Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ДОДАТКУ ДЛЯ ДОПОМОГИ ШКОЛЯРАМ У ВИРІШЕННІ ЗАДАЧ ТА ПОШУКУ ФІЗИЧНИХ ФОРМУЛ З ВИКОРИСТАННЯМ KOTLIN ТА FIREBASE

DEVELOPMENT OF AN APPLICATION TO ASSIST STUDENTS IN SOLVING PROBLEMS AND FINDING PHYSICS FORMULAS USING KOTLIN AND FIREBASE

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Верещагін Даниїл Олександрович

Керівник:
Корень Віталій Володимирович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Верещагіну Даниїлу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка додатку для допомоги школярам у вирішенні задач та пошуку фізичних формул з використанням Kotlin та Firebase»*
Керівник роботи: *асистент Корень В.В.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Firebase, Android Studio, Kotlin, GitHub, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *провести аналіз подібних додатків, розробити структуру, спроєктувати архітектуру мобільного додатку, реалізувати функціонал авторизації, реалізувати пошук формул, забезпечити адаптивний інтерфейс, підготувати супровідну документацію, провести тестування і налагодження системи.*
5. Перелік графічного матеріалу: *12 рисунків*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Корень В. В.*		
Специфікація вимог до розробленої системи	*Корень В. В.*		
Розробка об'єкта проєктування	*Корень В. В.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,92 %		
Академічна доброчесність	*Корень В. В.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Викон.
2	Аналіз проблеми розробки та впровадження об'єкту проєктування	до 01.03.2025 р.	Викон.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Викон.
4	Розробка функціонально-структурної схеми роботи об'єкта проєктування та проєктування бази даних	до 29.03.2025 р.	Викон.
5	Практична реалізація об'єкта проєктування та розробка бази даних	до 26.04.2025 р.	Викон.
6	Тестування та налагодження об'єкта проєктування	до 03.05.2025 р.	Викон.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Викон.

Здобувач вищої освіти _____ Даниїл ВЕРЕЩАГІН

Керівник кваліфікаційної роботи _____ Віталій КОРЕНЬ

АНОТАЦІЯ

Верещагін Д. О. Розробка додатку для допомоги школярам у вирішенні задач та пошуку фізичних формул з використанням Kotlin та Firebase. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі здійснено аналіз предметної області, визначено цільову аудиторію програми, а саме школярів, та досліджено сучасні мобільні застосунки, такі як Дуолінго. Сформульовано основні завдання розробки додатку, спрямованого на ефективне вивчення та використання фізичних формул.

В другому розділі розроблено архітектуру мобільного застосунку, обґрунтовано вибір інструментів розробки: Android Studio як середовище програмування, мову Kotlin для імплементації функціоналу та платформу Firebase для забезпечення аутентифікації, зберігання та синхронізації даних. Також описано вибір та інтеграцію сторонньої бібліотеки з GitHub для коректного та зручного відображення математичних формул.

У третьому розділі створено повністю функціональний прототип застосунку, проведено його тестування на мобільних пристроях та перевірено якість роботи бібліотеки для формул. Проаналізовано ефективність та зручність користування продуктом з точки зору цільової аудиторії.

У висновках підсумовано результати розробки, підтверджено успішність реалізації поставлених завдань, а також наведено рекомендації щодо подальшого розширення функціональних можливостей та вдосконалення інтерфейсу для покращення навчального процесу.

Ключові слова: Android Studio, Kotlin, Firebase, мобільний застосунок, фізичні формули, математична бібліотека, навчання.

ABSTRACT

Vereshchahin D. Development of an Application to Assist Students in Solving Problems and Finding Physics Formulas Using Kotlin and Firebase. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references.

The first section analyzes the subject area, identifies the target audience of the application, namely schoolchildren, and explores modern mobile applications such as Duolingo. The main tasks of developing an application aimed at effective learning and use of physical formulas are formulated.

In the second section, the architecture of the mobile application is developed, and the choice of development tools is substantiated: Android Studio as a programming environment, the Kotlin language for implementing functionality, and the Firebase platform for authentication, storage, and data synchronization. We also describe the selection and integration of a third-party library from GitHub for the correct and convenient display of mathematical formulas.

In the third section, a fully functional prototype of the application is created, tested on mobile devices, and the quality of the library for formulas is checked. The effectiveness and usability of the product from the point of view of the target audience are analyzed.

The conclusions summarize the results of the development, confirm the success of the implementation of the tasks, and provide recommendations for further expanding the functionality and improving the interface to improve the educational process.

Keywords: Android Studio, Kotlin, Firebase, mobile application, physical formulas, mathematical library, learning.

ЗМІСТ

ВСТУП

Актуальність теми. У процесі вивчення фізики учні регулярно стикаються з необхідністю швидко знаходити потрібні формули, проте для цього часто доводиться звертатися до різноманітних джерел – підручників, особистих конспектів або пошукових систем в інтернеті. Це може бути незручно, особливо коли потрібно швидко знайти потрібну формулу чи зрозуміти, що означає та чи інша змінна. Тому виникла ідея створити простий мобільний додаток, де всі формули будуть зібрані в одному місці й зручно структуровані за темами.

На сьогодні існує багато навчальних застосунків, але не всі з них зручні, українською мовою або відповідають програмі з фізики. Часто у них відсутній пошук або пояснення змінних у формулах. Це спонукало до розробки власного застосунку, який був би простим у використанні й корисним саме для школярів.

Метою цієї роботи є розробка мобільного Android-додатку, основним призначенням якого є надання учням зручного інструменту для швидкого доступу до фізичних формул, згрупованих за темами та підтемами. Програма має не лише зберігати формули, а й допомагати зрозуміти їх зміст завдяки коротким описам, поясненням змінних та зручному візуальному поданню математичних виразів. Таким чином, додаток виконує функцію електронного довідника, який можна використовувати під час навчання, виконання домашніх завдань або підготовки до контрольних робіт.

Об’єктом роботи є процес використання мобільних технологій у навчанні фізики, зокрема застосування мобільних додатків як допоміжного засобу для вивчення теоретичного матеріалу. Дослідження зосереджується на тому, як програмне забезпечення для Android може ефективно структурувати та подавати навчальний матеріал, сприяти запам’ятовуванню формул, а також покращити загальне сприйняття предмету через інтерактивність та візуалізацію.

Предметом роботи є структура та реалізація навчального застосунку для зручного доступу до фізичних формул. Окрема увага приділяється тому, як

додаток забезпечує навігацію між темами, як здійснюється пошук формул за ключовими словами та як адаптується інтерфейс для різних розмірів екранів.

Для досягнення поставленої мети були поставлені наступні завдання:

– провести аналіз предметної області та існуючих програмних рішень, що використовуються для навчання фізики;

– розробити структуру навчального матеріалу: класифікацію фізичних формул за темами та підтемами з поясненням кожної змінної;

– спроєктувати архітектуру мобільного додатку з урахуванням сучасних підходів до проєктування програмного забезпечення (MVVM, фрагментна навігація, ViewModel, Firebase);

– реалізувати функціонал авторизації та аутентифікації користувачів за допомогою Firebase Authentication;

– реалізувати пошук формул за ключовими словами та зручне відображення через бібліотеку MathView з підтримкою форматування математичних виразів;

– забезпечити адаптивний інтерфейс користувача з використанням RecyclerView для перегляду тем і формул;

– провести тестування додатку на відповідність функціональним вимогам та зручності використання;

– підготувати супровідну документацію та зробити висновки щодо перспектив подальшого розвитку проєкту.

РОЗДІЛ 1
АНАЛІЗ ДОДАТКІВ ДЛЯ ДОПОМОГИ ШКОЛЯРАМ У НАВЧАННІ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сучасна система освіти все активніше впроваджує цифрові технології в навчальний процес, особливо в галузі природничих наук. Зокрема, вивчення фізики вимагає засвоєння великого обсягу формул і понять, які не завжди просто запам’ятати чи швидко знайти. У цьому контексті мобільні додатки для навчання стають важливим інструментом підтримки учнів. Вони дозволяють отримувати доступ до навчального контенту в будь-який час і в будь-якому місці, адаптувати подачу інформації до потреб користувача та створювати інтерактивне середовище навчання.

Аналіз існуючих мобільних рішень показав наявність великої кількості навчальних застосунків, однак не всі з них забезпечують зручну класифікацію тем, структуровану базу формул або адаптивне відображення складних математичних виразів, нижче будуть наведені приклади.

Наприклад, додаток «Mechanical Engineering Calcula» від Trelleborg Sealing Solutions містить великий набір формул, що охоплює як базові, так і більш складні теми (рис. 1.1).

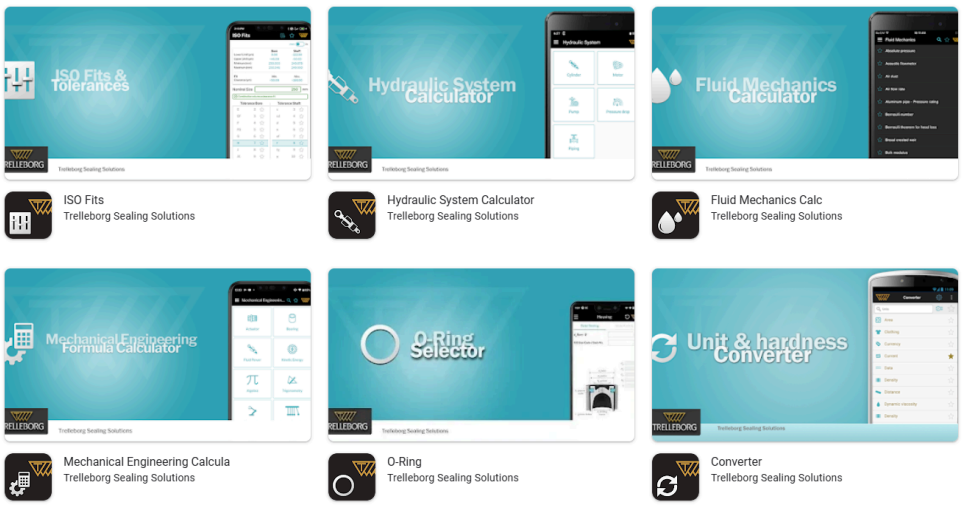

Рисунок 1.1 – Додатки від Trelleborg Sealing Solutions [1]

Проте подача інформації є частиною, і користувач змушений мати не один додаток а декілька, без можливості швидко перейти до потрібної теми. Відсутність одного єдиного додатку ускладнює використання додатку в умовах обмеженого часу, наприклад, під час підготовки до контрольної роботи чи ЗНО.

Інший додаток, Physics Formula від Codebug, містить формули з механіки, термодинаміки, електростатики, магнетизму, оптики, а також сучасної фізики. Він також дозволяє користувачам створювати власні примітки. Проте головна мета цього застосунку – надання великого обсягу формул як довідника, а не структуризація за темами чи навчальне пояснення. Інтерфейс досить базовий і менш зручний для новачків, що обмежує його ефективність у шкільному навчанні (рис. 1.2).

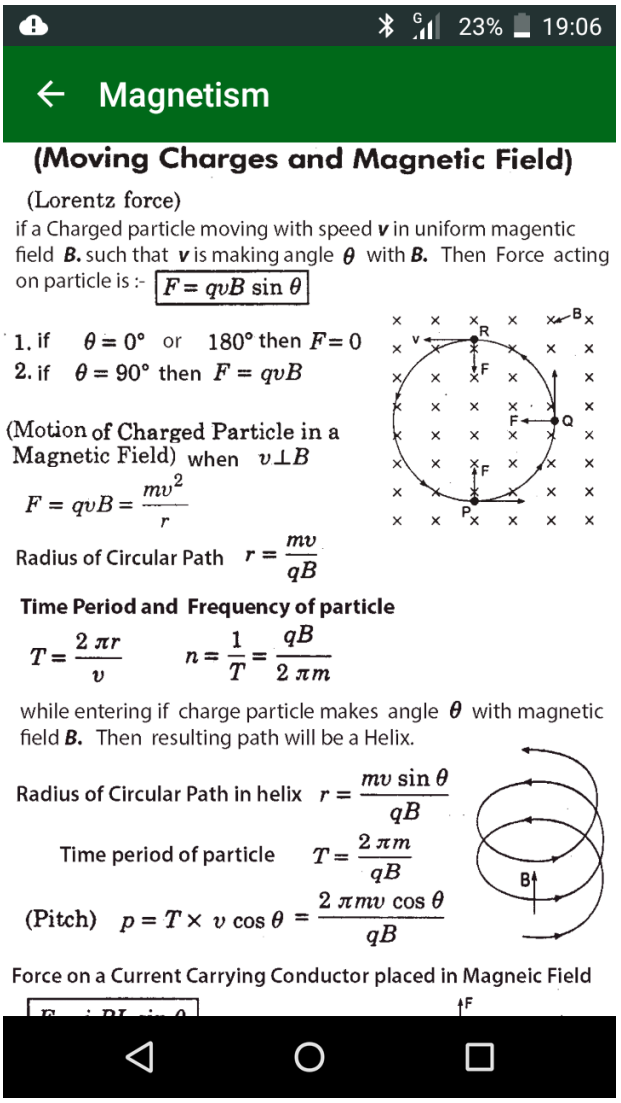

Рисунок 1.2 – Додаток Physics Formula від Codebug [2]

Крім того, в багатьох аналогах математичні вирази представлені у вигляді зображень, що унеможливлює динамічне форматування та адаптацію до різних екранів.

Враховуючи ці обмеження, постає потреба у створенні нового застосунку, який би враховував потреби сучасного користувача: підтримував українську мову, мав адаптивний інтерфейс, дозволяв пошук і фільтрацію формул, забезпечував їх якісне відображення у форматі математичних виразів, і мав структуру тем і підтем, наближену до шкільної програми. Розробка такого додатку дозволить не лише підвищити зручність доступу до навчального матеріалу, але й поліпшити засвоєння знань завдяки зручному та інтуїтивно зрозумілому інтерфейсу.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети розробки програми для допомоги школярам у вирішенні задач та пошуку фізичних формул потрібно виконати наступні завдання:

– провести аналіз предметної області та існуючих програмних рішень, що використовуються для навчання фізики;

– розробити структуру навчального матеріалу: класифікацію фізичних формул за темами та підтемами з поясненням кожної змінної;

– спроєктувати архітектуру мобільного додатку з урахуванням сучасних підходів до проєктування програмного забезпечення (MVVM, фрагментна навігація, ViewModel, Firebase);

– реалізувати функціонал авторизації та аутентифікації користувачів за допомогою Firebase Authentication;

– реалізувати пошук формул за ключовими словами та зручне відображення через бібліотеку MathView з підтримкою форматування математичних виразів;

– забезпечити адаптивний інтерфейс користувача з використанням RecyclerView для перегляду тем і формул;

– провести тестування додатку на відповідність функціональним вимогам та зручності використання;

– підготувати супровідну документацію та зробити висновки щодо перспектив подальшого розвитку проєкту.

Висновки до розділу 1

У результаті проведеного аналізу предметної області було встановлено, що існуючі мобільні застосунки для вивчення фізики мають низку обмежень. Більшість із них не забезпечують достатньої інтерактивності, не містять гнучкої структури поділу формул за темами, а також не надають пояснень до змінних, що ускладнює самостійне засвоєння матеріалу.

Проведене дослідження існуючих програмних рішень засвідчило, що вони або мають вузьку спеціалізацію (наприклад, лише тестування чи підготовка до ЗНО), або не є адаптованими до україномовного користувача. Також виявлено відсутність повноцінного комплексного рішення, яке б поєднувало структуровану базу знань, зручний пошук і сучасний інтерфейс користувача.

На основі проведеного аналізу було визначено актуальність і доцільність розробки нового мобільного додатку, що дозволить ефективно заповнити виявлені прогалини та підвищити якість вивчення фізики за рахунок інтерактивного та адаптивного підходу.

Таким чином, отримані результати аналізу стали підґрунтям для подальшого проєктування та реалізації мобільного застосунку, здатного значно покращити якість навчального процесу і сприяти успішному засвоєнню фізики учнями шкіл.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

На етапі аналізу і визначення вимог до мобільного застосунку важливо було виявлення потреб майбутніх користувачів, обґрунтування вибору технологічних засобів та побудови моделі функціонування майбутньої системи [3], [4]. Особливо важливим це є у контексті навчальних застосунків, де кінцевою метою є не просто функціональність, а ефективна підтримка освітнього процесу.

У якості моделі розробки програмного забезпечення для створення даного мобільного додатку було обрано елементи Agile-підходу, який забезпечує гнучке управління розробкою та дозволяє адаптувати систему до змін вимог на будь-якому етапі життєвого циклу. Реалізація проєкту розбивається на окремі ітерації (спринти), в межах яких розробляються, тестуються та вдосконалюються функціональні модулі системи. Такий підхід дозволяє враховувати відгуки кінцевих користувачів, поступово розширювати функціонал та забезпечувати високу якість продукту на кожному етапі реалізації. Використання Agile також сприяє ефективному розподілу задач у команді, регулярній перевірці виконання цілей і швидкій адаптації до нових вимог.

Архітектурною основою мобільного застосунку стала модель MVVM (Model-View-ViewModel), яка добре зарекомендувала себе у розробці Android-додатків. Вона забезпечує поділ відповідальностей між логікою користувацького інтерфейсу, бізнес-логікою та доступом до даних [5]. Завдяки цьому досягається покращена підтримуваність коду, можливість повторного використання компонентів, а також ефективна організація життєвого циклу даних.

Вимоги до програмного забезпечення були сформовані шляхом вивчення типових сценаріїв використання мобільних навчальних застосунків, а також порівняння з існуючими аналогами. Основними напрямами аналізу стали зручність навігації, швидкість доступу до інформації, гнучкість пошуку формул та зрозумілість подання навчального матеріалу.

Для реалізації поставлених цілей визначено такі функціональні вимоги:

– додаток повинен чітко структурувати формули за категоріями (кінематика, динаміка, молекулярна фізика тощо);

– забезпечення швидкого пошуку формули за ключовими словами або назвою;

– коректне і зрозуміле відображення формул за допомогою спеціалізованої бібліотеки, яка підтримує науково прийняту нотацію;

– кожна формула супроводжується поясненням, де описані позначення та змінні;

– користувач має можливість реєстрації та авторизації в додатку, що забезпечується використанням Firebase Authentication;

– можливість розширення бази формул.

Нефункціональні вимоги:

– інтерфейс програми має бути простим, зрозумілим і інтуїтивним;

– швидкий відгук на дії користувача, мінімальні затримки при пошуку та відображенні формул;

– мінімізація помилок, стабільна робота при тривалому використанні;

– легкість подальшого додавання нових категорій формул або додаткових функцій.

Для формалізації структури та логіки роботи майбутнього мобільного застосунку було використано підхід об’єктно-орієнтованого аналізу. Визначено ключові сценарії взаємодії користувача з системою, серед яких – авторизація, перегляд розділів фізики, вибір конкретної теми, перегляд формули з поясненням, а також пошук за ключовими словами. Ці сценарії дали змогу

виділити основні сутності та функціональні компоненти програмного забезпечення (рис. 2.1).

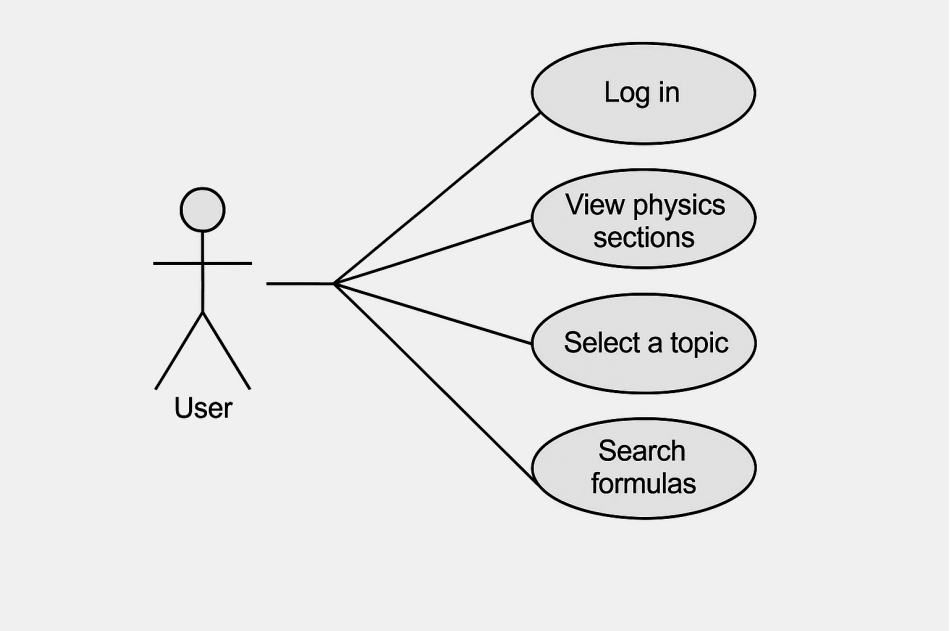

Рисунок 2.1 – Схема прецедентів

У процесі аналізу було сформовано структуру основних класів, що представляють такі об’єкти, як фізична формула, категорія, підтема, користувач, а також ViewModel-компоненти для взаємодії з інтерфейсом і даними. Для кожного класу визначено його поля, методи та взаємозв’язки з іншими елементами системи. Такий підхід дозволив чітко розмежувати функціональні зони відповідальності, забезпечити логічну побудову додатку та спростити його подальшу підтримку і розширення.

Окрему увагу було приділено аналізу послідовності дій користувача під час роботи з додатком – від запуску програми й авторизації до переходу між екранами, вибору розділу та перегляду конкретної формули. Це дало змогу побудувати послідовну логіку інтерфейсу, яка є зрозумілою для кінцевого користувача. Всі переходи між елементами інтерфейсу були організовані за допомогою навігаційної системи, яка передбачає передачу параметрів між екранами для відображення відповідного вмісту (рис. 2.2).

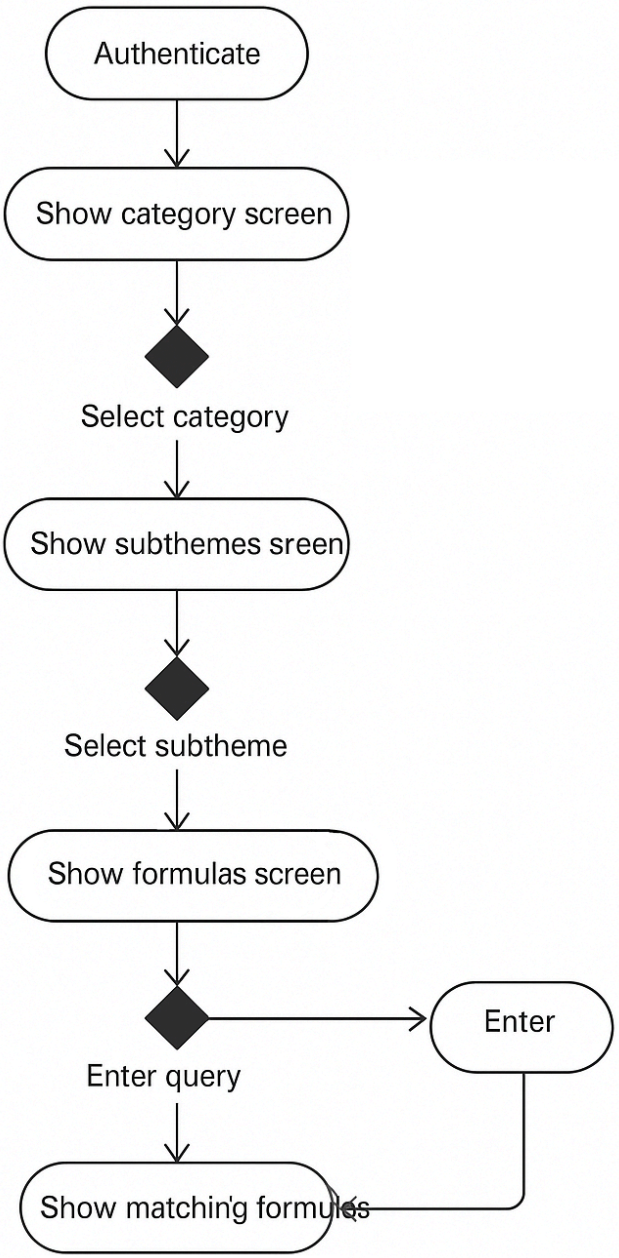

Рисунок 2.2 – Схема активності

Інтерфейс додатку розроблявся з дотриманням принципів зрозумілості, адаптивності та доступності для різних категорій користувачів. Його візуальна структура складається з трьох основних екранів: головного, де відбувається вибір категорії; екрану підтеми з переліком тем; та сторінки, де безпосередньо подається формула з поясненням. Реалізація навігації між цими елементами здійснюється засобами Android Navigation Component, що дозволяє забезпечити надійний механізм переходів і передачу даних між фрагментами.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Розробка мобільного додатку для вивчення фізичних формул вимагає використання сучасних і перевірених інструментів, які здатні забезпечити стабільність, масштабованість, зручність у використанні та підтримку розширеного функціоналу. Для цього було проведено аналіз наявних технологій і засобів, які найчастіше використовуються в Android-розробці освітніх застосунків.

У межах реалізації поставлених задач було розглянуто кілька платформ і фреймворків. Наприклад, Flutter, React Native та Kotlin/Android SDK. Після аналізу переваг і недоліків було обрано Kotlin у зв’язці зі стандартним Android SDK. Основною перевагою такого вибору є повна сумісність з Android API, висока продуктивність, підтримка сучасної архітектури Jetpack, а також активна підтримка з боку спільноти й Google. Kotlin дозволяє писати лаконічний, безпечний і добре підтримуваний код, що особливо важливо в умовах навчального застосунку з потенціалом до розширення.

Основною мовою програмування було обрано Kotlin [6] завдяки її сучасному синтаксису, зручності у використанні, кращій безпеці роботи з типами даних, простішому керуванню потоками і підтримці корутин. Kotlin є офіційно рекомендованою компанією Google для Android-розробки та дозволяє створювати стабільні та продуктивні застосунки.

Для авторизації користувачів було обрано Firebase Authentication. Цей сервіс надає зручні та безпечні засоби для входу до застосунку за допомогою електронної пошти та пароля. Його інтеграція з Android-додатками є простою та не потребує складної серверної логіки, що дозволяє швидко впровадити базову аутентифікацію користувачів.

Для коректного відображення математичних формул було обрано бібліотеку з відкритим вихідним кодом із GitHub (зокрема, бібліотеку MathView від zanvent) [7]. Вибір цієї бібліотеки пояснюється підтримкою простого та

зрозумілого синтаксису (зокрема jqMath-сумісного), можливістю швидкої інтеграції в Android Studio та високою продуктивністю візуалізації формул. Також цей підхід дозволяє коректно й красиво візуалізувати фізичні формули без необхідності ручного малювання або генерування зображень.

Крім того у розробці застосунку важливу роль відіграють компоненти бібліотеки Android Jetpack, які забезпечують стабільну та сучасну архітектуру. Зокрема, використано ViewModel, що дозволяє зберігати та обробляти дані незалежно від життєвого циклу фрагментів, що запобігає втраті інформації під час повороту екрана або перезапуску компонента. Для реалізації реактивного оновлення інтерфейсу при зміні даних використовується LiveData, завдяки чому відображення змін умісту відбувається автоматично без потреби в ручному оновленні.

Організація переходів між екранами реалізується через Navigation Component [8], який також забезпечує зручний механізм передачі аргументів між фрагментами та дотримання єдиної навігаційної структури. Для відображення колекцій даних, таких як списки тем, підтеми чи формул, було використано RecyclerView [9], [10], який дозволяє ефективно працювати з великими обсягами інформації та гнучко налаштовувати вигляд елементів.

Для реалізації архітектурної логіки застосовано паттерн MVVM (Model-View-ViewModel), що дозволяє ефективно відділити логіку представлення інтерфейсу (View) від бізнес-логіки (ViewModel) і моделі даних (Model). Це спрощує тестування та покращує підтримуваність проєкту.

Основні алгоритмічні рішення, що реалізовані в додатку, зосереджені на обробці пошуку формул і відображенні потрібного вмісту відповідно до вибору користувача. Зокрема, реалізовано:

– фільтрацію формул за ключовим словом з використанням простого алгоритму проходження по назвах формул і їхніх описах (без залучення індексації або сторонніх бібліотек);

– мапування підтем до відповідної категорії за допомогою Enum, що забезпечує ефективний доступ до потрібного набору формул;

– побудову навігаційного потоку між фрагментами на основі аргументів, переданих через SafeArgs, що є частиною Android Navigation Architecture Component.

Алгоритми не є складними з точки зору обчислювальної складності, однак вони ретельно оптимізовані під мобільне середовище: забезпечують швидкий відгук інтерфейсу, не блокують головний потік, і виконуються у ViewModel або фоновому режимі через coroutines.

Згідно з вимогами до структури, система моделюється як набір взаємодіючих компонентів:

– класи, що описують сутності (PhysicsFormula, SubTheme, CategoryItem);

– viewModel-класи, які керують даними (MainViewModel, LoginVM, SubTopicViewModel);

– адаптери, які передають дані у вигляді списків до елементів інтерфейсу (FormulaAdapter, CategoryItemAdapter, SubThemeAdapter);

– фрагменти, які є екранами застосунку (MainFragment, AnswerPage, LoginFragment).

Усі ці компоненти пов’язані між собою згідно з правилами MVVM і взаємодіють через спостережувані об’єкти (LiveData) або аргументи, що передаються через Navigation Graph.

Висновки до розділу 2

У результаті проведеного аналізу та проєктування було сформовано чітке уявлення про функціональні та нефункціональні вимоги до мобільного застосунку, визначено основні компоненти системи та їхню взаємодію. Застосування об’єктно-орієнтованого підходу дозволило структурувати програмну архітектуру та окреслити зони відповідальності кожного елемента. Були визначені ключові користувацькі сценарії, сформовано логіку взаємодії між користувачем і додатком, а також запропоновано архітектурну модель на

основі паттерну MVVM, яка є стабільною, масштабованою і добре підтримуваною в межах Android-платформи.

У рамках цього етапу також було обґрунтовано вибір засобів розробки, що найкраще відповідають поставленим задачам. Для побудови інтерфейсу та управління даними були обрані компоненти Android Jetpack, такі як ViewModel, LiveData, Navigation Component та RecyclerView. Вони забезпечують ефективну організацію логіки, гнучке керування станами та просту реалізацію навігації. Крім того, для відображення математичних виразів було адаптовано бібліотеку MathView, яка дозволяє подати формули у зрозумілому та естетичному вигляді.

Також важливою складовою системи стала реалізація модуля авторизації за допомогою Firebase Authentication. Цей сервіс забезпечує безпечний вхід до додатку за допомогою електронної пошти та пароля, з мінімальними витратами на інтеграцію та обслуговування. Його використання дозволило швидко впровадити базову автентифікацію, що є важливим кроком до персоналізації роботи з додатком та захисту користувацьких даних.

Під час проєктування також було враховано принципи доступності, адаптивності та зручності для кінцевого користувача. Всі інтерфейсні рішення були спрямовані на те, щоб зробити роботу з додатком максимально простою та логічною для учнів, які самостійно вивчають фізику.

Таким чином, у розділі було сформовано концептуальну та технічну базу для реалізації програмного продукту, що відповідає сучасним вимогам до мобільного освітнього застосунку. Вибрані засоби й методи забезпечують надійність, гнучкість і зручність у використанні, відкриваючи можливості для подальшого розвитку проєкту.

РОЗДІЛ 3
РОЗРОБКА ДОДАТКУ ДЛЯ ДОПОМОГИ У ВИРІШЕННІ ЗАДАЧ ТА ПОШУКУ ФІЗИЧНИХ ФОРМУЛ

3.1 Практична реалізація об’єкта проєктування

Практичний етап розробки мобільного додатку відбувався у середовищі Android Studio, що забезпечило повний цикл написання, компіляції та відлагодження коду Kotlin. У межах проєкту послідовно реалізовано кілька логічних підсистем: автентифікацію користувачів, навігацію між екранами, відображення ієрархії категорій, підтем, формул та візуалізацію математичних виразів. Нижче подано узагальнені кроки та характерні фрагменти вихідного коду, що ілюструють застосування мови Kotlin та бібліотек Android Jetpack.

Першим був створений модуль автентифікації на основі Firebase Authentication та Firebase UI. Це дало змогу уникнути написання власного backend – сервісу й одночасно підтримувати Email і Google-вхід. У LoginFragment оголошено Activity Result Launcher, який відкриває вже згенеровану Firebase-форму, а результат обробляється у колбеку. Код подано нижче, він демонструє використання Kotlin-синтаксису лямбда-виразів та ViewBinding (ліст. 3.1).

Лістинг 3.1 – Фрагмент системи логіну з LoginFragment

```
private val signInLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult())
{ result → if (result.resultCode == Activity.RESULT_OK) {
        FirebaseAuth.getInstance().currentUser?.let { user →
            Toast.makeText(requireContext(), "Успішний вхід:
${user.displayName}", Toast.LENGTH_SHORT).show()
findNavController().navigate
(R.id.action_loginFragment_to_mainFragment)} }
else {
        Toast.makeText(requireContext(), "Вхід скасовано",
Toast.LENGTH_SHORT).show()}
}
```

кінець лістингу 3.1

Аутентифікаційний стан користувача у додатку реалізовано за допомогою ViewModel у LoginVM, що відповідає за зберігання та обробку відповідних даних незалежно від життєвого циклу інтерфейсних компонентів. У цій ViewModel використовується спеціальна обгортка UserLiveDataFB (ліст. 3.2), яка є реалізацією LiveData і спостерігає за об’єктом FirebaseUser, що надається Firebase Authentication [11], [12]. Завдяки цьому рішення система реагує на зміну стану аутентифікації у режимі реального часу – наприклад, коли користувач входить у свій обліковий запис або виходить з нього.

ViewModel трансформує UserLiveDataFB у змінну authenticatedState, яка зберігає значення перерахування AuthenticationState з можливими станами AUTHENTICATED, UNAUTHENTICATED або INVALID_AUTHENTICATION. Такий підхід забезпечує чітке розділення логіки UI та обробки даних, спрощує тестування, а також дозволяє ефективно керувати поведінкою додатку залежно від того, чи авторизований користувач. Завдяки використанню LiveData компоненти інтерфейсу, такі як LoginFragment, можуть автоматично реагувати на зміну аутентифікаційного стану, не потребуючи ручного оновлення або опитування даних.

Лістинг 3.2 – Фрагмент коду з UserLiveDataFB

```
val authenticatedState = userLiveData.map { firebaseUser →
    if (firebaseUser == null) AuthenticationState.UNAUTHENTICATED
    else AuthenticationState.AUTHENTICATED
}
```

кінець лістингу 3.2

Після успішної авторизації користувач автоматично перенаправляється на головний екран додатку, представлений фрагментом MainFragment. Цей екран виконує роль центрального навігаційного хабу, де відображається список основних тематичних розділів фізики, таких як механіка, оптика, електродинаміка тощо. Для побудови такого списку використовується

компонент RecyclerView, що забезпечує ефективне та адаптивне відображення великої кількості однотипних елементів.

У поєднанні з RecyclerView працює спеціально розроблений адаптер CategoryItemAdapter (ліст. 3.3), який відповідає за зв’язок між даними та візуальним представленням кожного елементу списку. Цей адаптер використовує шаблон ViewHolder для оптимізації повторного використання в’юшок, що значно покращує продуктивність, особливо на менш потужних пристроях.

Кожен елемент списку не лише містить назву розділу та відповідну іконку, а й реалізує обробник кліку, який ініціює навігацію до наступного фрагменту – SubThemeFragment. Передача параметрів відбувається через SafeArgs, що дозволяє гарантовано передати тип обраної категорії (CategoryType) без ризику помилок у час компіляції. Такий підхід забезпечує типобезпечність і спрощує підтримку та масштабування коду.

Лістинг 3.3 – Фрагмент коду з CategoryItemAdapter

```
class CategoryItemAdapter( private val onClick: (CategoryItem) →
Unit ) : ListAdapter<CategoryItem,
CategoryItemAdapter.BaseVH>(Diff()) {
     sealed class BaseVH(itemView: View) :
RecyclerView.ViewHolder(itemView) { class CategoryVH(
     private val binding: ItemCategoryBinding) :
BaseVH(binding.root) {
                  fun bind(item: CategoryItem.Category, onClick:
(CategoryItem) → Unit) { binding.title.text = item.title
         binding.root.setOnClickListener { onClick(item) }}}}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
     BaseVH { val binding =
ItemCategoryBinding.inflate(LayoutInflater.from(parent.context),
parent, false)
             return BaseVH.CategoryVH(binding)
         }

         override fun onBindViewHolder(holder: BaseVH, position:
Int) {
             (holder as BaseVH.CategoryVH).bind(getItem(position) as
CategoryItem.Category, onClick)}}
```

кінець лістингу 3.3

Самі об’єкти, які формують вміст списку категорій на головному екрані, реалізовані з використанням конструкції sealed class у вигляді класу CategoryItem. Це дозволяє чітко визначити обмежену ієрархію можливих типів елементів, які можуть бути передані адаптеру для відображення. Такий підхід є особливо корисним у випадках, коли потрібно мати кілька підтипів даних, але водночас гарантувати, що всі вони контролюються централізовано – наприклад, категорія фізики, підкатегорія.

Для представлення переліку всіх основних фізичних розділів, які можуть бути вибрані користувачем, використовується enum class під назвою CategoryType. Цей клас перераховує всі підтримувані категорії, наприклад: механіка, термодинаміка, електродинаміка, оптика тощо. Кожен елемент enum може містити не лише текстову назву, а й додаткові параметри – наприклад, посилання на ресурс іконки або кольорову тему, що забезпечує гнучке візуальне оформлення кожного розділу (ліст. 3.4).

Лістинг 3.4 – Фрагмент коду з CategoryItem

```
sealed class CategoryItem {
    data class Category(val title: String, val type: CategoryType) :
CategoryItem()
}

enum class CategoryType { MECHANICS, THERMODYNAMICS,
ELECTRODYNAMICS, OPTICS, ATOM, MAGNETISM }
```

кінець лістингу 3.4

Використання sealed class у поєднанні з enum class демонструє потужність і виразність мови Kotlin при проєктуванні типобезпечних, модульних і легко масштабованих моделей даних. Це не лише покращує читабельність і структуру коду, але й дозволяє виявляти помилки ще на етапі компіляції, оскільки всі можливі варіанти типів вже відомі компілятору. Таким чином, система побудови категорій є не лише ефективною, але й відкритою до подальшого розширення, що є важливою вимогою для довгострокового розвитку додатку.

Передача даних із моделі в інтерфейс користувача у додатку реалізована за допомогою окремого класу CategoryItemViewModel, що відповідає за підготовку та надання списку категорій. Цей клас дотримується архітектурного підходу MVVM (Model-View-ViewModel), де саме ViewModel [13] виступає проміжною ланкою між логікою даних і відображенням на екрані.

Під час ініціалізації CategoryItemViewModel створює та публікує LiveData-список елементів, який формується на основі CategoryType.entries. Це означає, що всі доступні категорії, описані в перерахуванні CategoryType, автоматично обробляються й передаються до списку як елементи типу CategoryItem (ліст. 3.5). Таким чином, логіка побудови категорій є централізованою й не залежить від окремих фрагментів – що значно полегшує масштабування додатку.

Лістинг 3.5 – Фрагмент коду з CategoryItemViewModel

```
fun loadCategories(){
        _items.value = CategoryType.entries.map {
CategoryItem.Category(it) }
    }
fun loadSubCategories(parent: CategoryType)
{
        _items.value = SubCategoryType.entries.filter { it.parent ==
parent }.map {
     CategoryItem.SubCategory(it) }
}
```

кінець лістингу 3.5

Такий підхід не лише підтримує реактивну модель оновлення інтерфейсу, але й гарантує стабільність у роботі застосунку, оскільки ViewModel живе довше за фрагмент і зберігає дані навіть при зміні конфігурації (наприклад, при обертанні екрана). Це особливо важливо для мобільних додатків, які повинні бути максимально стійкими до подібних сценаріїв.

Після вибору категорії виконується навігація до SubThemeFragment. Такий перехід є важливою частиною взаємодії з користувачем, оскільки саме він відкриває доступ до глибшої структури навчального контенту. Передача

параметра здійснюється через SafeArgs – спеціальний механізм, який автоматично генерує класи для передачі аргументів між фрагментами. У даному випадку використовується клас MainFragmentDirections, що гарантує типобезпечність і захищає від помилок, пов’язаних з неправильним форматом чи відсутністю даних у навігаційному запиті. Завдяки цьому можна бути впевненим, що під час переходу до SubThemeFragment буде передано саме ту категорію, яку вибрав користувач, і система не зламається в разі внесення змін до структури даних у майбутньому (ліст. 3.6).

Лістинг 3.6 – Фрагмент коду з MainFragment

```
val action =
MainFragmentDirections.actionMainFragmentToSubTheme(category.type)
findNavController().navigate(action)
```

кінець лістингу 3.6

У SubThemeFragment також використовується RecyclerView, однак його функціональне призначення змінено відповідно до ієрархії даних. На цьому етапі програма повинна відображати список підтематичних блоків, які є підкатегоріями вибраної теми. Реалізація цієї логіки здійснюється за допомогою повторного використання CategoryItemAdapter, адаптованого для роботи як з основними категоріями, так і з підтемами. Це рішення дозволяє уникнути дублювання коду й забезпечує гнучкість при обробці різних типів контенту. Незважаючи на те, що початково передбачалося створення окремого адаптера для підтем, подальший аналіз показав доцільність централізації логіки в одному адаптері з умовною фільтрацією на основі переданого CategoryType. Завдяки цьому вдалося зменшити кількість класів, спростити обслуговування коду та пришвидшити початкову розробку, залишаючи можливість подальшого масштабування структури.

Після вибору підтеми відбувається перехід до AnswerPageFragment, у якому користувачеві демонструються всі формули, що належать до відповідної підкатегорії. Переданий параметр підтеми приймається через SafeArgs і

використовується для фільтрації відповідних даних у локальній структурі. Це дозволяє сформувати лише релевантний набір формул, який відповідає запиту користувача, уникаючи зайвого навантаження на інтерфейс. Наведений нижче код демонструє цю логіку в класі AnswerPage (ліст. 3.7).

Лістинг 3.7 – Фрагмент коду з AnswerPage

```
class AnswerPage : Fragment() {
    private lateinit var binding: FragmentAnswerPageBinding
    private val args: AnswerPageArgs by navArgs<AnswerPageArgs>()
    private lateinit var adapter: FormulaAdapter
    private var fullFormulaList = listOf<PhysicsFormula>()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentAnswerPageBinding.inflate(inflater,
container, false)
        return binding.root }
    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        val subcategory = args.subcategory
        fullFormulaList = PhysicsFormulaData.formulas.filter {
it.subcategory == subcategory }

        adapter = FormulaAdapter()
        adapter.submitList(fullFormulaList)

        binding.recyclerView.layoutManager =
LinearLayoutManager(requireContext())
        binding.recyclerView.adapter = adapter }
}
```

кінець лістингу 3.7

Для рендерингу кожної формули у списку використовується власний адаптер FormulaAdapter, реалізований як нащадок ListAdapter. Це дає змогу ефективно працювати зі змінами у списку завдяки використанню механізму DiffUtil, який оновлює лише ті елементи, що дійсно змінилися. У методі onCreateViewHolder створюється ViewHolder із прив’язаним макетом

ItemFormulaBinding, а в методі onBindViewHolder викликається метод bind, що відповідає за заповнення текстових полів та ініціалізацію компонента MathView.

Важливою особливістю FormulaAdapter є використання MathView – спеціального віджета для відображення математичних виразів. Кожна формула подається у вигляді рядка у форматі jqMath (наприклад, v = s / t), що дозволяє без зайвих залежностей забезпечити гарне та зрозуміле візуальне подання навіть складних рівнянь. MathView автоматично масштабує текст до потрібного розміру та застосовує колір, заданий у методі setTextColor, що підвищує читабельність. Код налаштування виглядає наступним чином (ліст. 3.8).

Лістинг 3.8 – Фрагмент коду з FormulaAdapter

```
val mathView = binding.root.findViewById<MathView>(R.id.mathview)
mathView.setText("$$${formula.formula}$$")
mathView.setPixelScaleType(MathView.Scale.SCALE_DP)
mathView.setTextSize(32)
mathView.setTextColor("#111111")
```

кінець лістингу 3.8

Окрім формул, у кожному елементі також відображається назва (formulaTitle) та короткий опис (formulaDescription) відповідного закону чи поняття, що дозволяє користувачам не лише бачити математичний вираз, а й розуміти його фізичний сенс. У поєднанні з ефективною структурою ListAdapter, такий підхід забезпечує високу швидкість рендерингу навіть для великих обсягів даних, що критично важливо для мобільного застосунку. Формули зберігаються у Kotlin-об’єктах як частина структури PhysicsFormulaData, що дозволяє централізовано керувати всім навчальним контентом і забезпечує масштабованість бази знань.

Також для зберігання інформації про кожну фізичну формулу було створено уніфікований клас даних котрий дозволив зберігати всю необхідну інформацію про формулу (назва, формулювання, опис змінних), зв’язати формулу з відповідною підтему, що критично важливо для фільтрації і також передавати об’єкти у списках (наприклад, у RecyclerView) без складної обробки.

Завдяки цій структурі можна ефективно обробляти формули у ViewModel, передавати їх у RecyclerView.Adapter, а також виконувати фільтрацію за темами без дублювання логіки або створення зайвих моделей.

Для централізованого зберігання всіх формул було створено об’єкт-репозиторій. Завдяки централізації даних, усі компоненти, які потребують доступ до формул (ViewModel, фрагменти, адаптери), можуть працювати з єдиним списком, не дублюючи код або логіку завантаження. Це також спрощує можливість експорту, сортування або пошуку.

Також для структурованого зберігання формул за темами було створено окремі об’єкти, кожен з яких відповідає за конкретний розділ фізики (рис. 3.1).

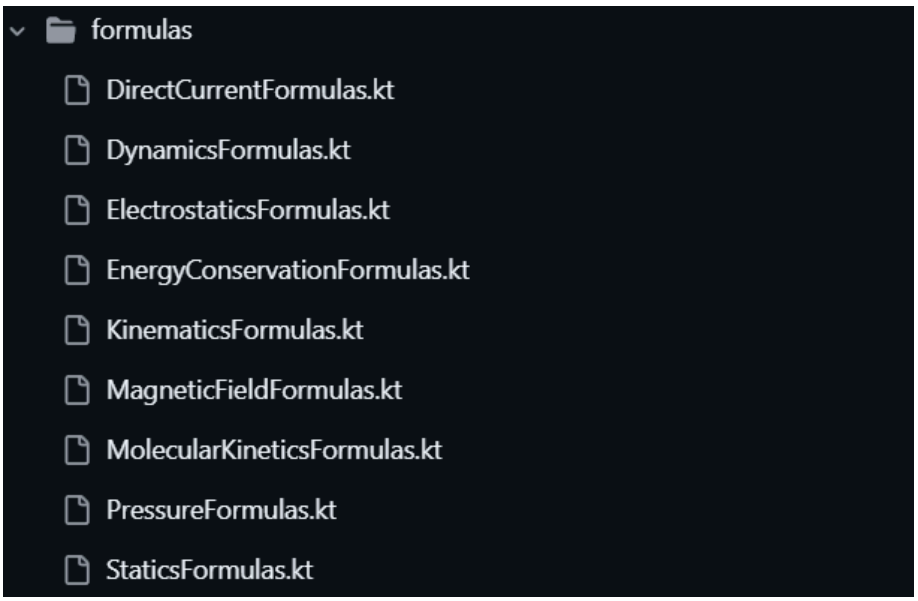

Рисунок 3.1 – Структура зберігання формул

Ці об’єкти містять списки PhysicsFormula, прив’язаних до відповідних SubCategoryType. Така модульна організація дозволила уникнути громіздких переліків формул в одному місці та полегшити читання, редагування та оновлення контенту.

Крім того, це створює передумови для подальшого впровадження тематичних розширень, додавання нових модулів (наприклад, «Оптика» чи «Ядерна фізика») без переробки існуючої логіки. Таким чином, було

забезпечено чисту, гнучку архітектуру даних, яка легко підтримується та масштабується.

Ці рішення не лише спростили роботу з великою кількістю даних, а й зробили застосунок легким у підтримці та розвитку, відповідаючи стандартам сучасної розробки Android-застосунків. У майбутньому планується перенесення цієї бази у віддалене сховище – Firebase Realtime Database. Це дозволить оновлювати контент без потреби перевипуску додатку, підтримувати персоналізацію для кожного користувача та розширити функціональність, наприклад, через можливості залишати нотатки. Завдяки такому підходу зберігається можливість швидкого масштабування, а додаток зможе легше адаптуватися до потреб різних груп користувачів.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Процес тестування додатку відбувався на етапах розробки кожного з функціональних модулів. Основним методом було ручне функціональне тестування, яке проводилось безпосередньо в емуляторі Android Studio. Особлива увага приділялась стабільності роботи при переходах між фрагментами, передачі параметрів за допомогою SafeArgs, рендерингу математичних формул через MathView, а також правильності авторизації через Firebase Authentication.

Налагодження виконувалося за допомогою вбудованих засобів Android Studio: Logcat для виведення діагностичних повідомлень, Layout Inspector – для візуального аналізу елементів інтерфейсу, а також Debug Tool, що дозволяв пошагово перевірити логіку ViewModel та адаптерів. Також використовувалися логи та брейкпоінти для моніторингу змін стану користувача, що значно полегшувало виявлення проблем синхронізації стану UI з даними.

Одним із ключових завдань стало тестування коректного відображення інтерфейсу на різних типах пристроїв – як смартфонах із малими діагоналями, так і планшетах або пристроях з нестандартним співвідношенням сторін. Це

тестування дозволило виявити та усунути проблеми з: масштабуванням тексту та вирівнюванням елементів списків.

Особливу увагу було приділено перевірці відображення формул на сторінці AnswerPage. Тестування охоплювало такі деталі як виведення формул відповідно до вибраної підтеми, наявність пояснення до кожної формули, перевірку пустих станів – тобто як поводиться інтерфейс, якщо у підтему ще не додано жодної формули. Також перевірялась повна відповідність між назвами підтем і формулами, що їм належать, що дозволило виявити логічні помилки при фільтрації.

Оскільки застосунок активно використовує Jetpack Navigation Component із передачею аргументів між фрагментами, важливим етапом тестування стала перевірка стабільності навігації після повороту екрана (щоб аргументи не втрачались), перевірка того, що параметри (SubCategoryType, CategoryType) передаються коректно та було проведено тестування випадків некоректного виклику переходу (наприклад, без аргументу – щоб переконатися, що додаток обробляє такі ситуації стабільно).

Серед виявлених проблем на етапі тестування можна також виділити некоректне повернення до попередніх фрагментів після авторизації – це було усунено завдяки перевірці стеку навігації та коригуванню дій після входу в систему.

Тестування охоплювало як позитивні сценарії (успішний вхід, перехід між категоріями, коректне відображення формул), так і негативні сценарії (скасування входу, відсутність відповідних формул, некоректні параметри навігації). Також перевірялися випадки повторного відкриття фрагментів, взаємодії з RecyclerView та поведінка при зміні орієнтації екрану. Це дозволяло переконатися, що додаток стабільно працює в умовах змін стану системи та вводів користувача.

Застосування різнопланових методів тестування дозволило всебічно оцінити працездатність і стабільність системи, виявити потенційні вузькі місця в інтерфейсі та логіці переходів. Завдяки детальному тестуванню було

забезпечено хорошу якість взаємодії користувача з додатком, а також закладено непогану основу для подальшого масштабування і вдосконалення.

3.3 Інтерфейс користувача

Інтерфейс користувача (UI) відіграє критично важливу роль у мобільному застосунку, особливо якщо йдеться про освітній інструмент, орієнтований на школярів. Основним завданням UI у цьому проєкті було забезпечення інтуїтивно зрозумілого, зручного та логічно структурованого середовища, яке сприятиме легкому доступу до фізичних формул, їх розумінню та подальшому практичному застосуванню.

Під час проєктування інтерфейсу були використані наступні принципи, такі як мінімалізм, тому кожен екран містить лише ті елементи, які є критично необхідними для виконання поточного завдання користувача, також матеріал розділений на теми, підтемі та конкретні формули, що дозволяє послідовно та логічно орієнтуватися у контенті [14]. Ще було використано однаковий шаблон для всіх екранів, це забезпечує сталість сприйняття та мінімізує час адаптації користувача.

Мобільний застосунок побудований за принципом ієрархічної навігації, де кожен екран виконує окрему роль у ланцюгу користувацької взаємодії – від моменту входу до перегляду навчального матеріалу. Така структура дозволяє уникати перевантаження інтерфейсу та забезпечує послідовне ознайомлення з контентом.

При своєму першому відкритті додатку користувач попадає на екран авторизації. Це забезпечення доступу до додатку лише автентифікованим користувачам, а також розмежування логіки навігації залежно від стану входу (рис. 3.2).

Рисунок 3.2 – Екран авторизації

Екран авторизації реалізований у межах MainActivity і містить базову форму для входу або реєстрації через Firebase Authentication. Якщо користувач уже автентифікований, екран автоматично пропускається, і система перенаправляє до основного інтерфейсу.

Після того як користувач зареєструвався, або вже є залогіненим, тоді він отримує доступ до головного екрану (рис. 3.3), де користувачу надають список основних тем фізики (наприклад, «Механіка», «Електродинаміка», «Оптика»),

які відповідають розділам шкільної програми. По кліку на тему, користувач переходить до відповідного SubThemeFragment, де відображаються підтеми.

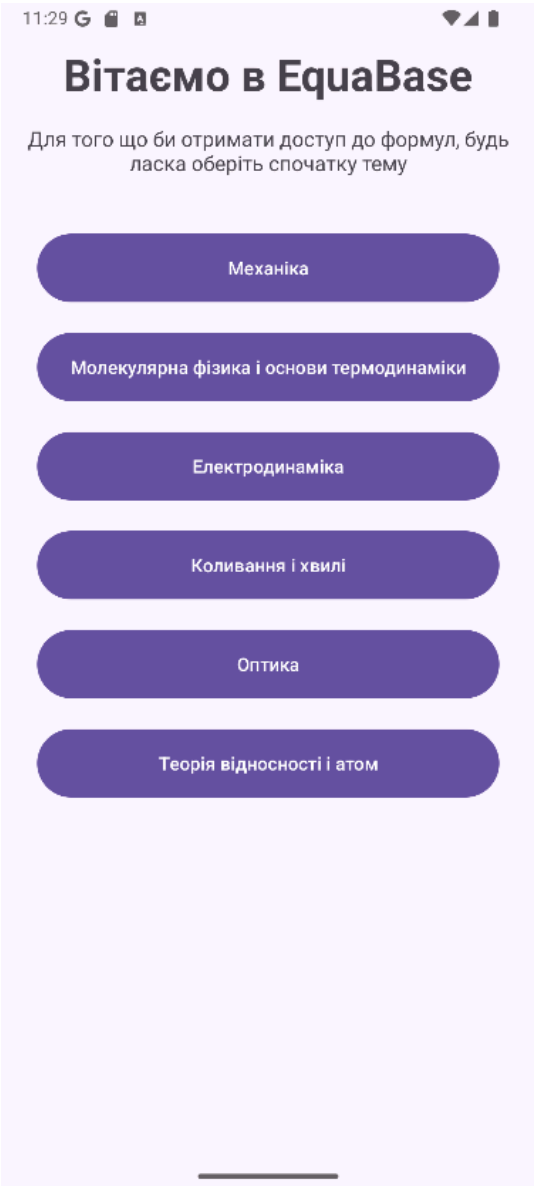

Рисунок 3.3 – Головний екран з темами

Після вибору теми, користувач переходить до списку підтематик (рис. 3.4), які деталізують загальну категорію. Цей екран динамічно формується на основі обраної теми. Реалізовано за тим самим принципом, що і ThemeFragment, але завантаження залежить від обраного CategoryType. Також тут використовується той самий RecyclerView.Adapter, що й для тем, завдяки структурі sealed class CategoryItem.

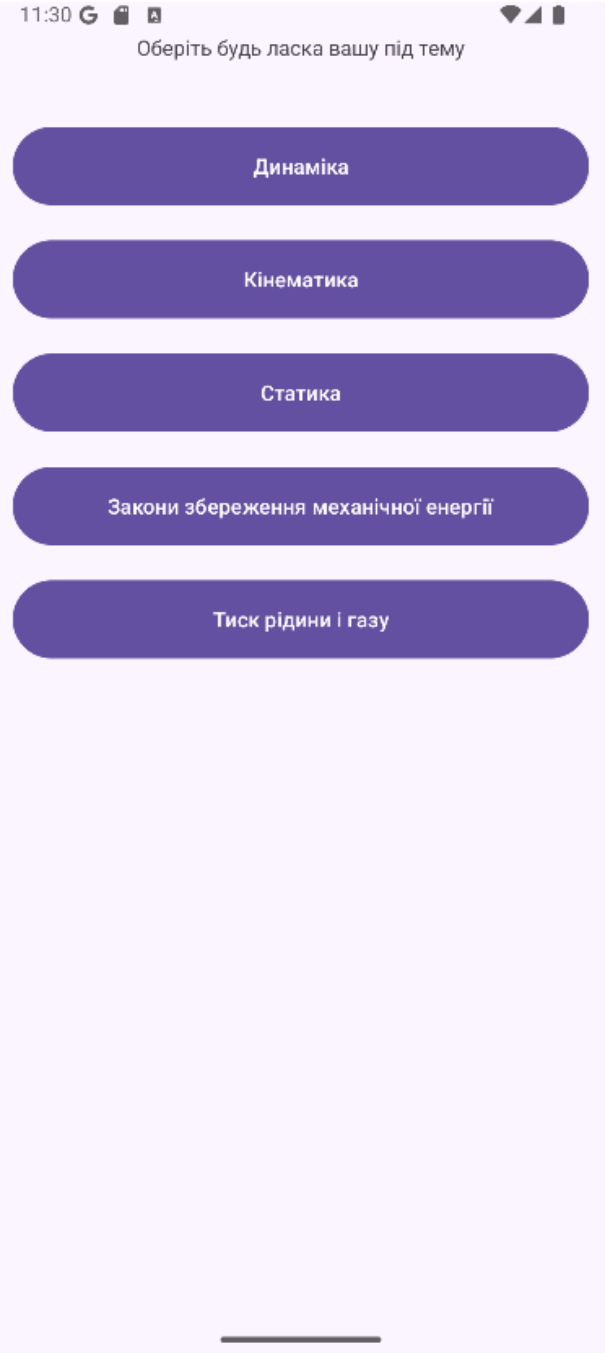

Рисунок 3.4 – Екран з підтемами

Після того як користувач обере свою підтему його перенаправить на останню сторінку, де відображаються всі формули, що належать до вибраної підтеми. Кожна формула виводиться у вигляді картки, яка містить: назву формули, саму формулу, візуалізовану через MathView, та пояснення змінних у текстовому вигляді. Це основний навчальний екран, на якому користувач безпосередньо працює з контентом. Відображення реалізовано через

RecyclerView, що дозволяє оптимізувати продуктивність навіть при великій кількості формул (рис. 3.5).

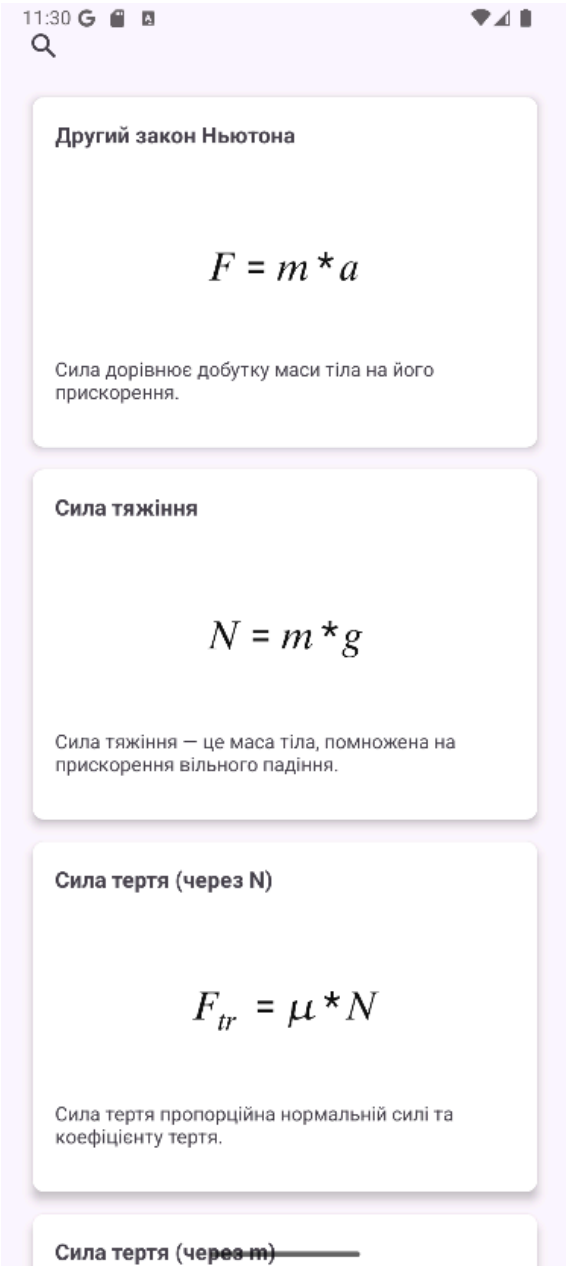

Рисунок 3.5 – Екран з формулами

Однією з важливих переваг використання Jetpack Navigation Component та побудови структури застосунку через NavGraph є можливість гнучкого налаштування анімацій переходів між екранами. Хоча ця можливість є факультативною з погляду функціональності, вона суттєво впливає на загальне

сприйняття інтерфейсу користувачем, підвищуючи комфорт роботи із застосунком.

У рамках реалізації навігації в nav_graph.xml можна задавати анімації входу, виходу, появи при поверненні назад тощо. Ці налаштування визначають, як саме будуть візуально поводитись екрани під час переходу між ними. Завдяки цьому застосунок виглядає живим, плавним та приємним у використанні, а взаємодія з ним стає більш природною та естетичною.

3.4 Розробка бази даних

У межах реалізації мобільного додатку зберігання навчального контенту реалізовано за допомогою вбудованих Kotlin-структур. Вся база формул на поточному етапі проєкту представлена у вигляді об’єкта PhysicsFormulaData, який містить загальний список об’єктів типу PhysicsFormula. Ці об’єкти включають такі атрибути, як назва формули, її математичний запис у форматі jqMath, короткий опис, а також категорію та підтему, до яких вона належить. Це дозволяє виконувати швидку фільтрацію за категорією або підтематикою.

Щоб зберегти модульність та розділення логіки за фізичними темами, формули згруповані у відповідні пакети, наприклад KinematicsFormulas.kt, DynamicsFormulas.kt, StaticsFormulas.kt тощо. Кожен із них містить список формул, що стосуються певної підтеми, які потім централізовано імпортуються до об’єкта PhysicsFormulaData. Поточна реалізація виглядає так (ліст. 3.9).

Лістинг 3.9 – Фрагмент коду з PhysicsFormulaData

```
object PhysicsFormulaData {
    val formulas: List<PhysicsFormula> = listOf(
        *KinematicsFormulas.list.toTypedArray(),
        *DynamicsFormulas.list.toTypedArray(),
        *StaticsFormulas.list.toTypedArray(),
        *EnergyConservationFormulas.list.toTypedArray(),
        //*OpticsFormulas.list.toTypedArray()
    ) }
```

кінець лістингу 3.9

Такий спосіб організації бази даних спрощує підтримку та оновлення контенту, а також дозволяє легко масштабувати систему – достатньо додати нову підтему у вигляді окремого об’єкта з формулами, і підключити її у PhysicsFormulaData. Завдяки модульній структурі, розробники можуть додавати нові теми без ризику порушити цілісність існуючих даних. Крім того, це значно полегшує локалізацію контенту для інших мов або адаптацію формул для різних рівнів складності – шкільного, ліцейного чи навіть університетського.

На наступних етапах розвитку додатку передбачено перенесення поточної структури бази формул у хмарне середовище, зокрема у Firebase Realtime Database [15]. Для цього необхідно буде реалізувати схему даних у форматі JSON, де кожна формула матиме власний унікальний ідентифікатор і зберігатиметься як окремий запис. Далі планується перенести вміст об’єктів, таких як KinematicsFormulas, DynamicsFormulas та інших, у відповідні колекції Firebase, зберігаючи структуру та розділення за темами.

Окрім цього, необхідно буде реалізувати механізм синхронізації: під час першого запуску програми дані мають автоматично завантажуватись із хмари та зберігатися локально для забезпечення офлайн-доступу. Також потрібно буде змінити джерело даних у класі PhysicsFormulaData – замість локальних списків, обробка повинна відбуватись через LiveData або Flow, які будуть безпосередньо пов’язані з Firebase-джерелом. Для безпечного використання контенту варто налаштувати правила доступу Firebase так, щоб звичайні користувачі мали змогу лише переглядати дані, без можливості їхнього редагування або видалення.

Такий підхід дозволить оновлювати базу формул без публікації нових версій додатку, реалізовувати гнучкі механізми пошуку, додавання коментарів, рейтингу формул, а в майбутньому – персоналізовані добірки або інтеграцію з системами управління навчанням. Перехід до зовнішньої бази – це також крок до розширення додатку у напрямку багатокористувацьких функцій та аналітики навчального процесу.

3.5 Розробка документації для програмного продукту

Для стабільної роботи мобільного додатку мінімальні системні вимоги включають пристрій з операційною системою Android 8.0 (API level 26) або вище, не менше 100 МБ вільного місця на внутрішньому сховищі, та активне підключення до інтернету при першому запуску або у випадку використання хмарних сервісів (наприклад, Firebase Authentication для того щоби увійти в систему).

З метою забезпечення зручності користування додатком було реалізовано чітку та послідовну логіку взаємодії з інтерфейсом. Після запуску програми користувач потрапляє на екран авторизації або реєстрації, оскільки доступ до основного функціоналу надається лише зареєстрованим користувачам. Цей етап реалізовано за допомогою Firebase Authentication, що дозволяє швидко здійснити вхід або створити новий обліковий запис. Після успішної автентифікації відкривається головний екран із тематичними категоріями фізики (рис. 3.6).

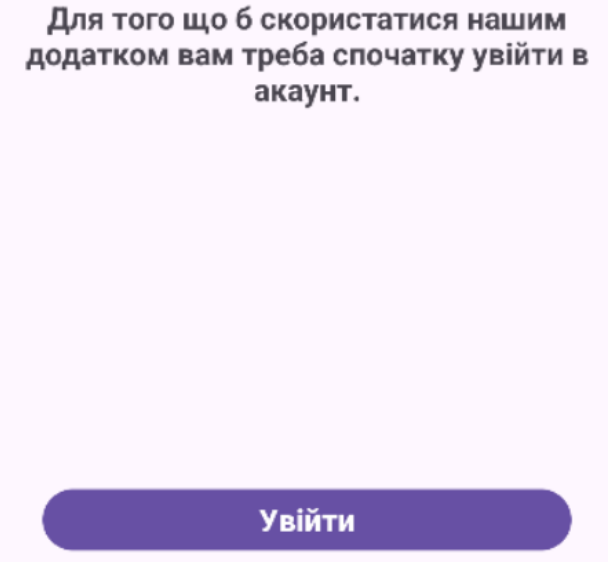

Рисунок 3.6 – Екран з проханням увійти до аккаунту

Після входу в додаток користувач потрапляє на головний екран, де його вітає повідомлення з короткою інструкцією: для перегляду формул необхідно

обрати одну з представлених тем. Категорії відображаються у вигляді інтерактивного списку, що реалізований через RecyclerView, а для кожного розділу вказано відповідну назву та іконку, що полегшує візуальне орієнтування (рис. 3.7).

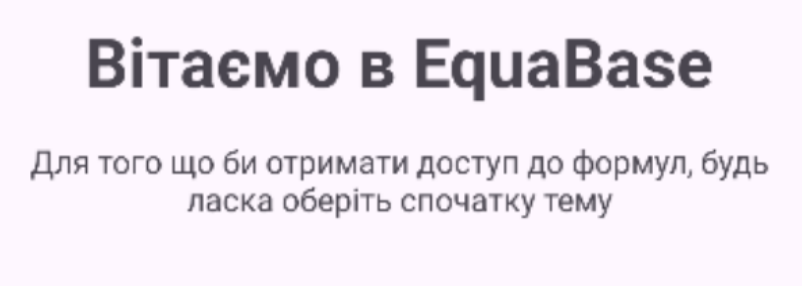

Рисунок 3.7 – Екран з проханням обрати тему

Після вибору категорії відкривається новий екран, у верхній частині якого користувачеві знову надається інструкція – тепер необхідно обрати конкретну підтему (рис. 3.8). Це дозволяє користувачу поступово занурюватися у потрібний контекст, не перевантажуючи його зайвою інформацією одразу. Лише після вибору підтеми відкривається основний функціональний екран, де подано перелік формул, що відповідають вибраній тематиці.

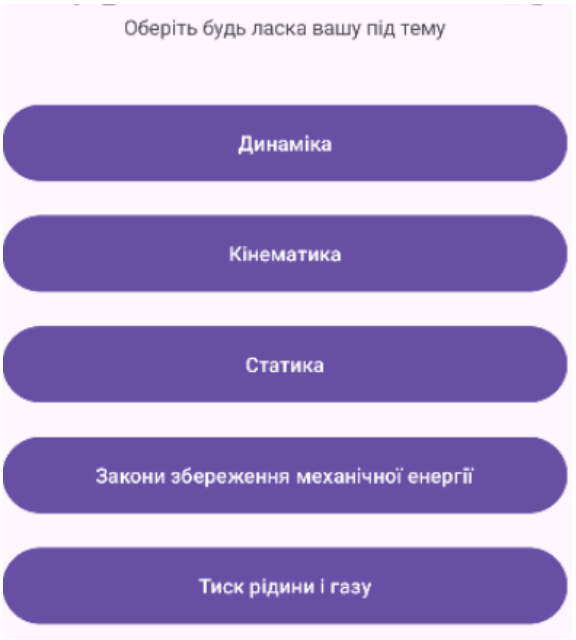

Рисунок 3.8 – Екран з проханням обрати тему

Такий багаторівневий підхід забезпечує логічну послідовність дій, зменшує ймовірність помилок при навігації та допомагає користувачеві сфокусуватись на потрібному матеріалі.

У наступних версіях планується додати окремий розділ «Допомога» з підказками щодо використання ключових функцій додатку, пошуку формул та збереження обраного матеріалу.

Висновки до розділу 3

У результаті вдалося повністю втілити основну функціональність мобільного додатку для роботи з фізичними формулами. У процесі розробки було створено адаптивний інтерфейс із чіткою логікою навігації між категоріями, підтемами та відповідними формулами. Для виведення динамічного вмісту застосовано RecyclerView з кастомними адаптерами, що дозволяють зручно масштабувати структуру знань та підтримувати її оновлення. Аутентифікація реалізована через Firebase Authentication, забезпечуючи надійний вхід користувачів через електронну пошту або аккаунт Google.

Отже, створений мобільний застосунок демонструє готовність до використання у навчальному процесі та може стати основою для подальшого розвитку. Його можна доповнити новими розділами фізики, реалізувати інтерактивні вправи або візуалізації, а також інтегрувати з зовнішніми освітніми платформами крім того як майбутнє оновлення додатку, можна добавити ще одне вікно, де при натисканні на саму плитку з формулою користувачеві відкриється ще одне вікно де буде подано матеріал з більшим описом та прикладами. Отриманий результат підтверджує доцільність вибраної архітектури, ефективність інструментів і високу прикладну цінність проєкту.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено повнофункціональний мобільний застосунок, призначений для полегшення вивчення фізики шляхом надання доступу до структурованої бази формул з інтерактивним інтерфейсом. Проєкт реалізовано на сучасній технологічній основі з використанням мови Kotlin, архітектурного підходу MVVM та інструментів Android Jetpack, що відповідає актуальним тенденціям розробки мобільних застосунків у світовій практиці.

Досягнуті результати повністю задовольняють поставлену задачу – створити зручну й масштабовану систему для швидкого доступу до фізичних формул із можливістю пошуку, фільтрації та відображення в математичному вигляді. У межах роботи впроваджено модульну структуру, що полегшує оновлення навчального контенту, та забезпечено інтеграцію з Firebase Authentication для обліку користувачів. Додаток протестовано в реальних умовах, що підтвердило його стабільність, швидкодію та відповідність функціональним вимогам.

Результати роботи мають практичне значення для шкільних та позашкільних освітніх установ, репетиторських центрів, а також індивідуального використання учнями. Завдяки інтуїтивному інтерфейсу додаток може бути адаптований для широкого кола користувачів, включаючи англомовну аудиторію. Перспективи подальшого розвитку передбачають інтеграцію з хмарною базою даних, розширення функціоналу (наприклад, додавання режиму тестування, збереження обраного, багатомовність) та можливість використання в межах національної освітньої платформи.

У результаті виконання кваліфікаційної роботи було повністю реалізовано всі етапи, визначені в індивідуальному завданні.

Під час виконання проєкту було детально проаналізовано наявні мобільні рішення, зокрема додатки «Mechanical Engineering Calcula» від Trelleborg Sealing Solutions та «Physics Formula» від Codebug. Перший з них не

орієнтований на шкільний курс фізики і розрахований переважно на інженерні розрахунки, в той час як другий є лише довідником без чіткої структуризації матеріалу за темами та без навчального підходу. Обидва продукти не враховують потреб українських школярів, що й стало підґрунтям для розробки власного спеціалізованого додатку з фокусом на зручність, адаптивність і навчальну цінність.

Особливу увагу приділено структурі навчального матеріалу: всі фізичні формули були класифіковані за темами та підтематичними розділами відповідно до шкільної програми. Кожен запис містить не лише математичний вираз, але й текстове пояснення до кожної змінної, що полегшує розуміння фізичних залежностей. Це дозволило створити базу знань, яка не є просто списком формул, а навчальним ресурсом із потенціалом для подальшого використання в освітньому процесі.

Архітектура додатку була спроєктована з урахуванням сучасних принципів розробки – модель MVVM забезпечує чітке розділення логіки та представлення, що спрощує підтримку й тестування. Для навігації між фрагментами використано Jetpack Navigation Component з SafeArgs, що підвищує безпечність передачі параметрів. Firebase було обрано як зовнішню платформу для реалізації користувацьких сервісів, оскільки вона надає зручні інструменти для аутентифікації, хмарного зберігання та аналітики.

Аутентифікація користувачів реалізована на основі Firebase Authentication, що дозволяє входити до системи за допомогою Google або електронної пошти. Для цього було створено окремий екран логіну з підтримкою ActivityResult API, а логіка авторизації інтегрована у ViewModel з підтримкою LiveData для відслідковування стану користувача. Такий підхід забезпечує гнучкість, розширюваність та безпечність облікових записів.

Для підвищення зручності використання додатку реалізовано функціонал пошуку формул за ключовими словами. Це дозволяє учням швидко знаходити потрібну формулу без потреби переходити вручну через всі підкатегорії. Відображення математичних виразів здійснюється за допомогою компонента

MathView із підтримкою синтаксису jqMath, що дає змогу рендерити формули у зручному для сприйняття форматі, максимально наближеному до вигляду друкованих підручників.

Інтерфейс додатку реалізовано з використанням RecyclerView, що дозволяє ефективно відображати списки категорій, підтематичних розділів та формул. Адаптивний дизайн забезпечує коректну роботу як на телефонах, так і на планшетах із різними розмірами екранів. Було також приділено увагу інтерактивності: кліки на елементах ведуть до відповідних екранів, а обробка станів здійснюється через ViewModel, що відповідає принципам реактивного програмування.

Додаток пройшов тестування на відповідність функціональним вимогам. Було перевірено роботу навігації, авторизації, фільтрації формул та відображення математичних виразів. Тестування проводилося вручну. Усі виявлені недоліки були оперативно виправлені, а результатом стала стабільна та продуктивна система, готова до використання в освітньому середовищі.

На завершальному етапі було підготовлено документацію до додатку, описано його структуру, системні вимоги, алгоритми роботи та інструкції для користувачів. Було також проаналізовано можливості подальшого розвитку: перенесення бази формул у Firebase, додавання функції закладок, інтеграція з тестовими модулями та локалізація інтерфейсу. Це створює ґрунт для подальшого масштабування продукту та його використання в ширшому освітньому контексті.

У результаті створено повноцінний, функціонально завершений мобільний застосунок, який може слугувати базою для подальшого розвитку – зокрема, розширення бази формул, додавання інтерактивних завдань, тестів чи візуалізацій фізичних процесів. Проведене тестування підтвердило стабільну роботу системи, коректну обробку навчального контенту та відповідність функціональності поставленим цілям. Розроблене програмне рішення має практичну цінність для освітньої сфери та потенціал до подальшої інтеграції в навчальні процеси.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Apps by Trelleborg Sealing Solutions on Google Play. URL: https://play.google.com/store/apps/developer?id=Trelleborg+Sealing+Solutions (date of access: 04.02.2025).

2. Codebug. Physics Formula - Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.codebug.physics.formulas (date of access: 05.02.2025).

3. Android Developers – Official Documentation. Android Developers. URL: https://developer.android.com (date of access: 12.02.2025).

4. Android Performance Patterns (Google Dev YouTube Series). URL: https://www.youtube.com/playlist?list=PLOU2XLYxmsIJGErt5rrCqaSGTMyyqNtH (date of access: 20.02.2025).

5. MindOrks – Android MVVM Architecture. URL: https://mindorks.com/post/mvvm-architecture-android-tutorial-for-beginners-step-by-step-guide (date of access: 05.03.2025).

6. Kotlin Documentation. URL: https://kotlinlang.org/docs/home.html (date of access: 08.03.2025).

7. Zanvent MathView GitHub. URL: https://github.com/zanvent/MathView (date of access: 15.03.2025).

8. Jetpack Navigation Component. Android Developers. URL: https://developer.android.com/guide/navigation (date of access: 25.03.2025).

9. Android Developers. RecyclerView Overview. URL: https://developer.android.com/guide/topics/ui/layout/recyclerview (date of access: 01.04.2025).

10. Raywenderlich – RecyclerView in Android. URL: https://www.raywenderlich.com/262-developing-android-apps-with-kotlin (date of access: 08.04.2025).

11. DigitalOcean – Firebase Authentication with Android. URL: https://www.digitalocean.com/community/tutorials/firebase-authentication-android (date of access: 14.04.2025).

12. Firebase Documentation. URL: https://firebase.google.com/docs (date of access: 18.04.2025).

13. UX Planet – Mobile App UX Best Practices. URL: https://uxplanet.org/mobile-app-ux-design-best-practices-7c54aafdff1b (date of access: 21.04.2025).

14. Material Design Guidelines. Material Design. URL: https://m3.material.io (date of access: 07.05.2025).

15. Read and Write Data on Android. Firebase Realtime Database. URL: https://firebase.google.com/docs/database/android/read-and-write (date of access: 19.05.2025).