсу програмного застосунку

Під час розробки інтерфейсу мобільного застосунку було прийнято рішення зробити його максимально простим та дружнім для користувача. Це допомагає забезпечити зручність у використанні та знижує поріг вхідного бар'єру для користувачів [8].

Основним кольором інтерфейсу було обрано бежевий. Цей вибір зумовлений тим, що бежевий колір асоціюється зі старовинним пергаментом, що додає елементи фентезі до візуальної тематики. Такий підхід створює атмосферу та додає художнього шарму до інтерфейсу.

Допоміжним кольором для інтерфейсу був обраний коричневий. Цей колір був вибраний через контрастність з основним бежевим кольором. Вони доповнюють один одного, створюючи візуальну гармонію та підкреслюючи важливі елементи інтерфейсу [9].

При відкритті додатку, користувача зустрічає сторінка із активними завданнями та назвою екрану (див. рисунок 2.1), з якою користувач буде взаємодіяти найбільше.

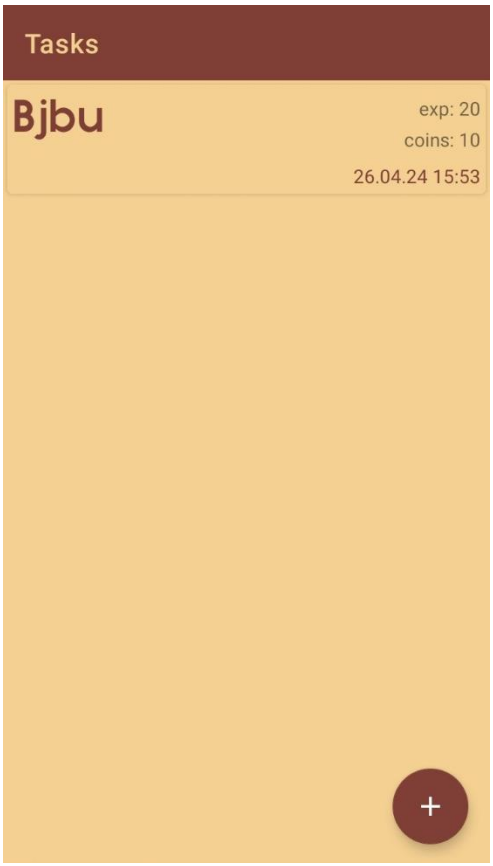

Рисунок 2.1 – Екран завдань

Навігаційне меню буде розташоване у нижній частині екрану, дозволяючи користувачам зручно переходити між різними розділами додатка.

Сторінка персонажу (див. рисунок 2.2) буде призначена для відображення інформації про поточного персонажа. Тут будуть показані характеристики персонажа, його рівень, його бали досвіду, бали досвіду для прокачки окремої характеристики та його аватар. Лінія прогресу бали досвіду персонажу та бали досвіду окремих характеристик заповнюються по мірі виконання завдань. Сам екран виконаний надзвичайно просто для розуміння і не потребує ніяких просунутих знань.

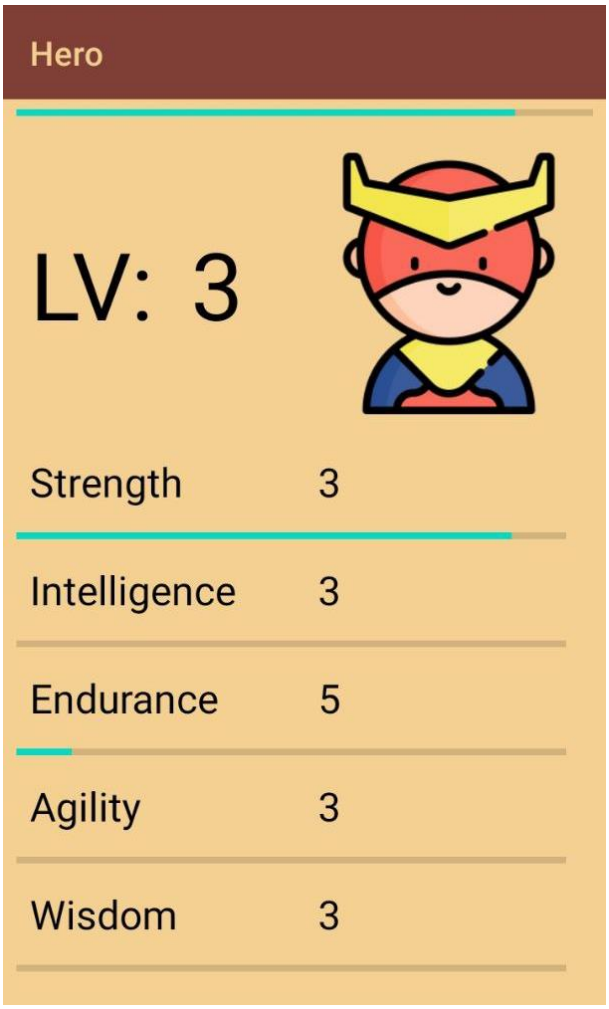

Рисунок 2.2 – Екран персонажу

На сторінці налаштувань завдань, зображеній на рисунку 2.3, користувачам надається можливість здійснювати різноманітні операції з управління завданнями. Основними функціями цього інтерфейсу є видалення обраного завдання: у випадку, коли завдання стає непотрібним або помилково створеним, користувачі мають змогу видалити його зі списку, це дозволяє підтримувати актуальність та впорядкованість завдань. Редагування завдань: якщо виникає необхідність у внесенні змін до існуючого завдання, користувачі можуть легко редагувати його параметри, це включає оновлення опису, зміни в плануванні та коригування відповідальних осіб. Завершення завдань у режимі адміністратора: в режимі адміністратора надається можливість завершувати завдання, незалежно від їхнього поточного стану, це особливо корисно для контролю і нагляду за виконанням завдань користувачами.

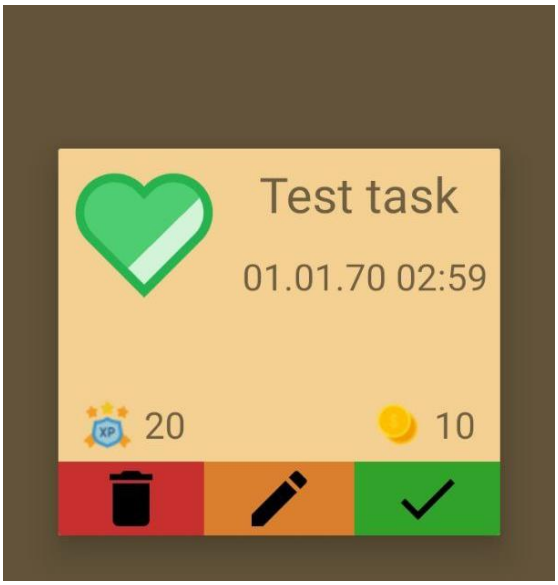

Рисунок 2.3 – Екран розширеної інформації про завдання

На екрані адміністрування застосунку користувачі мають змогу виконувати різноманітні операції, пов'язані з налаштуваннями та управлінням адміністративними функціями. Основні можливості цього екрану включають встановлення пароля адміністратора: користувачі можуть створити або змінити пароль адміністратора, що забезпечує додатковий рівень захисту та контролю доступу до адміністративних функцій застосунку. Вхід у режим адміністратора: після встановлення пароля, користувач може увійти у режим адміністратора. Скидання пароля адміністратора: У випадку втрати або забуття пароля, користувачі можуть скористатися функцією скидання пароля. Вихід з режиму адміністратора: Після завершення робіт у режимі адміністратора, користувачі можуть вийти з цього режиму, щоб повернутися до стандартного користувацького інтерфейсу.

Інтерфейс даного екрану є динамічним і змінюється в залежності від вже наявних даних та поточного стану застосунку. Наприклад, деякі елементи можуть бути додані або видалені в різних ситуаціях, забезпечуючи таким чином адаптивність та гнучкість застосунку.

На рисунку 2.4 представлено вигляд екрану адміністрування без будь-яких попередніх даних.

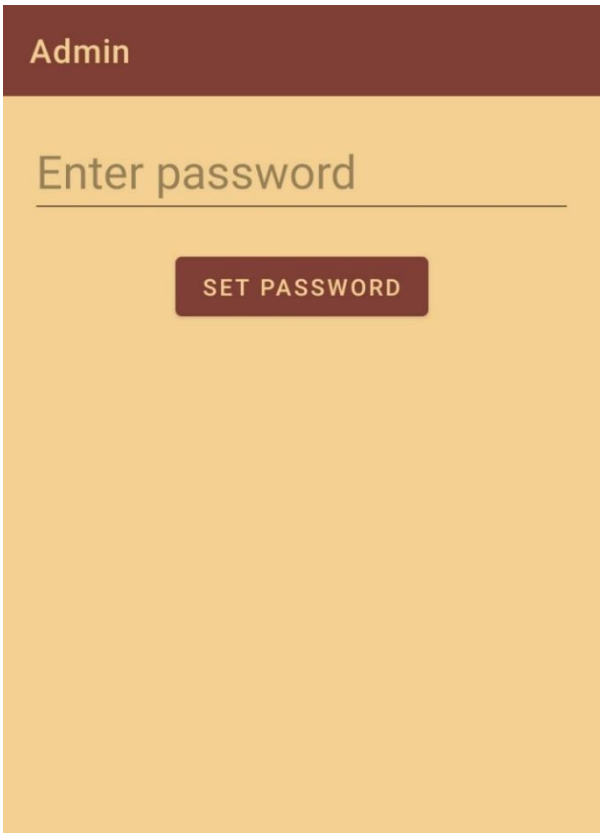

Рисунок 2.4 – Екран адміністрування без попередніх даних для ініціалізації

У випадках, коли в застосунку вже встановлено пароль адміністратора, інтерфейс зазнає певних змін, що можуть виявлятися у вигляді суперечливості між різними елементами. Ці зміни включають модифікацію елементів інтерфейсу: встановлення адміністративного пароля спричиняє адаптацію інтерфейсу до нового стану. Наприклад, деякі опції можуть стати доступними лише в режимі адміністратора, тоді як інші можуть бути заблоковані або змінити свою поведінку. Також відбувається видалення або заміна деяких елементів за їх непотрібністю.

На рисунку 2.5 показаний приклад інтерфейсу застосунку з попередньо встановленим паролем, де можна спостерігати зазначені вище зміни в елементах інтерфейсу, що відображають суперечливість між ними. Крім того видно що було додано елементи пов’язані з віртуальною валютою.

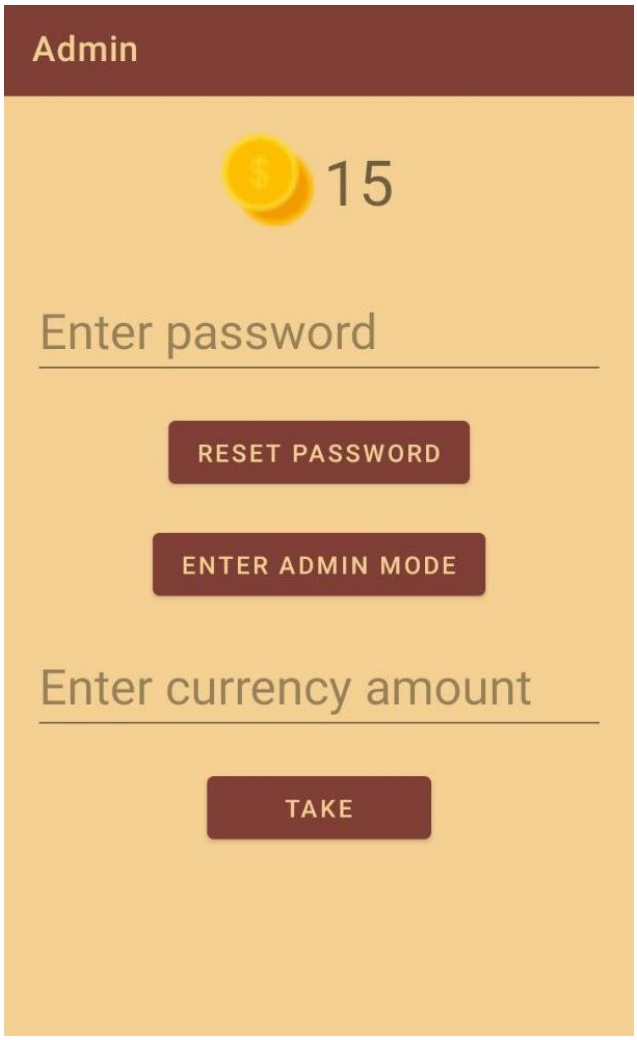

Рисунок 2.5 – Екран адміністрування із попередньо зазначеним паролем адміністратора

Ще одним важливим екраном у застосунку є екран додавання завдання. Цей екран має динамічний інтерфейс, який змінюється залежно від отриманих даних. Особливо помітні зміни відбуваються при переході з екрану розширеної інформації про завдання. У такому випадку всі поля на екрані додавання завдання автоматично заповнюються даними з обраного завдання. Це означає, що замість створення нового завдання, користувач має можливість редагувати вже існуюче завдання. Такий підхід значно спрощує процес коригування завдань, забезпечуючи зручність та економію часу для користувача.

Коли екран додавання завдання використовується без попередньо отриманих даних, він відкривається у своєму стандартному стані, готовий до введення нової

інформації. У цьому випадку всі поля залишаються порожніми, і користувач може заповнити їх необхідними даними для створення нового завдання. Така гнучкість інтерфейсу дозволяє ефективно використовувати екран як для створення нових завдань, так і для редагування існуючих.

Приклад екрану додавання завдання без попередньо отриманих даних можна побачити на рисунку 2.6. Цей приклад демонструє початковий стан інтерфейсу, що містить порожні поля, готові до заповнення користувачем. У реальних умовах використання, залежно від контексту, інтерфейс цього екрану може змінюватися, автоматично адаптуючись до поточних потреб користувача.

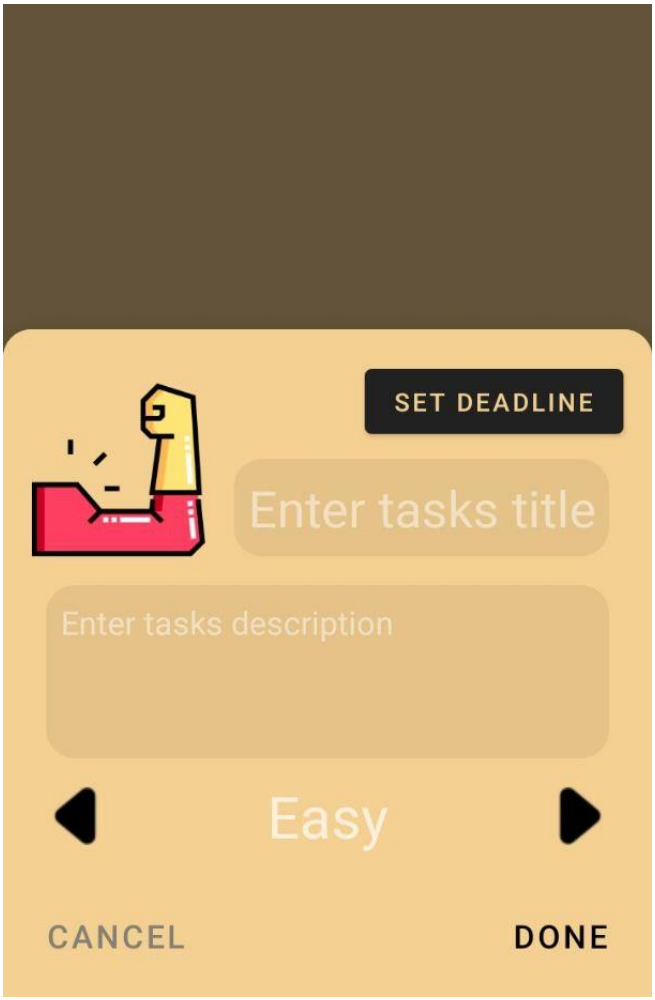

Рисунок 2.6 – Екран створення нового завдання без ініціаційних даних

У застосунку було розроблено спеціальний компонент для вибору характеристики для прокачки під час виконання завдання. Цей компонент

реалізовано у вигляді AlertDialog, який містить список усіх можливих характеристик. AlertDialog є модальним вікном, яке тимчасово блокує основний інтерфейс і вимагає від користувача вибору однієї з характеристик для подальшого налаштування завдання. Цей діалог викликається в момент, коли користувач натискає на зображення існуючої характеристики на екрані створення завдання. У відповідь на цю дію з’являється вікно AlertDialog, де користувач може переглянути та обрати потрібну характеристику зі списку.

На рисунку 2.7 зображено, як виглядає цей діалоговий екран у застосунку. На зображенні показано список можливих характеристик, які користувач може обрати для прокачки. Завдяки використанню AlertDialog, процес налаштування завдання стає більш структурованим і зрозумілим, дозволяючи швидко і ефективно обирати необхідні параметри.

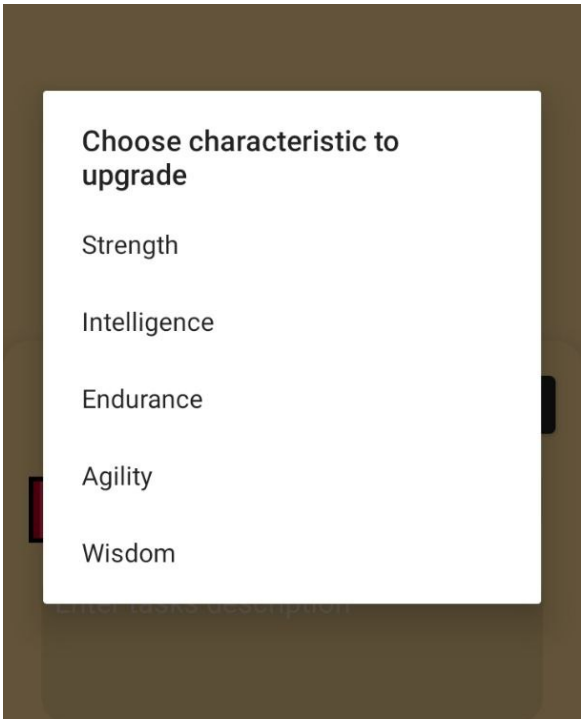

Рисунок 2.7 – Екран вибору характеристики

Одним із ключових компонентів застосунку є зручний екран вибору дати та часу. Цей екран розроблений для того, щоб забезпечити користувачеві максимальну зручність при виборі дати та часу для крайнього строку виконання

завдання. Головною метою цього екрану є полегшення процесу встановлення термінів та дедлайнів, дозволяючи користувачам швидко та без зусиль визначати необхідний часовий проміжок для виконання завдань.

Екран вибору дати та часу має інтуїтивний і зрозумілий інтерфейс. Він включає в себе календар для вибору конкретної дати та зручний інтерфейс для встановлення точного часу. Це дозволяє користувачам легко налаштовувати терміни виконання завдань, мінімізуючи можливість помилок і підвищуючи ефективність планування.

На рисунку 2.8 представлено вигляд інтерфейсу цього екрану, де видно що обрання дати реалізовано трьома скролінговими елементами, що є надзвичайно простою у розумінні реалізацією.

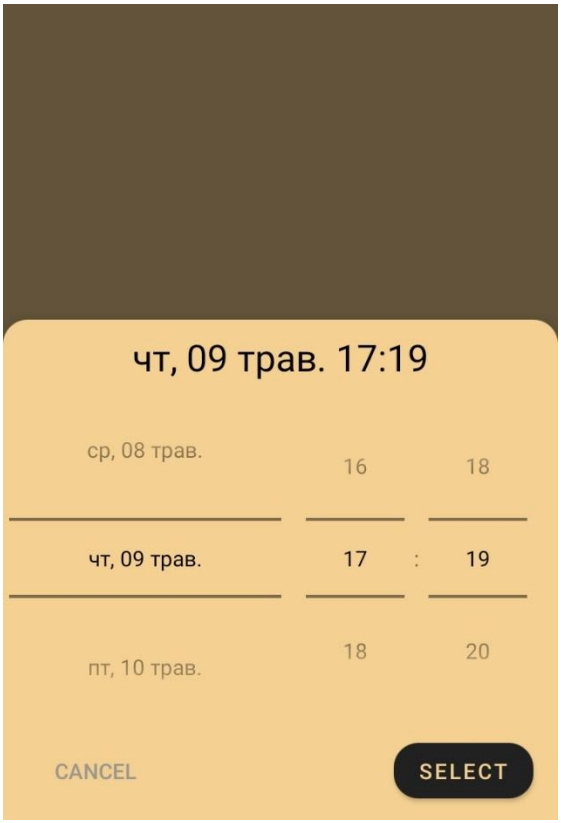

Рисунок 2.8 – Екран вибору дати та часу

Загалом, такий інтерфейс дозволить користувачам швидко і зручно керувати своїм персонажем та завданнями в застосунку.

2.3 Розробка алгоритму застосунку

Для кращого розуміння роботи програмних модулів мобільного застосунку з використанням технологій геймифікації було вирішено розробити модель роботи застосунку на прикладі UML діаграми. UML – це мова моделювання, яка використовується для створення діаграм для аналізу, проєктування та документування програмного забезпечення [10]. Діаграма діяльності застосунку - блок-схема діяльності. Вона представляє робочий процес між різними діями застосунку [11]. Розроблена схема зображена на рисунку 2.8.

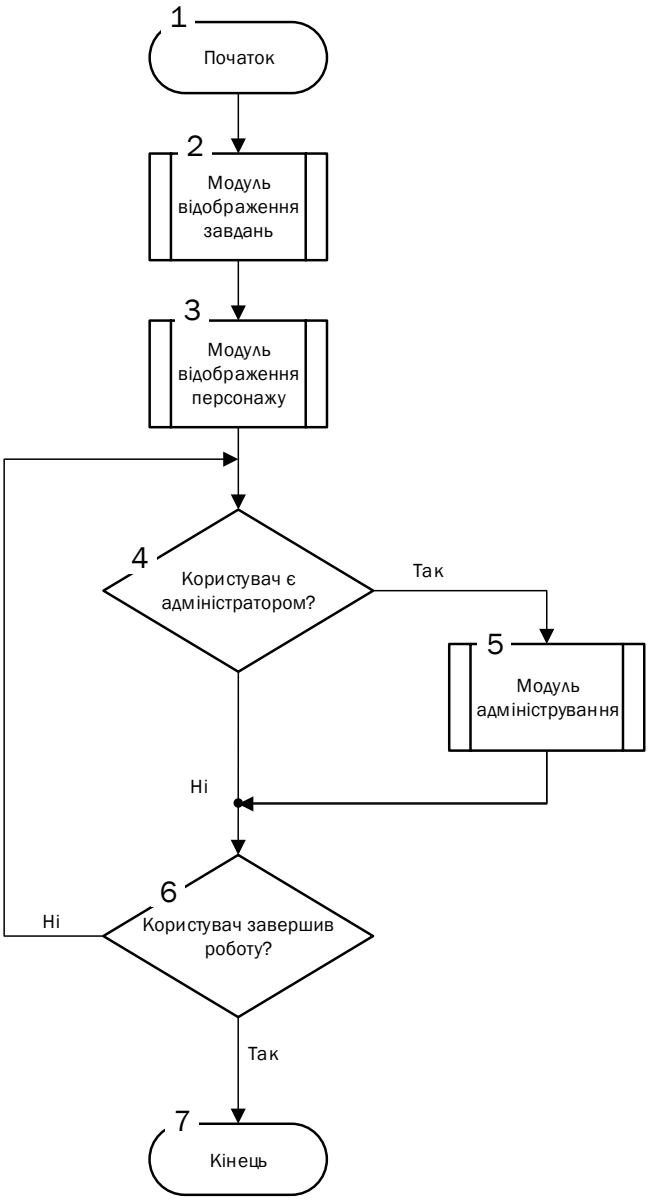

Рисунок 2.8 – Алгоритм мобільного застосунку

Згідно вказаної діаграми, при запуску застосунку спочатку відбувається отримання даних із бази даних. Після цього користувач отримує доступ до двох основних модулів: модулю завдань та модулю персонажу. Якщо користувач має права адміністратора, йому надається додатковий доступ до модуля адміністрування. Цей модуль дозволяє адміністратору керувати завданнями, в тому числі завершувати їх, а також отримувати або витрачати нагороди. Такий функціонал допомагає забезпечити ефективне управління завданнями та мотивацію користувачів до досягнення їх цілей.

Для розробки застосунку необхідно розробити загальний алгоритм, згідно з яким він буде працювати (див. рисунок 2.9).

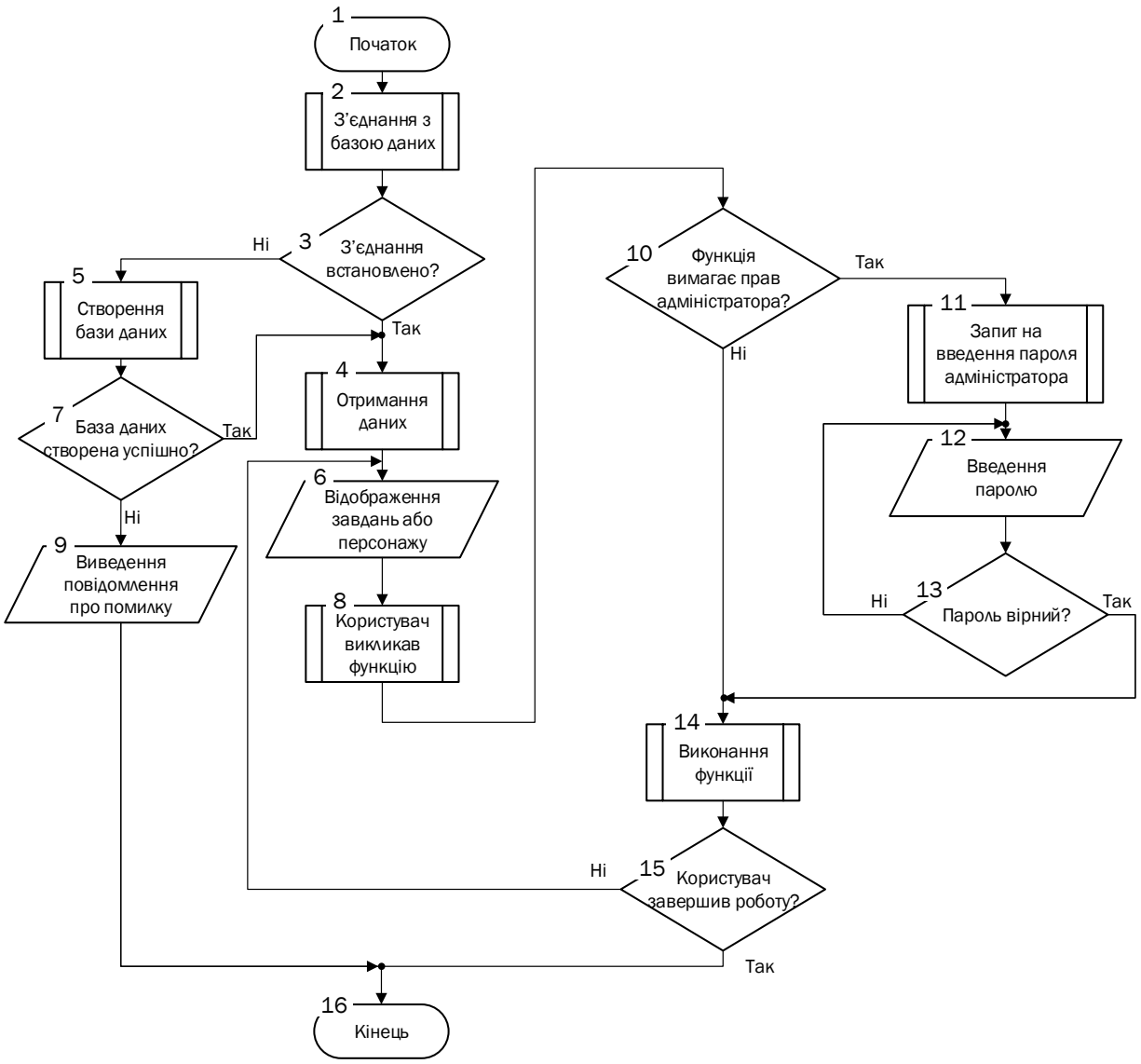

Рисунок 2.9 – Блок-схема алгоритму формування завдань

Згідно цьому алгоритму при запуску застосунку він спробує встановити з’єднання з базою даних, якщо цього не вдасться – база даних буде створена, якщо і цього не вдасться – буде виведено повідомлення про помилку і застосунок завершить свою роботу. Після успішного з’єднання з базою даних або її створення з неї будуть отримані і виведені для користувача дані. Після цього користувач може обрати що робити у застосунку через інтерфейс, якщо обрана користувачем дія вимагає прав адміністратора, потрібно буде ввести пароль адміністратора.

2.4 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних застосунку. Розглянуто основні необхідні модулі застосунку. Було розроблено простий та зручний графічний інтерфейс застосунку. Також було розроблено загальну модель роботи застосунку з використанням блок-схем, крім цього було розроблено загальний спрощений алгоритм роботи застосунку.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІХ МОБІЛЬНОГО ЗАСТОСУНКУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ГЕЙМІФІКАЦІЇ

3.1 Обґрунтування вибору засобів для реалізації програмного застосунку

Розробляючи програмні модулі для мобільного застосунку, яка використовує технології геймифікації, важливо ретельно обирати технології, що використовуються. Мова програмування Kotlin була обрана через свою високу продуктивність, чистоту синтаксису та сумісність з платформою Android. Kotlin надає розробникам широкі можливості для виразного програмування та дозволяє писати чистий та ефективний код. Крім того, Kotlin ідеально поєднується з іншими технологіями, що використовуються в розробці Android-застосунків. Як середовище розробки було обрано середовище Android studio через велику кількість можливостей генерації коду, зручний будівельник xml макетів та підтримка від google.

Рисунок 3.1 – Логотип мови програмування Kotlin

Бібліотека kotlinx надає розробникам Kotlin різноманітні корисні інструменти та розширення для роботи з різними платформами та технологіями [12]. Вона включає в себе такі модулі, як kotlinx.coroutines для

асинхронного програмування, kotlinx.serialization для серіалізації даних, а також kotlinx.html для роботи з HTML у Kotlin [13]. Використання інструментів kotlinx дозволяє розробникам писати більш компактний та ефективний код, що сприяє полегшенню розробки.

Android Jetpack – набір компонентів та інструментів, які допомагають розробникам швидше створювати високоякісні Android-застосунки [14]. Він включає в себе такі модулі, як Lifecycle, ViewModel, Navigation, WorkManager та інші, які спрощують розробку, тестування та підтримку додатків. Використання Android Jetpack дозволяє писати більш стабільний та масштабований код, що сприяє підвищенню продуктивності розробки.

Рисунок 3.2 – Логотип Android Jetpack

Бібліотека Room вбудована в Android Jetpack і надає зручний спосіб роботи з базами даних SQLite на платформі Android. Вона дозволяє використовувати об'єктно-орієнтований підхід до роботи з даними, використовуючи анотації Kotlin

для визначення структури бази даних. Room спрощує взаємодію з базою даних і забезпечує безпечний доступ до даних в Android-додатках [15].

Android Studio – це інтегроване середовище розробки, створене спеціально для розробки додатків для платформи Android. Воно базується на популярному інструменті розробки IntelliJ IDEA від JetBrains. Android Studio надає розробникам широкий спектр інструментів для створення, тестування, налагодження та оптимізації додатків для Android. Серед основних функцій Android Studio можна виділити графічний редактор інтерфейсу користувача, що зображений на рисунку 3.3, інтегровану систему збірки та розгортання, розширений відлагоджувальник, симулятори пристроїв та підтримку різних мов програмування, таких як Java та Kotlin. Android Studio надає зручне та потужне середовище для розробки додатків для Android, що дозволяє розробникам швидко та ефективно створювати якісне програмне забезпечення для мобільних пристроїв [16].

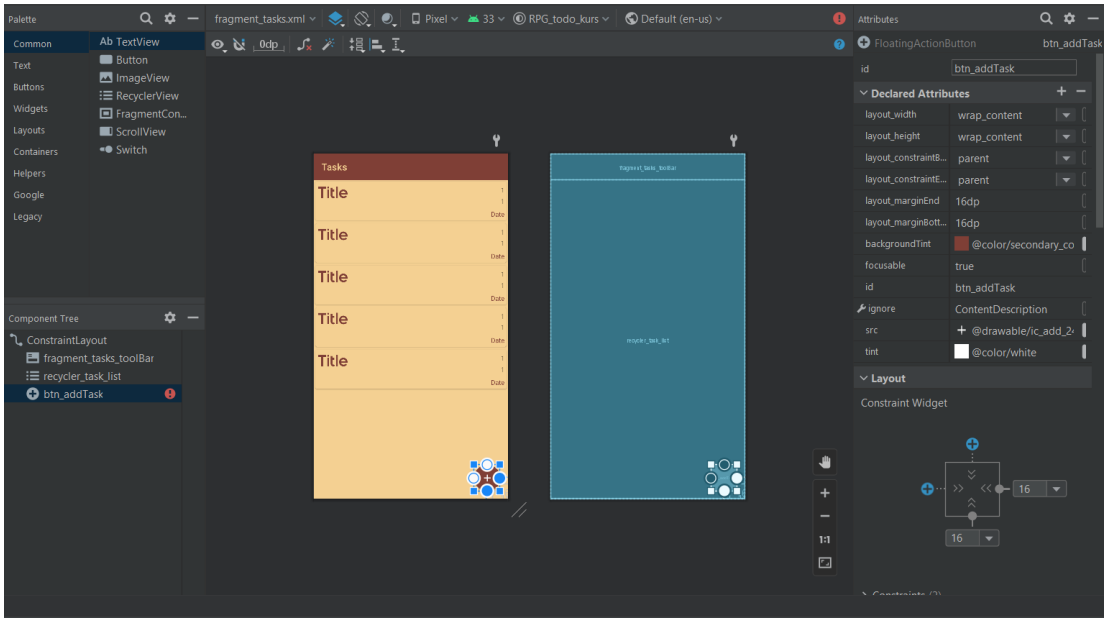

Рисунок 3.3 – Графічний будівельник інтерфейсу користувача Android studio

Будуючи інтерфейс у зручному графічному будівельнику інтерфейсу Android studio Layout editor автоматично генерує xml-розмітку інтерфейсу, частина такого коду зображена на рисунку 3.4 [17].

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/main_color"
    tools:context=".ui.tasks.TasksFragment">


    <androidx.appcompat.widget.Toolbar
        android:id="@+id/fragment_tasks_toolBar"

        android:layout_width="match_parent"
        android:layout_height="wrap_content"


        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"

        android:background="@color/secondary_color"
        app:title="Tasks"
        app:titleTextColor="@color/main_color" />
```

Рисунок 3.4 – Автоматично згенерований xml-код в результаті роботи у будівельнику інтерфейсу

Крім того середовище Android studio надає можливість створити віртуальний пристрій для миттєвого тестування розроблюваного застосунку у межах середовища розробки, приклад такого пристрою зображено на рисунку 3.5.

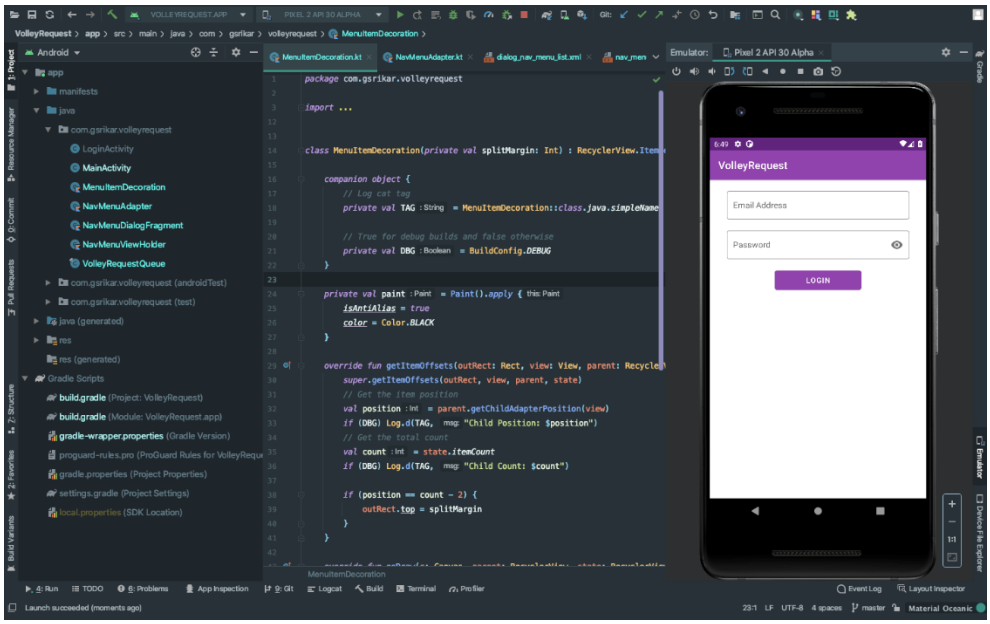

Рисунок 3.5 – Приклад віртуального пристрою у Android studio

Отже, вибір технологій для розробки програмних модулів для мобільного застосунку, що використовує технології геймифікації в Kotlin, kotlinx, Android Jetpack, Room, має вирішальне значення для створення надійного і захоплюючого досвіду. Поєднання цих технологій надає необхідні інструменти для розробки ефективної, надійної та простої в обслуговуванні мобільного застосунку.

3.2 Розробка модуля завдань та відображення персонажу мобільного застосунку

Більшість модулів даного застосунку розроблені у вигляді фрагментів або діалогових фрагментів та відповідних їм об’єктам ViewModel, де фрагмент відповідає за ініціалізацію інтерфейсу із відповідних їм файлів xml, а також за ініціалізацію обробників подій різних клікабельних елементів, а ViewModel – за збереження чутливих даних та зв’язок із іншими модулями, які не відносяться до інтерфейсу напряму, а також мають більші вимоги до ресурсів системи, реалізацію даної архітектури зображено на рисунку 3.6.

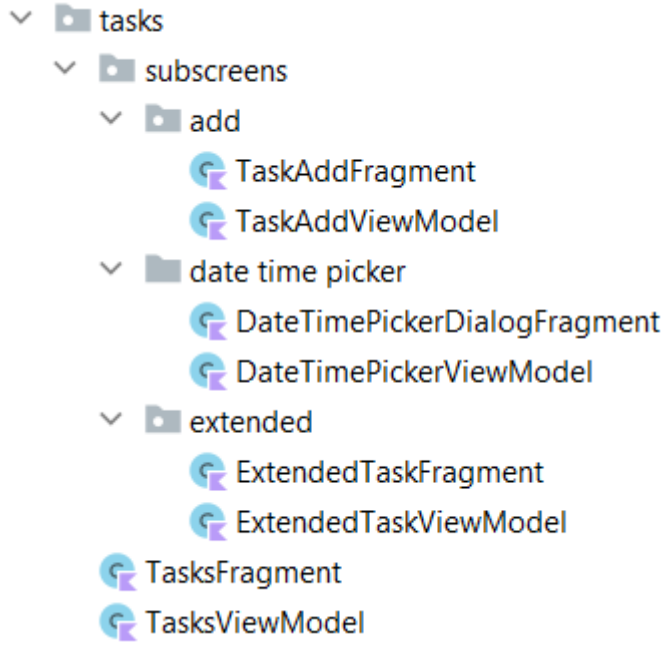

Рисунок 3.6 – Архітектура модулів застосунку

Усі ці фрагменти мають базуватися на головному елементі – MainActivity, на якому розташовано елемент інтерфейсу fragment, що є контейнером для фрагментів-модулів, які перемикаються завдяки елементу BottomNavigationView, що розташований під елементом fragment у розмітці MainActivity, основні дестинації навігації зображено на рисунку 3.7.

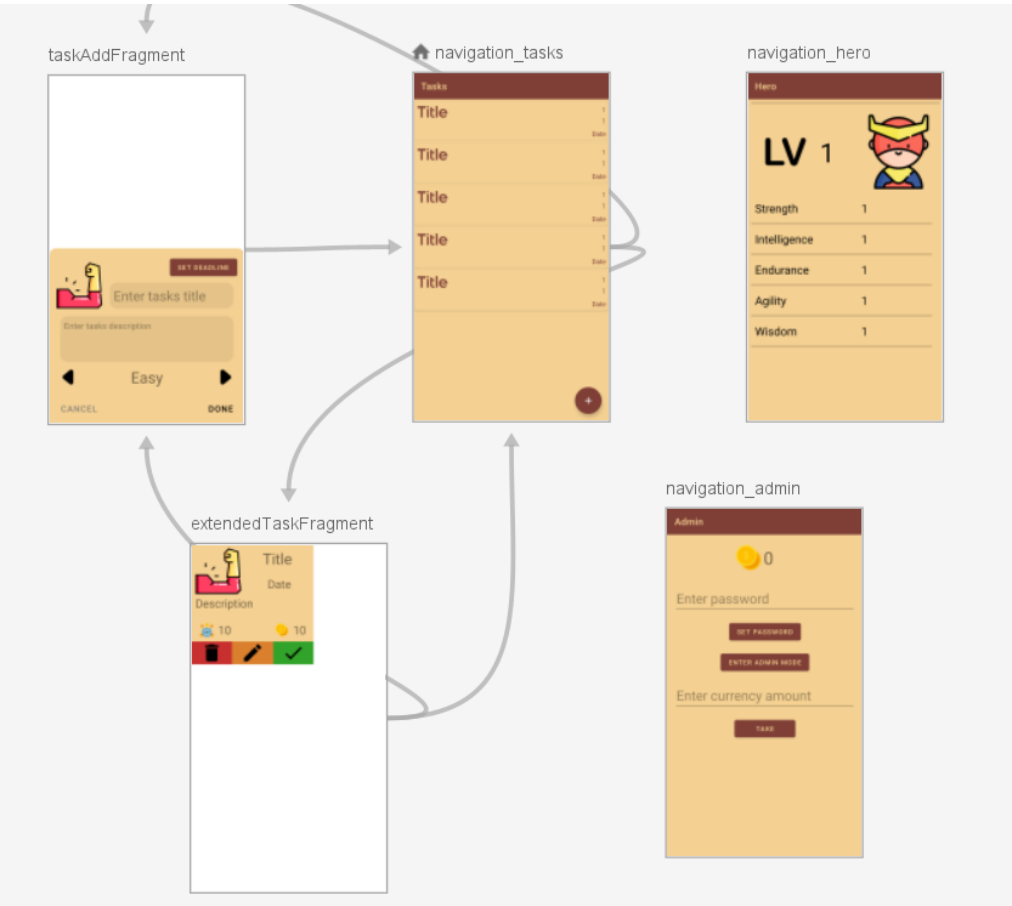

Рисунок 3.7 – Дестинації нижньої навігації

Задача даних модулів – створення та відображення для користувача завдань та персонажа із наданих даних. Для розробки цього модуля було використано інструменти Android jetpack та kotlinx задля реалізації швидкого та надійного побудування інтерфейсу та отримання даних із бази даних у окремому потоці завдяки kotlinx coroutines.

Функція модуля завдань, яка відповідає за головну сторінку для відображення завдань, є важливим компонентом застосунку. Ця функція виконує

кілька ключових завдань, що забезпечують коректне отримання даних для подальшого коректного відображення списку завдань для користувача.

Перш за все, через гарантоване відкриття даного вікна першим функція ініціалізує базу даних викликаючи метод прив’язаного до фрагменту об’єкту ViewModel, після чого починається ініціалізація адаптеру елементу відображення списку головної сторінки завдань та його прив’язка до елементу інтерфейсу.

Крім того у цій функції виконується постійне підключення до бази даних, що дозволяє миттєво отримувати у вигляді списку дані одразу після їх оновлення у базі даних, які після отримання одразу надсилаються у адаптер списку для оновлення інтерфейсу, це дозволяє зменшити код за рахунок відсутності потреби вручну запитувати оновлення даних у фрагменті та надає більш стабільної роботи застосунку. Дане з’єднання створене у окремому робочому потоці завдяки kotlinx.coroutines, через що отримання даних не блокує графічний потік, попереджаючи затримки відповіді на дії користувача.

Іншою важливою частиною модуля інтерфейсу є функції, що відповідають за ініціалізацію інтерфейсу та обробників клікабельних елементів у модулі відображення завдань. Ці функції забезпечують інтерактивність і зручність використання застосунку, дозволяючи користувачам взаємодіяти з різними елементами інтерфейсу.

Вони відповідають за натискання кнопки додання завдань та за натискання на елементи списку.

Оскільки за обробку елементів списку відповідає адаптер списку, ініціалізація обробнику натискання також має знаходиться у ньому, проте викликати елементи навігації з нього не коректно, через що було створено companion object для отримання даних про натиснутий елемент. Цей об’єкт отримує натиснутий екземпляр класу, після чого здійснює навігацію до відповідного вікна передаючи екземпляр завдання усередині об’єкта Bundle за ключем-константою вказаним у окремому файлі.

Модуль відображення завдань також пов’язаний з модулем додання завдань та модулем розширеної інформації про завдання.

Функція модуля додання завдань, що відповідає за ініціалізацію початкових даних та заповнення ними інтерфейсу починає свою роботу з перевірки на наявність надісланих до фрагменту даних, якщо дані наявні – ними буде заповнено пусті поля, це зроблено для того, щоб фрагмент додання завдань можливо було використовувати не тільки задля створення нових, а й для редагування старих завдань, що зменшить кількість окремих екранів, а відповідно і кількість коду. Після цього відбувається оновлення елементів які потребують оновлення у будь-якому випадку.

Крім функції ініціалізації інтерфейсу та даних було розроблено функцію ініціалізації обробників натискання клікабельних елементів інтерфейсу. У неї входять ініціалізації обробників таких клікабельних елементів як: кнопка «скасувати», яка відміняє створення або редагування завдань, кнопка «закінчити» яка зберігає нове або замінює старе завдання, кнопка «Встановити дату» яка відкриває вікно вибору дати та часу дедлайну, та іконка характеристики при натисканні якої відкривається вікно вибору характеристики для прокачки.

Функції деяких кнопок різняться в залежності від початкових даних, так, наприклад, при відкритті через кнопку редагування завдання кнопка «завершити» буде отримувати попереднє завдання, порівнювати його з новим і якщо відбулись якісь зміни – замінювати існуюче завдання, а при створенні нового буде тільки додавати новий запис у базу даних.

У додатку також розроблено фрагмент, який призначений для відображення розширеної інформації про завдання. Цей фрагмент дозволяє користувачам отримувати детальну інформацію про кожне завдання, що включає додаткові атрибути та деталі, які не відображаються на основному екрані. Цей модуль також надає користувачу можливість проводити операції із завданням.

Функція відповідальна за ініціалізацію даних та інтерфейсу цього фрагменту починає свою роботу із отримання надісланих до неї даних у вигляді Bundle, отримує дані про завдання із цього об’єкту, після чого ініціалізує інтерфейс отриманими даними і форматує дату яка надана у вигляді timestamp у зрозумілий

для людини вигляд, зважаючи на обрані користувачем системні налаштування регіону та формату часу.

Частина цього фрагменту відповідальна за керування виклику функцій при натисканні на клікабельні елементи ініціалізує три кнопки що знаходяться на цьому фрагменті – кнопку «Видалити», кнопку «Редагувати» та кнопку «Завершити».

При натисканні на кнопку «Видалити» або кнопку «Завершити» відбудуться такі дії як перевірка чи увімкнений режим адміністратора, після чого за умови увімкненого цього режиму відбудеться запит до об’єкта ViewModel, що прив’язаний до цього фрагменту про відповідну дію, який, у свою чергу створить запит у базу даних про ці дії у окремому потоці задля уникнення блокування графічного потоку завдяки coroutines. У випадку відсутності прав адміністратора буде відображено повідомлення у вигляді toast про помилку через неактивований режим адміністратора.

При натисканні на кнопку «Редагувати» перевірки відбуватись не буде, першою дією цієї функції є створення об’єкту Bundle задля передачі даних до іншого фрагменту, поміщення інформації про відкрите завдання до цього об’єкту та навігація із встановленими даними до фрагменту що відповідає за створення та редагування завдань.

Після виконання певних операцій із завданнями можуть відбуватися зміни у віртуальному персонажі, які відображаються у спеціальному модулі, що призначений для відображення персонажу.

Цей модуль надає можливість користувачам спостерігати за змінами, які стосуються їхнього віртуального образу, такі як зміна вигляду, характеристик або інших атрибутів. Відображення персонажу надає більше інтерактивності та відчуття живості у застосунку, а також дозволяє користувачам більш глибоко зануритися у віртуальний світ.

Дана функція отримує інформацію із попередньо ініціалізованого об’єкта ViewModel для цього модуля. Цей об’єкт слугує для зв’язку із SharedPreferences застосунку у яких зберігається інформація про персонажа за збереженим у застосунку ключем-константою.

Функція об’єкта ViewModel, що використовується для отримання даних персонажу виконується у головному потоці через відсутність на сторінці потреби постійного оновлення та клікабельних елементів що потребують постійного доступу до них навіть перед завантаженням даних. У ній виконується отримання даних із SharedPreferences застосунку за відповідним ключем, дані у сховищі зберігаються у вигляді строки Gson, після отримання якої вона перетворюється у об’єкт персонажа HeroModelUI, що дозволяє використовувати його у інтерфейсі застосунку.

При відкритті відповідної сторінки процес розпочинається з отримання даних із бази даних у робочому потоці. Після цього отримані дані перетворюються на відповідні об'єкти або об'єкт, для завдань та персонажа відповідно.

У випадку персонажа, коли дані оброблено і перетворено, вони відображаються на екрані персонажа користувача, щоб користувач міг бачити оновлену інформацію про персонажа.

Щодо завдань, після перетворення дані надсилаються до адаптеру RecyclerView, зручного та ефективного елементу для відображення різних видів списків, який відповідає за відображення списку у зручному та ефективному вигляді. Після певної обробки адаптер відображає ці дані на екрані у вигляді списку завдань.

Функція зазначеного адаптеру, що відповідає за ініціалізацію відображення кожного елементу завдання із списку RecyclerView отримує дані із об’єкту інтерфейсу про поточну позицію списку яку потрібно відобразити, після чого отримує відповідний елемент із списку даних та заповнює ним окремий елемент інтерфейсу списку, а також встановлює для нього обробник натискання на елемент задля подальшої передачі даних на головну сторінку для обробки.

Крім того у адаптері реалізовано додаткові функції отримання списку, яка після отримання даних сортує їх за датою та встановлює у внутрішній список об’єктів адаптера, після чого дає запит на оновлення даних інтерфейсу списку RecyclerView.

У даному застосунку завдання також можуть бути наділені можливістю вибору дедлайну для їх виконання. Отже, виробництво зручного та ефективного інструмента вибору дати та часу є важливим елементом функціоналу.

Через відсутність зручного та компактного елементу для отримання дати та часу був розроблений власний варіант обирача дати та часу, який був інтегрований у застосунок. Цей інструмент дозволяє користувачам зручно обирати та встановлювати дедлайни для своїх завдань, що сприяє покращенню організації та контролю за їх виконанням.

Цей елемент складається з двох головних елементів для вибору дати та часу – NumberPicker та TimePicker у форматі spinner та допоміжних елементів у вигляді кнопок встановлення дати та скасування встановлення, а також елементу що відображає поточно вибрану користувачем дату та час.

Після відкриття даного діалогу починається ініціалізація елементів NumberPicker та TimePicker для вибору дати та часу відповідно, селектор дати реалізовано у вигляді одного стовпця з елементами у форматі «день тижня, число, місяць», якщо обирається елемент іншого року – буде відображено і рік, таке відображення дійсне для всіх елементів крім перших двох, які представлено як «Сьогодні» та «Завтра» при зміні значення у цьому елементі відбувається заміна глобальної змінної цього фрагменту та запит на оновлення елементу що відображає поточно обрану дату та час.

Селектор часу зображено у вигляді двох або трьох стовпців залежно від того чи обрано 24 або 12 часовий формат часу, які відповідають за час, хвилину та частину дня відповідно. Як і селектор дати вони перезаписують глобальну змінну дати та часу і надають запит на оновлення елементу відображення.

Кнопки завершити та скасувати відповідають за завершення та повернення даних назад до елементу що викликав даний діалог або завершення вибору без збереження даних відповідно.

Крім того було створено підмодуль адаптера RecyclerView, який відповідає за визначення залишкового часу до дедлайну виконання завдання. Цей підмодуль

має на меті полегшити користувачам відстеження термінів та нагадувати про наближення кінцевих строків виконання завдань.

Функціональність підмодуля полягає у відстеженні залишкового часу до дедлайну кожного завдання. Якщо часу до кінця виконання завдання залишається менше 24 годин, підмодуль автоматично помічає дату жовтим кольором. Це візуальне попередження дозволяє користувачам швидко звернути увагу на завдання, які потребують негайної уваги. У випадку, якщо дедлайн вже минув, підмодуль змінює колір дати на червоний, що сигналізує про прострочене завдання.

Даний підмодуль використовує дати у вигляді timestamp, порівнюючи їх різниці з відповідними величинами і проводячи відповідні операції з інтерфейсом елементу.

Отже, модулі завдань та персонажу було розроблено у повному обсязі, вони включають у себе певну кількість менших підмодулів, таких як модуль відображення, створення та редагування, розширеного відображення завдань, адаптерів, селекторів дати та часу, а також модуль відображення завдання які працюючи разом та обмінюючись даними складають два основних модулі та основну та головну частину функціоналу програми.

3.3 Розробка модуля керування базою даних завдань та персонажу

Задача даного модуля – завантажувати дані у базу даних для зберігання та отримувати їх з цієї бази даних при необхідності. Крім того база даних повинна підтримувати постійний зв’язок з застосунком і автоматично надсилати дані при їх модифікації. Для реалізації цього модуля було використано бібліотеку room для спрощеного створення надійної бази даних та створення надійних запитів відповідно до строки SQL запиту. Крім того було використано androidx flow для зберігання даних із бази даних та можливості автоматичного оновлення даних при їх модифікації.

Модуль керування базою даних розподілений на декілька основних частей, серед яких:

– клас ініціалізації бази даних;

– клас операцій із базою даних;

– клас даних моделі завдань;

– клас що містить виклики функцій керування базою даних.

Архітектуру розроблюваної бази даних зображено на рисунку 3.8. Її можна змінювати в залежності від потреб розробника.

Завдяки використанні бібліотеки Room, усе що потрібно для ініціалізації та встановлення з'єднання з базою даних – це константа назви бази даних, дата клас який буде використаний як шаблон для нової таблиці бази даних SQL. Більшість шаблонного коду що потрібна для цього генерується автоматично самою бібліотекою, ми ж тільки вказуємо необхідні нам параметри, це дозволяє значно скоротити та очистити код від надлишкових елементів та уникнути небажаних помилок.

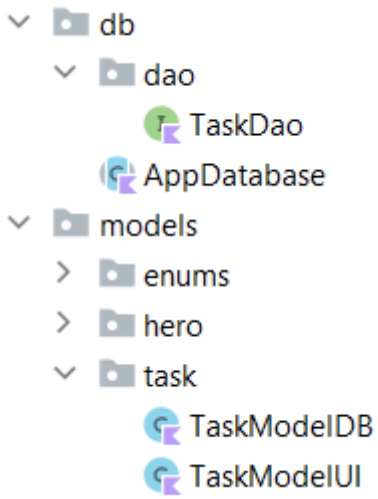

Рисунок 3.8 – Архітектура класів бази даних

Частину модуля керування базою даних відповідальною за операції із базою даних реалізовано у вигляді інтерфейсу, цю частину зображено на рисунку 3.9. Даний інтерфейс визначає які команди будуть надсилатись у базу даних для операцій з нею, код який необхідний для виконання цих функцій також генерується бібліотекою Room.

```
@Dao
interface TaskDao {
    @Query("SELECT * FROM $TASK_DB_TABLE_NAME")
    fun getAll(): Flow<List<TaskModelDB>>

    @Insert
    suspend fun insert(user: TaskModelDB)

    @Delete
    suspend fun delete(user: TaskModelDB)

}
```

Рисунок 3.9 – Частина модулю керування базою даних відповідальна за операції

Як можна побачити, інтерфейс виділяється певними тегами «@Dao», «@Query», «@Insert» та «@Delete», вони слугують відмітками для бібліотеки room, що позначають за що відповідають конкретні елементи, що дозволяє бібліотеці генерувати запити та шаблонний код.

Клас, відповідальний за шаблон таблиці бази даних реалізовано як дата клас TaskModelDB, він має тег «@Entity» з параметром «tableName» задля вказання назви таблиці що буде зберігати об’єкти даного класу.

Даний клас має такі поля як:

1. ID: цей стовпець відображає ідентифікатор завдань у базі даних та у програмі, у базі даних він виступає як PrimaryKey, він не може співпадати з іншими записами та генерується автоматично, про що свідчить переданий до бібліотеки параметр «autoGenerate = true».

2. Title: цей стовпець відображає заголовок завдання яке слугує задля скороченого викладення змісту завдання.

3. Desc: цей стовпець відображає опис завдання, він слугує для розширеного опису завдання або його кроків.

4. Creation_date: цей стовпець відображає дату коли завдання було створене, зазвичай за цим критерієм завдання сортуються у списку.

5. Expiration_date: цей стовпець відображає дату, до якої завдання потрібно виконати, від нього залежить відображення елементу у списку.

6. Currency_reward: цей стовпець відображає винагороду віртуальною валютою за завдання, яку потім можна списати у фрагменті адміністрування для отримання нагород.

7. Exp_reward: цей стовпець відображає нагороду балами досвіду, що при завершенні нараховуються на рахунок віртуального персонажу.

Цей клас має встановлені значення за замовчуванням на випадок якщо користувач не введе їх, що запобігає помилкам. Він використовується як шаблон для створення таблиці бібліотекою Room, про що свідчить константа яка визначає назву таблиці бази даних.

Частини класів що викликають функції бази даних зберігаються у об’єктах ViewModel різних класів, вони викликаються здебільшого з використанням kotlinx.coroutines задля уникнення блокування користувацького інтерфейсу у момент роботи з базою даних [18].

Задля уникнення повторення коду об’єкт бази даних було ініціалізовано як глобальну змінну для всього додатку, що робить її доступною з будь-якого модулю застосунку.

При запуску програми і завантаженні стартової сторінки одразу викликається метод модуля керування базою даних для створення з’єднання з базою даних, у випадку якщо база даних не знайдена, нова база даних буде створена на пристрої. Після успішного встановлення з’єднання база даних готова для отримання запитів від інших модулів.

3.4 Розробка модуля адміністрування завдань та нагород

Метою даного модуля є контроль над статусом адміністратора та його паролем, щоб забезпечити можливість завершення або видалення завдань. Крім того цей модуль передбачає керування віртуальною валютою на рахунку персонажа.

Для реалізації цієї функціональності застосовуються різноманітні інструменти, зокрема управління SharedPreferences. SharedPreferences дозволяє

зберігати дані безпосередньо на пристрої користувача, що забезпечує зручний та швидкий доступ до необхідної інформації. Однак, оскільки зберігання паролів є вразливим місцем, необхідно вживати додаткових заходів безпеки.

Для уникнення несанкціонованого доступу та зламу, використовуються сучасні інструменти шифрування, які забезпечують надійне та безпечне зберігання паролів. Шифрування даних дозволяє захистити конфіденційну інформацію від потенційних загроз та гарантує, що навіть у разі зламу пристрою, зловмисники не зможуть отримати доступ до захищених даних.

Застосування цих інструментів не лише підвищує рівень безпеки, але й забезпечує конфіденційність під час управління адміністративними правами та паролем у застосунку. Такий підхід гарантує, що лише авторизовані користувачі мають доступ до важливої інформації, тим самим знижуючи ризик виникнення витоків даних та інших загроз [19].

Цей модуль складається із трьох основних частин:

– частина відповідальна за шифрування та розшифрування;

– частина відповідальна за керування правами адміністратора та віртуальною валютою;

– частина відповідальна за відображення елементів інтерфейсу.

Частина модуля адміністрування відповідальна за шифрування реалізована окремим класом PasswordManager, функція цього класу відповідальна за шифрування отримує пароль у вигляді строки, після чого генерує «сіль» із константи, секретний ключ та шифрувальник за запитом, після чого шифрує пароль та повертає його до попереднього методу для подальшого збереження.

Для шифрування використовується метод шифрування AES, оскільки він є одним із найбільш поширених та надійних методів шифрування даних, використовуваних сьогодні. Він був прийнятий як стандарт шифрування і здобув визнання у всьому світі завдяки своїй ефективності та безпеці.

Крім того для отримання паролю для подальшої перевірки потрібен метод для розшифрування збереженого паролю.

Функція відповідальна за розшифрування отримує введений користувачем пароль, після чого перевіряє чи збережений зашифрований пароль, після чого отримує його та розшифровує. Ця функція викликається також при виклику функції сбросу паролю, при цьому вона ще додатково замість повернення змінної Boolean видаляє збережений пароль із SharedPreferences.

Частина модуля адміністрування відповідальна за встановлення або скидання паролю реалізована однією функцією, яка змінює свою поведінку в залежності від потреби користувача.

У цій функції відбувається отримання, розшифрування та порівняння збереженого та введеного паролю, якщо вони співпадають – вмикається режим адміністратора, якщо ж ні – відображається повідомлення про помилку.

Одним із ключових аспектів модуля адміністрування є його здатність ініціювати та керувати інтерфейсом додатку. Це означає визначення конкретних елементів інтерфейсу, які будуть відображатись на екрані користувача, а також визначення їх функціональності.

У даній частині модуля реалізується логіка відображення інтерфейсу та його взаємодія з користувачем. Вона також відповідає за надання необхідного функціоналу для кожного елемента інтерфейсу, що забезпечує його коректну роботу та взаємодію з іншими частинами додатку.

При відкритті сторінки адміністрування перевіряється наявність збереженого паролю, відповідно до результату перевірки ініціалізується інтерфейс та функції кнопок. При виконанні дій на цій сторінці відбуваються операції з SharedPreferences або з глобальною змінною відповідальною за стан адміністратора.

Крім цього було реалізовано функції керування віртуальною валютою, яка відображається у верхній частині модуля адміністрування.

Функції керування валютою працюють схожим чином до інших функцій цього модулю і потребують режима адміністратора. При введенні валюти та натисканні кнопки зняття валюти перевіряється чи увімкнено режим адміністратора, якщо його увімкнено, через об’єкт ViewModel із SharedPreferences

отримується об’єкт персонажу, після чого знімається задана кількість валюти, і персонаж зберігається знову.

3.5 Висновки

У третьому розділі було детально обґрунтовано вибір мови програмування та технологій, що використовувалися для розробки застосунку. Проаналізувавши специфічні потреби програмного продукту, було вирішено зупинитися на мові програмування Kotlin, яка на сьогоднішній день є одним із найсучасніших і найзручніших інструментів для розробки під платформу Android. Завдяки своїй простоті та виразності, Kotlin значно полегшує процес написання коду та покращує його читабельність.

Для зручності створення та налаштування графічного інтерфейсу було обрано стандартний метод використання xml файлів у середовищі розробки Android Studio. Це дозволяє легко розробляти та змінювати компоненти інтерфейсу, забезпечуючи водночас їхню високу адаптивність та сумісність з різними пристроями.

В якості бази даних було обрано бібліотеку Room, яка є офіційною бібліотекою для роботи з базами даних у середовищі Android. Room забезпечує простоту інтеграції, високу продуктивність і безпеку даних, що є критично важливим для сучасних мобільних застосунків. Для забезпечення ефективної роботи застосунку було реалізовано асинхронну обробку більш затратних по часу завдань. Це дозволяє уникнути блокування головного потоку, забезпечуючи плавну та безперебійну роботу користувацького інтерфейсу навіть під час виконання складних операцій.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування застосунку

Тестування мобільного застосунку на Android – це процес перевірки функціональності, продуктивності, безпеки та сумісності програмного забезпечення на пристроях, які працюють під управлінням операційної системи Android. Під час тестування проводяться різні види тестів, включаючи модульні тести, інтеграційні тести, системні тести, взаємодійні тести та інші.

Метою тестування є забезпечення якості програмного забезпечення, виявлення та виправлення помилок та недоліків перед випуском продукту на ринок. Тестування мобільних застосунків на Android допомагає забезпечити оптимальну роботу програм на різних пристроях і забезпечити задоволення користувачів від їх використання [20].

Проведення тестування мобільного застосунку на Android – це складний процес, який вимагає систематичного та методичного підходу для забезпечення високої якості продукту. Перш за все, необхідно ретельно спланувати всі етапи тестування. Це включає визначення обсягу тестування, типів тестів та встановлення пріоритетів для них.

Далі важливим кроком є створення тестових сценаріїв. Кожен сценарій повинен містити набір тестових кейсів, які охоплюють різні функції та можливі сценарії використання застосунку. Ці тестові сценарії допомагають систематизувати процес тестування та забезпечити вичерпне охоплення всіх можливих варіантів використання програмного забезпечення.

Після цього необхідно підготувати тестові середовища. Це означає забезпечення доступу до різних пристроїв з різними версіями Android та встановлення необхідного програмного забезпечення для тестування.

Саме тестування включає в себе виконання підготовлених тестових сценаріїв згідно з планом тестування. Під час виконання тестів важливо ретельно

реєструвати результати, виявляти помилки та їх документувати для подальшого виправлення.

Опціональним етапом може бути автоматизація тестування, що допомагає автоматизувати повторювані сценарії тестування та прискорити процес виявлення помилок.

Після виконання тестів важливо провести аналіз результатів. Це включає оцінку покриття тестами, аналіз виявлених помилок та їх пріоритетів для виправлення, а також підготовку звіту про результати тестування.

На завершальному етапі може бути необхідне повторне тестування після виправлення помилок для перевірки їх ефективності. Такий цикл тестування допомагає забезпечити високу якість мобільного застосунку на Android та задовольнення користувачів від її використання.

Результат тестування використання ресурсів системи при роботі застосунку зображено на рисунку 4.1.

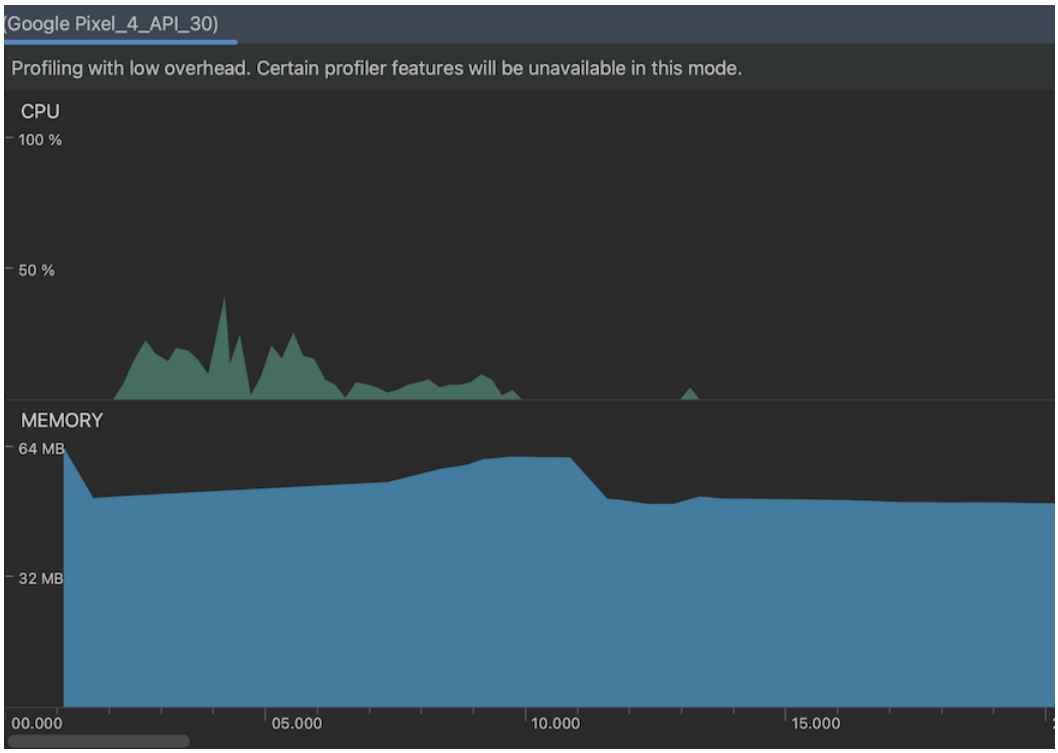

Рисунок 4.1 – Тестування використання ресурсів системою

Першим кроком тестування інтерфейсу є перевірка головної навігації застосунку, зокрема Bottom Navigation View, та коректності її транзицій. Bottom Navigation View є ключовим елементом інтерфейсу, який забезпечує швидкий доступ до основних розділів застосунку. Важливо, щоб перехід між цими розділами був плавним і коректним, без будь-яких помилок або затримок.

На рисунку 4.2 зображено екрани транзицій застосунку. На цьому рисунку показані три основні вікна, які відповідають різним розділам застосунку, а також призначенні їм кнопки навігації. Кнопки навігації обведені червоним кольором, що дозволяє легко ідентифікувати до якого вікна вони прив’язані.

Рисунок 4.2 – Транзиції bottom navigation view

Як можна побачити – транзиції навігації працюють коректно, посилаючись на відповідні їм екрани.

Наступним кроком буде перевірка сторінки завдань та її транзицій. При натисканні кнопки з плюсом має відкриватись сторінка створення завдання без

даних та зникати нижня навігація. Інша транзакція – при натисканні на вже створене завдання має відкриватись не редагуєма сторінка опису даного завдання, де зображена більш розширена інформація про завдання та можливі дії з цим завданням. Клікабельні елементи цієї сторінки зображено на рисунку 4.3.

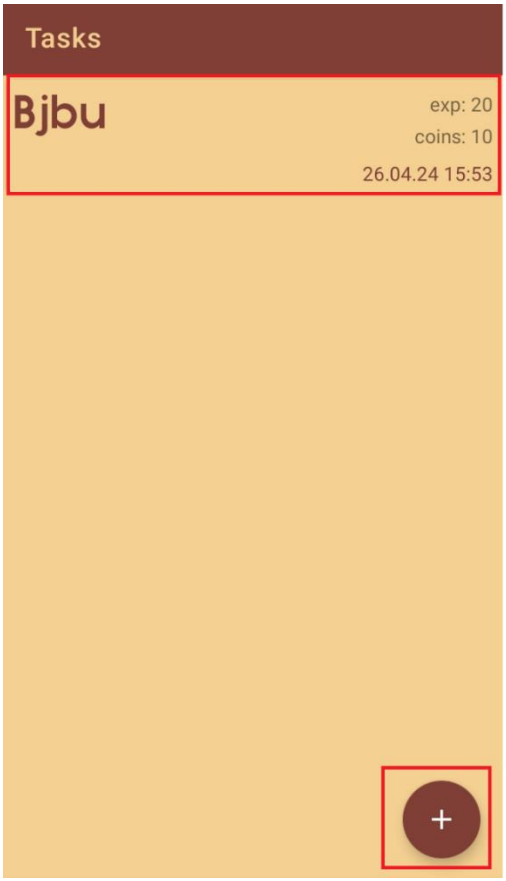

Рисунок 4.3 – Сторінка завдань та її клікабельні елементи

Переконавшись що сторінки на які посилаються дані клікабельні елементи відображаються коректно та ініціалізують правильні дані, можна переходити до тестування похідних сторінок з цього модуля.

Першою такою сторінкою є сторінка створення завдання. При відкритті даної сторінки через сторінку відображення завдань усі поля мають бути пусті і відображати підказки, складність завдання має бути встановлена як «проста», характеристика для прокачки – сила, а кнопка вибору дати дедлайну писати «встановити дату». Клікабельні елементи даної сторінки – повернення, завершення, встановлення дати, дві кнопки зміни складності та іконка

характеристики. Дана сторінка з виділеними клікабельними елементами зображена на рисунку 4.4.

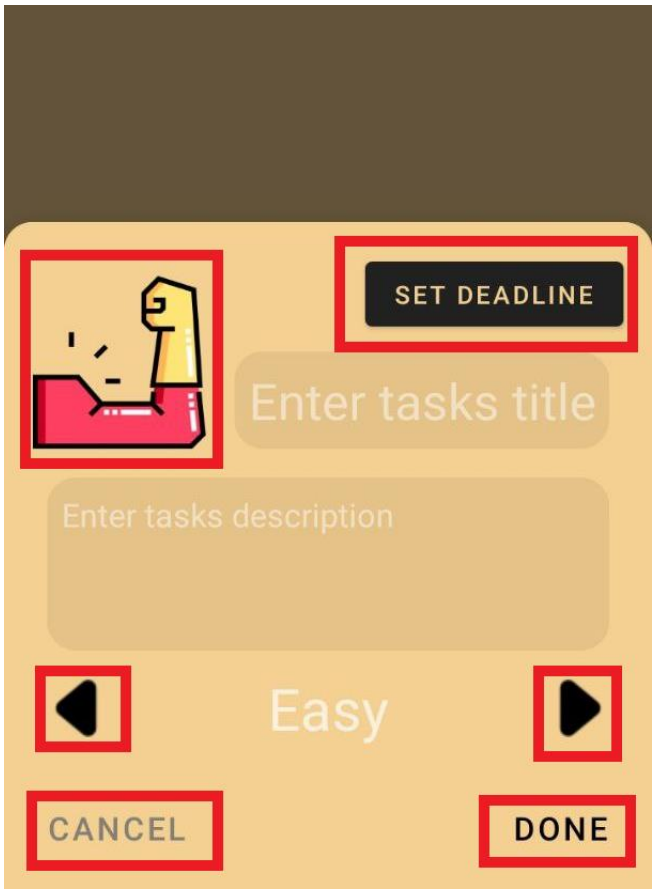

Рисунок 4.4 – Сторінка додання завдання

При натисканні на кнопку повернення створення завдання скасовується та відбувається транзакція на головну сторінку завдань. При натисканні на кнопку завершення створюється завдання при наявності заголовку або опису завдання та відображається на головній сторінці. При натисканні кнопки встановлення дати відкривається кастомний елемент вибору дати DateTimePicker. При натисканні на стрілки змінюється складність завдання, а, відповідно і нагороди за це завдання. При натисканні на картинку характеристики відкривається меню вибору характеристики.

Усі клікабельні елементи відображаються, ініціалізують дані та працюють коректно, отже можна переходити до наступного екрану.

Наступним кроком тестування даної частини програми є тестування сторінки розширеного опису завдання. Дана сторінка має містити більше інформації про завдання та мати три клікабельних елементи для здійснення певних дій з нею. Перший елемент – видалення, має спрацьовувати тільки при увімкненому режимі адміністратора та повертати до головного меню. Редагування має переносити на сторінку з створення завдання із заповненими полями. Завершення має спрацьовувати тільки при увімкненому режимі адміністратора та повертати до головного меню пересилаючи нагороди до персонажу. Клікабельні елементи даної сторінки зображено на рисунку 4.5.

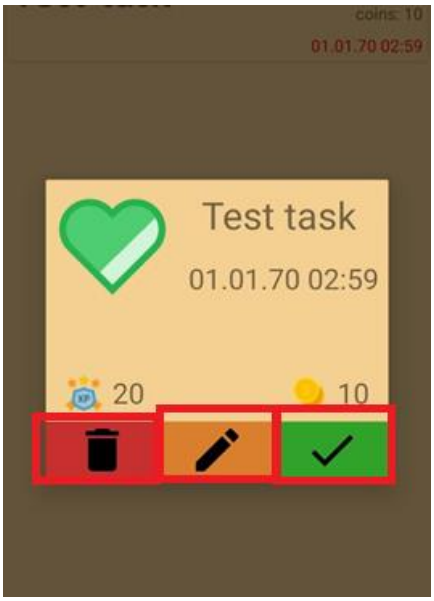

Рисунок 4.5 – Клікабельні елементи сторінки розширеної інформації про завдання

Як видно з рисунку, дані про завдання відображаються коректно, отже ініціалізація даних працює як очікувано. При натисканні на усі клікабельні елементи даної сторінки відбуваються зазначені вище очікувані події, отже усі елементи даного екрану працюють коректно.

Сторінка відображення віртуального персонажа має відображати усю інформацію про персонажа користувача у простій для розуміння формі. Після виконання завдань повинні бути видимі зміни у характеристиках чи полосі досвіду персонажу.

Екран відображення інформації про персонажа після виконання деякої кількості завдань зображено на рисунку 4.6.

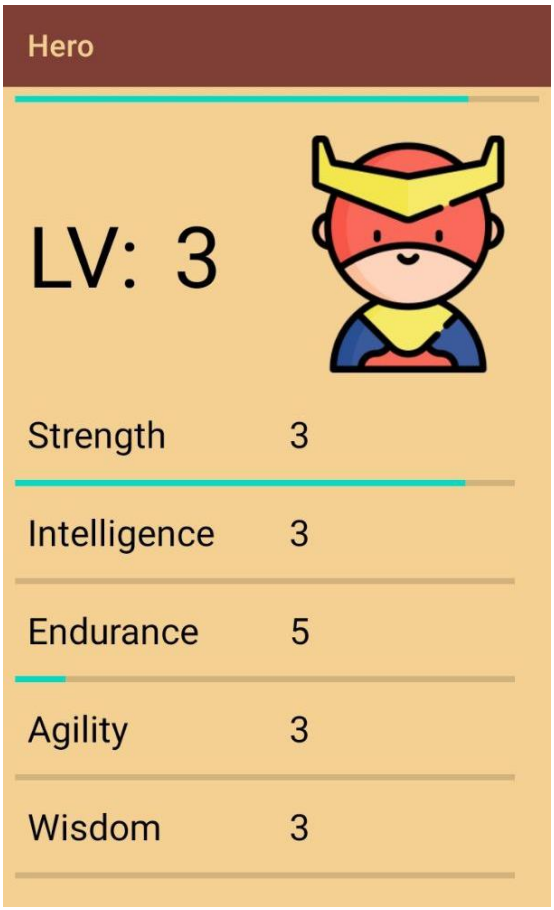

Рисунок 4.6 – Сторінка відображення персонажу

Як можна побачити – інтерфейс ініціалізується нормально, усі бали які мали бути зараховані – зберігаються та відображаються коректно.

Після досягнення нового рівня персонаж отримає прибавку до усіх характеристик. При досягненні 100 очок у окремій характеристиці ця характеристика збільшиться на 1, що відбулося вже двічі для характеристики витривалості.

Сторінка адміністрування пропонує назначити пароль та при його наявності – увійти у режим адміністратора чи скинути пароль при вказанні вірного паролю, а також елементи для зняття певної кількості віртуальної валюти з рахунку

персонажу у режимі адміністратора. При вказуванні невірного паролю буде відображено повідомлення про невірний пароль, це зображено на рисунку 4.7.

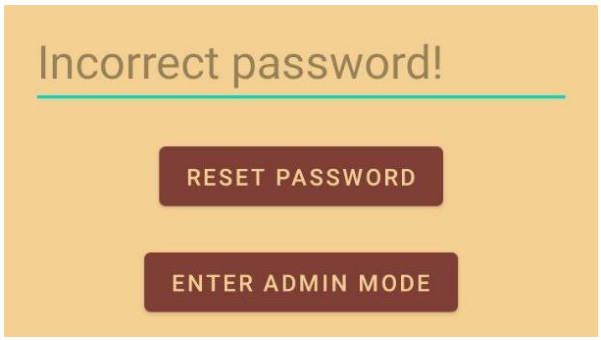

Рисунок 4.7 – Повідомлення про невірний пароль на сторінці адміністрування

При внесені різних змін у стані адміністрування та паролю відбуваються відповідні зміни у інтерфейсі застосунку, що відповідає очікуваному.

При обранні кількості валюти для зняття при увімкненому режимі адміністратора певна кількість валюти буде знята з рахунку, перевірка коректності зняття зображена на рисунку 4.8.

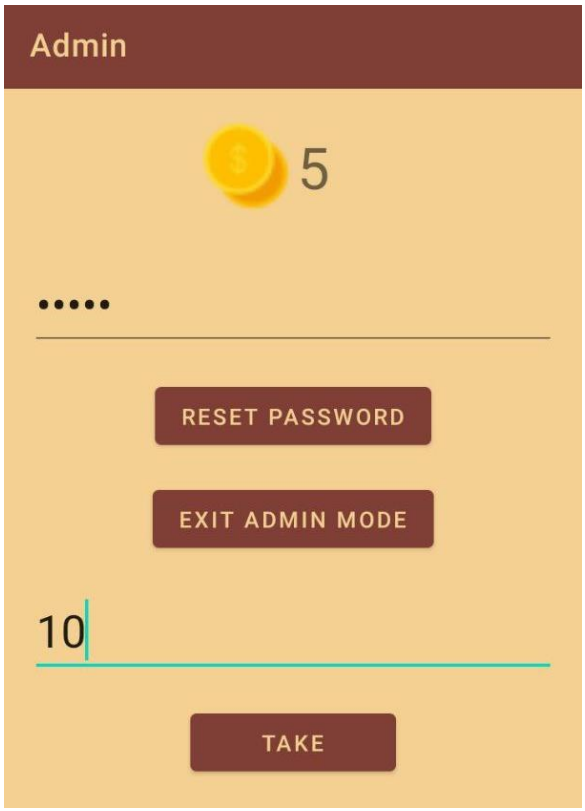

Рисунок 4.8 – Перевірка зняття віртуальної валюти з рахунку

При введенні суми валюти, що перевищує наявну або при вимкненому режимі адміністратора з’являється повідомлення про помилку у вигляді елементу Toast-повідомлення, що є невеликим спливаючим вікном, яке на короткий час з’являється на екрані і містить текстове повідомлення. Це повідомлення не блокує інтерфейс користувача і автоматично зникає через кілька секунд. Ця поведінка є очікуваною, отже даний екран працює коректно.

Екран вибору характеристики являє собою список, при натисканні елементу якого буде змінена іконка інтерфейсу відповідаюча за зображення характеристики та при завершенні завдання визначаюча яка характеристика отримає прибавку очків. Результат тестування зображено на рисунку 4.9.

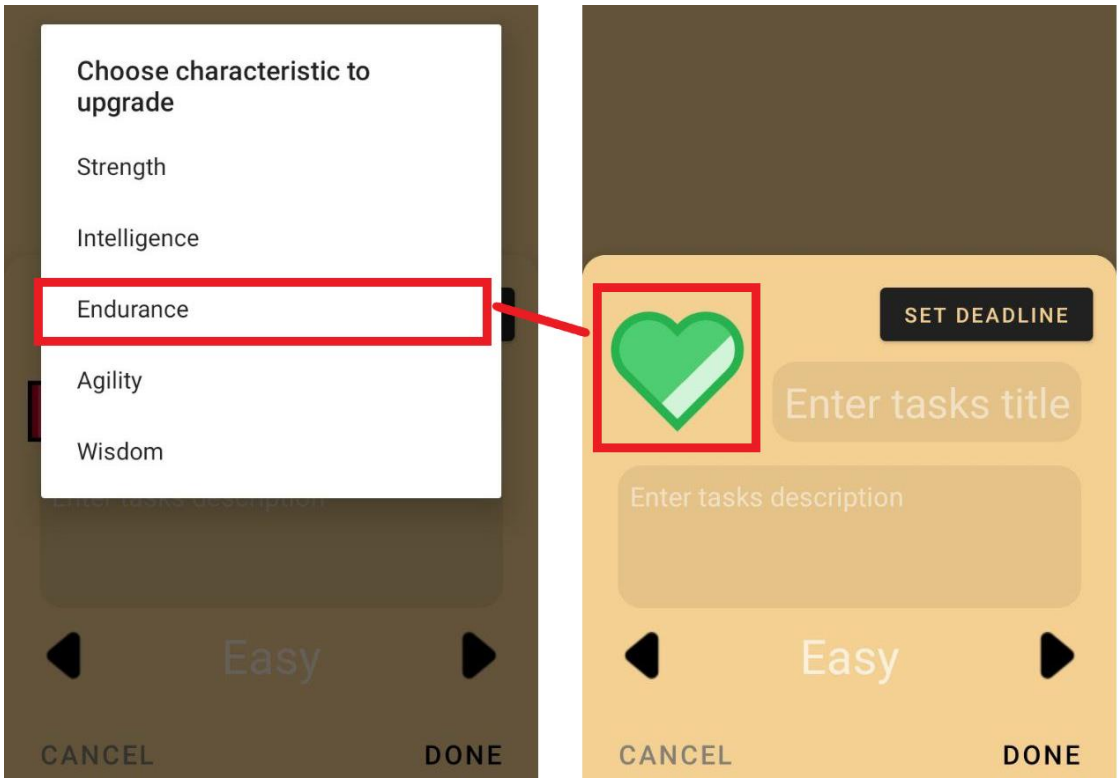

Рисунок 4.9 – Обрання іншої характеристики

Після обрання іншої характеристики у спливаючому діалозі відповідно до обраної характеристики змінюється і зображення характеристики та цільова

характеристика для прокачки при завершенні завдання, що є коректною роботою даного діалогу та екрану, отже дані екрани працюють коректно.

Модуль відображення завдань також має функції попередження користувача про закінчення терміну виконання завдань з наявним дедлайном, у завданнях, які скоро будуть прострочені дата помічається жовтим, у вже прострочених завданнях вона помічається червоним.

Результат тестування цього підмодулю зображено на рисунку 4.10, де відображено 3 завдання – без терміну виконання або ж із значним запасом часу, термін виконання якого спливає та термін виконання якого вже сплив.

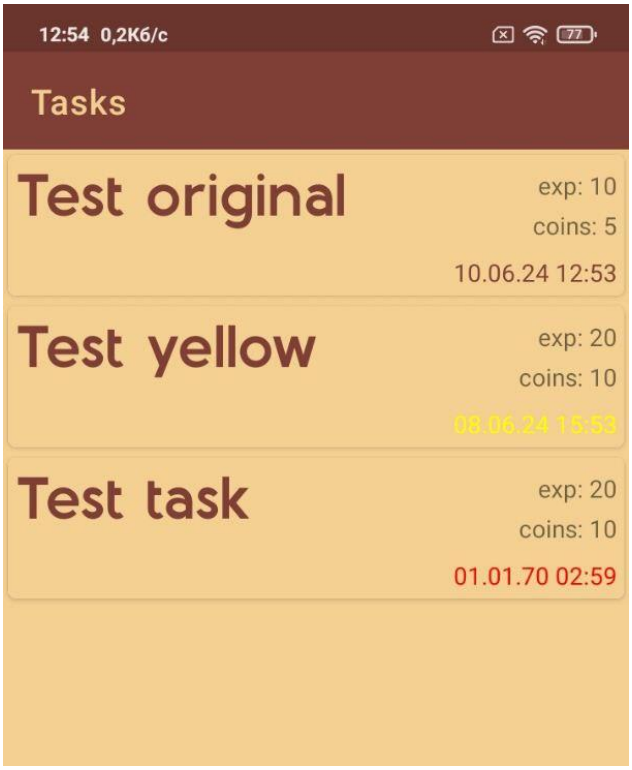

Рисунок 4.10 – Відображення різних станів наближення дедлайнів

Як видно із рисунку, усі стани наближення дедлайну відображаються коректно.

Отже, усі екрани та модулі протестовані та відповідають вимогам та потребам.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного застосунку [21]. Деталі щодо мінімальної конфігурації пристрою наведено в таблиці 4.1.

Таблиця 4.1 – Мінімальна конфігурація:

Процесор	Snapdragon 400 серії 4x1.2ГГц
Об’єм оперативної пам’яті	512 ГБ
Вільне місце на локальному сховищі	1 ГБ
Операційна система	Android 7.0

Окрім того для комфортного використання застосунку розроблено рекомендовані характеристики для використання, їх наведено в таблиці 4.2.

Таблиця 4.2 – Рекомендована конфігурація:

Процесор	Snapdragon 400+ серії 4x1.45ГГц
Об’єм оперативної пам’яті	1 ГБ
Вільне місце на локальному сховищі	2 ГБ
Операційна система	Android 7.0+

Після запуску застосунку користувач знаходиться на сторінці відображення завдань, де він може створювати нові завдання чи редагувати вже наявні. Для відмічання завдань як виконаних або їх видалення потрібно мати права адміністратора, які можна отримати ввівши пароль на сторінці адміністрування, на цій же сторінці пароль можна встановити або скинути, а також побачити скільки

віртуальної валюти було накопичено та можливо також її списати з рахунку. Усі завдання мають певну винагороду за їх виконання, при завершенні завдання ця винагорода нараховується на рахунок віртуального персонажа користувача. Побачити прогрес прокачки віртуального персонажа можна на сторінці відображення персонажу.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів мобільного застосунку з використанням технологій геймифікації. Було перевірено використання ресурсів пристрою під час використання застосунку, валідацію даних та коректність роботи різних елементів інтерфейсу та обробників подій. Також було розроблено інструкцію користувача по встановлення та початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмні модулі реалізації мобільного застосунку з використанням технологій геймифікації. Роботу розроблено згідно методичних вказівок. Для розробки було використано середовище програмної розробки Android studio.

Під час виконання роботи було проаналізовано стан проблеми на сьогоднішній день. Було розглянуто основні аналоги розроблюваного програмного застосунку. Було визначено недоліки та порівняно з власним застосунком. Базуючись на порівнянні було встановлено основні задачі роботи.

Було проаналізовано вхідні та вихідні дані застосунку, розроблено інтерфейс згідно вимог та модель і алгоритм роботи застосунку.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Kotlin та бібліотеки kotlinx, Android jetpack, room.

У роботі було зроблено наступне:

– визначено та розроблено загальний алгоритм застосунку;

– розроблено модель віртуального персонажа для застосування в якості технологій геймифікації;

– розроблено базу даних для збереження інформації про персонажа та нагороди;

– розроблено зручний та адаптивний графічний інтерфейс мобільного застосунку, що може бути використаний учнями молодших класів;

– реалізовано мобільний застосунок з використанням технологій геймифікації;

– здійснено тестування мобільного застосунку згідно поставлених задач.

Було розроблено схеми загального алгоритму роботи мобільного застосунку. Також було розроблено схему роботи застосунку з використанням блок-схем.

Тестування програми довело повну працездатність даного програмного застосунку та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

Бакалаврську дипломну роботу було оформлено відповідно до методичних вказівок [22].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yu-kai Chou. Actionable Gamification, 2019. 500 с.

2. Кучеренко М.В. Програмна модель віртуального персонажа у застосунках з використанням технологій геймифікації / М. В. Кучеренко, Г. Б. Ракитянська // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20324.

3. Вплив комп'ютерних ігор на мозкові процеси [Електроний ресурс] – Режим доступу до ресурсу: https://wz.lviv.ua/blogs/420657-yak-kompiuterni-ihry-vplyvaiut-na-mozok.

4. Classcraft [Електроний ресурс] – Режим доступу до ресурсу: https://www.classcraft.com.

5. Do-it-now: RPG To-Do list [Електроний ресурс] – Режим доступу до ресурсу: https://do-it-now.app.

6. Kahoot [Електроний ресурс] – Режим доступу до ресурсу: https://kahoot.com.

7. DailyHabits [Електронний ресурс] – Режим доступу до ресурсу: https://www.dailyhabits.xyz.

8. Основи якісного інтерфейсу [Електроний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/power-platform/well-architected/experience-optimization/user-interface-content.

9. Поєднання кольорів [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/uxuiua/колір-в-ui-дизайні-практичний-фреймворк-30112c900bd3.

10. UML [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html.

11. Діаграма діяльності [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/uk/uml-activity-diagram.html.

12. Kotlinx [Електронний ресурс] – Режим доступу до ресурсу: https://www.naukri.com/code360/library/features-of-kotlin-and-kotlinx.

13. Coroutines [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/api/kotlinx.coroutines/kotlinx-coroutines-core/kotlinx.coroutines/.

14. Android Jetpack [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/jetpack.

15. Room [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/training/data-storage/room.

16. Android studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio.

17. Layout editor [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/write/layout-editor.

18. Асинхронність [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/asynkhronne-prohramuvannia/.

19. Шифрування [Електронний ресурс] – Режим доступу до ресурсу: https://experience.dropbox.com/uk-ua/resources/what-is-encryption.

20. Тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/.

21. Інструкція користувача [Електронний ресурс] – Режим доступу до ресурсу: https://uk.itpedia.nl/2018/06/25/een-gebruikershandleiding-maken/.

22. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклал. : О. Н. Романюк. Г. О. Черноволик. - Вінниця : ВНТУ. 2022. - 51 с.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

" 12 " березня 2024 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації»

за спеціальністю – 121 Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

к.т.н., доц. Ракитянська Г.Б.

" 12 " березня 2024 р.

Виконав:

студент гр. 3ПІ-20б Кучеренко М.В.

" 12 " березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації».

Галузь застосування – освіта молодших класів.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою розробки є удосконалення наявних мобільних інструментів мотивації до виконання завдань за допомогою використання технологій геміфікації у вигляді віртуального персонажа що асоціюється з користувачем.

4. Вихідні дані для проведення НДР

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

1. Jesse Schell The Art of Game Design, 2019. 652 с.

2. Yu-kai Chou. Actionable Gamification, 2019. 500 с.

3. Róbert Nagy, Simplifying Application Development with Kotlin Multiplatform Mobile, 2022. 184 с.

5. Технічні вимоги

Вхідні дані – інформація від користувача про завдання; вихідні дані – графічний інтерфейс, що відображає завдання у списку, та віртуальний персонаж.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні. Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану мобільних систем з використанням технологій геймифікації	14.03.2024 – 28.03.2024
2	Розробка інтерфейсу та алгоритмів програмного продукту	29.03.2024 – 10.04.2024
3	Розробка модулів програмного продукту	11.04.2024 – 15.05.2024
4	Тестування програми	16.05.2024 – 26.05.2024
5	Оформлення матеріалів до захисту БДР	26.05.2024– 30.05.2024

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Дозволяється корегування бакалаврської дипломної роботи.

Додаток Б – Протокол на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільного застосунку для організації навчального процесу учнів молодших класів із застосуванням технологій геймифікації

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Ракитянська Г.Б.

Оригінальність	94,8%
Схожість	5,2%

Аналіз звіту подібності

■ **Запозичення, виявлені 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 автора. Роботу направити на доопрацювання.

□ 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 _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Кучеренко М.В.

Керівник роботи _____ Ракитянська Г.Б.

Додаток В – Лістинг програми

MainActivity.kt

```
package vnt.dip.rpg_to_do_kurs

import android.os.Bundle
import com.google.android.material.bottomnavigation.BottomNavigationView
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.NavController
import androidx.navigation.findNavController
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.setupActionBarWithNavController
import androidx.navigation.ui.setupWithNavController
import vnt.dip.rpg_to_do_kurs.controllers.NavVisibilityController
import vnt.dip.rpg_to_do_kurs.controllers.PasswordManager
import vnt.dip.rpg_to_do_kurs.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    lateinit var navController: NavController

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        MAIN = this

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)



        val navView: BottomNavigationView = binding.navView

        navController = findNavController(R.id.nav_host_fragment_activity_main)
        // Passing each menu ID as a set of Ids because each
```

```
        // menu should be considered as top level destinations.
        val appBarConfiguration = AppBarConfiguration(
            setOf(
                R.id.navigation_tasks, R.id.navigation_hero, R.id.navigation_admin
            )
        )

//        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)

        val navigationVisibilityController = NavVisibilityController(binding.navView)
        navController.addOnDestinationChangedListener(navigationVisibilityController)
    }
}
```

Constants.kt

```
package vnt.dip.rpg_to_do_kurs

import vnt.dip.rpg_to_do_kurs.db.AppDatabase

lateinit var MAIN : MainActivity
lateinit var DB: AppDatabase
var ADMIN_MODE = false

const val TASK_DB_TABLE_NAME = "task_table"
const val HERO_DB_TABLE_NAME = "hero_table"
const val DB_NAME = "task-hero-database"
const val BUNDLE_KEY_TASK_TO_DESC = "key_task_open_info"
const val BUNDLE_KEY_DESC_TO_EDIT = "key_task_edit"
const val PREFS_NAME = "hero_prefs"
const val CHARACTER_KEY = "hero"
```

TasksFragment.kt

```
package vnt.dip.rpg_to_do_kurs.ui.tasks

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.lifecycleScope
import androidx.recyclerview.widget.RecyclerView
import kotlinx.coroutines.launch
import vnt.dip.rpg_to_do_kurs.BUNDLE_KEY_TASK_TO_DESC
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.R
import vnt.dip.rpg_to_do_kurs.adapters.task.TaskAdapter
import vnt.dip.rpg_to_do_kurs.databinding.FragmentTasksBinding
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB

class TasksFragment : Fragment() {

    private var _binding: FragmentTasksBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!
    private lateinit var tasksViewModel : TasksViewModel
    private lateinit var rvTaskList: RecyclerView
    private lateinit var taskAdapter: TaskAdapter

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
```

```
    savedInstanceState: Bundle?
): View {
    tasksViewModel = ViewModelProvider(this).get(TasksViewModel::class.java)

    _binding = FragmentTasksBinding.inflate(inflater, container, false)
    val root: View = binding.root

    return root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    init()
}

fun init(){
    initData()
    initButtonListeners()

}

fun initButtonListeners(){
    binding.btnAddTask.setOnClickListener(){
        MAIN.navController.navigate(R.id.action_navigation_tasks_to_taskAddFragment)

    }
}



fun initData(){
    tasksViewModel.initDatabase()
    rvTaskList = binding.recyclerTaskList
    taskAdapter = TaskAdapter()
    rvTaskList.adapter = taskAdapter
```

```
            viewLifecycleOwner.lifecycleScope.launch{
                tasksViewModel.allTasks().collect { tasks ->
                    taskAdapter.setList(tasks)
                }
            }

        }

        companion object{
            fun itemClicked(itemModel: TaskModelDB){
                val bundle = Bundle()
                bundle.putSerializable(BUNDLE_KEY_TASK_TO_DESC, itemModel)

MAIN.navController.navigate(R.id.action_navigation_tasks_to_extendedTaskFragment, bundle)
            }
        }

        fun updateInterface(){

        }

        override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
        }


    }
```

TasksViewModel.kt

```
    package vnt.dip.rpg_to_do_kurs.ui.tasks

    import android.content.Context
```

```
import androidx.lifecycle.ViewModel
import com.google.gson.Gson
import kotlinx.coroutines.flow.Flow
import vnt.dip.rpg_to_do_kurs.CHARACTER_KEY
import vnt.dip.rpg_to_do_kurs.DB
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.PREFS_NAME
import vnt.dip.rpg_to_do_kurs.db.AppDatabase
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelUI
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB

class TasksViewModel : ViewModel() {

    fun initDatabase(){
        DB = AppDatabase.getInstance(MAIN)
        createCharacterIfNotExists(MAIN)
    }

    fun allTasks() : Flow<List<TaskModelDB>> = DB.getTaskDao().getAll()

    private fun createCharacterIfNotExists(context: Context) {
        val prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
        if (!prefs.contains(CHARACTER_KEY)) {
            saveCharacter(context, HeroModelUI(name = "Hero"))
        }
    }

    private fun saveCharacter(context: Context, character: HeroModelUI) {
        val prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
        val editor = prefs.edit()
        val gson = Gson()
        val characterJson = gson.toJson(character)
        editor.putString(CHARACTER_KEY, characterJson)
        editor.apply()
    }
```

```
}
```

TaskAddFragment.kt

```
package vnt.dip.rpg_to_do_kurs.ui.tasks.subscreens.add

import android.os.Build
import androidx.lifecycle.ViewModelProvider
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import
com.simpleNotes.simplenotelist.screens.DateTimePickerDialog.DateTimePickerDialogFragment
import vnt.dip.rpg_to_do_kurs.BUNDLE_KEY_DESC_TO_EDIT
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.R
import vnt.dip.rpg_to_do_kurs.databinding.FragmentTaskAddBinding
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB
import java.io.Serializable
import java.text.DateFormat
import java.time.Instant
import java.time.LocalDateTime
import java.time.ZoneId
import java.time.format.DateTimeFormatter
import java.time.format.FormatStyle

class TaskAddFragment : Fragment(), DateTimePickerDialogFragment.DataTransferListener
{

    private lateinit var binding: FragmentTaskAddBinding
    private lateinit var viewModel: TaskAddViewModel
```

```
private lateinit var taskData: TaskModelDB
var selectedDateTimeTimestamp : Long = -1
var currDifficulty = 1

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    binding = FragmentTaskAddBinding.inflate(inflater, container, false)
    if (arguments != null) {
        taskData = arguments?.customGetSerializable(BUNDLE_KEY_DESC_TO_EDIT)!!
    }
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    viewModel = ViewModelProvider(this).get(TaskAddViewModel::class.java)

    init()

}

private fun init(){
    initListeners()
    initDataAndInterface()

}

private fun initDataAndInterface(){
    if (arguments != null) {
        binding.AddTaskFragmentEtTitle.setText(taskData.title)
        binding.AddTaskFragmentEtDesc.setText(taskData.desc)
        currDifficulty = getDifficulty()
        selectedDateTimeTimestamp = taskData.dateEnd
```

```
            }

            updateDateButton()

            updateDifficultyView()

        }

        private fun updateDateButton(){
            if (selectedDateTimeTimestamp > 0){
                binding.fragmentAddTaskBtnClearDate.visibility = View.VISIBLE
                binding.fragmentAddTaskBtnSetDate.text =
DateFormat.getInstance().format(selectedDateTimeTimestamp)
            }else{
                binding.fragmentAddTaskBtnClearDate.visibility = View.GONE
                binding.fragmentAddTaskBtnSetDate.setText(R.string.default_task_date_button_text)
            }
        }

        private fun initListeners(){
            binding.fragmentAddTaskBackBtn.setOnClickListener(){
                returnToTaskScreen()
            }

            binding.fragmentAddTaskDoneBtn.setOnClickListener(){
                val newTask = getNewTaskData()

                if (arguments != null){
                    if (checkTaskForSimilarity(newTask)){
                        return@setOnClickListener
                    }
                    viewModel.tryReplace(taskData, newTask)
                }else{
                    viewModel.tryInsert(newTask)
                }
```

```
            returnToTaskScreen()
        }

        binding.fragmentAddTaskBtnSetDate.setOnClickListener(){
            val dateTimePickerDialog = DateTimePickerDialogFragment()
            dateTimePickerDialog.setDataTransferListener(this)
            dateTimePickerDialog.show(parentFragmentManager, "dateTimePicker")
        }

        binding.fragmentAddTaskBtnHigherDifficulty.setOnClickListener(){
            if (currDifficulty < 3){
                ++currDifficulty
                updateDifficultyView()
            }
        }

        binding.fragmentAddTaskBtnLowerDifficulty.setOnClickListener(){
            if (currDifficulty > 1){
                --currDifficulty
                updateDifficultyView()
            }
        }

        binding.fragmentAddTaskBtnClearDate.setOnClickListener(){
            selectedDateTimeTimestamp = -1
            updateDateButton()
        }

    }

    private fun getDifficulty() : Int{
        return (taskData.rewardExp)/(10)
    }
```

```
    private fun updateDifficultyView(){
        binding.fragmentAddTaskTvDifficulty.text =
resources.getStringArray(R.array.task_difficulty).get(currDifficulty-1)
    }

    private fun checkTaskForSimilarity(tempNewTask: TaskModelDB): Boolean {
        if (taskData.title != tempNewTask.title){
            return false
        }

        if (taskData.desc != tempNewTask.desc){
            return false
        }

        if (taskData.dateEnd != tempNewTask.dateEnd){
            return false
        }

        if (taskData.rewardExp != tempNewTask.rewardExp){
            return false
        }

        if (taskData.rewardCurrency != tempNewTask.rewardCurrency){
            return false
        }

        return true
    }

    private fun getNewTaskData(): TaskModelDB{
        val newTitle = binding.AddTaskFragmentEtTitle.text.toString()
        val newDesc = binding.AddTaskFragmentEtDesc.text.toString()
        val newCurrentDate = Instant.now().toEpochMilli()
        val newDeadlineDate = Instant.now().toEpochMilli()
```

```
            val newExp = currDifficulty*10
            val newCurrency = currDifficulty*5


            val tempNewTask = TaskModelDB(
                title = newTitle,
                desc = newDesc,
                dateCreated = newCurrentDate,
                dateEnd = newDeadlineDate,
                rewardExp = newExp,
                rewardCurrency = newCurrency
            )

            return tempNewTask

        }

        //прийом даних(LocalDateTime) з кастомного DateTimePicker, зміна тексту кнопки, збереження у вигляді TimeStamp
        override fun onDataTransfer(data: LocalDateTime) {
            binding.fragmentAddTaskBtnSetDate.text = data.format(DateTimeFormatter.ofLocalizedDateTime(
                FormatStyle.SHORT))
            selectedDateTimeTimestamp = data.atZone(ZoneId.systemDefault()).toInstant().toEpochMilli()
            updateDateButton()

        }

        private inline fun <reified T : Serializable> Bundle.customGetSerializable(key: String): T? {
            return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                getSerializable(key, T::class.java)
            } else {
                @Suppress("DEPRECATION")
```

```
            getSerializable(key) as? T
        }
    }

    private fun returnToTaskScreen(){
        MAIN.navController.navigate(R.id.action_taskAddFragment_to_navigation_tasks)
    }

}
```

TaskAddViewModel.kt

```
package vnt.dip.rpg_to_do_kurs.ui.tasks.subscreens.add

import android.content.Context
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.google.gson.Gson
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import vnt.dip.rpg_to_do_kurs.CHARACTER_KEY
import vnt.dip.rpg_to_do_kurs.DB
import vnt.dip.rpg_to_do_kurs.PREFS_NAME
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelUI
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB

class TaskAddViewModel : ViewModel() {
    fun tryInsert(task: TaskModelDB) {
        viewModelScope.launch(Dispatchers.IO) {
            if (!task.title.isBlank() || !task.desc.isBlank()) {
                DB.getTaskDao().insert(task)
            }
        }
    }
```

```
    fun tryReplace(oldTask: TaskModelDB, newTask: TaskModelDB){
        viewModelScope.launch(Dispatchers.IO) {
            if (!newTask.title.isBlank() || !newTask.desc.isBlank()) {
                DB.getTaskDao().delete(oldTask)
                DB.getTaskDao().insert(newTask)
            }
        }
    }

}
```

ExtendedTaskFragment.kt

```
package vnt.dip.rpg_to_do_kurs.ui.tasks.subscreens.extended

import android.os.Build
import androidx.lifecycle.ViewModelProvider
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import vnt.dip.rpg_to_do_kurs.*
import vnt.dip.rpg_to_do_kurs.databinding.FragmentExtendedTaskBinding
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB
import java.io.Serializable
import java.text.DateFormat

class ExtendedTaskFragment : Fragment() {

    private lateinit var viewModel: ExtendedTaskViewModel
    private lateinit var binding: FragmentExtendedTaskBinding
```

```
    private lateinit var taskData: TaskModelDB

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentExtendedTaskBinding.inflate(inflater,container,false)
        taskData = arguments?.customGetSerializable(BUNDLE_KEY_TASK_TO_DESC)!!

        init()
        return binding.root
    }

    private fun init() {
        initDataAndInterface()
        initListeners()
    }

    private fun initDataAndInterface() {
        binding.fragmentExtendedTaskTitle.text = taskData.title
        binding.fragmentExtendedTaskDescription.text = taskData.desc
        binding.fragmentExtendedTaskTvDateTo.text =
DateFormat.getInstance().format(taskData.dateEnd)
        binding.fragmentExtendedTaskTvExpReward.text = taskData.rewardExp.toString()
        binding.fragmentExtendedTaskTvCurrencyReward.text =
taskData.rewardCurrency.toString()


    }

    private fun initListeners() {
        binding.fragmentExtendedTaskBtnComplete.setOnClickListener(){
            if (ADMIN_MODE) {
                viewModel.tryComplete(taskData)
                navigateToTaskFragment()
```

```
                }else{

Toast.makeText(MAIN,R.string.admin_mode_required,Toast.LENGTH_LONG).show()
                }
            }

            binding.fragmentExtendedTaskBtnDelete.setOnClickListener(){
                if (ADMIN_MODE) {
                    viewModel.tryDelete(taskData)
                    navigateToTaskFragment()
                }else{

Toast.makeText(MAIN,R.string.admin_mode_required,Toast.LENGTH_LONG).show()
                }
            }

            binding.fragmentExtendedTaskBtnEdit.setOnClickListener(){
                val bundle = Bundle()
                bundle.putSerializable(BUNDLE_KEY_DESC_TO_EDIT, taskData)

MAIN.navController.navigate(R.id.action_extendedTaskFragment_to_taskAddFragment, bundle)
            }

        }

        private fun navigateToTaskFragment(){
            MAIN.navController.navigate(R.id.action_extendedTaskFragment_to_navigation_tasks)
        }

        override fun onActivityCreated(savedInstanceState: Bundle?) {
            super.onActivityCreated(savedInstanceState)
            viewModel = ViewModelProvider(this).get(ExtendedTaskViewModel::class.java)
        }

        override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
```

```
            super.onViewCreated(view, savedInstanceState)



        }

        private inline fun <reified T : Serializable> Bundle.customGetSerializable(key: String): T?
{
            return if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                getSerializable(key, T::class.java)
            } else {
                @Suppress("DEPRECATION")
                getSerializable(key) as? T
            }
        }

    }
```

ExtendedTaskViewModel.kt

```
package vnt.dip.rpg_to_do_kurs.ui.tasks.subscreens.extended

import android.content.Context
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.google.gson.Gson
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import vnt.dip.rpg_to_do_kurs.CHARACTER_KEY
import vnt.dip.rpg_to_do_kurs.DB
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.PREFS_NAME
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelUI
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB

class ExtendedTaskViewModel : ViewModel() {
    fun tryDelete(task: TaskModelDB) {
        viewModelScope.launch(Dispatchers.IO) {
            if (!task.title.isBlank() || !task.desc.isBlank()) {
                DB.getTaskDao().delete(task)
            }
        }
    }
```

```
    fun tryComplete(task: TaskModelDB) {
        viewModelScope.launch(Dispatchers.IO) {
            val hero = getCharacter(MAIN)
            hero.exp += task.rewardExp
            hero.currency += task.rewardCurrency

            if (hero.exp>=hero.expNeeded){
                hero.lvlUp()
            }

            saveCharacter(MAIN, hero)

            if (!task.title.isBlank() || !task.desc.isBlank()) {
                DB.getTaskDao().delete(task)
            }
        }
    }

    fun getCharacter(context: Context): HeroModelUI {
        val prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
        val gson = Gson()
        val characterJson = prefs.getString(CHARACTER_KEY, null)
        return gson.fromJson(characterJson, HeroModelUI::class.java)
    }

    private fun saveCharacter(context: Context, character: HeroModelUI) {
        val prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
        val editor = prefs.edit()
        val gson = Gson()
        val characterJson = gson.toJson(character)
        editor.putString(CHARACTER_KEY, characterJson)
        editor.apply()
    }
}
```

DateTimePickerDialogFragment.kt

```
package com.simpleNotes.simplenotelist.screens.DateTimePickerDialog

import android.os.Build
import android.os.Bundle
import android.text.format.DateFormat
import android.view.*
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.get
import com.google.android.material.bottomsheet.BottomSheetDialogFragment
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.databinding.FragmentDateTimePickerDialogBinding
import java.time.LocalDate
```

```
import java.time.LocalDateTime
import java.time.LocalTime
import java.time.format.DateTimeFormatter
import java.util.*

class DateTimePickerDialogFragment : BottomSheetDialogFragment() {
    private lateinit var binding: FragmentDateTimePickerDialogBinding
    private lateinit var viewModel: DateTimePickerViewModel
//    private var arrActualDates = ArrayList<LocalDate>()
    private var selectedDate = LocalDate.now()
    private var selectedTime = LocalTime.now()
    private var selectedDateTime = LocalDateTime.now()
    private var dataTransferListener: DataTransferListener? = null

    interface DataTransferListener {
        fun onDataTransfer(data: LocalDateTime)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentDateTimePickerDialogBinding.inflate(inflater,container,false)

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {

ViewCompat.setOnApplyWindowInsetsListener(requireDialog().window?.decorView!!) { _, insets ->
                val imeHeight = insets.getInsets(WindowInsetsCompat.Type.ime()).bottom
                val navigationBarHeight =
                    insets.getInsets(WindowInsetsCompat.Type.navigationBars()).bottom
                binding.root.setPadding(0, 0, 0, imeHeight - navigationBarHeight)
                insets
            }
        } else {
            @Suppress("DEPRECATION")

requireDialog().window?.setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_ADJUST
_RESIZE)
        }
        init()
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        requireDialog().setCancelable(false)
        requireDialog().setCanceledOnTouchOutside(true)

        requireView().setOnKeyListener { _, keyCode, event ->
            if (keyCode == KeyEvent.KEYCODE_BACK && event.action ==
KeyEvent.ACTION_UP) {
                // Закриття BottomSheetDialogFragment
```

```
            dismiss()
            return@setOnKeyListener true
        }
        return@setOnKeyListener false
    }

}

private fun init(){
    viewModel = ViewModelProvider(this).get()
    initDatePickerValues()
    updateSelectedTime(selectedDate,selectedTime)


    binding.timePicker.setIs24HourView(DateFormat.is24HourFormat(MAIN))

    //date picker listener
    binding.datePicker.setOnValueChangedListener { numberPicker, _, _ ->
        //selectedDate = arrActualDates.get(numberPicker.value)
        selectedDate = LocalDate.now().plusDays(numberPicker.value.toLong())

        updateSelectedTime(selectedDate, selectedTime)
    }


    //timepicker listener

    binding.timePicker.setOnTimeChangedListener {_,hour,minute ->
        selectedTime = LocalTime.of(hour,minute)

        updateSelectedTime(selectedDate,selectedTime)
    }

    //done buttons
    binding.selectTimeButton.setOnClickListener{
        dataTransferListener?.onDataTransfer(selectedDateTime)

        dismiss()
    }

    binding.cancelSelectingButton.setOnClickListener{
        dismissAllowingStateLoss()
    }


}

private fun updateSelectedTime(selectedDate: LocalDate, selectedTime: LocalTime) {
    val formatter : DateTimeFormatter
    val formatPattern = getDateFormatPattern()

    formatter = DateTimeFormatter.ofPattern(formatPattern)
```

```
        selectedDateTime = LocalDateTime.of(selectedDate,selectedTime).withSecond(0)
        binding.pickerTitle.text = selectedDateTime.format(formatter)
    }

    private fun getDateFormatPattern():String{
        var formatPattern = "EE, dd MMM HH:mm"

        if (selectedDate.year != LocalDate.now().year) {
            formatPattern = "EE, dd MMM yyyy, HH:mm"
        }

        if (!DateFormat.is24HourFormat(MAIN)){
            formatPattern = formatPattern.replace('H', 'h')
            formatPattern = formatPattern.plus(" a")
        }

        return formatPattern
    }

    private fun initDatePickerValues(){
        val arrDates = ArrayList<String>()
        val formatter = DateTimeFormatter.ofPattern("EE, dd MMM")
        val currentDate = LocalDate.now()

        arrDates.add("Today")
        //arrActualDates.add(currentDate)
        arrDates.add("Tomorrow")
        //arrActualDates.add(currentDate.plusDays(1))

        for (i in 2..365) {
            val date = currentDate.plusDays(i.toLong())
            //arrActualDates.add(date)
            arrDates.add(date.format(formatter))
        }

        binding.datePicker.minValue = 0
        binding.datePicker.maxValue = arrDates.size - 1
        binding.datePicker.wrapSelectorWheel = false
        binding.datePicker.isClickable = false
//      binding.datePicker.isFocusable = false
//      binding.timePicker.isVerticalScrollBarEnabled = false

        val stringArray = arrDates.toTypedArray()
        binding.datePicker.displayedValues = stringArray
    }

    fun setDataTransferListener(listener: DataTransferListener) {
        dataTransferListener = listener
    }

}
```

HeroFragment.kt

```
package vnt.dip.rpg_to_do_kurs.ui.hero

import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.databinding.FragmentHeroBinding
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelUI

class HeroFragment : Fragment() {

    private var _binding: FragmentHeroBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!
    private lateinit var heroViewModel : HeroViewModel
    private lateinit var hero : HeroModelUI

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        heroViewModel = ViewModelProvider(this).get(HeroViewModel::class.java)
        _binding = FragmentHeroBinding.inflate(inflater, container, false)
        val root: View = binding.root
        init()

        return root
    }

    private fun init() {
        initDataAndInterface()
        initListeners()
    }

    private fun initDataAndInterface() {
        hero = heroViewModel.getCharacter(MAIN)
        Log.d("heroPage", " " + hero.exp + " " + hero.expNeeded)

        binding.fragmentHeroTvStatStrengthCounter.text = hero.strength.toString()
        binding.fragmentHeroTvStatIntelligenceCounter.text = hero.intelligence.toString()
        binding.fragmentHeroTvStatEnduranceCounter.text = hero.endurance.toString()
```

```
        binding.fragmentHeroTvStatAgilityCounter.text = hero.agility.toString()
        binding.fragmentHeroTvStatSpeedCounter.text = hero.speed.toString()
        binding.fragmentHeroTvStatWisdomCounter.text = hero.wisdom.toString()
        binding.fragmentHeroPbExpLvlupBar.post(
            Runnable {
                binding.fragmentHeroPbExpLvlupBar.max = hero.expNeeded
                binding.fragmentHeroPbExpLvlupBar.progress = hero.exp
            }
        )


        binding.fragmentHeroTvStatLevelCounter.text = hero.lvl.toString()

    }

    private fun initListeners() {

    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

HeroViewModel.kt

```
package vnt.dip.rpg_to_do_kurs.ui.hero

import android.content.Context
import androidx.lifecycle.ViewModel
import com.google.gson.Gson
import vnt.dip.rpg_to_do_kurs.CHARACTER_KEY
import vnt.dip.rpg_to_do_kurs.PREFS_NAME
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelDB
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelUI

class HeroViewModel : ViewModel() {
    fun getHero(): HeroModelDB{
        return HeroModelDB(name = "hero")
    }

    fun getCharacter(context: Context): HeroModelUI {
        val prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)
        val gson = Gson()
        val characterJson = prefs.getString(CHARACTER_KEY, null)
```

```
        return gson.fromJson(characterJson, HeroModelUI::class.java)
    }

}
```

AdminFragment.kt

```
package vnt.dip.rpg_to_do_kurs.ui.admin

import android.content.res.Resources
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import androidx.lifecycle.lifecycleScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import vnt.dip.rpg_to_do_kurs.ADMIN_MODE
import vnt.dip.rpg_to_do_kurs.MAIN
import vnt.dip.rpg_to_do_kurs.R
import vnt.dip.rpg_to_do_kurs.controllers.PasswordManager
import vnt.dip.rpg_to_do_kurs.databinding.FragmentAdminBinding

class AdminFragment : Fragment() {

    private var _binding: FragmentAdminBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!
    private lateinit var viewModel : AdminViewModel
    val passwordManager = PasswordManager(MAIN)
    var correctPassword = true

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        viewModel = ViewModelProvider(this).get(AdminViewModel::class.java)
        _binding = FragmentAdminBinding.inflate(inflater, container, false)
        val root: View = binding.root


        return root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
```

```
        super.onViewCreated(view, savedInstanceState)
        init()
    }

    private fun init() {
        initDataAndInterface()
        initListeners()
    }

    private fun initListeners() {
        binding.fragmentAdminBtnSetResetPassword.setOnClickListener(){
            lifecycleScope.launch(Dispatchers.Main) {

                withContext(Dispatchers.Default) {
                    if (passwordManager.isPasswordSaved()) {
                        if
(passwordManager.isPasswordCorrect(binding.fragmentAdminEtPassword.text.toString())) {
                            passwordManager.deletePassword()
                            correctPassword = true
                        } else {
                            correctPassword = false
                        }
                    } else {

passwordManager.savePassword(binding.fragmentAdminEtPassword.text.toString())
                    }
                }

                initDataAndInterface()
            }
        }

        binding.fragmentAdminBtnToggleAdmin.setOnClickListener(){
            lifecycleScope.launch(Dispatchers.Main) {

                withContext(Dispatchers.Default) {
                    if (!ADMIN_MODE) {
                        if
(passwordManager.isPasswordCorrect(binding.fragmentAdminEtPassword.text.toString())) {
                            ADMIN_MODE = true
                            correctPassword = true
                        } else {
                            correctPassword = false
                        }

                    } else {
                        ADMIN_MODE = false
                    }
                }

                initDataAndInterface()
            }
```

```
            }
        }

        private fun initDataAndInterface() {
            if(passwordManager.isPasswordSaved()){

binding.fragmentAdminBtnSetResetPassword.setText(R.string.admin_password_reset)
                binding.fragmentAdminBtnToggleAdmin.visibility = View.VISIBLE
            }else{
                binding.fragmentAdminBtnSetResetPassword.setText(R.string.admin_password_set)
                binding.fragmentAdminBtnToggleAdmin.visibility = View.GONE
            }

            if (ADMIN_MODE){

binding.fragmentAdminBtnToggleAdmin.setText(R.string.admin_btn_toggle_turned_on)
            }else{

binding.fragmentAdminBtnToggleAdmin.setText(R.string.admin_btn_toggle_turned_off)
            }

            if (correctPassword){
                binding.fragmentAdminEtPassword.setHint(R.string.admin_et_password_hint)
            }else{
                binding.fragmentAdminEtPassword.hint = "Incorrect password!"
            }

        }

        override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
        }
    }
```

TaskModelDB.kt

```
package vnt.dip.rpg_to_do_kurs.models.task

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import vnt.dip.rpg_to_do_kurs.TASK_DB_TABLE_NAME
import java.io.Serializable

@Entity(tableName = TASK_DB_TABLE_NAME)
data class TaskModelDB (
    @PrimaryKey(autoGenerate = true) var id: Int = 0,
    @ColumnInfo var title: String = "",
    @ColumnInfo var desc: String = "",
    @ColumnInfo(name = "creation_date") var dateCreated: Long = -1,
```

```
    @ColumnInfo(name = "expiration_date") var dateEnd: Long = -1,
    @ColumnInfo(name = "currency_reward") var rewardCurrency: Int = 0,
    @ColumnInfo(name = "exp_reward") var rewardExp: Int = 0
        ) : Serializable
```

HeroModelUI.kt

```
package vnt.dip.rpg_to_do_kurs.models.hero

data class HeroModelUI(
    var name: String,
    var lvl: Int = 1,
    var expNeeded: Int = 100,
    var freePoints: Int = 0,
    var exp: Int = 0,
    var currency: Int = 0,
    var strength: Int = 10,
    var intelligence: Int = 10,
    var endurance: Int = 10,
    var agility: Int = 10,
    var wisdom: Int = 10,
    var speed: Int = 10
) {
    fun toHash(): Int{
        return name.hashCode()+
                lvl.hashCode()+
                exp.hashCode()+
                strength.hashCode()+
                intelligence.hashCode()+
                endurance.hashCode()+
                agility.hashCode()+
                wisdom.hashCode()+
                speed.hashCode()
    }

    fun lvlUp() {
        ++lvl
        exp -= expNeeded
        expNeeded += 5
        freePoints += 2

        strength += 1
        intelligence += 1
        endurance += 1
        agility += 1
        wisdom += 1
        speed += 1

    }
}
```

AppDatabase.kt

```
package vnt.dip.rpg_to_do_kurs.db

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import vnt.dip.rpg_to_do_kurs.DB_NAME
import vnt.dip.rpg_to_do_kurs.db.dao.HeroDao
import vnt.dip.rpg_to_do_kurs.db.dao.TaskDao
import vnt.dip.rpg_to_do_kurs.models.hero.HeroModelDB
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB

@Database(
    entities = [TaskModelDB::class, HeroModelDB::class],
    version = 1
)
abstract class AppDatabase : RoomDatabase() {
    abstract fun getTaskDao():TaskDao
    abstract fun getHeroDao():HeroDao

    companion object{
        private var database: AppDatabase ?= null

        @Synchronized
        fun getInstance(context: Context):AppDatabase{
            return if(database == null){
                database = Room.databaseBuilder(context, AppDatabase::class.java,
DB_NAME).build()

                database as AppDatabase
            }else{
                database as AppDatabase
            }
        }
    }

}
```

TaskDao.kt

```
package vnt.dip.rpg_to_do_kurs.db.dao

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.Query
import kotlinx.coroutines.flow.Flow
import vnt.dip.rpg_to_do_kurs.HERO_DB_TABLE_NAME
import vnt.dip.rpg_to_do_kurs.TASK_DB_TABLE_NAME
```

```
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB

@Dao
interface TaskDao {
    @Query("SELECT * FROM $TASK_DB_TABLE_NAME")
    fun getAll(): Flow<List<TaskModelDB>>

    @Insert
    suspend fun insert(user: TaskModelDB)

    @Delete
    suspend fun delete(user: TaskModelDB)

//    @Query("SELECT * FROM $HERO_DB_TABLE_NAME")
//    fun getAll(): Flow<List<>>

}
```

NavVisibilityController.kt

```
package vnt.dip.rpg_to_do_kurs.controllers

import android.os.Bundle
import android.view.View
import androidx.collection.arraySetOf
import androidx.navigation.NavController
import androidx.navigation.NavDestination
import com.google.android.material.bottomnavigation.BottomNavigationView
import vnt.dip.rpg_to_do_kurs.R

class NavVisibilityController(
    private val bottomNavigationView: BottomNavigationView
) : NavController.OnDestinationChangedListener {

    override fun onDestinationChanged(
        controller: NavController,
        destination: NavDestination,
        arguments: Bundle?
    ) {
        // Список ID фрагментів, для яких Bottom Navigation View буде видимим.
        val visibleBottomNavDestinations = arraySetOf(R.id.navigation_hero,
R.id.navigation_tasks, R.id.navigation_admin)
//      val visibleActionBarDestinations = arraySetOf(R.id.tasksFragment, R.id.notesFragment,
R.id.nav_bar_item_settings)

        if (destination.id in visibleBottomNavDestinations) {
            bottomNavigationView.visibility = View.VISIBLE
        } else {
            bottomNavigationView.visibility = View.GONE
        }
```

```
//        if (destination.id in visibleActionBarDestinations) {
//            MAIN.supportActionBar?.show()
//        } else {
//            MAIN.supportActionBar?.hide()
//        }


    }
}
```

PasswordManager.kt

```
package vnt.dip.rpg_to_do_kurs.controllers

import android.content.Context
import android.util.Base64
import javax.crypto.Cipher
import javax.crypto.SecretKeyFactory
import javax.crypto.spec.PBEKeySpec
import javax.crypto.spec.SecretKeySpec

class PasswordManager(private val context: Context) {

    private val PASSWORD_PREFS_NAME = "password_prefs"
    private val PASSWORD_KEY = "encrypted_password"

    fun isPasswordSaved(): Boolean {
        val sharedPreferences = context.getSharedPreferences(PASSWORD_PREFS_NAME,
Context.MODE_PRIVATE)
        return sharedPreferences.contains(PASSWORD_KEY)
    }

    fun deletePassword() {
        val sharedPreferences = context.getSharedPreferences(PASSWORD_PREFS_NAME,
Context.MODE_PRIVATE)
        sharedPreferences.edit().remove(PASSWORD_KEY).apply()
    }

    fun savePassword(password: String) {
        val encryptedPassword = encryptPassword(password)
        val sharedPreferences = context.getSharedPreferences(PASSWORD_PREFS_NAME,
Context.MODE_PRIVATE)
        sharedPreferences.edit().putString(PASSWORD_KEY, encryptedPassword).apply()
    }

    fun isPasswordCorrect(inputPassword: String): Boolean {
        val sharedPreferences = context.getSharedPreferences(PASSWORD_PREFS_NAME,
Context.MODE_PRIVATE)
        val storedPassword = sharedPreferences.getString(PASSWORD_KEY, null) ?: return
false
        val decryptedPassword = decryptPassword(storedPassword, inputPassword)
```

```
        return inputPassword == decryptedPassword
    }

    private fun encryptPassword(password: String): String {
        val salt = "R,£3kX%0XI|4".toByteArray() // Change this to your own salt
        val factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256")
        val spec = PBEKeySpec("63^7&3HH2aq-".toCharArray(), salt, 65536, 256)
        val tmp = factory.generateSecret(spec)
        val secretKey = SecretKeySpec(tmp.encoded, "AES")

        val cipher = Cipher.getInstance("AES/ECB/PKCS5Padding")
        cipher.init(Cipher.ENCRYPT_MODE, secretKey)
        val encrypted = cipher.doFinal(password.toByteArray())

        return Base64.encodeToString(encrypted, Base64.DEFAULT)
    }

    private fun decryptPassword(encryptedPassword: String, password: String): String {
        val salt = "R,£3kX%0XI|4".toByteArray() // Change this to your own salt
        val factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256")
        val spec = PBEKeySpec("63^7&3HH2aq-".toCharArray(), salt, 65536, 256)
        val tmp = factory.generateSecret(spec)
        val secretKey = SecretKeySpec(tmp.encoded, "AES")

        val cipher = Cipher.getInstance("AES/ECB/PKCS5Padding")
        cipher.init(Cipher.DECRYPT_MODE, secretKey)
        val decrypted = cipher.doFinal(Base64.decode(encryptedPassword, Base64.DEFAULT))

        return String(decrypted)
    }
}
```

TaskAdapter.kt

```
package vnt.dip.rpg_to_do_kurs.adapters.task

import android.annotation.SuppressLint
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import vnt.dip.rpg_to_do_kurs.databinding.ItemTaskLayoutBinding
import vnt.dip.rpg_to_do_kurs.models.task.TaskModelDB
import vnt.dip.rpg_to_do_kurs.ui.tasks.TasksFragment
import java.text.DateFormat
import java.time.Instant
import java.time.LocalDateTime

class TaskAdapter(
//    private val showMenuDelete: (Boolean) -> Unit
): RecyclerView.Adapter<TaskAdapter.TaskViewHolder>() {
```

```
    private var itemList = ArrayList<TaskModelDB>()
    //    var isSelectionEnabled = false
    private var selectedItemList = mutableListOf<Int>()


    class TaskViewHolder(val binding: ItemTaskLayoutBinding): RecyclerView.ViewHolder(binding.root)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TaskViewHolder {
        return TaskViewHolder(ItemTaskLayoutBinding.inflate(LayoutInflater.from(parent.context),parent,false))
    }

    override fun onBindViewHolder(holder: TaskViewHolder, position: Int) {
        val item = itemList[position]

        holder.binding.taskTitle.text = item.title
        holder.binding.taskDesc.text = item.desc
        holder.binding.rvitemTaskTvExpReward.text = "exp: " + item.rewardExp.toString()
        holder.binding.rvitemTaskTvCurrencyReward.text = "coins: " + item.rewardCurrency.toString()
        holder.binding.taskDate.text = DateFormat.getInstance().format(item.dateCreated)

        holder.itemView.setOnClickListener {
            TasksFragment.itemClicked(itemList[position])
        }

    }

    override fun getItemCount(): Int {
        return itemList.size
    }

    @SuppressLint("NotifyDataSetChanged")
    fun setList(newList: List<TaskModelDB>){
        itemList.clear()
        for (task in newList){
            this.itemList.add(task)
        }

        itemList.reverse()
        notifyDataSetChanged()
    }


}
```

Додаток Г – Ілюстративна частина

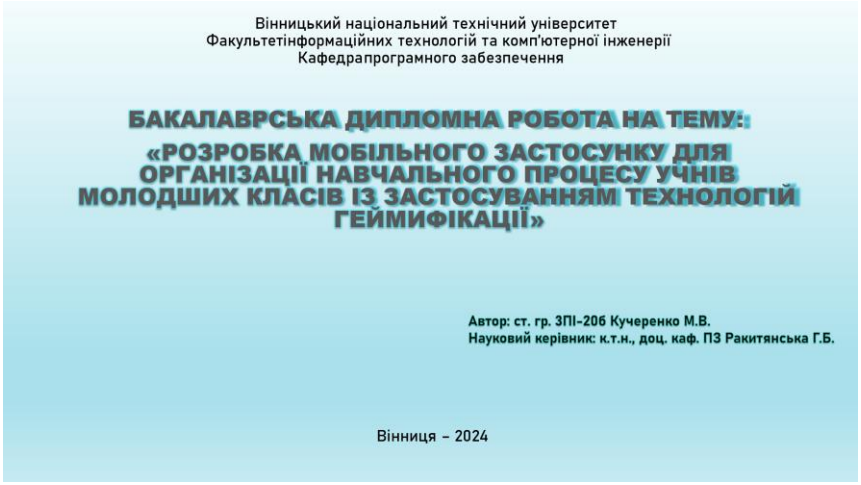

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Слайд «актуальність теми»

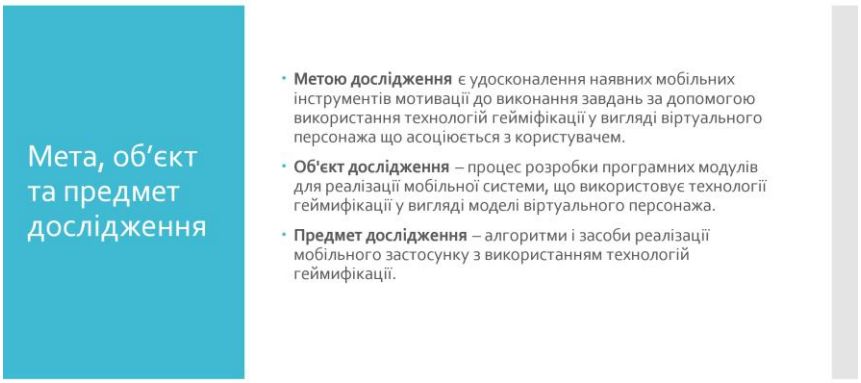

Рисунок Г.3 – Слайд «Мета, об’єкт та предмет дослідження»

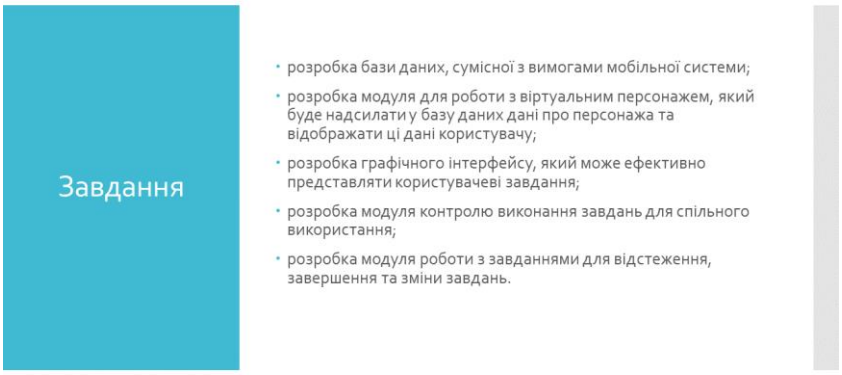

Рисунок Г.4 – Слайд «завдання»

Рисунок Г.5 – Слайд «Новизна результатів»

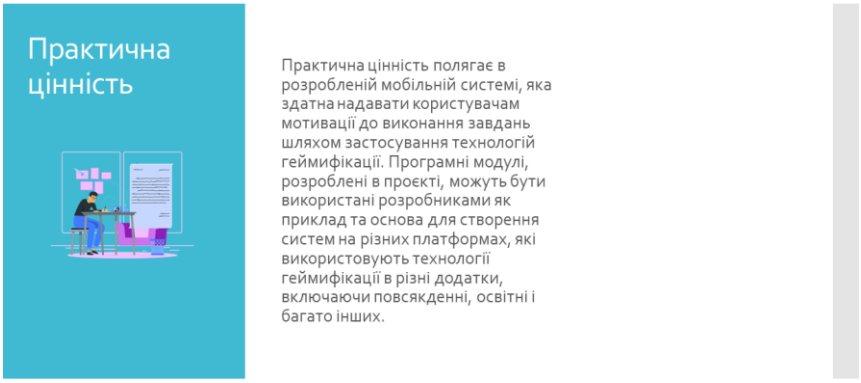

Рисунок Г.6 – Слайд «Практична цінність»

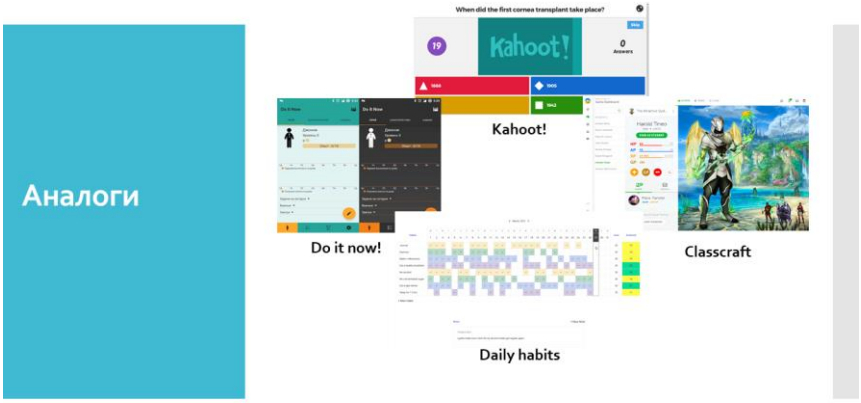

Рисунок Г.7 – Слайд «Аналоги»

Рисунок Г.8 – Слайд «Технології та засоби розробки»

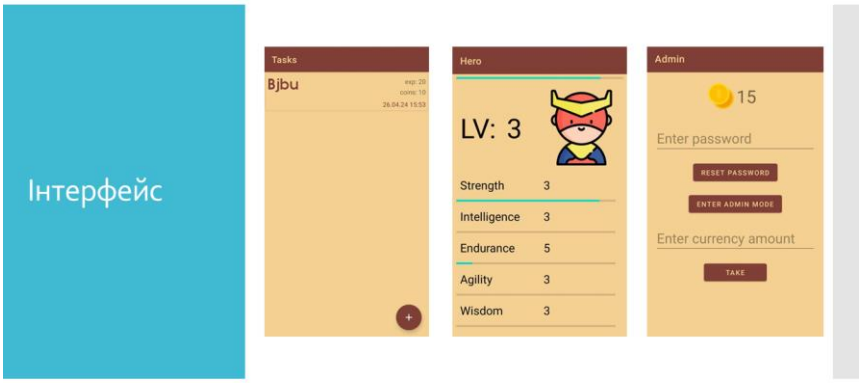

Рисунок Г.9 – Слайд «Інтерфейс»

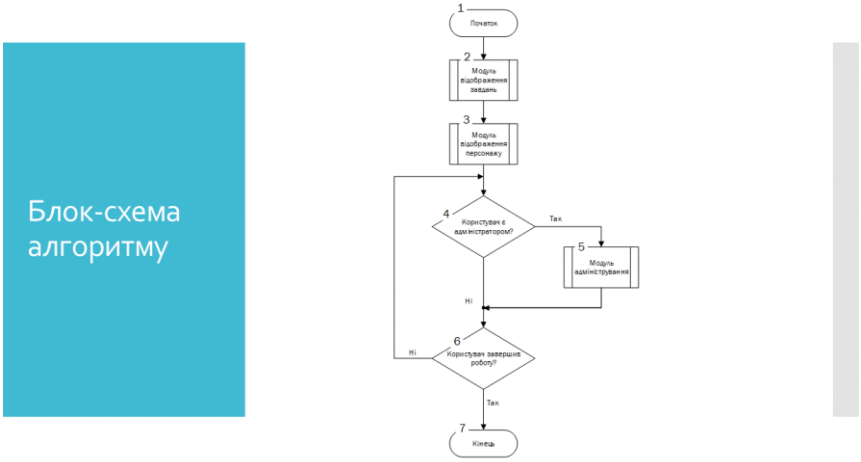

Рисунок Г.10 – Слайд «Блок-схема алгоритму»

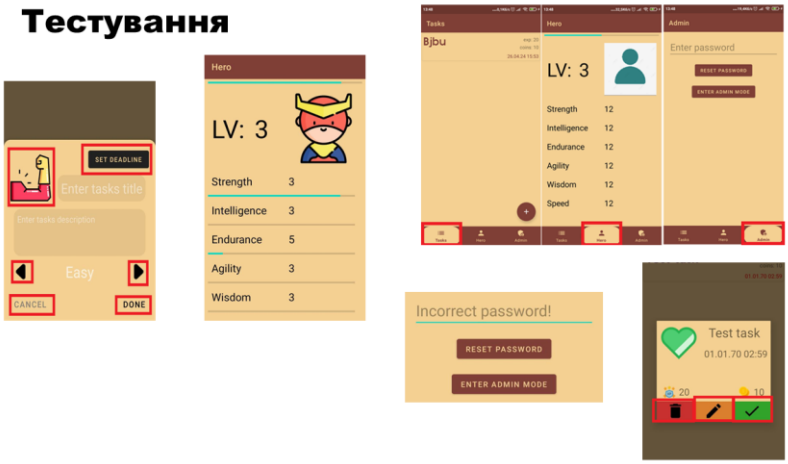

Рисунок Г.11 – Слайд «Тестування»

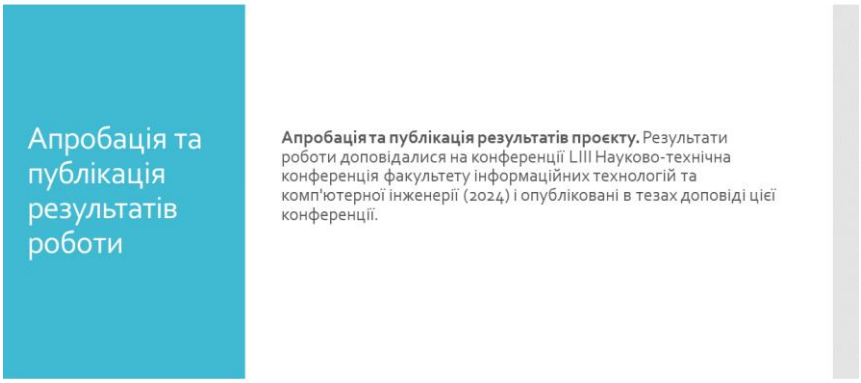

Рисунок Г.12 – Слайд «Апробація та публікація результатів роботи»

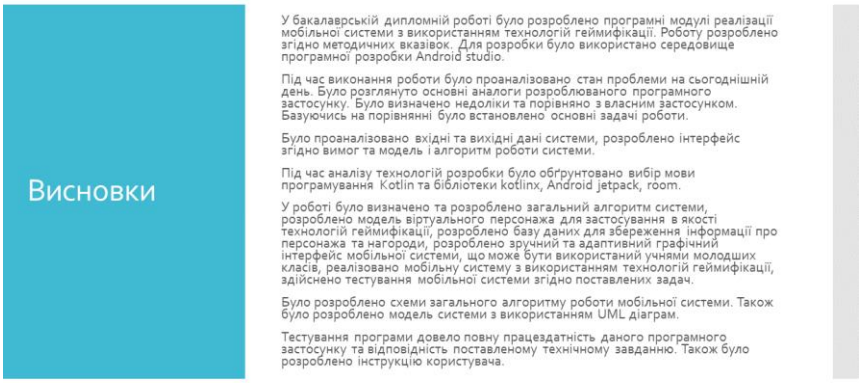

Рисунок Г.13 – Слайд «Висновки»