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

Хмарно-стрімінговий медіа плеєр для лекцій та навчальних матеріалів. Частина 1.Compose система

ПОЯСНЮВАЛЬНА ЗАПИСКА

до магістерської дипломної роботи

освітній ступінь **магістер**

08-23.КМКР.027.00.000 ПЗ

Виконала: студентка 2 курсу, групи 1КІ-20м
напряму підготовки (спеціальності)
123 - «Комп'ютерна інженерія»

 Левицька Ю.Р.

Керівник ___ ___к.т.н., доц. Ткаченко О.М.
(прізвище та ініціали)

2021р

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень - магістр

Спеціальність 123 - «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

обчислювальної техніки

д.т.н., проф. Азаров О.Д.

_____

«____» _____2021 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

_____Левицькій Юлії Русланівні_____
(прізвище, ім'я, по батькові)

1 Хмарно-стрімінговий медіа плеєр для лекцій та навчальних матеріалів. Частина 2. Compose система
керівник роботи Ткаченко Олександр Миколайович, кандидат технічних наук, доцент, кафедри обчислювальної техніки затверджені наказом Вінницького національного технічного університету від 26.09.2021 року №_277.

2 Строк подання студентом роботи: 05.12.2021

3 Вихідні дані до роботи _Проаналізувати методи і засоби розробки додатків на основі операційної системи Android. Розробити способи побудови сучасного користувацького макету .

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): _розглянути та проаналізувати методи та засоби

розробки додатків на основі операційної системи Android; розробити макет користувацького інтерфейсу; реалізувати програмно всі складові.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Лістинги скетчів та фото розроблювальних пристроїв

6 Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Ткаченко О.М к.т.н., доцент каф. ОТ		
4	Кавецький В. В., к.е.н., доцент каф. ЕПВМ		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Огляд і аналіз джерел інформації	6.09.2021	Виконано
	Розробка технічного завдання	29.09.2021	Виконано
	Огляд та аналіз методів та засобів побудови сучасних макетів у системах Android	21.10.2021	Виконано
	Розробка кресленого та програмного графу макету	27.10.2021	Виконано
	Реалізація програмного додатку системи Android	24.11.2021	Виконано
	Підготовка до презентації і захист роботи	25.11.2021	Виконано

Студент _____ _____Левицька Ю.Р._____
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Ткаченко О.М.___
(підпис) (прізвище та ініціали)

Консультант з економічної частини _____ ___Кавецький В. В.___
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Проаналізовано сучасні макети на основі Compose. Обрано оптимальний макет та програмне забезпечення. Проаналізовано існуючі додатки на ОС Andrioid , макети побудови сучасного та інтуітивно зрозумілого користувацького інтерфейсу.

Розроблено графічний UI та UX програмного забезпечення для додатку, розроблений функціональний навігаційний граф додатку. Проведено тестування на AVD та APD. Наведено характеристики, переваги і недоліки, рекомендації щодо їх застосування.

ABSTRACT

Modern Compose-based layouts are analyzed. The optimal layout and software are selected.

The existing applications on the Andrioid OS, layouts for building a modern and intuitive user interface are analyzed.

Developed graphical UI and UX software for the application, developed a functional navigation graph of the application. AVD and APD testing performed. The characteristics, advantages and disadvantages, recommendations for their use are given..

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

OS (ОС) — операційна система

ПЗ — програмне забезпечення

API— Application Programming Interfaces

APK — Application Package Kit

GUI — Graphical User Interface

JSON — JavaScript Object Notation

SDK — Software Development Kit

XML — Extensible Markup Language — розширювана мова розмітки

AVD — Android Virtual Device

APD — Android Physical Device

ВСТУП

Актуальність теми**.** З появою сучасних операційних систем, таких як Android і iOS, стали актуальними завдання, які раніше вирішувались лише для комп'ютерних операційних систем. Правельний вибір макета життєво необхідний для призначеного користувача та інтерфейсу Android.

Макет дозволяє позиціонувати та розміщувати компоненти. Добре представлення того, що використовується в кожному з наявних паттернів, може реалізувати різницю між простим й складним призначеним для користувача UI та UX як з точки зору розробника , так і як користувача. Це особливо доречно, якщо потрібно підтримувати деякий ряд пристроїв.

Для кожного додатка є необхідність використання кількох макетів, що допомагає створити інтерфейс простим у використанні, сучасним та цікавим. Тому актуальним є дослідження існуючих технологій для створення інтерфейсу, вибір найбільш оптимального вирішення в залежності від поставленої умови. Операційна система Android підтримує такі мови програмування: Kotlin та Java. Для проведення дослідження буде використано мову програмування Kotlin.

Об’єкт дослідження — процес створення макету користувацького інтерфейсу та відповідного програмного забезпечення.

Предмет дослідження — методи та засоби створення гнучких макетів користувацького інтерфейсу.

Мета роботи — розширення функціональних можливостей мобільних пристроїв за рахунок розроблення мобільного додатку на ОС Android, що забезпечує сучасний макет користувацького інтерфейсу. Для досягнення вищевказаної мети необхідно виконати такі **задачі:**

— проаналізувати галузі застосування мобільних пристроїв;

— класифікувати макети;

— розглянути існуючі засоби строрення користувацьких макетів ;

— розробити мобільний додаток на ОС Andriod ;

— розробити UI та UX для мобільного додатку;

— провести діагностику роботи розробленого додатку;

— розробити рекомендації та пропозиції про застосування додатку на практиці.

Наукова новизна роботи полягає в тому, що в ній вдосконалено Compose модель написання додатків, в якій, на відміну від існуючих, дані зберігаються у хмарній базі данних Firebase та у локальній базі даних Room, що дозволило підвищити швидкість відображення даних завдяки відсутнсті затримок у передачі даних.

Практична цінність роботи полягає в тому, що на базі проведених теоретичних досліджень розроблено мобільний додаток для доступу до навчальних матеріалів на ОС Android.

Апробація. Основні результати роботи доповідалися та були схвалені на Всеукраїнській науково-практичній інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2021) (м. Вінниця, 01.05.2021 — 14.05.2021)

Публікації. За результатами дослідження було опубліковано 2 тези доповіді: ДОСЛІДЖЕННЯ BACKEND AS A SERVICE ПЛАТФОРМ. РОЗРОБКА ЗВ'ЯЗКІВ З ПЛАТФОРМОЮ FIREBASE.. [Текст] Ю.Р.Левицька // Молодь в науці: дослідження, проблеми, перспективи (МН-2021) Тез. доп. - Вінниця, 2021. - 1. Режим https://conferences.vntu.edu.ua/index.php /mn/mn2021/paper/view/13144/11053 [1].

ДОСЛІДЖЕННЯ JETPACK COMPOSE UI FRAMEWORK ДЛЯ ANDROID. РОЗРОБКА ЗВ'ЯЗКІВ З FRAMEWORK COMPOSE.. [Текст] Ю.Р.Левицька // Молодь в науці: дослідження, проблеми, перспективи (МН-2021) Тез. доп. - Вінниця, 2021. - 1. Режим https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/14244

1 ЗАСОБИ РОЗРОБКИ

1.1 Огляд написання додатків на ОС Android

Програми для Android можна писати за допомогою мов Kotlin, Java та C++. Інструменти Android SDK компілюють ваш код разом із будь-якими даними та файлами ресурсів у файл .apk або пакет Android App Bundle.

Пакет Android, який є архівним файлом із суфіксом .apk, містить вміст програми для Android, необхідний під час виконання, і це файл, який пристрої на базі Android використовують для встановлення програми.[1]

Android App Bundle, який є архівним файлом із суфіксом .aab, містить вміст проекту програми Android, включаючи деякі додаткові метадані, які не потрібні під час виконання. AAB — це формат публікації, який не можна встановити на пристроях Android, він відкладає створення файлів .apk і підписання на більш пізній етап. Наприклад, під час розповсюдження вашої програми через Google Play сервери Google Play генерують оптимізовані файли .apk, які містять лише ресурси та код, необхідні для конкретного пристрою, який запитує встановлення програми.

Кожна програма Android працює у власному пісочнику безпеки, захищений такими функціями безпеки Android. Операційна система Android — це багатокористувацька система Linux, в якій кожен додаток є іншим користувачем. За замовчуванням система призначає кожній програмі унікальний ідентифікатор користувача Linux (ідентифікатор використовується тільки системою і невідомий програмі). Система встановлює дозволи для всіх файлів у програмі, щоб лише ідентифікатор користувача, призначений цій програмі, мав доступ до них. Кожен процес має свою власну віртуальну машину (VM), тому код програми працює ізольовано від інших програм. [2]

За замовчуванням кожна програма запускається в власному процесі Linux. Система Android запускає процес, коли потрібно виконати будь-який з

компонентів програми, а потім завершує процес, коли він більше не потрібен або коли системі потрібно відновити пам’ять для інших програм.

Система Android реалізує принцип найменших привілеїв. Тобто кожен додаток за замовчуванням має доступ лише до компонентів, які йому потрібні для виконання своєї роботи, і не більше. Це створює дуже безпечне середовище, в якому програма не може отримати доступ до частин системи, на які їй не надано дозволу. Однак є способи для програми обмінюватися даними з іншими програмами і для програми доступу до системних служб.

Можна організувати, щоб дві програми використовували один і той самий ідентифікатор користувача Linux, і в цьому випадку вони могли отримувати доступ до файлів один одного. Щоб заощадити системні ресурси, програми з однаковим ідентифікатором користувача також можуть працювати в одному процесі Linux і використовувати одну віртуальну машину. Програми також мають бути підписані тим самим сертифікатом.

Додаток може запитувати дозвіл на доступ до даних пристрою, таких як місцезнаходження пристрою, камера та з’єднання Bluetooth. Користувач повинен явно надати ці дозволи. Додаткову інформацію див. у розділі Робота з системними дозволами.Решта цього документа вводить такі поняття. Основні компоненти платформи, які визначають вашу програму. Файл маніфесту, у якому оголошуєте компоненти та необхідні функції пристрою для вашої програми. Ресурси, які відокремлені від коду програми та дозволяють вашій програмі витончено оптимізувати свою поведінку для різних конфігурацій пристрою. Компоненти програми є основними будівельними блоками програми для Android. Кожен компонент є точкою входу, через яку система або користувач можуть увійти у вашу програму. Активність - це точка входу для взаємодії з користувачем. Він являє собою один екран з інтерфейсом користувача, програма електронної пошти може мати одну дію, яка показує список нових листів, іншу дію для створення електронного листа та іншу дію для читання електронних

листів. Незважаючи на те, що ці дії працюють разом, щоб сформувати згуртований досвід роботи користувачів у програмі електронної пошти, кожна з них не залежить від інших. Таким чином, інша програма може розпочати будь-яку з цих дій, якщо програма електронної пошти дозволяє це. Наприклад, програма для камери може розпочати дію в програмі електронної пошти, яка створює нову пошту, щоб дозволити користувачеві поділитися фотографією. Діяльність сприяє такій ключовій взаємодії між системою та програмою.

Відстеження того, що в даний момент цікавить користувача (що відображається на екрані), щоб система продовжувала виконувати процес, на якому розміщена активність. Знаючи, що раніше використовувані процеси містять речі, до яких користувач може повернутися (припинені дії), і, таким чином, надає більший пріоритет збереження цих процесів. Допомагає програмі впоратися з припиненням процесу, щоб користувач міг повернутися до діяльності з відновленим попереднім станом. Забезпечує додаткам спосіб реалізовувати потоки користувачів між собою, а система координувати ці потоки. Найкласичнішим прикладом тут є share. Реалізуєте дію як підклас класу Activity. Щоб отримати докладнішу інформацію про клас Activity, перегляньте посібник для розробників Activities.

Служба — це точка входу загального призначення для підтримки роботи програми у фоновому режимі з різних причин. Це компонент, який працює у фоновому режимі для виконання довготривалих операцій або роботи для віддалених процесів. Сервіс не надає інтерфейс користувача. Наприклад, служба може відтворювати музику у фоновому режимі, коли користувач перебуває в іншій програмі, або може отримувати дані через мережу, не блокуючи взаємодію користувача з діяльністю. Інший компонент, наприклад, діяльність, може запустити службу і дозволити їй запуститися або прив’язатися до неї, щоб взаємодіяти з нею. [5]

Запущені служби повідомляють системі продовжувати їх роботу, поки їх робота не буде завершена. Це може бути синхронізація деяких даних у фоновому режимі або відтворення музики навіть після того, як користувач покине програму. Синхронізація даних у фоновому режимі або відтворення музики також представляють два різних типи запущених служб, які змінюють те, як система їх обробляє. Відтворення музики - це те, про що користувач безпосередньо знає, тому програма повідомляє про це системі, кажучи, що хоче бути на передньому плані з сповіщенням, щоб повідомити про це користувачеві; у цьому випадку система знає, що їй слід дуже старатися, щоб процес цієї служби працював, тому що користувач буде незадоволений, якщо він зникне. Звичайна фонова служба не є тим, що користувач безпосередньо знає як запущений, тому система має більше свободи в управлінні своїм процесом. Це може дозволити його знищити (а потім перезапустити службу через деякий час), якщо йому потрібна оперативна пам’ять для речей, які найбільше турбують користувача. Прив’язані служби запускаються, тому що якась інша програма (або система) повідомила, що хоче скористатися послугою. В основному це служба, що надає API для іншого процесу. Таким чином, система знає, що між цими процесами існує залежність, тому, якщо процес А прив’язаний до служби в процесі В, вона знає, що їй потрібно підтримувати процес В (і його службу) в роботі для А. Крім того, якщо процес А є чимось користувачеві турбується, тоді він також знає, що процес В має розглядатися як те, що його також хвилює. [3]

Завдяки своїй гнучкості (на краще чи на гірше) послуги виявилися дійсно корисним будівельним блоком для всіх видів системних концепцій вищого рівня. Живі шпалери, прослуховувачі сповіщень, заставки, методи введення, служби доступності та багато інших основних системних функцій створені як сервіси, які реалізують програми, і система прив’язується до того, коли вони повинні бути запущені. Сервіс реалізується як підклас Сервісу. Щоб отримати додаткові відомості про клас Service, перегляньте посібник розробника Services.

1.2 Методи написання UI/UX частини у ОС Android

UX це буквально означає «досвід користувача». У більш широкому сенсі це поняття про весь досвід, який отримує користувач при взаємодії із сайтом або програмою.

UX дизайн відповідає за функції, адаптивність продукту та те, які емоції він викликає у користувачів. Чим зрозуміліший інтерфейс, тим легше отримати результат і зробити цільову дію. UX дизайн - це проектування інтерфейсу на основі досліджень досвіду користувача і поведінки. Розробляти візуально привабливі проекти, але й створювати новий позитивний досвід користувачів та змінювати мислення бізнесу про інтерфейси. Перший Macintosh — один із яскравих прикладів роботи UX дизайнерів. Ідея використати вікна замість командного рядка існувала і до 1984 року, але саме проектувальники Apple зробили графічний інтерфейс масово доступним. Графічний інтерфейс Macintosh у 1984 році.

Деякі дизайнери вважають, що UX — це тільки про робота сайту або програми. Насправді досвід користувача цим не обмежується. Наприклад, якщо клієнт залишив заявку, але не отримав СМС з підтвердженням або дзвінок від менеджера, це симптоми поганого UX. UX не закінчується на красивій та зрозумілій формі на сайті. UX це шлях користувача від точки входу до точки виходу, з пункту А до пункту Б. UX це враження від роботи з інтерфейсом. [10]

Досвід користувача залежить від різних компонентів, а саме від архітектури сайту, графічного дизайну, зрозумілого тексту та чуйності інтерфейсу на конкретні дії користувача. Так як враження користувачів абстрактні, в UX дизайні потрібно вивчати їхні звички, розробляти прототипи поведінки та проводити тестування. Всею цією роботою займається UX дизайнер.

UX дизайнер це проектувальник, який вивчає потреби користувачів, будує логічні схеми роботи інтерфейсу, тестує прототипи на цільовій аудиторії та складає технічне завдання для UI дизайнера.

UX дизайнер це інженер-маркетолог, який досліджує досвід користувачів: вивчає аналітику, продумує зв'язки між елементами інтерфейсу та їх розташування, складає технічні завдання для редакторів. І на основі дослідження розробляє найефективніший прототип. [9]

UI перекладається як інтерфейс користувача. І необов'язково лише графічний тактильний, голосовий чи звуковий. розглянемо лише графічний інтерфейс, оскільки дизайнери переважно працюють із ним. Процес візуалізації прототипу, який розробили на основі користувальницького досвіду та дослідження цільової аудиторії. UI дизайн включає роботу над графічною частиною інтерфейсу анімацією, ілюстраціями, кнопками, меню, слайдерами, фотографіями і шрифтами. UI дизайнер визначає колірну палітру та розташування об'єктів в інтерфейсі чи зручно натискати чи правильно працює меню, чи легко заповнювати форму, чи добре читається текст зі смартфона, яке повідомлення видає сайт при тій чи іншій дії.

UI дизайнер інтерфейсів, який візуалізує робочий прототип, малює кнопки, іконки, форми та інші його компоненти і збирає їх в гармонійний макет. UI дизайнер відповідає за те, як виглядає інтерфейс продукту та як користувач взаємодіє з його елементами. Для цього необхідно грамотно організувати елементи інтерфейсу та витримати єдині стиль та логіку їхньої взаємодії. Концепція інтерфейсу для мобільних додатків Clean & Clear. Завдання допомогти користувачеві швидко і без стресу зрозуміти, як користуватися продуктом: сайтом, програмою, платіжним терміналом, мікрохвильовою піччю або пультом від телевізора. Для цього дизайнер стежить, щоб інтерфейс відповідав основним вимогам.

1.2.1 Метод з використанням XML технології

XML (/ˌeks em ˈel/ англ. eXtensible Markup Language) — мова розмітки, що розширюється. Рекомендований Консорціумом Всесвітньої мережі (W3C). Специфікація XML визначає XML-документи і частково визначає поведінку XML-процесорів (програм, що читають XML-документи і забезпечують доступ до їх вмісту). XML розроблявся як мова з простим формальним синтаксисом, зручний для створення та обробки документів як програмами, так і людиною з акцентом на використання в Інтернеті. Мова називається розширюваною, оскільки вона не фіксує розмітку, яка використовується в документах: розробник вільний створити розмітку відповідно до потреб до конкретної області, будучи обмеженою лише синтаксичними правилами мови. Розширення XML конкретна граматика, створена на базі XML і представлена словником тегів та їх атрибутів, а також набором правил, які визначають, які атрибути та елементи можуть входити до складу інших елементів. Поєднання простого формального синтаксису, зручності для людини, розширюваності, а також базування на кодуваннях Юнікод для представлення змісту документів призвело до широкого використання як власне XML, так і безлічі похідних спеціалізованих мов на базі XML у найрізноманітніших програмних засобах.

XML (Extensible Markup Language) використовується для опису даних.

Стандарт XML — це гнучкий спосіб створення інформаційних форматів та електронного обміну структурованими даними через загальнодоступний Інтернет, а також через корпоративні мережі. [4]

XML - це мова розмітки, заснована на стандартній узагальненій мові розмітки (SGML), яка використовується для визначення мов розмітки.

Основною функцією XML є створення форматів для даних, які використовуються для кодування інформації для документації, записів бази даних, транзакцій та багатьох інших типів даних. Дані XML можуть

використовуватися для створення різних типів вмісту, які генеруються шляхом створення різного типу вмісту, включаючи веб-, друкований та мобільний вміст, які базуються на даних XML. Як і мова гіпертекстової розмітки (HTML), яка також заснована на стандарті SGML, документи XML зберігаються як файли американського стандартного коду для обміну інформацією (ASCII) і їх можна редагувати за допомогою будь-якого текстового редактора. Основною функцією XML є надання «простого текстового формату для представлення структурованої інформації», згідно з Консорціумом World Wide Web Consortium (W3C), органом стандартів для Інтернету, в тому числі для наступного основні формати даних для програм, наприклад, у Microsoft Office. W3C визначає стандарт XML і рекомендує використовувати його для веб-контенту. Хоча XML і HTML обидва засновані на платформі SGML, W3C також визначив формати документів XHTML і XHTLM5, які відображають, відповідно, стандарти HTML і HTML5 для веб-вмісту. [7]

XML працює, надаючи передбачуваний формат даних. XML суворо стосується форматування; якщо форматування вимкнено, програми, які обробляють або відображають закодовані дані, повертатимуть помилку. Щоб XML документ вважався добре сформованим, тобто відповідним синтаксису XML і здатним бути прочитаним і зрозумілим аналізатором XML, він повинен бути дійсним кодом XML. Усі документи XML складаються з елементів; елемент діє як контейнер для даних. Початок і кінець елемента визначаються відкриваючими і закриваючими тегами з іншими елементами або простими даними всередині.

XML працює, надаючи належним чином відформатовані дані, які можуть надійно оброблятися програмами, розробленими для обробки введених даних XML. Наприклад, технічна документація може містити елемент , подібний до того, що показано в наступному фрагменті коду XML. Дані інтерпретуються та відображаються по-різному, залежно від формфактора технічної документації.

На веб-сторінці цей елемент може відображатися таким чином. Той самий код XML відображається по-різному в інтерфейсі користувача пристрою (UI) або в друкованому вигляді. Цей елемент можна інтерпретувати як відображення тексту, позначеного як наголос, по-різному, наприклад, його відображення червоним кольором і миготливими виділеннями. У друкованому вигляді вміст може бути наданий іншим шрифтом та іншим форматом. [8]

1.2.2 Метод з використанням Compose технології

Стан у програмі — це будь-яке значення, яке може змінюватися з часом. Це дуже широке визначення, яке охоплює все, від бази даних Room до змінної в класі. Усі програми Android відображають стан користувача. Кілька прикладів стану в програмах Android. Панель закусок, яка показує, коли не вдається встановити з'єднання з мережею. Допис у блозі та пов'язані з ним коментарі. Анімація пульсації на кнопках, які відтворюються, коли користувач натискає їх. Наклейки, які користувач може намалювати поверх зображення.

Jetpack Compose допоможе вам чітко пояснити, де і як зберігаєте та використовуєте стан у програмі Android. У цьому посібнику зосереджено на зв'язку між станом і компонованими, а також на API, які Jetpack Compose пропонує для легшої роботи зі станом. [8]

Compose є декларативним, і, таким чином, єдиний спосіб оновити його - це викликати той самий composable з новими аргументами. Ці аргументи є представленнями стану UI. Щоразу, коли стан оновлюється, відбувається перекомпозиція. В результаті такі речі, як TextField, не оновлюються автоматично, як у імперативних представленнях на основі XML. Компоненту має бути чітко вказано новий стан, щоб він відповідним чином оновився.

Компонований, який використовує запам'ятовування для зберігання об'єкта, створює внутрішній стан, роблячи компонуване функцією стану. HelloContent є прикладом компонування з визначенням стану, оскільки він

зберігає та змінює свій стан імені всередині. Це може бути корисно в ситуаціях, коли абоненту не потрібно контролювати стан і він може використовувати його, не керуючи станом самостійно. Однак компоновці з внутрішнім станом, як правило, менш придатні для повторного використання і їх важче перевірити.

Компонований без стану - це компонований, який не має жодного стану. Простий спосіб отримати статус без громадянства — це використання державного підйому. [1]

Коли розробляється багаторазові композиційні файли, вам часто потрібно розкрити як версії того самого компону, що зберігається, так і версії без стану. Версія з визначенням стану зручна для абонентів, які не піклуються про стан, а версія без стану необхідна для абонентів, яким потрібно контролювати або піднімати стан.

Підвищення стану в Compose — це шаблон переміщення стану до виклику компонованого, щоб зробити компонооване без стану. Загальний шаблон для підйому стану в Jetpack Compose полягає в заміні змінної стану двома параметрами подія, яка вимагає зміни значення, де T — запропоноване нове значення. Однак не обмежуєтесь параметрами onValueChange. Якщо більш конкретні події підходять для компонованого, повинні визначити їх за допомогою лямбда, як це робить ExpandingCard з onExpand і onCollapse.

Стан, який піднімається таким чином, має деякі важливі властивості. Єдине джерело істини: змінюючи стан замість того, щоб його дублювати, гарантуємо, що існує лише одне джерело істини. Це допомагає уникнути помилок. [11]

Інкапсульовані тільки компоновані файли, які зберігають стан, зможуть змінювати свій стан. Це абсолютно внутрішньо. Можливість спільного використання: піднятий стан можна спільно використовувати з кількома компонованими. Скажімо, хотіли назвати інше компонооване, підйом дозволить нам це зробити.

Перехоплювані абоненти, які викликають композиційні файли без стану, можуть вирішити ігнорувати або змінювати події перед зміною стану.
Відокремлено: стан ExpandingCard без стану може зберігатися де завгодно. Наприклад, тепер можна перемістити ім’я в ViewModel. Витягуєте ім’я та значення onValueChange з HelloContent і переміщуєте їх вгору по дереву до компонованого HelloScreen, який викликає HelloContent.

1.3 Обгрунтування вибору макету програмування

Щороку з’являються нові технології, які полегшують життя розробника. Декларативний інтерфейс користувача став одним із найвизначніших трендів останніх років, оскільки значно підвищує продуктивність розробників зменшує вартість розробки полегшує націлювання на кілька платформ і пристроїв за допомогою одного коду. Однак Google знадобився деякий час, щоб створити власну декларативну структуру інтерфейсу користувача, перш ніж вони нарешті випустили Jetpack Compose для рідних програм Android. Jetpack Compose відрізняється від традиційної розробки інтерфейсу користувача Починаючи з Java Swing і Win32, переважна більшість інтерфейсу користувача була написана в імперативному стилі. Це також стосується розробки інтерфейсу користувача для Android і iOS. Розробники створювали повнофункціональний інтерфейс користувача, вручну описуючи, як елементи реагують на зміни, і оновлюючи їх пізніше за допомогою сетерів, коли змінюється стан. [12]

Однак React, Flutter, SwiftUI та Jetpack Compose застосували інший підхід. За допомогою цих фреймворків вказуєте, що має представляти інтерфейс користувача, а не як мають бути створені елементи. Частина how залишається для фреймворку, а весь підхід називається декларативним інтерфейсом користувача.

Декларативний інтерфейс користувача — це промислова тенденція в Інтернеті та мобільних пристроях. Імперативний інтерфейс був на місці протягом тривалого часу. Він звичайний, він надійний, але він різко не працює для великих

і складних реактивних додатків. Декларативний інтерфейс користувача продемонстрував чудове прийняття спільнотою та продуктивність розробки за допомогою Flutter і React. З огляду на такий успіх, інші фреймворки невдовзі почали використовувати цей підхід. Насправді, Jetpack Compose потрапив у центр уваги досить пізно, на відміну від Apple, яка випустила SwiftUI ще у вересні 2019 року. Jetpack Compose має переваги перед традиційним інтерфейсом користувача. Перегляди Android хороші, але не ідеальні. Існує багато незворотних технічних проблем, які накопичувалися роками, і Jetpack Compose може нарешті вирішити їх. [5]

Jetpack Compose повний редизайн інтерфейсу Android, створений з нуля, щоб підвищити якість і швидкість розробки. Він вирішує численні проблеми, оскільки UI є окремою змінною сутністю. Jetpack Compose має повну сумісність з XML і навпаки. Потрібно менше коду. Він підходить для однонаправленого потоку даних, оскільки композитні елементи приймають стан і видають події.Роз'єднаний набір інструментів, тому він не залежить від випусків платформи, як це було з бібліотекою підтримки перегляду.

Продуктивність Jetpack compose чудова. Він використовує інтелектуальні перекомпозиції, забезпечує єдиний прохід для макета та позбавляється від проблем XML, таких як інфляція перегляду. [5]

Крива навчання. Підхід відрізняється від того, до якого звикли розробники, і для ознайомлення з технологією потрібен час. Обмежене усиновлення громадою. Незважаючи на те, що Jetpack Compose добре сприйнятий і стає дійсно модним, більшість команд по праву віддасть перевагу традиційному інтерфейсу користувача з XML. Так само, як Kotlin не відразу завоював популярність, Jetpack Compose знадобиться кілька років, щоб добре засвоїтися. Відсутність документації. Значна частина Android Views полягає в тому, що більшість проблем, імовірно, уже вирішено кимось або існує багато ресурсів, які дозволяють вирішити їх самостійно. Однак знайти все необхідне за допомогою

Jetpack Compose складно, оскільки він ще досить свіжий. Тим не менш, після релізу з’явиться набагато більше документів. Менше інструментів підтримує його, оскільки більшості з них знадобиться час, щоб прийняти цю технологію. [2]

Декларативний інтерфейс користувача — це мегатренд. І Web, і Mobile активно використовують цей підхід. Google чітко дав зрозуміти, що Jetpack Compose є одним із головних пріоритетів. Потрібні роки, щоб технологія замінила традиційний XML. Багато розробників залишаться неохоче перемикатися, оскільки підхід зовсім інший і поки що обмежене застосування.

2 РОЗРОБКА ТЕХНОЛОГІЇ COMPOSE У ПОЄДНАННІ З МЕДІА ПЛЕЄРОМ ДЛЯ ЛЕКЦІЙ ТА НАВЧАЛЬНИХ МАТЕРІАЛІВ

2.1 Реалізація технології Separation of concerns

Поділ відповідальності (Separation of concerns) — це добре відомий принцип розробки програмного забезпечення. Це одна з фундаментальних речей, яку як розробники додатків дізнаємося. Незважаючи на те, що цей принцип добре відомий, часто важко зрозуміти, чи дотримується цей принцип на практиці. Може бути корисно думати про цей принцип як термін типу «зчеплення» або «пов'язаність» дивитись рисунок 2.1.

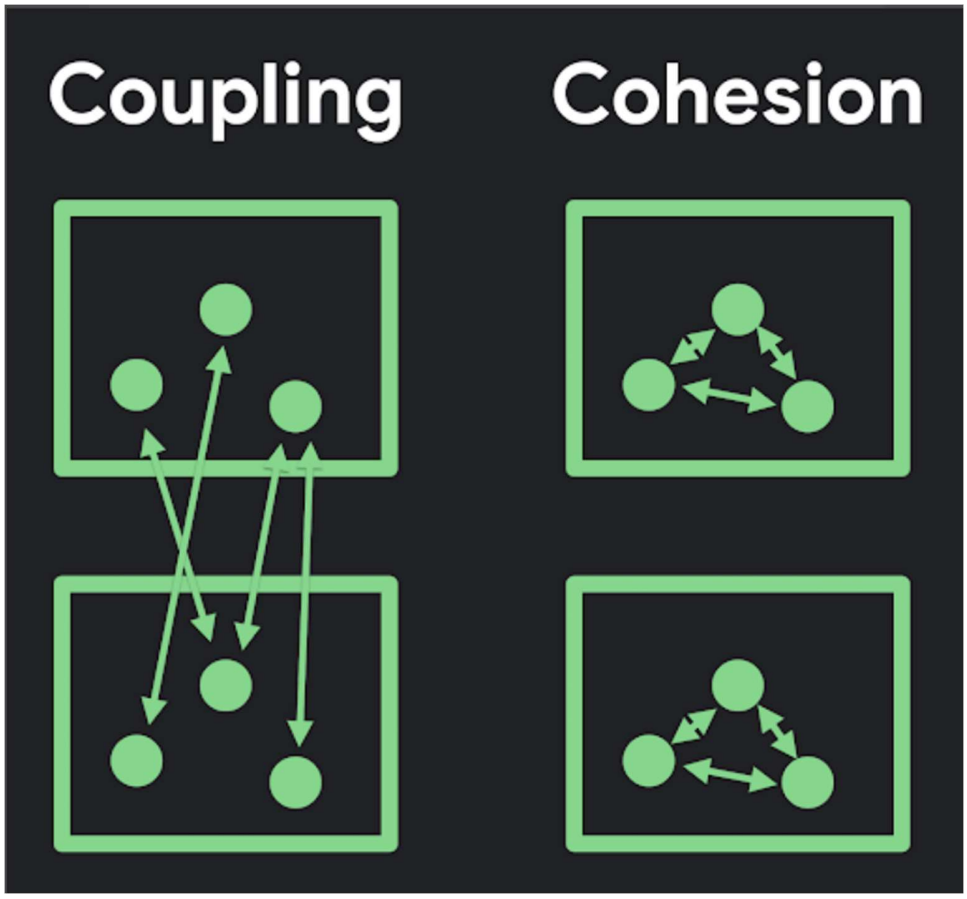

Рисунок 2.1 — Архітекрура поділу відповідальності

При написвнні коду створюємо модулі, які складаються з кількох сутностей (унітів). Пов'язаність (Coupling) — це залежність між сутностями в різних модулях, яка відображає те, що частини одного модуля впливають на частини інших модулів. Цілісність (Cohesion) — це, навпаки, взаємозв'язок між сутностями (юнітами) у модулі і показує, наскільки добре згруповані юніти у модулі. При написанні програмного забезпечення, що підтримується, наша мета — мінімізувати пов'язаність і максимізувати цілісність. [6]

Коли у нас є дуже пов'язані модулі, внесення змін до коду в одному місці означає необхідність внесення множини змін до інших модулів. Що ще гірше, зв'язок часто може бути неявним, тому речі ламаються в несподіваних місцях через зміну, яка здається зовсім не пов'язаною.

Поділ відповідальності полягає в тому, щоб згрупувати якомога більше пов'язаного коду, щоб наш код можна було легко підтримувати та масштабувати у міру зростання програми.

2.2 ViewModel и XML-лейаут

ViewModel надає дані лейаут. Виявляється, тут можна заховати багато залежностей: великий взаємозв'язок між ViewModel і лейаутом. Один з найчастіших і добре знайомих нам випадків сильного взаємозв'язку - це використання API (від Android або сторонніх бібліотек - прим. перекладача), в яких потрібне знання про нутрощі самого XML-макета, наприклад метод findViewByID дедальніше на рисунку 2.2. [9]

Використання таких API вимагає знання того, як влаштований XML-макет і створює взаємозв'язок між ними. Оскільки наша програма з часом зростає, повинні стежити за тим, щоб жодна з цих залежностей не застаріла.

Більшість сучасних додатків відображають інтерфейс користувача динамічно і змінюються в процесі виконання.

В результаті необхідно не тільки перевірити, що ці залежності (тобто View-елементи) надаються XML-макетом, а також те, що вони будуть надаватися під час роботи програми. Якщо елемент залишає ієрархію перегляду під час виконання, деякі з цих залежностей можуть бути порушені і можуть призвести до таких проблем, як NulReferenceExceptions.

Рисунок 2.2 — Взаємозв'язок між ViewModel та лейаутом

Зазвичай ViewModel визначається мовою програмування Kotlin, а макет у XML. Через цю різницю в мові існує примусове поділ, хоча ViewModel і XML-макет іноді можуть бути тісно пов'язані. Інакше кажучи, вони дуже тісно пов'язані зображено на рисунку 2.3, а також особливість цього методі можна переглянути на рисунку 2.2.

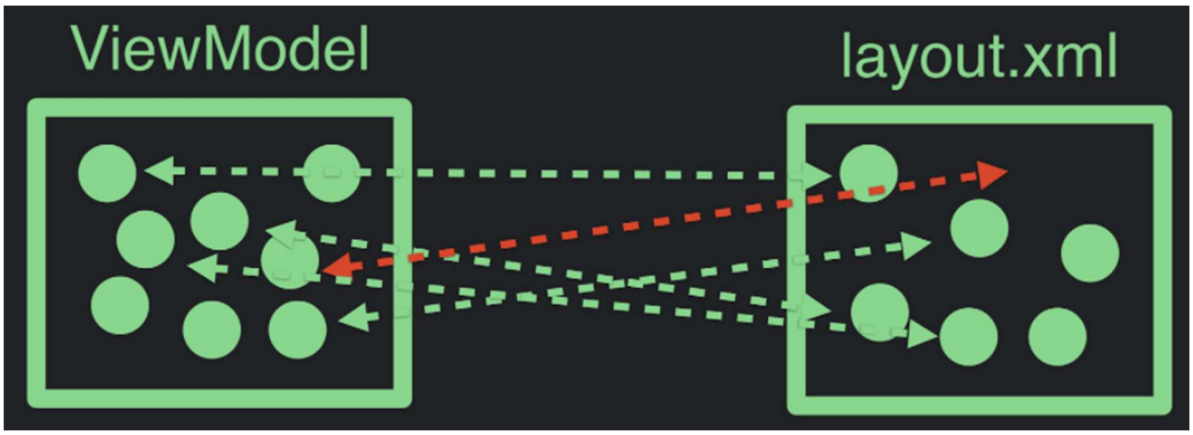

Рисунок 2.3 — Проблеми взаємозв'язка між ViewModel та лейаутом

Оскільки в цьому випадку будемо працювати однією мовою, деякі із залежностей, які раніше були неявними, можуть стати більш очевидними.

Отже можемо провести рефакторинг коду та перемістити речі туди, де вони зменшать взаємозв'язок та збільшать узгодженість. [6]

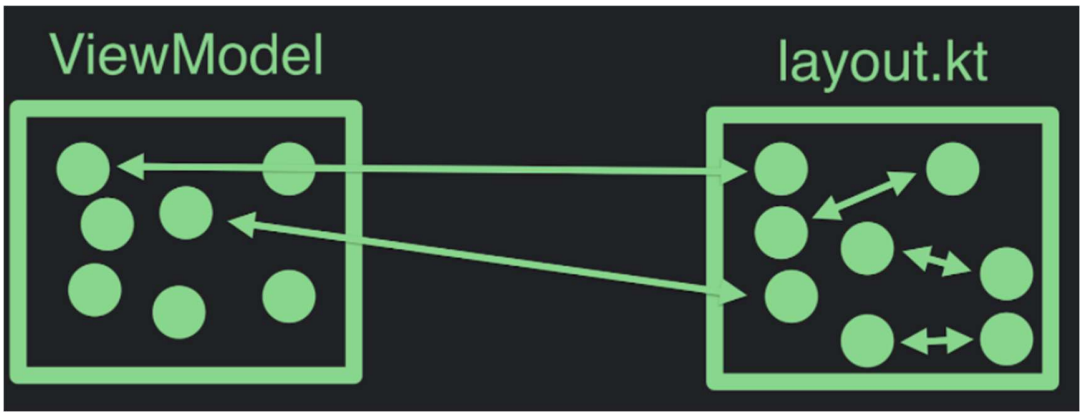

Рисунок 2.4 — Спрощення розділення між ViewModel та лейаутом

Тепер можна подумати, що змішуєте логіку з інтерфейсом користувача. Реальність така, що у вас буде логіка, пов'язана з інтерфейсом користувача, у вашому додатку, незалежно від того, як воно структуроване. Сама структура неспроможна цього змінити що показано на рисунку 2.4.

Але те, що може зробити фреймворк, це надати вам інструменти, що спрощують поділ цим інструментом, є функція Composable. Функції — це те, що ви, ймовірно, використовували протягом тривалого часу, щоб розділити завдання в інших місцях вашого коду. Навички, які придбали для такого роду рефакторингу та написання надійного, підтримуваного, чистого коду — ті ж навички застосовні і до Composable -функцій. [7]

2.3 Пристрій Composable-функції

2.3.1 Composable методи

Тут функція отримує дані як параметри класу appData. В ідеалі це незмінні дані, які функція Composable не змінює: Composable-функція повинна бути

функцією перетворення для цих даних. Отже, можемо використовувати будь-який код Kotlin, щоб взяти ці дані і використовувати їх для опису нашої ієрархії, наприклад викликавши функції Header() і Body().

Це означає, що викликаємо інші Composable функції, і ці виклики відображають структуру нашого UI. можемо використовувати всі примітиви, що надаються Kotlin-ом. можемо включити оператори if і цикли for для управління структурою UI, щоб впоратися з більш складною логікою інтерфейсу користувача.

Composable — функції часто використовують кінцевий лямбда-синтаксис Kotlin, тому Body() — це Composable-функція, яка приймає composable-лямбду як параметр. Це має на увазі ієрархію або структуру, тому Body() обгортає тут набір елементів. У розробці програмного забезпечення композиція — це те, як кілька частин простішого коду можуть об'єднуватися в складніший блок коду. В объектно-ориентированной моделі програмування однією з найпоширеніших форм композиції є успадкування з урахуванням класів. У світі Jetpack Compose, оскільки працюємо лише з функціями, а не з класами, метод композиції дуже відрізняється, але має багато переваг перед успадкуванням. Давайте подивимося на приклад. [2]

Припустимо, у нас є перегляд і хочемо створити поле введення. У разі спадкування наш код може виглядати так

View — це базовий клас. ValidatedInput є підкласом Input. Для перевірки дати DateInput успадковується ValidatedInput. Але тоді виникає проблема: хочемо створити компонент із введенням діапазону дат, отже, нам потрібно здійснювати перевірку за двома датами — датою початку та датою закінчення. можете створити підклас DateInput, але вам потрібно це зробити двічі, а не можете цього зробити. Це обмеження спадкування: у нас має бути єдиний батько, від якого успадковуємо.

У Compose це не так складно. Допустимо, починаємо з базового composable-компонента Input

Коли створюємо наш ValidatedInput, просто викликаємо Input у тілі нашої функції. Потім можемо доповнити його чимось для перевірки. Потім для DataInput можемо викликати ValidatedInput.

Тепер, коли стикаємося з введенням діапазону дат, більше немає проблеми, це лише два виклики замість одного. [3]

При створенні UI-компонентів за допомогою Compose, у них немає єдиного батька, і це вирішує проблему, яка виникла у випадку з використанням успадкування. Інший тип проблеми композиції — це абстрагування від типу декотратора.

FancyBox — це View, яке прикрашає інші View, у разі Story і EditForm. хочемо створити FancyStory та FancyEditForm, але як? успадковуємо від FancyBox чи успадковуємо від Story? Це неясно, тому що, знову ж таки, у нас може бути лише один батько в ланцюжку наслідування. Compose справляється з цим дуже добре. [3] [4]

У Composamble-функції є лямбда, у якій описуємо дочірні View, тобто визначаємо View, яка обертає інші View. Отже, тепер, коли хочемо створити FancyStory, викликаємо Story всередині FancyBox і можемо зробити те саме з FancyEditForm. Це спосіб композиції Compose.

2.3.2 Перемальовування компонентів

Це спосіб сказати, що будь-яку функцію Composable можна повторно викликати в будь-який час. Якщо у вас дуже велика ієрархія Composable, коли частина вашої ієрархії змінюється, вам не потрібно перераховувати всю ієрархію. Оскільки Composable-функції можна викликати повторно, можете використовувати цю особливість для деяких корисних речей. Наприклад, ось

функція bind, яку можете зустріти сьогодні у розробці для Android, жображено на рисунку 2.5.

```
@Composable
fun Messages(liveMsgs: LiveData<MessageData>) {
  val msgs by liveMsgs.observeAsState()
  for (msg in msgs) {
    Message(msg)
  }
}
```

Рисунок 2.5 — Composable-компонент Messages

LiveData, на яку хочемо підписати оновлення View. Для цього викликаємо метод observe у класі, що має життєвий цикл (LifecycleOwner — Activity або Fragment), а потім передаємо лямбду. Лямбда викликається щоразу при оновленні LiveData і коли це відбувається, хочемо оновлювати і View.За допомогою Compose можемо змінити цей спосіб взаємодії з LiveData

Це аналогічний Composable-компонент Messages , який отримує LiveData і викликає compose-метод observationAsState. Метод observeAsState перетворює LiveData<T> на State<T>. Це означає, що можна використовувати отримане значення в тілі функції. Екземпляр State підписаний на екземпляр LiveData, що означає, що оновлюватиметься при кожному оновленні LiveData. Це також означає, що будь-де читався екземпляр State, навколишня compose-функція, в якій читається, буде автоматично підписуватися на ці зміни. Кінцевим результатом є те, що більше не потрібно вказувати LifecycleOwner або коллбек для оновлення, оскільки Composable може неявно виконувати функцію їх обох. [4]

Ще одна річ, яку добре виконує Compose — це інкапсуляція. Це те, що повинні думати, коли робите загальнодоступні API-інтерфейси Composable-функцій: публічний API-інтерфейс Composable-функцій - це набір параметрів, які

вона отримує, тому не може їх контролювати. З іншого боку, Composable-компонент може керувати станом і створювати його, а потім передавати цей стан разом з будь-якими даними, які отримав, до інших Composable-компонентів як параметри.

Тепер, оскільки керує цим станом, якщо хочете змінити стан, можете дозволити дочірнім компонентам передавати сигнал про цю зміну за допомогою колбека

2.4 Створення моделі написання додатків на базі Compose для навчальних матеріалів

Запропоновано нову модель роботи з данними для навчальних матеріалів та відображення користувацького інтерфкейсу.

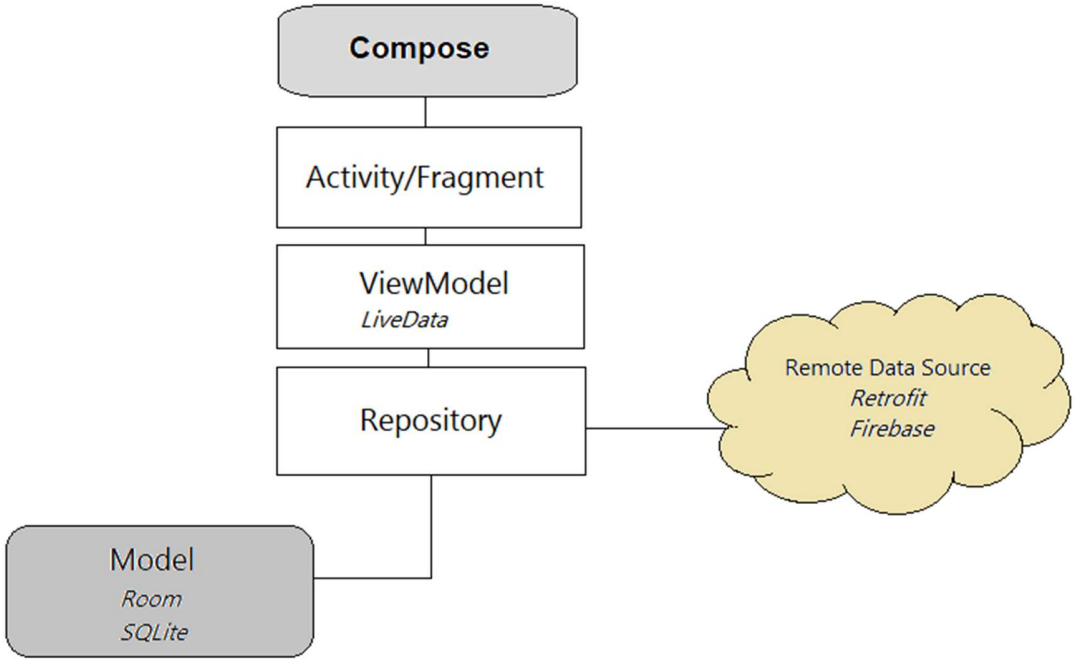

Рисунок 2.6 — Compose додаток з комбінованим зберіганням данних

У даній роботі реалізується робота з Compose та хмарно стрімінговий медіаплеєр. Новизна даної роботи полягає у тому що дані зберігаються у хмарній

базі данних Firebase та у локальній базі даних Room.Також особливістю є те що данні одразу відображаються за допомогою нової технології компом без затримки та передачі даних у лейаут за допомогою рисунку 2.6 відображено створення нової моделі. [5]

Дана розробка є першою на ринку та не має аналогів. Цей макет роботи з даними для максимально швидкої роботи з навчальними матеріалами також та розробки, пропонує максимальну цілісність даних так як використовується хмарна база даних firebase також локальна база даних Room, яка допомагає зберігати дані локально та редагувати дані локально іншої сторони firebase допомагає зберігати дані хмарно також допомагає редагувати дані хмарно.

Через те що не потрібно зберігати величезні макети layauts, програма буде займати набагато меншу кількість пам'яті, а також меншу кількість оперативної пам'яті. Так як за допомогою даної технології UI та UX частина пишеться програмно, та відображається як і у звичайному додатку. Також дана технологія дуже добре показують себе зі сторони швидкості роботи програми.

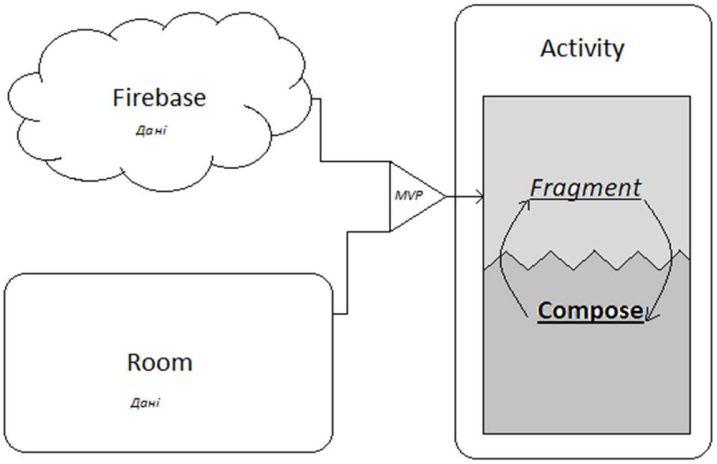

Рисунок 2.7 — Передача даних за допомогою розробленої моделі

Унікальність даної роботи полягає у тому що до хмарного локальних базах даних добавляється нова технологія Compose, що допомагає відображати дані максимально швидко , а також максимально заощаджувати кількість витраченої пам'яті на збереження великої кількості навчальних даних. Також підвищується швидкість даного додатка так як не використовується система xml леяут, що зображено на рисунку 2.7.

Після того як студент, або викладач, вносить якісь дані, Вони спочатку зберігається у локальній базі даних, далі дані проходять через архітектуру патерна MVVM або MVP. Після чого розділяється на дві гілки, перша гілка відправляє дані на віддалену базу даних firebase, інша гілка передає дані у viewmodel, там дані проходять спеціальну обробку, якщо вона потрібна, і передаються на фрагмент або activity.

За допомогою технології відображається на екрані. Особливість даного проекту у тому, що при отриманні даних програма не буде витрачати зайвого часу на передавання їх у макет лейаут, а буде зберігати одразу у фрагменті, так як технологія compose розміщується одразу під фрагментом. Після закінчення самого класу фрагменту, розробляється його UI та UX частина, за допомогою compose нижче цього ж файлу. [2]

Розроблений макет ідеально підходить для навчальних матеріалів та лекцій, так як у цьому випадку потрібно працювати з великою кількістю даних, а також відображати велику кількість різних елементів, що дуже затратно по часу та пам'яті.

Дана комбінація технічних елементів, а також вже відомих технологій допомагає набагато краще працювати з подібними додатками та нейтралізувати дані проблеми. [8]

Отже, телефону або іншому девайсу, не потрібно завантажувати графічні елементи, а потрібно тільки реалізувати декілька строчок коду, що набагато швидше ніж графічні макети.

У данному розділі було розглянуто типи макету програмного забезпечення мобільних додатків. Також було проаналізовани технології створення UI та UX. Для данного мобільного додатка є набільш доцільним використання макету Compose. Це надає сучасний підхід до створення UI, дозволяючи ефективно розділяти відповідальність у коді. Оскільки compose-функції дуже схожі на звичайні функції Kotlin, використовувати ті ж інструменти для рефакторингу, що і для звичайного Kotlin-коду.

3 РОЗРОБКА COMPOSE ІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Розробка UI та UX додатка

У даній роботі було розроблено новий унікальний UI та UX додатка на основі імпорт compost у Android студії. Розроблено кілька основних напрямки у додатку. Перший це реалізація реєстрації у додатку. Наступний це чат, а також групи і канали у додатку . Останній знаходиться у стадії розробки це медіаплеєр з багатьма відео. [4]

Фреймворк працює на основі Composable функцій. Ці функції дозволяють програмно визначати користувальницький інтерфейс ваших додатків, описуючи його форму та залежність, а не сосредотачиваться в процесі створення користувальницького інтерфейсу. Щоб створити «составну» функцію, просто додайте анотацію @Composable до функції імені.

Для початку слідуйте інструкціям по налаштуванню Jetpack Compose і створіть додаток, використовуючи шаблон Empty Compose Activity. Шаблон за умовчанням уже містить деякі елементи Compose, але створюємо його крок за кроком.

Спочатку видалити функції Greeting і Preview за замовчуванням і видалити блок setContent з MainActivity, залишив дію пустим. Скомпілюйте та запустіть пусте додаток.

Блок setContent визначає макет активіті. Замість визначення вмісту макету за допомогою XML-файлу, ми викликаємо складні функції. Jetpack Compose використовує кастомний плагін компілятора Kotlin для перетворення цих складових функцій в елементи інтерфейсу програми. [1]

Наприклад, функція Text() визначається бібліотекою Compose UI - ви викликаєте цю функцію, щоб оголосити текстовий елемент у своїй програмі , деталі на рисунку 3.1

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Text("Hello world!")
        }
    }
}
```

Рисунок 3.1 — Приклад написання функції з використанням setContent

Поточна canary збірка Android Studio дозволяє вам попередньо переглядати ваші складові функції прямо в IDE, замість того, щоб завантажувати програму на пристрій Android або емулятор. Основне обмеження у тому, що складова функція має приймати жодних параметрів. З цієї причини ви не можете переглянути функцію Greeting() безпосередньо. Натомість створіть другу функцію з ім'ям PreviewGreeting(), яка викликає Greeting() з відповідним параметром. Реалізацію зображено на рисунку 3.2.

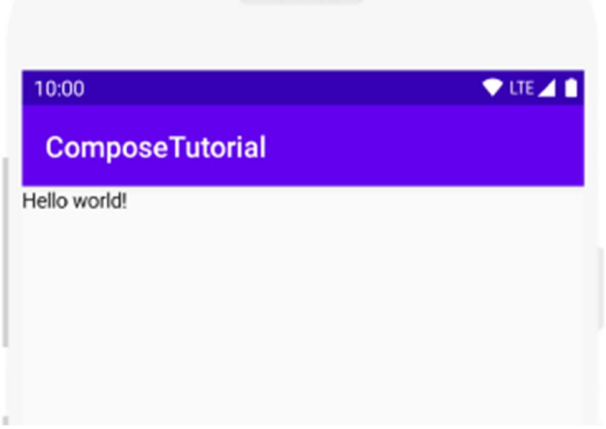

Рисунок 3.2 — Відображення функції з використанням setContent

3.2 Розробка ідеї постановки додатка

Перед розробкою даного додатку було розроблено навігаційний граф та план даного додатку без використання технологій, зображено на рисунку 3.3.

Також заздалегідь було передбачено увесь функціонал та усі технології які будуть використовуватись у даному додатку. Було передбачено використання таких технологій як екзо плеєр, для розробки мультиплеєра, також ж було обрано хмарно базу даних Firebird та локальну базу даних Room. [11]

Перед створенням даного проекту було проаналізовано всі методи написання сучасних макетів та вирішено обрати найбільш актуальний та найбільш сучасний макет побудови даних, а саме технологію compose. При виборі спеціально архітектури додатку було розглянуто кілька патернів архітектури а саме MVVM, MVP тощо.

Для того щоб у даному додатку показати на прикладі, як відображаються дані, та як працює вся система було обрано спеціально локальну систему, для збереження тимчасової інформації, а саме Json файл.

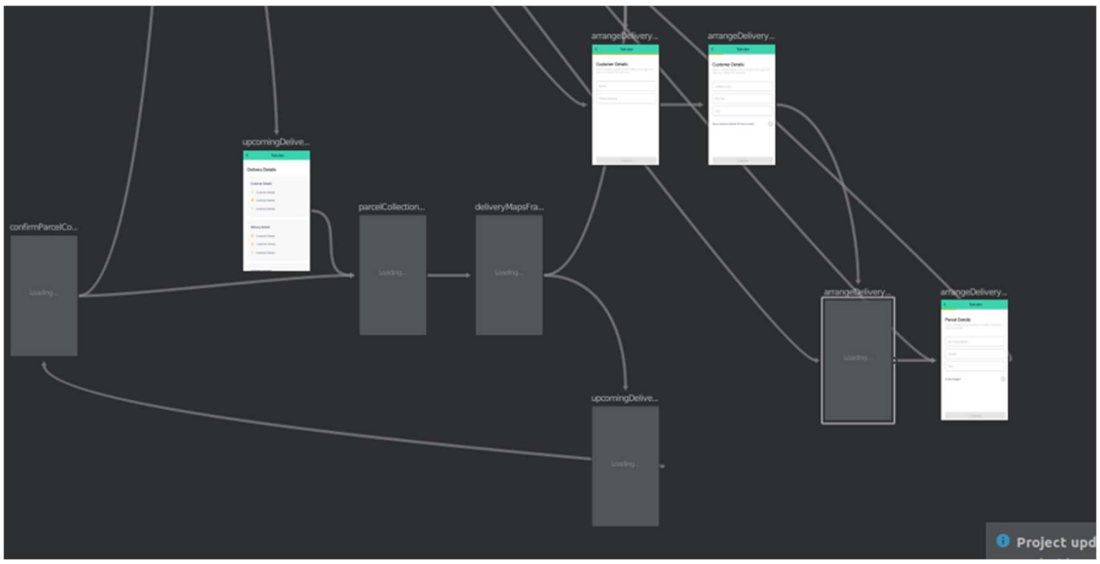

Рисунок 3.3 — Постановка навігаційного графу та елементів додатку

Щоб передавати дані конкретно між фрагментами та між усіма додатком було обрано спеціально дата змінну яка дозволяє у реальному часі обновляти та підтягувати дані, а саме це Livedata. Також було розглянуто можливість нотифікації для подальшого користування. [10]

3.2.1 Розробка системи чату та каналів

При розробці данного розділу було розглянуто кілька різних макетів та дизайн компонентів. Обрано стандартний дизайн паттерн для данного чату . Особливостями якого є додаткові можливості для перегляду лекцій.

Данний чат містить у собі :

— назву каналу ;

— кількість учасників;

— можливість пошуку по чату;

— повідомлення;

— міні-профіль користувача з іменем та прізвищем;

— час повідомлення;

— кнопку переходу у відеобібліотеку;

— поле для написання повідомлення;

— логотип;

— панель швидкого доступу при написанні повідомлення;

— додаткова динамічна кнопка;

— кнопка відправлення повідомлення з перевіркою на пусте поле.

Для зручності користування данним чатом була застосована стандартна технологія переходу до актуальних данних, дивитись рисунок 3.4.

Наразі для створення повідомлення доступні кілька функцій перше з них це використання емоції а також стікерів наступна це відмінювання конкретних осіб за допомогою значка собачка. Також доступний пошук по символам який в онлайн-режимі знаходить та підкреслює ключові слова з пошуку, яку можна побачити на рисунку 3.4.

При натисканні на відміну особу за допомогою ключового символа собачка буде відбуватися перехід на профіль даної особи що допоможе детальніше так краще працювати з повідомленнями.

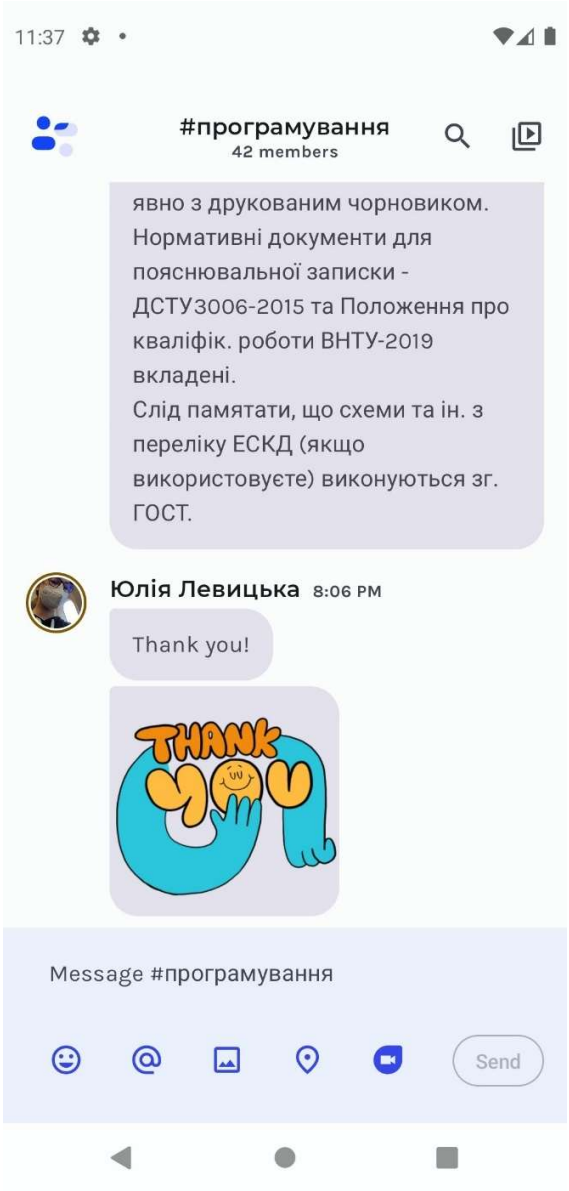

Рисунок 3.4 — UI реалізація чату

Додана динамічна кнопка яка допомагає корректно слідкувати за навігацією, детальніше зображено на рисунку 3.5.

Як буде ведення повідомлення у полі message можна побачити назву чата що допомагає святкувати за за конкретним введенням тексту у даний чат, деталі зображені на рисунку 3.5.

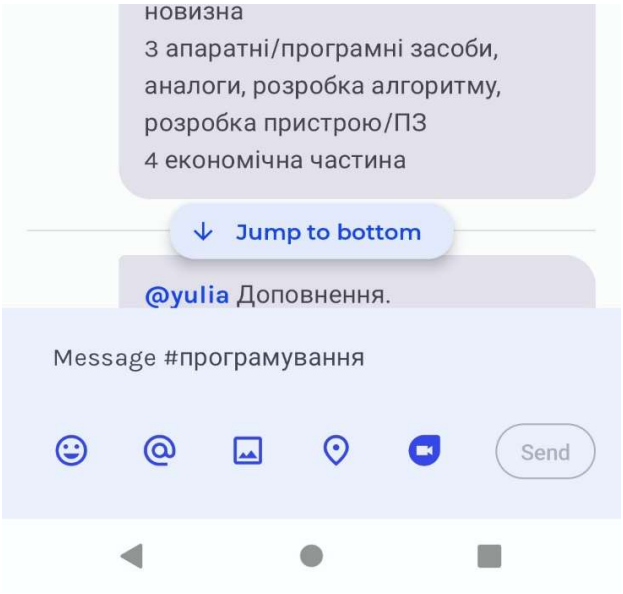

Рисунок 3.5 — Реалізація кнопки навігації чату

Також кнопка відправлення повідомлення не буде активною до тих пір поки не почнеться введення самого повідомлення, що допомагає попередити відправлення пустих повідомлень. [9]

Наразі для створення повідомлення доступні кілька функцій перше з них це використання емоції а також стікерів наступна це відмінювання конкретних осіб за допомогою значка собачка. Також доступний пошук по символам який в онлайн-режимі знаходить та підкреслює ключові слова з пошуку, яку можна побачити на рисунку 3.5.

При натисканні на відміну особу за допомогою ключового символа собачка буде відбуватися перехід на профіль даної особи що допоможе детальніше так краще працювати з повідомленнями.

Наступне що було реалізовано у даному чаті це можливість відмічати власні повідомлення також підсвічувати час час та дата відправлення також було Додано можливість слідкування за чатом. [12]

До того як буде ведення повідомлення у полі message можна побачити назву чата що допомагає святкувати за за конкретним введенням тексту у даний чат, деталі зображені на рисунку 3.6

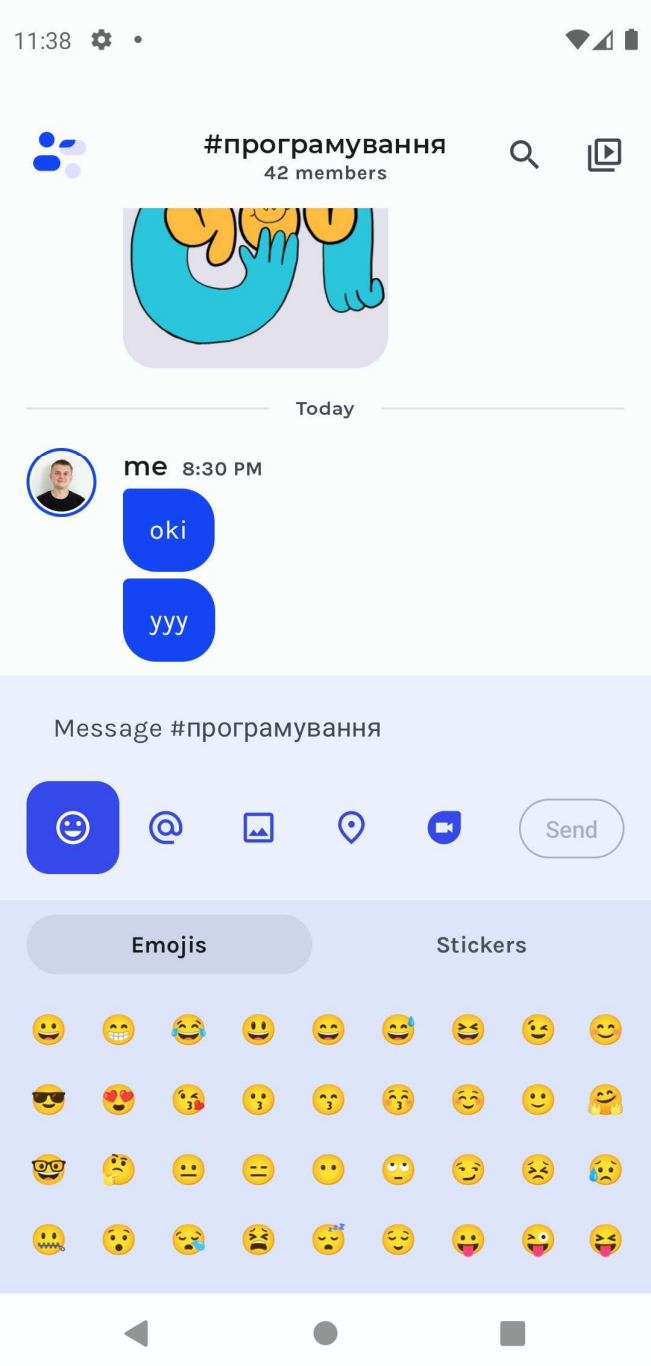

Рисунок 3.6 — Введення та відправлення повідомлення у чаті

Також у даному чаті буде реалізовано кілька додаткових можливостей такі як передача фото та відео передача локальних даних а також онлайн записування відео та фото, зображено на рисунку 3.7.

Якщо користувач спробуйте використати функціонал який ще не добавлений він отримає спеціальне повідомлення. [12]

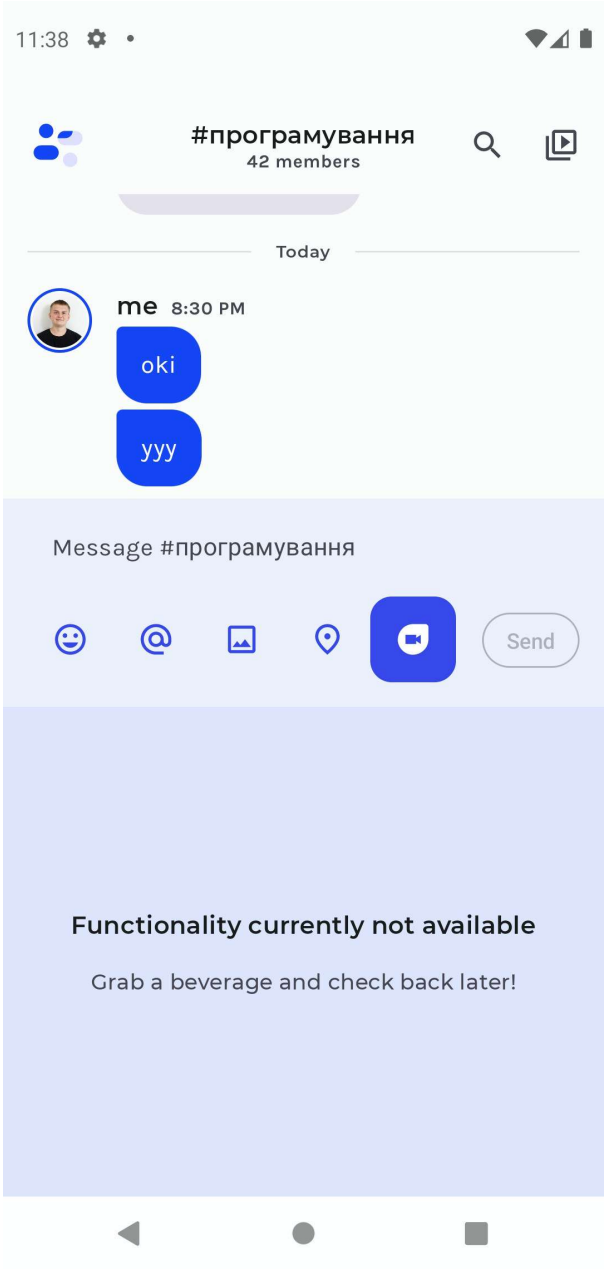

Рисунок 3.7 — Використання недоступного функціоналу

При натисканні на логотип даного додатку користуватись одразу перейде на на реалізацію бокового навігаційного фрагменту що допоможе краще виконувати навігацію по чатах та отримати доступ до всіх учасників даної групи.

При повторному натисканні користувач зможе звернути навігацію бокового фрагменту що повернення його до чату та дозволить продовжувати обмінюватися повідомленнями. [8]

3.2.2 Розробка бокового навігаційного фрагменту

У даному розділі було розроблено панель бокової навігації фрагменту що дозволяє коректно переміщатись між чатами

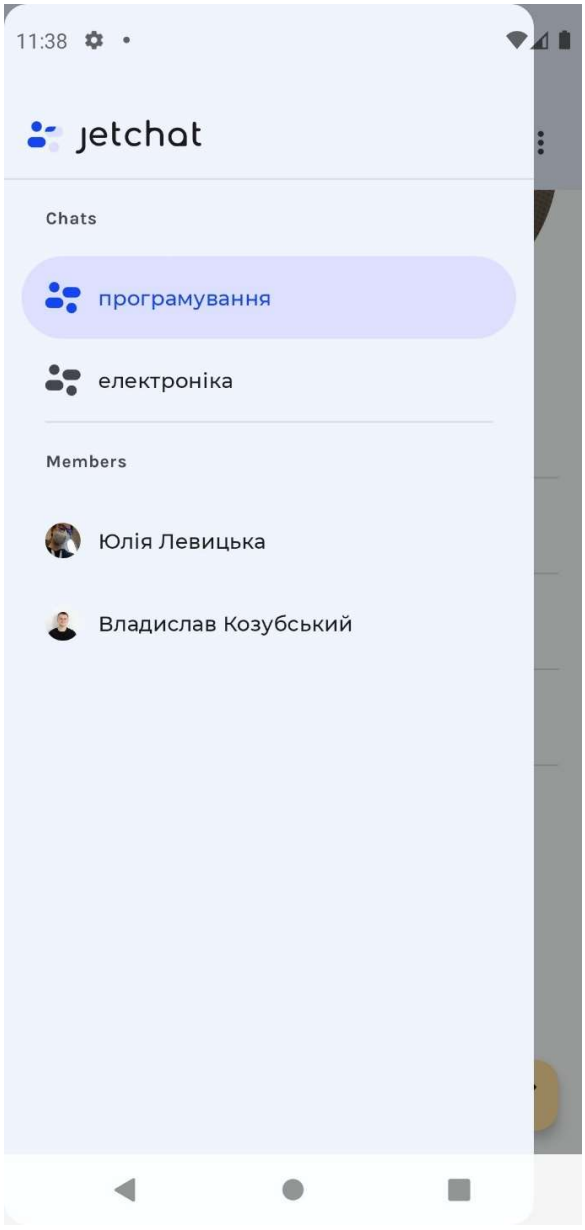

Рисунок 3.8 — Панель навігації фрагменту

. Для розробки даної панелі навігації було використано стандартний старі метод, тільки використовуючи нову технологію compose. Що дозволило прискорити відображення й швидкість роботи програми та самої навігації, данна панель навігаційного фрагмкнту зображена на рисунку 3.8.

У дані навігації було вирішено відобразити логотип та назву даної додатку також обрані чати та учасників у конкретних чатах. При натисненні на конкретного учасника можна перейти до профілю та дізнатися більше інформації. За допомогою навігейшн графа можна здійснювати перехід з одного фрагмента на інший. Бокова панель допомагає користувачу зорієнтуватися у даному додатку та обрати те що потрібно. Дана панель просто відкривається та закривається що допомагає одразу орієнтуватися у додатку. [7]

Коли користувач вибирає конкретний чат змінюється колір логотипу та назви даного чату, це допомагає зорієнтуватися користувачу у якому чаті конкретно він зараз знаходиться. При закриті та відкриті даної панелі навігації ця позначка не змінюється поки користувач не перейде до іншого чату.

У різних чатах різна кількість учасників та різні самі учасники при зміні чату змінюється і кількість учасників та викладачів змінюється склад даної групи. Викладачі також відображуються у складі групи але з позначкою викладач щоб студенти могли зорієнтуватися та направити до правильного чату.

В даній панелі навігації може відображатися більше двадцяти різних чатів та більше сімдесяти учасників, що допомагає зробити дану програму більш обширною та корисною. На рисунку 3.9 зображено два різних чата та два різних учасники.

Технологія навігації яка використовується у даному додатку перейшла ще від леяцтів та xml. Також прекрасно реалізується при технології compose та не втрачає своїх властивостей, використовує свої плюси та мінуси. Отже, при перенесенні цієї технології було збережено саму навігацією. При технології

compose це є взагалі єдина технологія навігації яка досі використовується з новими технологіями.

3.2.3 Розробка макету медіа плеєра

Стаття охоплює як ручне, так і автоматичне відтворення відео в ефективний спосіб, збереження/відновлення позиції останнього відтвореного відео, призупинення відтворення, якщо відеокарта не видима для користувача, і обробки життєвого циклу програми. ExoPlayer буде використовуватися для відтворення відео, а ці відео як тестовий набір даних. Coil буде використовуватися для відображення мініатюр відео, на рисунку 3.9.

Рисунок 3.9 — Медіа-плеєр exoPlayer

Як тільки елемент не видно, викликаємо onPlayVideoClick, який виконає описану раніше логіку збереження позиції відтворення та призупиняє програвач.

Давайте почнемо впроваджувати всі ці функції крок за кроком. Спочатку ми створюємо компонований файл, який міститиме екземпляр exoPlayer, список відео та playingItemIndex. Дуже важливою частиною тут є використання ключів для lazyColumn. Коротше кажучи: ми надаємо ключ, який дозволяє узгоджувати стан елемента в усіх змінах набору даних.

isPlaying ми завжди можемо знати, відтворюється відео чи ні. Якщо він грає playingItemIndex буде представляти позицію елемента в списку, якщо нічого не грає нуль. Нам потрібне це поле, щоб знати, чи потрібно показувати піктограму відтворення/паузи, мініатюру зображення або playerView. [5]

Відеокарта компонована, яка показує подію кліку на піктограмі відтворення/паузи. Клацання оброблятиметься всередині viewModel. Зверніть увагу, що ми передаємо поточну позицію відтворення з exoPlayer, а також індекс натиснутого елемента. Будемо використовувати його для збереження та відновлення позиції відтворення для кожного відео. Щоразу, коли натискається значок відтворення/паузи, перевіряємо три сценарії. [6]

CurrentPlayingIndex має значення null — відео наразі не відтворюється, тому призначаємо videoIndex для currentPlayingIndex. CurrentPlayingIndex - це те саме, що натиснуто videoIndex — це означає, що те саме відео вже відтворюється, і ми хочемо призупинити відтворення, деталі на рисунку 3.10.

Ось чому ми призначаємо null для currentPlayingIndex. Щоб зберегти позицію останнього відтвореного відео, ми змінюємо список і зберігаємо позицію, яку ми отримали від exoPlayer. Якщо відтворюється поточний елемент, ми створюємо VideoPlayer і надаємо йому екземпляр exoPlayer, щоб останній можна було приєднати до playerView.

VideoPlayer composable відкриє лямбда, яка вказує нам, чи є інтерфейс програвача видимим чи ні (поточний час відтворення, тривалість відео та панель пошуку). Це дозволяє нам синхронізувати поведінку між піктограмою відтворення/паузи та попереднім інтерфейсом системи перегляду, наданим із playerView.

IsPlayingButtonVisible — визначає, показувати чи приховувати піктограму відтворення/паузи. VideoThumbnail показує зображення з URL-адреси.

Для цього нам потрібно внести кілька налаштувань у snapshotFlow. Раніше він відповідав за надання нам інформації про те, видимий елемент для гри чи ні. Тепер він скаже нам, який елемент «у фокусі». Якщо бути точним - якщо ми на вершині списку - перший елемент буде відтворюватися. Якщо ми прокрутили список до кінця - відтвориться останній елемент. Все, що між ними, буде грати за допомогою стратегії, найближчої до центру.

Цей підхід базується на індексах, але ви завжди можете використовувати посилання на об’єкт або ідентифікатори, якщо це краще відповідає вашим потребам. LazyListState.layoutInfo.visibleItemsInfo повідомляє нам, які індекси є видимими, ми конвертуємо їх у відеоелементи та знаходимо, чи є відповідність із відео, яке зараз відтворюється.

3.2.4 Розробка функціоналу профілю

Профіль є невід'ємною частиною будь-якого додатку з кількома або більше користувачами, тому особливу увагу було надато саме цьому розділу.

Було використано різні анімації для кращого розуміння даного розділу, користувач за допомогою продуманого ux дизайну може з легкістю орієнтуватися у даному розділі, , що можна побачити на рисунку 3.10.

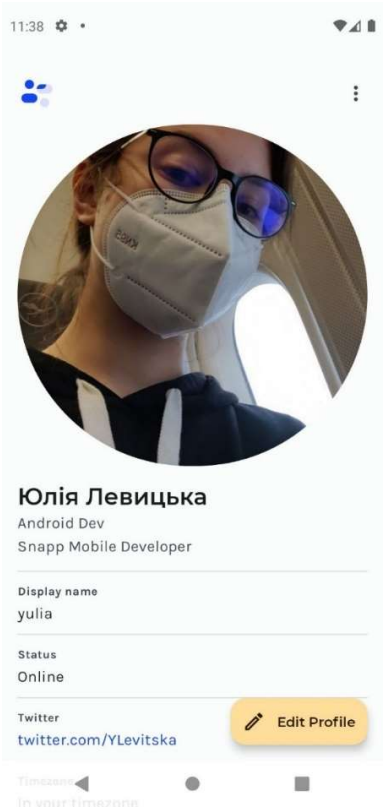

Рисунок 3.10 — UI та UX профілю

Даному розділі було вирішено добавити деякі особисті дані, такі як фотографія, посада на роботі або університеті, онлайн та оффлайн статус та твіттер канал.

Була реалізована анімація переходу повноцінної фотографії у режимі верхній статус бар, а також з деформацією форми даної фотографії. Розробки профілю було звернено увагу на різні профілі власний та способи особливістю цього є додаткові клавіши які допомагають або написати даному безсоння або диктувати свій профіль, , приклад на рисунку 3.11.

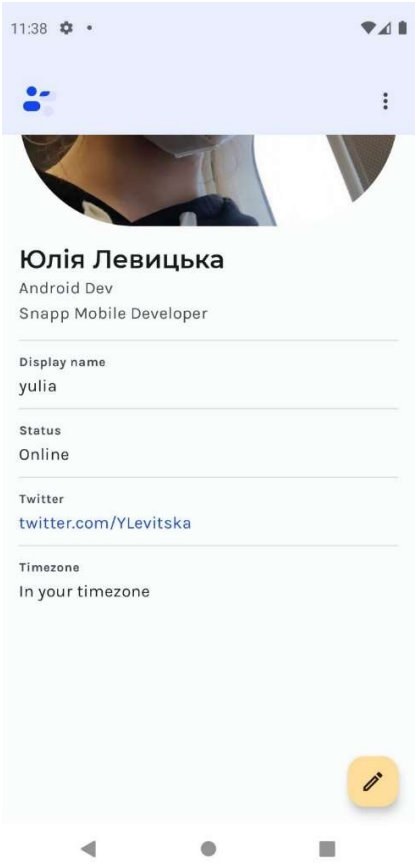

Рисунок 3.11 — Динамічна зміна профілю

Незалежно у якому профілю знаходиться користувач, чи у власному, чи іншого користувача. Динамічна кнопка, яка відрізняється у цих двох випадках,

буде додатково змінюватись. Незалежно який саме профіль спостерігаємо, вона буде відкриватися та звертатися. [3]

При відкритті кнопки ми можемо побачити іконку, а також текстове пояснення. У згорнутій клавіші ми можемо побачити тільки іконку даної кнопки, що робить її більш контактною та дозволяє відображати більше інформації стосовно профілю. Дана кнопка не буде заступати будь-яку інформацію про користувача, що робить її більш функціональною, що відтворено на рисунку 3.12.

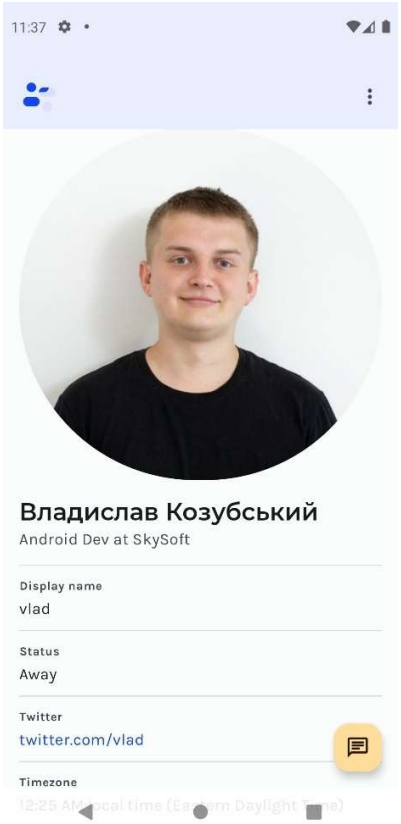

Рисунок 3.12 — Профіль іншого користувача

При переході на іншого користувача профіль можемо отримати такий функціонал як написання повідомлення у приватні повідомлення. Кнопка також є динамічною та дозволяє відкрити приватні повідомлення із таким самим функціоналом, як і у чаті. [4]

Деякі інші повідомлення є приватними, коли приватний чат закриваємо то отримаємо вихід на головну сторінку.

3.2.5 Розробка реєстрації додатку

Реєстрація є невід'ємною частиною більшості цифрових продуктів. Навіть круті компанії не можуть зробити зручну та безпечну систему логіну та відновлення доступу Здається, що всі продуктові дизайнери настільки звикли до того, що реєстрація - це просто, що навіть не докладають особливих зусиль для її проектування.

Реалізувати схему реєстрації, аутентифікації та відновлення доступу за допомогою email та соцмереж. Через універсальність інструкції, не допускається прив'язка до конкретних кадрів, баз даних або програмних рішень, розроблена UI та UX частина на рисунку 3.13.

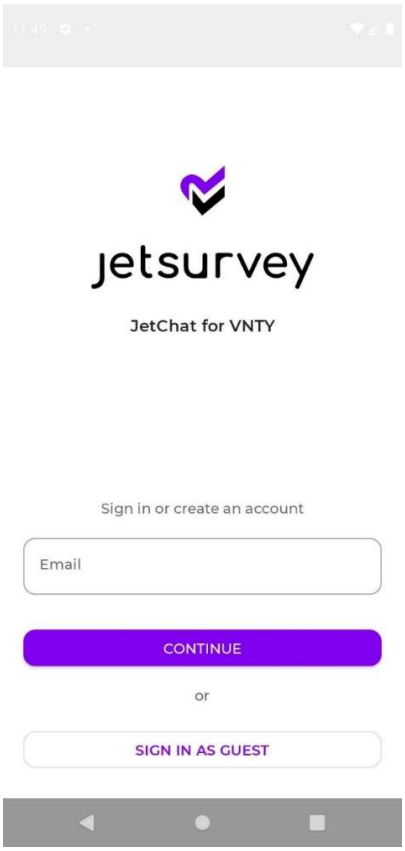

Рисунок 3.13 — Головна сторінка та вхід у додаток

Крім того, у схемі не реалізована подвійна автентифікація, оскільки вона передбачає певні технічні рішення (наприклад, TOTP або SMS), які можуть бути продиктовані особливостями продукту або бізнес-логіки, зображено на рисунку 3.14

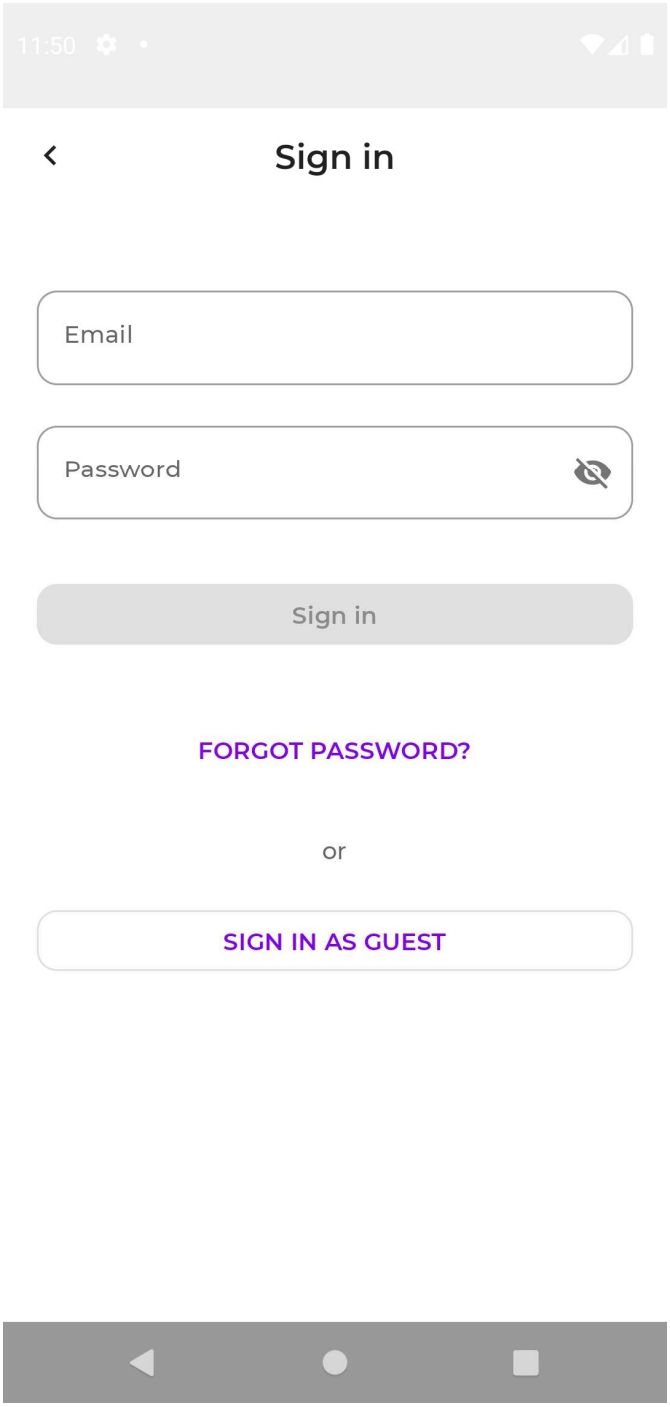

Рисунок 3.14 — Реєстрація за допомогою Copmose технології

При реєстрації через email, проводиться перевірка на наявність пошти в базі даних. Якщо користувач з таким email вже зареєстрований, відбувається передача управління процесом загальної функції перевірка на прив'язані соцмереж».

Тимчасовий додатковий акаунт з деякими можливостями та унікальним входом у систему. До того, як користувач не підтвердив свій email, його обліковий запис є тимчасовим. Це дозволяє більш гнучке управління правами доступу, таким користувачам можна обмежити набір дозволених дій, автоматично очищати базу від неактивованих акаунтів, після певного часу. [2]

У базі даних тимчасовий обліковий запис може відрізнятися від звичайного простим бульовим прапором - або такі облікові записи можуть взагалі зберігатися в окремій таблиціУ данному додатку відбувається різниця за допомогою простого прапору.

3.2.6 Розробка розділу з навчальними матеріалами та лекціями

Основний розділ даного додатку є розділ з навчальними матеріалами та лекціями. Саме для цього був розроблений даний додаток. Було реалізовано кілька методів та можливостей працювати з лекціями, як і студентам так і викладачам. Було реалізовано медіаплеєр, за допомогою якого можна переглядати навчальні матеріали, та лекції. Також було розроблено список для того щоб корректно зберігати всі навчальні данні.

На рисунку 3.15 зображено приклад зі збереженими навчальними матеріалами та лекціями. На даному прикладі можна побачити кілька відео, а також посилання на міт. Також було вирішено притримуватися стилістики даного додатку і при натисненні на логотип, який знаходиться у лівому верхньому углу, буде відбуватися перехід на бокову навігацію. Також передбачена можливість додаткового функціоналу, за допомогою іконки три крапки, який знаходиться у правому верхньому углу, можна редагувати лекцій та добавляти нові.

Щодо студентів, то вони можуть помічати лекції та зберігати собі на локальний пристрій, у цьому випадку було передбачено такий функціонал для студентів і добавлена локальна база даних room, що допоможе студента ознайомлюватися з лекціями без доступу до інтернету , тобто офлайн. Для цього достатньо зберегти завчасно собі дану лекцію яка буде доступна до тих пір поки студент вручну не видалить її.

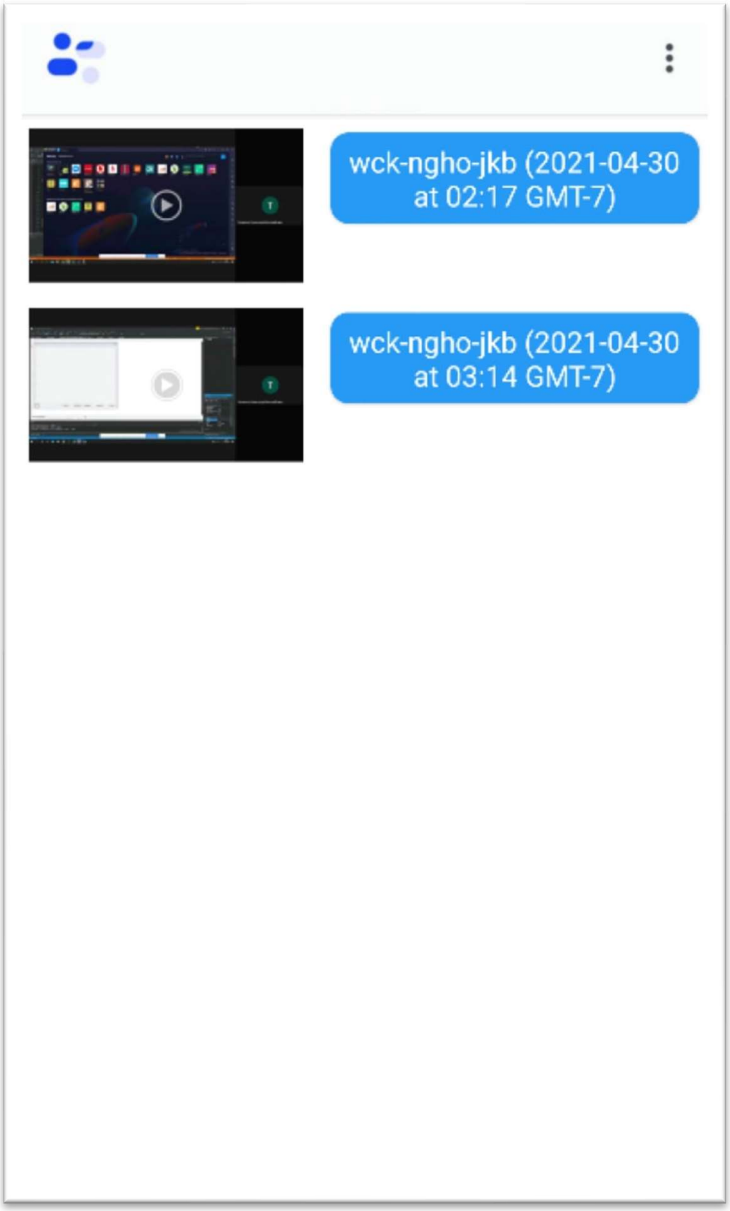

Рисунок 3.15 — Список навчальних матеріалів та лекцій розробленого додатку

Переглянути лекцію можна натиснувши на даний відео або на назву, після чого відкриється плеєр, за допомогою технології exoplayer, що допоможе переглянути дану лекцію. Для більш зручного користування даним додатком було передбачено можливість ділитися лекціями, а також позначати деяку цікаву інформацію.

За допомогою чату можна пересилати а також відзначати цікаві моменти разом зі всією групою студентів або скількома викладачами. Представлено відтворення лекції на рисунку 3.16.

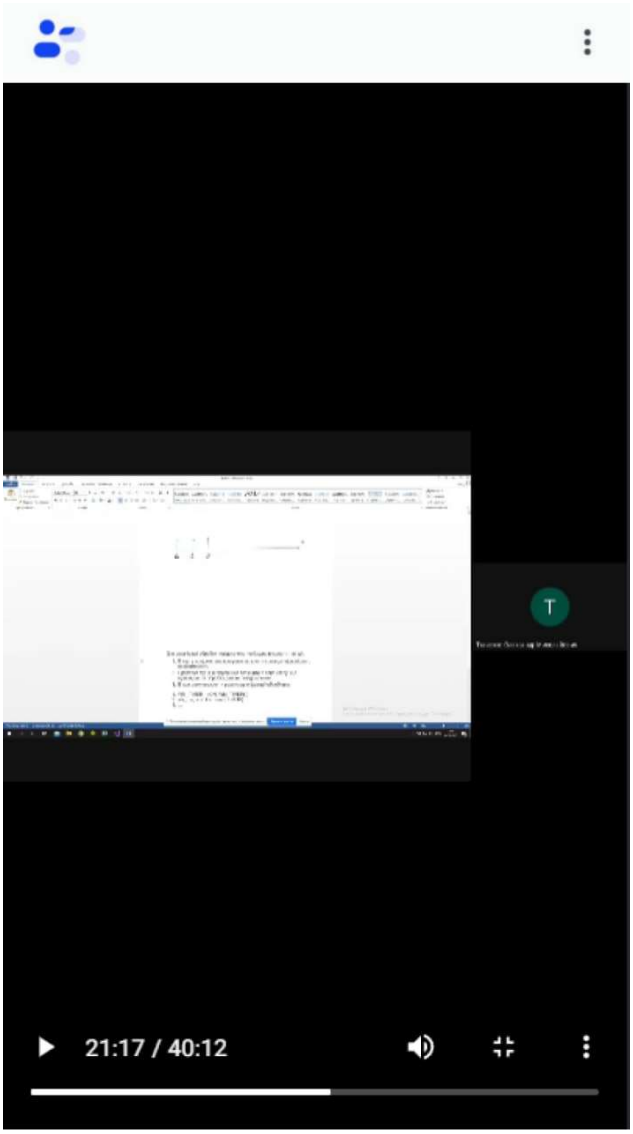

Рисунок 3.16 — Відтворення лекції за допомогою exoPlayer

Використовуючи технологію плеєра було відображено відео, а також можливість регулювати базові елементи, такі як час, звук, розмір, і тд. Для того щоб відкрити у повний розмір даному лекцію достатньо перевернути екран та увімкнути можливість перегляду в горизонтальні орієнтації. Це допоможе студента краще бачити матеріал який транслює викладач. [11]

3.2.7 Розробка додоткового функціоналу

Було передбачено додатковий функціонал данного додатку. Так як додаток знаходиться на стадії розробки та містить у собі не всі реалізовані функції та ідеї. При відкритті такої ідеї або функції, буде з'являтися повідомлення.

За допомогою діалогового вікна, яке буде повідомляти користувача про недоступний функціонал. Також це вікно можна буде закрите за допомогою спеціальної кнопки клоус. [3]

Діалогове вікно, яке було використано, запрограмоване так, щоб його не можна було закрити простим натисканням в не полі цього вікна. Це зроблено для того щоб користувач зміг прочитати все повністю та зі всією відповідальністю закрити повідомлення вручну.

Щоб не було деяких моментів, коли користувач випадково не звернувши увагу закрив данне діалогове вікно та не встиг прочитати про не доступний функціонал або не реалізовано ідею. Таке функуіональне діалогове вікно зображено нижче , а саме на рисунку 3.17. На даному прикладі можна побачити кілька відео, а також посилання на міт. Також було вирішено притримуватися стилістики даного додатку і при натисненні на логотип, який знаходиться у лівому верхньому углу, буде відбуватися перехід на бокову навігацію.

Також передбачена можливість додаткового функціоналу, за допомогою іконки три крапки, який знаходиться у правому верхньому углу, можна редагувати лекцій та добавляти нові.

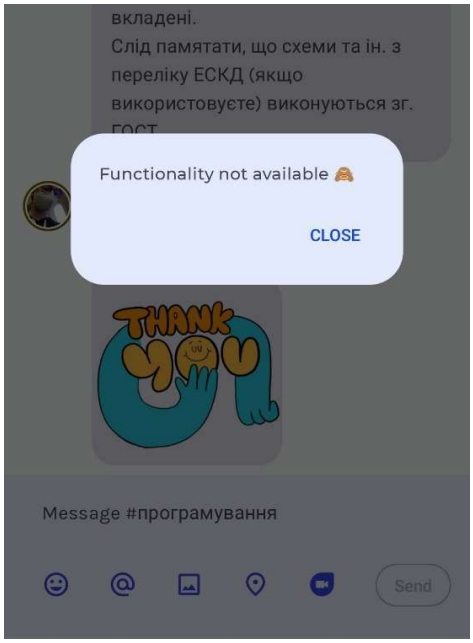

Рисунок 3.17 — Функціональне діалоговек вікно.

Такі діалогові повідомлення було використано у всіх неалізованих функціях та ідеях .у самому чаті та на головній сторінці даного додатку чи то при відкриття навігації, діалогові повідомлення не відрізняються один від одного та викликається й закриваються однаково.

3.3 Автоматизація побудови із Gradle та файл маніфесту

Додатки Android є строго визначений спеціальною системою життєвий цикл. При стартовому запуску користувачем данного додатка, система дає цьому йому високий пріоритет.

Кожен андроїд додаток запускається спеціально у вигляді окремого процесу, який дозволяє системі надавати одним процесам вищий пріоритет, на відміну від інших процесів, файл якого відображено на рисунку 3.18. Та якщо запускається операція, яка буде вказувати режим запуску сінглТаск, то вся задача буде перекладатись на передній план, і якщо екземпляр данної операції існує в задачі фоновій.

```
apply plugin: 'kotlin-kapt'
apply plugin: 'kotlinx-serialization'

android {
    compileSdkVersion 29

    defaultConfig {
        applicationId "com.example.frog"
        minSdkVersion 16
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt')
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

repositories {
    mavenCentral()
    jcenter()
}
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.2.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
```

Рисунок 3.18 — Побудова Gradle файлу

Завдяки цьому, при роботі з одним додатком не буде блокуватись вхідні дзвінки. Після зупинення роботи з додатком, андроїд система звільняє деякі або всі пов'язані ресурси та переводить той чи інший додаток в розряд низкоприоритетного та з часом закриває його, що детально відображенно на рисунку 3.19.

Система файлів градл є найважливішою файловою системою додатку, яка містить у собі версії коду та підключені бібліотеки. Якщо бібліотека застаріла , або взагалі була вилучена з доступу то у цій файловій системі градл можна легко побачити всі негаразди. Також перед запуском додатку програма буде проводити зборку гадл файла, щоб засвідчитись чи усі модулі впорядку.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.frog">

    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".BasicActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Рисунок 3.19 —Побудова маніфест файлу

При запуску донної операції в новій або іншій задачі, та при запуску операції у деякій існуючій задачі, кнопка бехг завжди повертає користувача на головний екран або до попередньої операції. Та якщо запускається операція, яка буде вказувати режим запуску сінглТаск, то вся задача буде перекладатись на передній план, і якщо екземпляр данної операції існує в задачі фоновій. У цей спеціальний момент, стек переходів назад буде поміщати всі операції із задачі, яка була перекладена на передній план, або на вершину стека. [1]

3.4 Порівняння з існуючими аналогами

Сучасному світу і сучасним реаліям потрібні спеціальні пристосування для роботи онлайн. У наші дні це найбільш актуально. Необхідно проаналізувати переваги та недоліки альтернативних додатків. Хоча їх і дуже мало але кілька з них ми можемо представити. Розроблений додаток був порівняний з таким гігантом як google classroom , а також з jetiq.

Звичайно, кожен з данних наведених прикладів має власні відмінності від своїх аналогічних додатків, в чомусь вони виграють, а у чомусь ні. Для того щоб

краще зрозуміти, яким чином нова розробка обходить аналогові застосунки на сьогоднішній день на ринку,потрібно проаналізувати недоліки цих додатків та реалізувати їх виправлення у своїй розробці,а також додати вжеготові переваги.

Проведено ряд досліджень, який допоможе визначити та проаналізувати існуючі аналоги . Основними вимогами буде швидкість та обсяг пам'яті, а також додаткові можливості додатка. Для цього буде проводитися два дослідження. Технічний , який допоможе визначити швидкість кожного додатку, а також його затрати пам'яті.

Наступне дослідження , функціональне, яке допоможе проаналізовати різноманітний функціонал, а також зручність використання.

Порівняння функціональної частини та характеристик переваг/недоліків конкурентних мобільних додатків наведена нижче, а саме у таблиці 3.2. На основі даних з цієї таблиці можна зробити висновки та правельні дії щодо впровадження конкретних пунктів у проектований додаток.

Провівши огляд конкуруючих аналогових андроїд додатків, можна проаналізувати їх переваги та недоліки між один одним, та на основі даної інформації зробити корисні висновки щодо концепцій, які потрібно включити або не допускати таких же помилок при виконанні даної комплексної магістерської кваліфікаційної роботи.

Таблиця 3.1 — Технічні переваги та недоліки альтернативних додатків

Назва програми	Швидкість відкриття додатку та лекції	Кількість затраченої пам яті
Google Classroom	-1890 мс	40 Мб
JetIq	-3120 мс	32 Мб
Jetchat for VNTU	-200 мс	9 Мб

Таблиця 3.2 — Функціональні переваги та недоліки альтернативних додатків

Назва програми	Переваги	Недоліки
Google Classroom	-можливість менеджити студентів у спеціальних групах; -можливість перевіряти завдання; -захищеність; -можливість залишати коментарі;	-не користувацько дружельбний інтерфейс; -немає можлиіості створювати онлайн чат; -не розвинена система додаткової інформації про студента (профіль); -не має можливості одразу додавати кілька зображень у завдання;
JetIq	-присутній розклад для студента; -інтуєтивно зрозумілий інтерфейс; - розвинена система додаткової інформації про студента; -можливість менеджити студентів у групи;	-немає можлиіості створювати онлайн чат; -немає можливості залишати коментарі; -не має можливості завантажити матеріали; -не можливість працювати офлайн; -часті проблеми при вході у додаток;
Jetchat for VNTU	-інтуєтивно зрозумілий інтерфейс; - розвинена система додаткової інформації про студента; -можливість менеджити студентів у групи; -захищеність; - можливість додавати кілька зображень та відео у завдання; - можливість працювати офлайн;	-відсутній розклад для студента; -немає можливості залишати коментарі; -не має конкретної інформації про підгрупи студента.

З технічної точки зору даний додаток немає аналогів, тому як являється найшвидшим та найменшим витратним по пам'яті. Це завдяки новітній технології compose, яка допомагає відображати це все з феноменальною швидкістю, а також займати мінімальний обсяг пам'яті. Як можемо бачити у таблиці 3.2 інші додатки не мають шансів у конкуренції з данною розробкою.

У данному розділі було розроблено мобільний додаток на основі ОС Android , також створено коректний навігаційний граф додатку . Розроблено користувацький інтерфейс з зрозумілим UI та UX. Для вирішення поставленої задачі узгоджено та застосовано такі технології створення макетів як Compose. Саме така технологія поєднання методів дає найбільші можливості для роботи з додатками, не навантажуючи UI-потік та UX, а також дозволяє швидко обробляти різні сценарії подій.

4 ЕКОНОМІЧНА ЧАСТИНА

Темою комплексної магістерської кваліфікаційної роботи є «Хмарно-стрімінговий медіа плеєд для навчальних матеріалів та лекцій. Частина 1 Compose». За цією темою в економічній частині буде проводитись розрахунки економічних показників для розробки програмного продукту й впровадження його на ринок аналогічних додатків .

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення технічного на функціонального аудиту є оцінювання комерційного потенціалу данного додатку на основі технології Compose , для навчальних матеріалів та лекції.

Для проведення технологічного аудиту було зарошено незалежних експертів. Здійснюємо оцінювання комерційного потенціалу розробки за 12-макритеріями, наведеними у таблиці 4.1, за 5-ти бальною шкалою.

Таблиця 4.1 — Технічні переваги та недоліки альтернативних додатків

Бали (за 5-ти бальною шкалою)					
Кри-те-рій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність проду-кту в реаль-них умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому рин-ку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно до-рівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

4	Технічні та споживчі властивості продукту значно гірші,ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші,ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі,ніж в аналогів	Технічні та споживчі властивості продукту значно кращі,ніж в аналогів
5	Експлуатаційні витрати значно вищі,ніж в аналогів	Експлуатаційні витрати дещо вищі,ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі,ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної,так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної,так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси,які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси.Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали,що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10років	Термін реалізації ідеї більший за 5років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х Років

Кінець таблиці 4.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту,що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 4.2

Таблиця 4.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Ткаченко О.М, к.т.н доц. кафедри ОТ	2. Черняк О.І, к.т.н, доц. кафедри ОТ
	Бали, виставлені експертами:	
1	4	4
2	0	1
3	4	4
4	2	3
5	4	3
6	2	3
7	2	3
8	3	3
9	4	4
10	4	4
11	4	3
12	4	4
Сума балів	$СБ_1 = 37$	$СБ_2 = 39$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 38$	

Отже, з отриманих даних таблиці 4.1 видно, що рівень потенціалу нової розробки — вище середнього. Дана розробка є конкурентоспроможною з аналогами, так як для її розробки було проаналізовано недоліки та переваги аналогових продуктів, та на основі цього впроваджену у розробку. Вона має

соціологічний вплив, так як покращує ефективність контролю бюджету користувачів, що є важливим у житті людей з обмеженими фінансами.

4.2 Прогнозування витрат на виконання науково-дослідної роботи таконструкторсько-технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати. Основна заробітна плата для розробників визначається за формулою (4.1):

$$Зо = \frac{М}{Тр} t \quad , \tag{4.1}$$

де М — місячний посадовий оклад конкретного розробника;

Тр — кількість робочих днів у місяці, Тр = 20 днів;

t — число днів роботи розробника, t = 37 днів.

Розрахунки заробітних плат для керівника і програміста в таблиці 4.3

Таблиця 4.3 — Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи t	Витрати на оплату праці, грн.
Науковий керівник	11000	525	33	17325
Інженер-програміст	9600	480	33	15840
Всього:				33165

Розрахуємо додаткову заробітну плату, вона розраховується як 10-12% від суми основної заробітної плати всіх розробників:

$$З_{дод}=0,1*33165=3316.5(грн)$$

Нарахування на заробітну плату Нзп для працівників бюджетної сферистановить 22% від суми основної та додаткової заробітної плати:

$$H_{3n} = (3o + 3p) \cdot \frac{\beta}{100}, \quad (4.3)$$

$$H_{3П} = (33165 + 3316) \cdot \frac{22}{100} = 8{,}025{,}93(\text{грн})$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$A = \frac{Ц \cdot H_a}{100} \cdot \frac{T}{12}, \quad (4.4)$$

де Ц — балансова вартість обладнання, грн;

На — річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т — Термін використання (Т=2 міс.).

Таблиця 4.4 — Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс	Величина амортизаційних відрахувань, грн
Персональний комп’ютер	12700	25	2	529,1
Мобільний пристрій	4300	15	2	107,5
Всього:				636,6

Розрахуємо інші витрати Він. Інші витрати Ів можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$B_{iH} = (1 \dots 3) \cdot (3o + 3p), \quad (4.5)$$

Отже, розрахуємо інші витрати:

$$B_{iH} = 1 \cdot (17340 + 1734) = 19074(\text{грн})$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$B = 3e + 3g + H_{3П} + A + K + Be + I_B, \quad (4.6)$$

В=17340+1734+4196,28+636,6+196,9+292,15+19074=43469,83(грн)

Розрахуємо загальну вартість наукової роботи Взаг за формулою:

$$B_{\Sigma} = \frac{B_{iH}}{a}, \quad (4.7)$$

Де а — частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$B_{\Sigma} = \frac{43469,83}{1} = 43469,83 \text{ (грн)}$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$3B = \frac{B_{\Sigma}}{\beta}, \quad (4.8)$$

де В — коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$3B = \frac{43469,83}{0,9} = 48299,8$$

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо кількісно, яку вигоду можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

Виконання наукової роботи та впровадження її результатів буде здійснюватися протягом одного року. Основні позитивні результати від впровадження розробки очікуються протягом 3-х років після її впровадження. Одним із основних позитивних результатів є зростання величини прибутку.

При реалізації результатів наукової розробки покращується якість програмного продукту, що дозволяє підвищити ціну його реалізації на 150 грн. Кількість одиниць реалізації програмного засобу також збільшиться: протягом першого року — на 700 шт., протягом другого року — ще на 450 шт., протягом третього року — ще на 300 шт.

Реалізація продукції до впровадження результатів наукової розробки складала 50 шт., а ціна — 250 грн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового за такою формулою 4.2:

$$\Delta П_i = \sum_1^n (\Delta П_k \cdot N + П_k \Delta N) i, \qquad (4.9)$$

де Пі — покращення основного оціночного показника від впровадження результатів розробки у даному році. Зазвичай таким показником може бути ціна одиниці нової розробки;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

Д N — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

Цо — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт 0,8333.

ρ — коефіцієнт, який враховує рентабельність продукту. Рекомендує ться приймати = 0,2...0,3;

υ — ставка податку на прибуток. υ = 18%.

Тоді, збільшення чистого прибутку підприємства протягом першого року складе:

$$\Delta\text{Пз} = [150 * 50 + (250 + 150) * 700] * 0{,}8333 * 0{,}2 * \left(1 - \frac{18}{100}\right) = 39290{,}09$$

Протягом другого року:

$$\Delta\text{Пз} = [150 * 50 + (250 + 150) * (700 + 450)] * 0{,}8333 * 0{,}2 * \left(1 - \frac{18}{100}\right) = 61\,710$$

Протягом третього року:

$$\Delta\text{Пз} = [150 * 50 + (250 + 150) * (700 + 450 + 300)] * 0{,}8333 * 0{,}2 * \left(1 - \frac{18}{100}\right) = 79\,900$$

Отже, протягом трьох років підприємство може розраховувати на збільшення чистого прибутку від реалізації наукової розробки.

4.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність Еабсвкладених інвестицій розраховується за формулою:

$$Egrad = (ПП - PV), \quad (4.10)$$

де ПП — приведена вартість всіх чистих прибутків, які отримає підприємство (організація) від реалізації результатів наукової розробки, грн.;

PV — теперішня вартість інвестицій PV = ЗВ = 48 299,8 грн.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд.

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним, оскільки Ток<3 років.

Отже, результати проведених розрахунків дають можливість зробити висновок про доцільність розробки та впровадження нашої наукової роботи. Це підтверджують такі показники як:

— абсолютна ефективність вкладених інвестицій, яка дорівнює 133 175,07 грн., що є більшим 0 і вказує на те, що інвестор може бути зацікавленим у нашій розробці;

— відносна ефективність наукової розробки становить 56%, що є вищим за мінімальну ставку дисконтування (30%), тому вкласти кошти у нашу розробку є вигідніше, ніж покласти кошти на депозит;

— термін окупності вкладених у реалізацію наукового проекту інвестицій складе 1,7 років, що є менше 3-ох і вказує швидку окупність інвестицій.

Крім того, розраховано, що наукова розробка приноситиме підприємству додатковий прибуток протягом 3-х років за рахунок покращення її якості порівняно з існуючими аналогами.Усе це, узяте разом, забезпечує прийняття рішення про доцільність виготовлення нового продукту.

ВИСНОВКИ

У комплексній магістерскій дипломній роботі спроектовано та розроблено мобільний додаток на основі ОС Android.

У першому розділі проведений аналіз мов програмування та технологій побудови UI та UX, що використовуються для створення мобільних додатків. Для даної розробки найбільш доцільним є використання мови програмування Kotlin та середовище розробки Android Studio. Також рекомендується використування APD, так як данна розробка потребує роботи з камерою та відеоплеєром.

У другому розділі було розглянуто типи архітектури програмного забезпечення мобільних додатків. Також було проаналізовани життєві цикли Activity та Fragment . Для данного мобільного додатка є набільш доцільним використання архітектури MVVM . Обрано життєвий цикил Activity , на бажі якого відбувається виклик Fragment.

У третьому розділі було розроблено мобільний додаток на ОС Android , створено коректний навігаційний граф додатку . Розроблено користувацький інтерфейс з кількома макетама. Для вирішення поставленої задачі узгоджено та застосовано такі технології: Compose, Room, Firebase. Саме таке поєднання методів дає найбільш гнучкі можливості для роботи з додатком, не навантажуючи UI-потік, а також дозволяє обробляти різні сценарії подій.

Отже, всі завдання роботи виконано, мету досягнуто.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jetpack Compose. [Електронний ресурс] : [Веб-сайт] – Електронні дані. - Режим доступу: https://apptractor.ru/info/articles/kontseptsii-jetpack-compose-kotorye-dolzhen-znat-kazhdyy-razrabotchik.html

2. Google выпустила стабильную версию Jetpack Compose 1.0. [Електронний ресурс] : [Веб-сайт] – Електронні дані. - Режим доступу: https://tproger.ru/news/google-nakonec-to-vypustila-jetpack-compose-jetot-instrument-anonsirovali-2-goda-nazad/

3. Jetpack Compose is now 1.0: announcing Android's modern toolkit for building native UI. [Електронний ресурс] : [Веб-сайт] – Електронні дані. - Режим доступу: https://android-developers.googleblog.com/2021/07/jetpack-compose-announcement.html

4. Android Programming In Kotlin: RelativeLayout. [Електронний ресурс] : [Веб-сайт] – Електронні дані. - Режим доступу: http://developer.alexanderklimov.ru/android/layout/ relativelayout.php

5. TableLayout. [Електронний ресурс] : [Веб-сайт] – Електронні дані. - Режим доступу: https://developer.android.com/reference/kotlin/android/widget/ TableLayout 6. AIDE- IDE for Android Java C++ [Електронний ресурс] - Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.aide.ui

7. Java Editor [Електронний ресурс] - Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=air.JavaEditor

8. JavaIDEdroid [Електронний ресурс] - Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=ch.tanapro.JavaIDEdroid

9. The Professional Android IDE [Електронний ресурс] - Режим доступу до ресурсу: http://www.jetbrains.com/idea/features/android.html

10 NBAndroid [Електронний ресурс] - Режим доступу до ресурсу: http://plugins.netbeans.org/plugin/19545/nbandroid

11. Android Studio [Електронний ресурс] - Режим доступу до ресурсу: http://developer.android.com/sdk/index.html

12. Backup & restore Android apps using adb [Електронний ресурс] - Режим доступу до ресурсу: http://jonwestfall.com/2009/08/backup-restore-android-apps-using-adb/

13. SDK Tools [Електронний ресурс] - Режим доступу до ресурсу: http://developer.android.com/tools/sdk/tools-notes.html

14. Dalvik Executable format [Електронний ресурс] - Режим доступу до ресурсу: https://source.android.com/devices/tech/dalvik/dex-format.html

15. Android — Invoke JNI Based Methods (Bridging C/C++ And Java) [Електронний ресурс] - Режим доступу до ресурсу: https://davanum.wordpress.com/2007/12/09/android-invoke-jni-based-methodsbridging-cc-and-java

16. Native C applications for Android [Електронний ресурс] - Режим доступу до ресурсу: http://benno.id.au/blog/2007/11/13/android-native-apps

17. Android NDK [Електронний ресурс] - Режим доступу до ресурсу: https://developer.android.com/tools/sdk/ndk/index.html

18. ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ДЛЯ РОЗПІЗНАВАННЯ ОБРАЗІВ НА ОСНОВІ НЕЙРОМЕРЕЖІ. РОЗРОБКА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ. [Текст] Ю.Р.Левицька // XLVII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2019): Тез. доп. - Вінниця, 2019. - 1. Режим доступу https:// conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9470/7736

ДОДАТОК А

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
Азаров О. Д.
«____» _____2021 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи на тему:
«Хмарно-стрімінговий медіа плеєр для навчальних матеріалів та лекцій. Частина1. Compose система»
08-23.КМКР.027.00.000 ПЗ

Науковий керівник: _____Ткаченко О.М.
(підпис)
студент групи _____ Левицька Ю.Р.
(підпис)

Вінниця 2021

1. Підстава для виконання комплексної магістерської кваліфікаційної роботи (КМКР)

а) актуальність досліджень;

б) наказ про затвердження теми дипломної роботи.

2. Мета і призначення КМКР

Мета полягає у розширення функціональних можливостей мобільних пристроїв за рахунок розроблення мобільного додатку на ОС Android

3. Джерела розробки КМКР:

— Google Play Hits 1 Million Apps [Електронний ресурс] - Режим доступу до ресурсу: http://mashable.com/2013/07/24/googleplay-1-million/

— Android App Stats [Електронний ресурс] - Режим доступу до ресурсу: http://www.androlib.com/appstats.aspx

4. Технічні вимоги до виконання МКР

Для кожного додатка є необхідність використання кількох макетів, що допомагає створити інтерфейс простим у використанні, сучасним та цікавим.

5. Етапи МКР та очікувані результати

Таблиця А.1 — Етапи виконання роботи

Етап	Назва етапу	Початок	Кінець	Очікувані результати
1.	Інформаційний пошук та огляд літературних джерел.	26.09.2021	07.10.2020	Розділи 1, 2 та 3.
2.	Дослідження підходів та розробка мобільного додатку на ОС Android	11.10.2020	03.11.2020	Чернетки матеріалів.
3.	Підготовка матеріалів пояснювальної записки.	20.11.2020	05.12.2020	Пояснювальна записка.

6. Матеріали, що подаються до захисту КМКР

Пояснювальна записка КМКР, графічні та ілюстративні матеріали, протокол попереднього захисту КМКР на кафедрі, відгук наукового керівника, відгук рецензента, анотації до КМКР українською та іноземною мовами, відповідність оформлення КМКР діючим вимогам.

7. Порядок контролю виконання та захисту КМКР

Виконання етапів графічної та розрахункової документації КМКР контролюється науковим керівником згідно зі встановленими термінами. Захист КМКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8. Вимоги до оформлення КМКР

Вимоги викладені в МЕТОДИЧНИХ ВКАЗІВКАХ до дипломного проектування, ДСТУ 3008-95, ДСТУ 3974-2000 «Правила виконання дослідно-конструкторських робіт. Загальні положення» та діючого ГОСТ 2.114-95 ЕСКД.

9. Вимоги щодо технічного захисту інформації в ДР з обмеженим доступом

Відсутні.

Технічне завдання до виконання прийняла _____

ДОДАТОК Б

Лістинг програми

```
private const val EMAIL_VALIDATION_REGEX = "^(.+)@(.+)\$"

class EmailState :

    TextFieldState(validator = ::isEmailValid, errorFor = ::emailValidationError)

private fun emailValidationError(email: String): String {

    return "Invalid email: $email"

}private fun isEmailValid(email: String): Boolean {

    return Pattern.matches(EMAIL_VALIDATION_REGEX, email)

}class PasswordState :

    TextFieldState(validator = ::isPasswordValid, errorFor = ::passwordValidationError)

class ConfirmPasswordState(private val passwordState: PasswordState) : TextFieldState() {

    override val isValid

        get() = passwordAndConfirmationValid(passwordState.text, text)

    override fun getError(): String? {

        return if (showErrors()) {

            passwordConfirmationError()

        } else {

            null

        }

    }}
```

```
        private fun passwordAndConfirmationValid(password: String, confirmedPassword: String): Boolean {
            return isPasswordValid(password) && password == confirmedPassword
        }private fun isPasswordValid(password: String): Boolean {
            return password.length > 3
        }@Suppress("UNUSED_PARAMETER")
        private fun passwordValidationError(password: String): String {
            return "Invalid password"
        }private fun passwordConfirmationError(): String {
            return "Passwords don't match"
        }class SignInFragment : Fragment() {
            private val viewModel: SignInViewModel by viewModels { SignInViewModelFactory() }
            override fun onCreateView(
                inflater: LayoutInflater,
                container: ViewGroup?,
                savedInstanceState: Bundle?
            ): View? {
                viewModel.navigateTo.observe(viewLifecycleOwner) { navigateToEvent -> navigateToEvent.getContentIfNotHandled()?.let { navigateTo ->
                    navigate(navigateTo, Screen.SignIn)
                } } return ComposeView(requireContext()).apply {
                // In order for savedState to work, the same ID needs to be used for all instances.
```

```
            id = R.id.sign_in_fragment
            layoutParams = ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.MATCH_PARENT
            )           setContent {
                JetsurveyTheme {
                    SignIn(
                        onNavigationEvent = { event ->
                            when (event) {
                                is SignInEvent.SignIn -> {
                                    viewModel.signIn(event.email, event.password)
                                } SignInEvent.SignUp -> {
                                    viewModel.signUp()
                                } SignInEvent.SignInAsGuest -> {
                                    viewModel.signInAsGuest()
                                }
                                SignInEvent.NavigateBack -> {
                                    activity?.onBackPressedDispatcher?.onBackPressed()
                                }}}
                    )}}}}}
sealed class SignInEvent {
    data class SignIn(val email: String, val password: String) : SignInEvent()
    object SignUp : SignInEvent()
```

```
    object SignInAsGuest : SignInEvent()
    object NavigateBack : SignInEvent()
}
@OptIn(ExperimentalMaterialApi::class)
@Composable
fun SignIn(onNavigationEvent: (SignInEvent) -> Unit) {
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()
    val snackbarErrorText = stringResource(id = R.string.feature_not_available)
    val snackbarActionLabel = stringResource(id = R.string.dismiss)
    Scaffold(
        topBar = {
            SignInSignUpTopAppBar(
                topAppBarText = stringResource(id = R.string.sign_in),
                onBackPressed = { onNavigationEvent(SignInEvent.NavigateBack) }
            )
        },
        content = {
            SignInSignUpScreen(
                modifier = Modifier.supportWideScreen(),
                onSignedInAsGuest = { onNavigationEvent(SignInEvent.SignInAsGuest) }
            ) {
```

```
            Column(modifier = Modifier.fillMaxWidth()) {
                SignInContent(
                    onSignInSubmitted = { email, password ->
                        onNavigationEvent(SignInEvent.SignIn(email, password))
                    }
                )
                Spacer(modifier = Modifier.height(16.dp))
                TextButton(
                    onClick = {
                        scope.launch {
                            snackbarHostState.showSnackbar(
                                message = snackbarErrorText,
                                actionLabel = snackbarActionLabel
                            )}
                    },
                    modifier = Modifier.fillMaxWidth()
                ) {
                    Text(text = stringResource(id = R.string.forgot_password))
                }}
        }})
    Box(modifier = Modifier.fillMaxSize()) {
        ErrorSnackbar(
            snackbarHostState = snackbarHostState,
```

```
                onDismiss = { snackbarHostState.currentSnackbarData?.dismiss() },
                modifier = Modifier.align(Alignment.BottomCenter)
            )}}
        @Composable
        fun SignInContent(
            onSignInSubmitted: (email: String, password: String) -> Unit,
        ) {
            Column(modifier = Modifier.fillMaxWidth()) {
                val focusRequester = remember { FocusRequester() }
                val emailState = remember { EmailState() }
                Email(emailState, onImeAction = { focusRequester.requestFocus() })
                Spacer(modifier = Modifier.height(16.dp))
                val passwordState = remember { PasswordState() }
                Password(
                    label = stringResource(id = R.string.password),
                    passwordState = passwordState,
                    modifier = Modifier.focusRequester(focusRequester),
                    onImeAction = { onSignInSubmitted(emailState.text,
passwordState.text) }
                )
                Spacer(modifier = Modifier.height(16.dp))
                Button(
                    onClick = { onSignInSubmitted(emailState.text, passwordState.text) },
```

```
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 16.dp),
            enabled = emailState.isValid && passwordState.isValid
        ) {
            Text(
                text = stringResource(id = R.string.sign_in)
            )}}}
@OptIn(ExperimentalMaterialApi::class)
@Composable
fun ErrorSnackbar(
    snackbarHostState: SnackbarHostState,
    modifier: Modifier = Modifier,
    onDismiss: () -> Unit = { }
) {
    SnackbarHost(
        hostState = snackbarHostState,
        snackbar = { data ->
            Snackbar(
                modifier = Modifier.padding(16.dp),
                content = {
                    Text(
                        text = data.message,
```

```
                style = MaterialTheme.typography.body2
            )
        },
        action = {
            data.actionLabel?.let {
                TextButton(onClick = onDismiss) {
                    Text(
                        text = stringResource(id = R.string.dismiss),
                        color = MaterialTheme.colors.snackbarAction
                    )}}})},
    modifier = modifier
        .fillMaxWidth()
        .wrapContentHeight(Alignment.Bottom)
)}
@Preview(name = "Sign in light theme")
@Composable
fun SignInPreview() {
    JetsurveyTheme {
        SignIn {}
    }}
@Preview(name = "Sign in dark theme")
@Composable
fun SignInPreviewDark() {
```

```
    JetsurveyTheme(darkTheme = true) {
        SignIn {}
    }}
@Composable
fun SignInSignUpScreen(
    onSignedInAsGuest: () -> Unit,
    modifier: Modifier = Modifier,
    content: @Composable() () -> Unit
) {
    LazyColumn(modifier = modifier) {
        item {
            Spacer(modifier = Modifier.height(44.dp))
            Box(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 20.dp)
            ) {
                content()
            }
            Spacer(modifier = Modifier.height(16.dp))
            OrSignInAsGuest(
                onSignedInAsGuest = onSignedInAsGuest,
                modifier = Modifier
```

```
                .fillMaxWidth()
                .padding(horizontal = 20.dp)
            )
        }
    }
}
@Composable
fun SignInSignUpTopAppBar(topAppBarText: String, onBackPressed: () -> Unit) {
    TopAppBar(
        title = {
            Text(
                text = topAppBarText,
                textAlign = TextAlign.Center,
                modifier = Modifier
                    .fillMaxSize()
                    .wrapContentSize(Alignment.Center)
            )
        },
        navigationIcon = {
            IconButton(onClick = onBackPressed) {
                Icon(
                    imageVector = Icons.Filled.ChevronLeft,
```

```
                contentDescription = stringResource(id = R.string.back)
            )
        }
    },
    // We need to balance the navigation icon, so we add a spacer.
    actions = {
        Spacer(modifier = Modifier.width(68.dp))
    },
    backgroundColor = MaterialTheme.colors.surface,
    elevation = 0.dp
  )
}
@Composable
fun Email(
  emailState: TextFieldState = remember { EmailState() },
  imeAction: ImeAction = ImeAction.Next,
  onImeAction: () -> Unit = {}
) {
  OutlinedTextField(
    value = emailState.text,
    onValueChange = {
      emailState.text = it
    },
```

```
            label = {
                CompositionLocalProvider(LocalContentAlpha provides
ContentAlpha.medium) {
                    Text(
                        text = stringResource(id = R.string.email),
                        style = MaterialTheme.typography.body2
                    )
                }
            },
            modifier = Modifier
                .fillMaxWidth()
                .onFocusChanged { focusState ->
                    emailState.onFocusChange(focusState.isFocused)
                    if (!focusState.isFocused) {
                        emailState.enableShowErrors()
                    }
                },
            textStyle = MaterialTheme.typography.body2,
            isError = emailState.showErrors(),
            keyboardOptions = KeyboardOptions.Default.copy(imeAction =
imeAction),
            keyboardActions = KeyboardActions(
                onDone = {
                    onImeAction()
```

```
            }
        )
    )

    emailState.getError()?.let { error -> TextFieldError(textError = error) }
}
@Composable
fun Password(
    label: String,
    passwordState: TextFieldState,
    modifier: Modifier = Modifier,
    imeAction: ImeAction = ImeAction.Done,
    onImeAction: () -> Unit = {}
) {
    val showPassword = remember { mutableStateOf(false) }
    OutlinedTextField(
        value = passwordState.text,
        onValueChange = {
            passwordState.text = it
            passwordState.enableShowErrors()
        },
        modifier = modifier
            .fillMaxWidth()
```

```
            .onFocusChanged { focusState ->
                passwordState.onFocusChange(focusState.isFocused)
                if (!focusState.isFocused) {
                    passwordState.enableShowErrors()
                }
            },
            textStyle = MaterialTheme.typography.body2,
            label = {
                CompositionLocalProvider(LocalContentAlpha provides
ContentAlpha.medium) {
                    Text(
                        text = label,
                        style = MaterialTheme.typography.body2
                    )
                }
            },
            trailingIcon = {
                if (showPassword.value) {
                    IconButton(onClick = { showPassword.value = false }) {
                        Icon(
                            imageVector = Icons.Filled.Visibility,
                            contentDescription = stringResource(id =
R.string.hide_password)
                        )
```

```
                }
            } else {
                IconButton(onClick = { showPassword.value = true }) {
                    Icon(
                        imageVector = Icons.Filled.VisibilityOff,
                        contentDescription = stringResource(id = R.string.show_password)
                    )
                }
            }
        },
        visualTransformation = if (showPassword.value) {
            VisualTransformation.None
        } else {
            PasswordVisualTransformation()
        },
        isError = passwordState.showErrors(),
        keyboardOptions = KeyboardOptions.Default.copy(imeAction = imeAction),
        keyboardActions = KeyboardActions(
            onDone = {
                onImeAction()
            }
        )
```

```
    )

    passwordState.getError()?.let { error -> TextFieldError(textError = error) }
}
@Composable
fun TextFieldError(textError: String) {
    Row(modifier = Modifier.fillMaxWidth()) {
        Spacer(modifier = Modifier.width(16.dp))
        Text(
            text = textError,
            modifier = Modifier.fillMaxWidth(),
            style = LocalTextStyle.current.copy(color = MaterialTheme.colors.error)
        )
    }
}
@Composable
fun OrSignInAsGuest(
    onSignedInAsGuest: () -> Unit,
    modifier: Modifier = Modifier
) {
    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally
```

```
        ) {
            Surface {
                CompositionLocalProvider(LocalContentAlpha provides
ContentAlpha.medium) {
                    Text(
                        text = stringResource(id = R.string.or),
                        style = MaterialTheme.typography.subtitle2,
                        modifier = Modifier.paddingFromBaseline(top = 25.dp)
                    )
                }
            }
            OutlinedButton(
                onClick = onSignedInAsGuest,
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(top = 20.dp, bottom = 24.dp)
            ) {
                Text(text = stringResource(id = R.string.sign_in_guest))
            }
        }
    }
    @Preview
    @Composable
```

```
fun SignInSignUpScreenPreview() {
    SignInSignUpScreen(
        onSignedInAsGuest = {},
        content = {}
    )
}
class SignInViewModel(private val userRepository: UserRepository) : ViewModel() {
    private val _navigateTo = MutableLiveData<Event<Screen>>()
    val navigateTo: LiveData<Event<Screen>>
        get() = _navigateTo
    fun signIn(email: String, password: String) {
        userRepository.signIn(email, password)
        _navigateTo.value = Event(Survey)
    }
    fun signInAsGuest() {
        userRepository.signInAsGuest()
        _navigateTo.value = Event(Survey)
    }
    fun signUp() {
        _navigateTo.value = Event(SignUp)
    }
}
```

```
class SignInViewModelFactory : ViewModelProvider.Factory {
    @Suppress("UNCHECKED_CAST")
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(SignInViewModel::class.java)) {
            return SignInViewModel(UserRepository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
class SignUpFragment : Fragment() {
    private val viewModel: SignUpViewModel by viewModels { SignUpViewModelFactory() }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        viewModel.navigateTo.observe(viewLifecycleOwner) { navigateToEvent ->
            navigateToEvent.getContentIfNotHandled()?.let { navigateTo ->
                navigate(navigateTo, Screen.SignUp)
            }
        }
```

```
        return ComposeView(requireContext()).apply {
            // In order for savedState to work, the same ID needs to be used for all
instances.
            id = R.id.sign_up_fragment

            layoutParams = ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.MATCH_PARENT
            )
            setContent {
                JetsurveyTheme {
                    SignUp(
                        onNavigationEvent = { event ->
                            when (event) {
                                is SignUpEvent.SignUp -> {
                                    viewModel.signUp(event.email, event.password)
                                }
                                SignUpEvent.SignIn -> {
                                    viewModel.signIn()
                                }
                                SignUpEvent.SignInAsGuest -> {
                                    viewModel.signInAsGuest()
```

```
                        }
                        SignUpEvent.NavigateBack -> {
                            activity?.onBackPressedDispatcher?.onBackPressed()
                        }
                    }
                }
            )
        }
    }
}
sealed class SignUpEvent {
    object SignIn : SignUpEvent()
    data class SignUp(val email: String, val password: String) : SignUpEvent()
    object SignInAsGuest : SignUpEvent()
    object NavigateBack : SignUpEvent()
}
@Composable
fun SignUp(onNavigationEvent: (SignUpEvent) -> Unit) {
    Scaffold(
        topBar = {
            SignInSignUpTopAppBar(
```

```
                topAppBarText = stringResource(id = R.string.create_account),
                onBackPressed = { onNavigationEvent(SignUpEvent.NavigateBack)
}
            )
        },
        content = {
            SignInSignUpScreen(
                onSignedInAsGuest = { onNavigationEvent(SignUpEvent.SignInAsGuest) },
                modifier = Modifier.supportWideScreen()
            ) {
                Column {
                    SignUpContent(
                        onSignUpSubmitted = { email, password ->
                            onNavigationEvent(SignUpEvent.SignUp(email, password))
                        }
                    )
                }
            }
        }
    )
}
@Composable
fun SignUpContent(
```

```
        onSignUpSubmitted: (email: String, password: String) -> Unit,
    ) {
        Column(modifier = Modifier.fillMaxWidth()) {
            val passwordFocusRequest = remember { FocusRequester() }
            val confirmationPasswordFocusRequest = remember { FocusRequester() }
            val emailState = remember { EmailState() }
            Email(emailState, onImeAction = { passwordFocusRequest.requestFocus()
})
            Spacer(modifier = Modifier.height(16.dp))
            val passwordState = remember { PasswordState() }
            Password(
                label = stringResource(id = R.string.password),
                passwordState = passwordState,
                imeAction = ImeAction.Next,
                onImeAction = { confirmationPasswordFocusRequest.requestFocus() },
                modifier = Modifier.focusRequester(passwordFocusRequest)
            )
            Spacer(modifier = Modifier.height(16.dp))
            val confirmPasswordState = remember {
ConfirmPasswordState(passwordState = passwordState) }
            Password(
                label = stringResource(id = R.string.confirm_password),
                passwordState = confirmPasswordState,
```

```
            onImeAction = { onSignUpSubmitted(emailState.text,
passwordState.text) },
            modifier =
Modifier.focusRequester(confirmationPasswordFocusRequest)
        )
        Spacer(modifier = Modifier.height(16.dp))
        CompositionLocalProvider(LocalContentAlpha provides
ContentAlpha.medium) {
            Text(
                text = stringResource(id = R.string.terms_and_conditions),
                style = MaterialTheme.typography.caption
            )
        }
        Spacer(modifier = Modifier.height(16.dp))
        Button(
            onClick = { onSignUpSubmitted(emailState.text, passwordState.text) },
            modifier = Modifier.fillMaxWidth(),
            enabled = emailState.isValid &&
                passwordState.isValid && confirmPasswordState.isValid
        ) {
            Text(text = stringResource(id = R.string.create_account))
        }
    }
}
```

```
@Preview(widthDp = 1024)
@Composable
fun SignUpPreview() {
    JetsurveyTheme {
        SignUp {}
    }
}
class SignUpViewModel(private val userRepository: UserRepository) : ViewModel() {
    private val _navigateTo = MutableLiveData<Event<Screen>>()
    val navigateTo: LiveData<Event<Screen>>
        get() = _navigateTo
    fun signUp(email: String, password: String) {
        userRepository.signUp(email, password)
        _navigateTo.value = Event(Survey)
    }
    fun signInAsGuest() {
        userRepository.signInAsGuest()
        _navigateTo.value = Event(Survey)
    }
    fun signIn() {
        _navigateTo.value = Event(SignIn)
    }
```

```
}
class SignUpViewModelFactory : ViewModelProvider.Factory {
    @Suppress("UNCHECKED_CAST")
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(SignUpViewModel::class.java)) {
            return SignUpViewModel(UserRepository) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}
==========
open class TextFieldState(
    private val validator: (String) -> Boolean = { true },
    private val errorFor: (String) -> String = { "" }
) {
    var text: String by mutableStateOf("")
    // was the TextField ever focused
    var isFocusedDirty: Boolean by mutableStateOf(false)
    var isFocused: Boolean by mutableStateOf(false)
    private var displayErrors: Boolean by mutableStateOf(false)

    open val isValid: Boolean
        get() = validator(text)
```

```
    fun onFocusChange(focused: Boolean) {
        isFocused = focused
        if (focused) isFocusedDirty = true
    }
    fun enableShowErrors() {
        // only show errors if the text was at least once focused
        if (isFocusedDirty) {
            displayErrors = true
        }
    }
    fun showErrors() = !isValid && displayErrors
    open fun getError(): String? {
        return if (showErrors()) {
            errorFor(text)
        } else {
            null
        }
    }
}
sealed class User {
    @Immutable
    data class LoggedInUser(val email: String) : User()
    object GuestUser : User()
```

```
    object NoUserLoggedIn : User()
}
object UserRepository {
    private var _user: User = User.NoUserLoggedIn
    val user: User
        get() = _user
    @Suppress("UNUSED_PARAMETER")
    fun signIn(email: String, password: String) {
        _user = User.LoggedInUser(email)
    }
    @Suppress("UNUSED_PARAMETER")
    fun signUp(email: String, password: String) {
        _user = User.LoggedInUser(email)
    }
    fun signInAsGuest() {
        _user = User.GuestUser
    }
    fun isKnownUserEmail(email: String): Boolean {
        // if the email contains "sign up" we consider it unknown
        return !email.contains("signup")
    }
}
```

```
enum class SurveyActionType { PICK_DATE, TAKE_PHOTO, SELECT_CONTACT }

sealed class SurveyActionResult {

    data class Date(val date: String) : SurveyActionResult()

    data class Photo(val uri: Uri) : SurveyActionResult()

    data class Contact(val contact: String) : SurveyActionResult()

}

sealed class PossibleAnswer {

    data class SingleChoice(val optionsStringRes: List<Int>) : PossibleAnswer()

    data class SingleChoiceIcon(val optionsStringIconRes: List<Pair<Int, Int>>) : PossibleAnswer()

    data class MultipleChoice(val optionsStringRes: List<Int>) : PossibleAnswer()

    data class MultipleChoiceIcon(val optionsStringIconRes: List<Pair<Int, Int>>) : PossibleAnswer()

    data class Action(

        @StringRes val label: Int,

        val actionType: SurveyActionType

    ) : PossibleAnswer()


    data class Slider(

        val range: ClosedFloatingPointRange<Float>,

        val steps: Int,

        @StringRes val startText: Int,
```

```
            @StringRes val endText: Int,
            @StringRes val neutralText: Int,
            val defaultValue: Float = 5.5f
        ) : PossibleAnswer()
    }
    sealed class Answer<T : PossibleAnswer> {
        object PermissionsDenied : Answer<Nothing>()
        data class SingleChoice(@StringRes val answer: Int) :
Answer<PossibleAnswer.SingleChoice>()
        data class MultipleChoice(val answersStringRes: Set<Int>) :
            Answer<PossibleAnswer.MultipleChoice>()

        data class Action(val result: SurveyActionResult) :
Answer<PossibleAnswer.Action>()
        data class Slider(val answerValue: Float) : Answer<PossibleAnswer.Slider>()
    }
    fun Answer.MultipleChoice.withAnswerSelected(
        @StringRes answer: Int,
        selected: Boolean
    ): Answer.MultipleChoice {
        val newStringRes = answersStringRes.toMutableSet()
        if (!selected) {
            newStringRes.remove(answer)
        } else {
```

```
            newStringRes.add(answer)
        }
        return Answer.MultipleChoice(newStringRes)
    }
    class SurveyFragment : Fragment() {
        private val viewModel: SurveyViewModel by viewModels {
SurveyViewModelFactory(PhotoUriManager(requireContext().applicationContext))
        }
        private val takePicture = registerForActivityResult(TakePicture()) {
photoSaved ->
            if (photoSaved) {
                viewModel.onImageSaved()
            }
        }
        override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
        ): View {
            return ComposeView(requireContext()).apply {
                // In order for savedState to work, the same ID needs to be used for all
instances.
                id = R.id.sign_in_fragment
```

```
            layoutParams = ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.MATCH_PARENT
            )
            setContent {
                JetsurveyTheme {
                    viewModel.uiState.observeAsState().value?.let { surveyState ->
                        when (surveyState) {
                            is SurveyState.Questions -> SurveyQuestionsScreen(
                                questions = surveyState,
                                shouldAskPermissions = viewModel.askForPermissions,
                                onAction = { id, action -> handleSurveyAction(id, action) },
                                onDoNotAskForPermissions = {
viewModel.doNotAskForPermissions() },
                                onDonePressed = { viewModel.computeResult(surveyState)
},
                                onBackPressed = {
                                    activity?.onBackPressedDispatcher?.onBackPressed()
                                },
                                openSettings = {
                                    activity?.startActivity(
                                        Intent(

Settings.ACTION_APPLICATION_DETAILS_SETTINGS,
```

```
                                Uri.fromParts("package", context.packageName,
null)
                            )
                        )
                    }
                )
                is SurveyState.Result -> SurveyResultScreen(
                    result = surveyState,
                    onDonePressed = {
                        activity?.onBackPressedDispatcher?.onBackPressed()
                    }
                )
            }
        }
    }
}

private fun handleSurveyAction(questionId: Int, actionType:
SurveyActionType) {
    when (actionType) {
        SurveyActionType.PICK_DATE -> showDatePicker(questionId)
        SurveyActionType.TAKE_PHOTO -> takeAPhoto()
```

```
            SurveyActionType.SELECT_CONTACT -> selectContact(questionId)
        }
    }

    private fun showDatePicker(questionId: Int) {
        val date = viewModel.getCurrentDate(questionId = questionId)
        val picker = MaterialDatePicker.Builder.datePicker()
            .setSelection(date)
            .build()
        activity?.let {
            picker.show(it.supportFragmentManager, picker.toString())
            picker.addOnPositiveButtonClickListener {
                viewModel.onDatePicked(questionId, picker.selection)
            }
        }
    }

    private fun takeAPhoto() {
        takePicture.launch(viewModel.getUriToSaveImage())
    }

    @Suppress("UNUSED_PARAMETER")
    private fun selectContact(questionId: Int) {
```

```
        // TODO: unsupported for now
    }
}
@Composable
fun SurveyQuestionsScreen(
    questions: SurveyState.Questions,
    shouldAskPermissions: Boolean,
    onDoNotAskForPermissions: () -> Unit,
    onAction: (Int, SurveyActionType) -> Unit,
    onDonePressed: () -> Unit,
    onBackPressed: () -> Unit,
    openSettings: () -> Unit
) {
    val questionState = remember(questions.currentQuestionIndex) {
        questions.questionsState[questions.currentQuestionIndex]
    }

    Surface(modifier = Modifier.supportWideScreen()) {
        Scaffold(
            topBar = {
                SurveyTopAppBar(
                    questionIndex = questionState.questionIndex,
                    totalQuestionsCount = questionState.totalQuestionsCount,
```

```
        onBackPressed = onBackPressed
    )
},
content = { innerPadding ->
    Question(
        question = questionState.question,
        answer = questionState.answer,
        shouldAskPermissions = shouldAskPermissions,
        onAnswer = {
            if (it !is Answer.PermissionsDenied) {
                questionState.answer = it
            }
            questionState.enableNext = true
        },
        onAction = onAction,
        openSettings = openSettings,
        onDoNotAskForPermissions = onDoNotAskForPermissions,
        modifier = Modifier
            .fillMaxSize()
            .padding(innerPadding)
    )
},
bottomBar = {
```

```
                SurveyBottomBar(
                    questionState = questionState,
                    onPreviousPressed = { questions.currentQuestionIndex-- },
                    onNextPressed = { questions.currentQuestionIndex++ },
                    onDonePressed = onDonePressed
                )
            }
        )
    }
}
@Composable
fun SurveyResultScreen(
    result: SurveyState.Result,
    onDonePressed: () -> Unit
) {
    Surface(modifier = Modifier.supportWideScreen()) {
        Scaffold(
            content = { innerPadding ->
                val modifier = Modifier.padding(innerPadding)
                SurveyResult(result = result, modifier = modifier)
            },
            bottomBar = {
                OutlinedButton(
```

```
                onClick = { onDonePressed() },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = 20.dp, vertical = 24.dp)
            ) {
                Text(text = stringResource(id = R.string.done))
            }
        }
    )
  }
}
@Composable
private fun SurveyResult(result: SurveyState.Result, modifier: Modifier = Modifier) {
    LazyColumn(modifier = modifier.fillMaxSize()) {
        item {
            Spacer(modifier = Modifier.height(44.dp))
            Text(
                text = result.surveyResult.library,
                style = MaterialTheme.typography.h3,
                modifier = Modifier.padding(horizontal = 20.dp)
            )
            Text(
```

```
                text = stringResource(
                    result.surveyResult.result,
                    result.surveyResult.library
                ),
                style = MaterialTheme.typography.subtitle1,
                modifier = Modifier.padding(20.dp)
            )
            Text(
                text = stringResource(result.surveyResult.description),
                style = MaterialTheme.typography.body1,
                modifier = Modifier.padding(horizontal = 20.dp)
            )
        }
    }
}
@Composable
private fun TopAppBarTitle(
    questionIndex: Int,
    totalQuestionsCount: Int,
    modifier: Modifier = Modifier
) {
    val indexStyle = MaterialTheme.typography.caption.toSpanStyle().copy(
        fontWeight = FontWeight.Bold
```

```
    )
    val totalStyle = MaterialTheme.typography.caption.toSpanStyle()
    val text = buildAnnotatedString {
        withStyle(style = indexStyle) {
            append("${questionIndex + 1}")
        }
        withStyle(style = totalStyle) {
            append(stringResource(R.string.question_count, totalQuestionsCount))
        }
    }
    Text(
        text = text,
        style = MaterialTheme.typography.caption,
        modifier = modifier
    )
}

@Composable
private fun SurveyTopAppBar(
    questionIndex: Int,
    totalQuestionsCount: Int,
    onBackPressed: () -> Unit
) {
```

```
Column(modifier = Modifier.fillMaxWidth()) {
    Box(modifier = Modifier.fillMaxWidth()) {
        TopAppBarTitle(
            questionIndex = questionIndex,
            totalQuestionsCount = totalQuestionsCount,
            modifier = Modifier
                .padding(vertical = 20.dp)
                .align(Alignment.Center)
        )

        CompositionLocalProvider(LocalContentAlpha provides
ContentAlpha.medium) {
            IconButton(
                onClick = onBackPressed,
                modifier = Modifier
                    .padding(horizontal = 20.dp, vertical = 20.dp)
                    .fillMaxWidth()
            ) {
                Icon(
                    Icons.Filled.Close,
                    contentDescription = stringResource(id = R.string.close),
                    modifier = Modifier.align(Alignment.CenterEnd)
                )
```

```
            }
        }
    }
    val animatedProgress by animateFloatAsState(
        targetValue = (questionIndex + 1) / totalQuestionsCount.toFloat(),
        animationSpec = ProgressIndicatorDefaults.ProgressAnimationSpec
    )
    LinearProgressIndicator(
        progress = animatedProgress,
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 20.dp),
        backgroundColor = MaterialTheme.colors.progressIndicatorBackground
    )
  }
}

@Composable
private fun SurveyBottomBar(
  questionState: QuestionState,
  onPreviousPressed: () -> Unit,
  onNextPressed: () -> Unit,
  onDonePressed: () -> Unit
```

```
        ) {
            Surface(
                elevation = 7.dp,
                modifier = Modifier.fillMaxWidth() // .border(1.dp, MaterialTheme.colors.primary)
            ) {

                Row(
                    modifier = Modifier
                        .fillMaxWidth()
                        .padding(horizontal = 16.dp, vertical = 20.dp)
                ) {
                    if (questionState.showPrevious) {
                        OutlinedButton(
                            modifier = Modifier
                                .weight(1f)
                                .height(48.dp),
                            onClick = onPreviousPressed
                        ) {
                            Text(text = stringResource(id = R.string.previous))
                        }
                        Spacer(modifier = Modifier.width(16.dp))
                    }
```

```
        if (questionState.showDone) {
            Button(
                modifier = Modifier
                    .weight(1f)
                    .height(48.dp),
                onClick = onDonePressed,
                enabled = questionState.enableNext
            ) {
                Text(text = stringResource(id = R.string.done))
            }
        } else {
            Button(
                modifier = Modifier
                    .weight(1f)
                    .height(48.dp),
                onClick = onNextPressed,
                enabled = questionState.enableNext
            ) {
                Text(text = stringResource(id = R.string.next))
            }
        }
    }
}
```

```
}
@Stable
class QuestionState(
    val question: Question,
    val questionIndex: Int,
    val totalQuestionsCount: Int,
    val showPrevious: Boolean,
    val showDone: Boolean
) {
    var enableNext by mutableStateOf(false)
    var answer by mutableStateOf<Answer<*>?>(null)
}

sealed class SurveyState {
    data class Questions(
        @StringRes val surveyTitle: Int,
        val questionsState: List<QuestionState>
    ) : SurveyState() {
        var currentQuestionIndex by mutableStateOf(0)
    }

    data class Result(
        @StringRes val surveyTitle: Int,
```

```
        val surveyResult: SurveyResult
    ) : SurveyState()
}
@Composable
fun JetchatAppBar(
    modifier: Modifier = Modifier,
    scrollBehavior: TopAppBarScrollBehavior? = null,
    onNavIconPressed: () -> Unit = { },
    title: @Composable () -> Unit,
    actions: @Composable RowScope.() -> Unit = {}
) {
    val backgroundColors = TopAppBarDefaults.centerAlignedTopAppBarColors()
    val backgroundColor = backgroundColors.containerColor(
        scrollFraction = scrollBehavior?.scrollFraction ?: 0f
    ).value
    val foregroundColors = TopAppBarDefaults.centerAlignedTopAppBarColors(
        containerColor = Color.Transparent,
        scrolledContainerColor = Color.Transparent
    )
    Box(modifier = Modifier.background(backgroundColor)) {
        CenterAlignedTopAppBar(
            modifier = modifier,
```

```
                actions = actions,
                title = title,
                scrollBehavior = scrollBehavior,
                colors = foregroundColors,
                navigationIcon = {
                    JetchatIcon(
                        contentDescription = stringResource(id = R.string.navigation_drawer_open),
                        modifier = Modifier
                            .size(64.dp)
                            .clickable(onClick = onNavIconPressed)
                            .padding(16.dp)
                    )
                }
            )
        }
    }
    @Preview
    @Composable
    fun JetchatAppBarPreview() {
        JetchatTheme {
            JetchatAppBar(title = { Text("Preview!") })
        }
```

```
}

@Preview
@Composable
fun JetchatAppBarPreviewDark() {
    JetchatTheme(isDarkTheme = true) {
        JetchatAppBar(title = { Text("Preview!") })
    }
}
@Composable
private fun ProfileItem(text: String, @DrawableRes profilePic: Int?, onProfileClicked: () -> Unit) {
    Row(
        modifier = Modifier
            .height(56.dp)
            .fillMaxWidth()
            .padding(horizontal = 12.dp)
            .clip(CircleShape)
            .clickable(onClick = onProfileClicked),
        verticalAlignment = CenterVertically
    ) {
        val paddingSizeModifier = Modifier
            .padding(start = 16.dp, top = 16.dp, bottom = 16.dp)
```

```
            .size(24.dp)
        if (profilePic != null) {
            Image(
                painter = painterResource(id = profilePic),
                modifier = paddingSizeModifier.then(Modifier.clip(CircleShape)),
                contentScale = ContentScale.Crop,
                contentDescription = null
            )
        } else {
            Spacer(modifier = paddingSizeModifier)
        }
        Text(
            text,
            style = MaterialTheme.typography.bodyMedium,
            color = MaterialTheme.colorScheme.onSurface,
            modifier = Modifier.padding(start = 12.dp)
        )
    }
}
@Composable
fun DividerItem(modifier: Modifier = Modifier) {
    // TODO (M3): No Divider, replace when available
    Divider(
```

```
        modifier = modifier,
        color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.12f)
    )
}
@Composable
@Preview
fun DrawerPreview() {
    JetchatTheme {
        Surface {
            Column {
                JetchatDrawer({}, {})
            }
        }
    }
}
@Composable
@Preview
fun DrawerPreviewDark() {
    JetchatTheme(isDarkTheme = true) {
        Surface {
            Column {
                JetchatDrawer({}, {})
            }
```

```
        }
    }
}
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun JetchatScaffold(
    scaffoldState: ScaffoldState = rememberScaffoldState(),
    onProfileClicked: (String) -> Unit,
    onChatClicked: (String) -> Unit,
    content: @Composable (PaddingValues) -> Unit
) {
    JetchatTheme {
        Scaffold(
            scaffoldState = scaffoldState,
            drawerContent = {
                JetchatDrawer(
                    onProfileClicked = onProfileClicked,
                    onChatClicked = onChatClicked
                )
            },
            content = content
        )
    }}
```

ДОДАТОК В

Compose додаток з комбінованим зберіганням данних

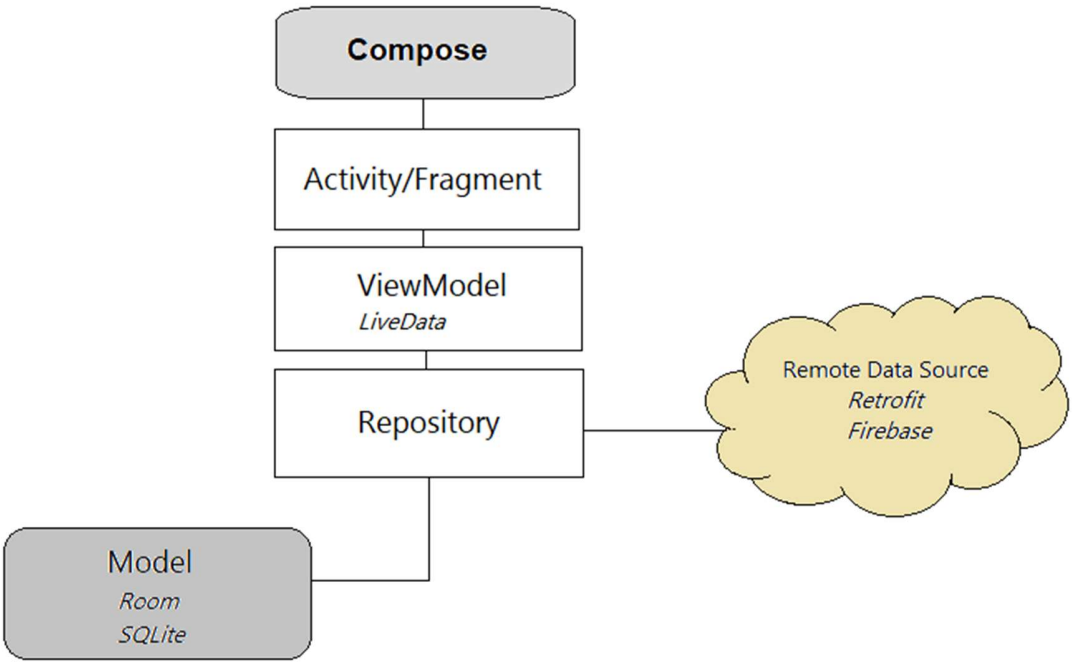

Рисунок В.1 — Compose додаток з комбінованим зберіганням данних

ДОДАТОК Г

Передача даних за допомогою розробленої моделі

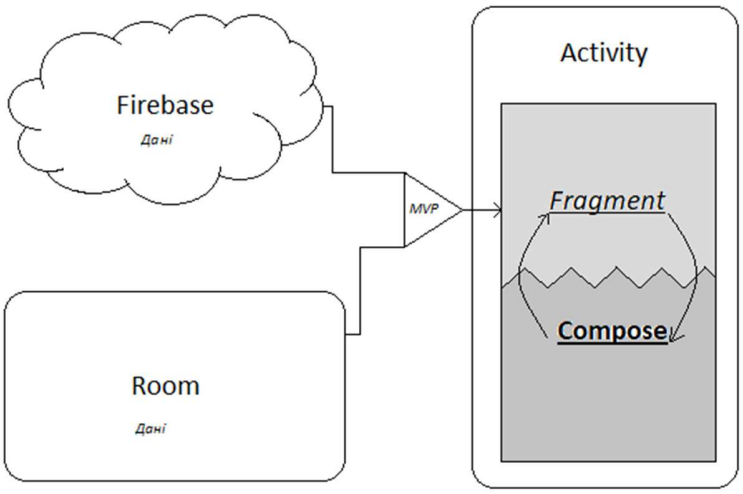

Рисунок Г.1 — Передача даних за допомогою розробленої моделі

ДОДАТОК Д

Рузультат роботи додатку

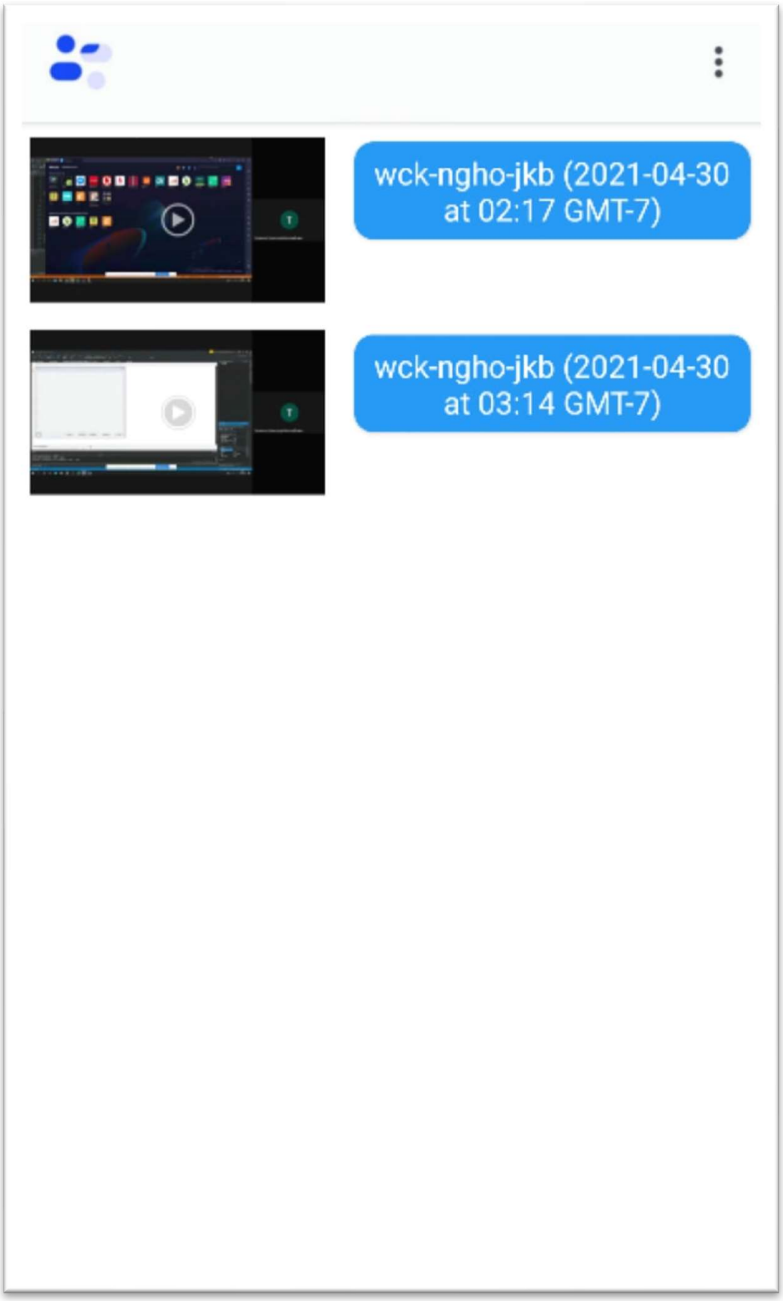

Рисунок Д.1 — Список навчальних матеріалів та лекцій розробленого додатку

Відтворення лекції в портативному режимі за допомогою exoPlayer

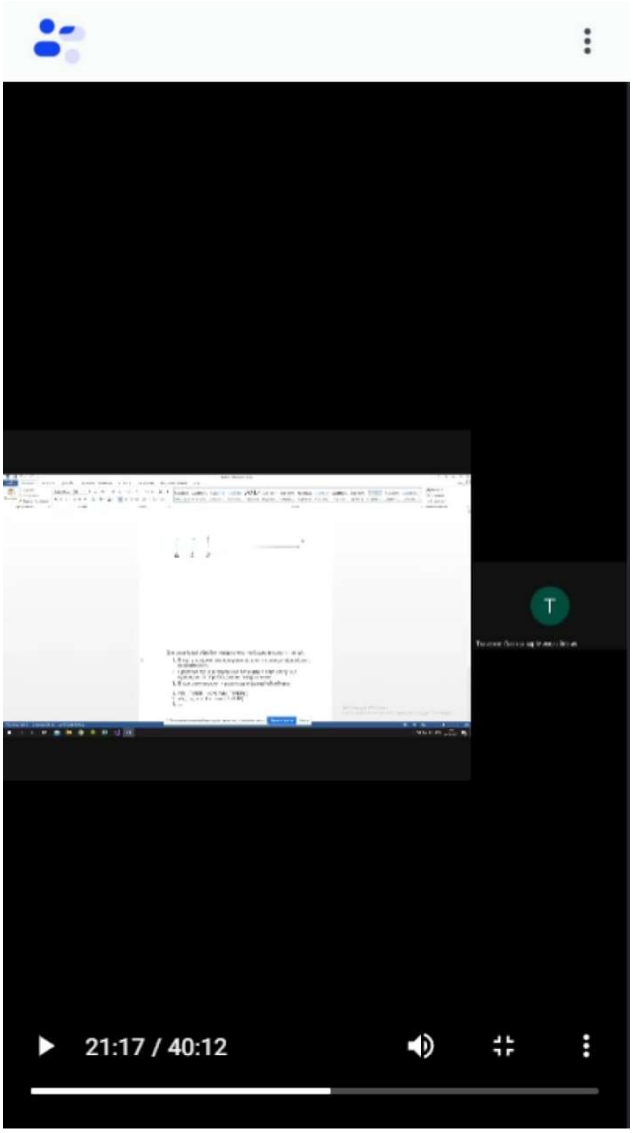

Рисунок Д.2 — Відтворення лекції за допомогою exoPlayer

ДОДАТОК Е

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:____ Хмарно-стрімінговий медіа плеєр для навчальних матеріалів та лекцій . Частина 1. Compose система_____.

Тип роботи:_____ магістерська кваліфікаційна робота _____

(кваліфікаційна роботи, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ _____кафедра обчислювальної техніки_____

(кафедра, факультет (інститут), навчальна група)

Науковий керівник _____ к.т.н., доц. Ткаченко О.М._____

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		Unicheck	
КП1			
КП2		Оригінальність	91,4
Тривога/Білі знаки	/	Схожість	8,6

Аналіз звіту подібності (відмінити подібне) ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності і відсутності самостійності її автора. Робот 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.

Заявляю, що ознайомлений(-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ _____Левицька Ю.Р.___

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_Ступінь оригінальності роботи відповідає вимогам, що висуваються до МКР

_____

_____

Особа, відповідальна за перевірку_____ _Захарченко С.М.__

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали)