ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку з елементами інклюзивного дизайну для відображення розкладу університету»

на здобуття освітнього ступеня бакалавра
за спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Хоанг Ву ФАМ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Хоанг Ву ФАМ

Керівник: Вячеслав САВІЦЬКИЙ
викладач кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Хоангу Ву Фаму

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку з елементами інклюзивного дизайну для відображення розклау університету»
керівник кваліфікаційної роботи викладач кафедри ІПЗ Вячеслав САВІЦЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки мобільного застосунку, опис інструментів для розробки мобільного застосунку.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Пошук та аналіз існуючих мобільних застосунків для відображення розкладу університету.
2. Проєктування застосунку з елементами інклюзивного дизайну для відображення розкладу університету.

3. Програмна реалізація та опис функціонування застосунку з елементами інклюзивного дизайну для відображення розкладу університету.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Екранні форми.
7. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих застосунків для відображення розкладу університету	14.03-19.03.2025	
4	Проектування застосунку з елементами інклюзивного дизайну для відображення розкладу університету	20.03-02.04.2025	
5	Програмна реалізація застосунку	03.04-21.04.2025	
6	Тестування застосунку	22.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Хоанг Ву ФАМ
(підпис)

Керівник
кваліфікаційної роботи _____ Вячеслав САВІЦЬКИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 2 табл., 18 рис., 18 джерел.

Мета роботи - покращення сприйняття розкладу занять для студентів із порушенням зору за рахунок використання принципів інклюзивного дизайну.

Об'єкт дослідження - процес відображення розкладу занять та його доступність для користувачів із порушенням зору.

Предмет дослідження - мобільний застосунок з елементами інклюзивного дизайну для відображення розкладу університету.

Короткий зміст роботи: У роботі проаналізовано сучасні тенденції та підходи до розробки до мобільних застосунків на платформі Android. Розглянуто варіанти створення застосунків, мови програмування та середовища розробки. Проведено огляд архітектурних шаблонів, що використовуються в Android-розробці. Розроблено мобільний застосунок із використанням фреймворку Jetpack Compose, що забезпечує декларативний підхід до побудови інтерфейсу. Програмно реалізовано основні функціональні можливості відповідно до визначених завдань. Проведено функціональне та модульне тестування розробленого застосунку. У процесі реалізації використано мову програмування Kotlin, архітектуру MVVM, середовище розробки Android Studio та бібліотеки Jetpack.

КЛЮЧОВІ СЛОВА*:* ANDROID, KOTLIN, МОБІЛЬНИЙ ЗАСТОСУНОК, ІНКЛЮЗИВНИЙ ДИЗАЙН.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

5G – Fifth generation

AOT – Ahead-Of-Time

API – Application Programming Interface

APK - Android Application Package

CSS - Cascading Style Sheets

Dp – Density-independent Pixels

HTML – HyperText Markup Language

JIT – Just-In-Time

JVM – Java Virtual Machine

PWA - Progressive Web Apps

SDK – Software Development Kit

UI/UX – User Interface / User Experience

VR – Virtual Reality

WCAG – Web Content Accessibility Guidelines

XML – eXtensible Markup Language

ООП – Об’єктно орієнтоване програмування

ОС – Операційна Система

ВСТУП

У сучасному світі, де технології стрімко інтегруються в усі сфери життя, освіта не є винятком. Мобільні застосунки стали невід'ємною частиною повсякденності, пропонуючи зручні рішення для комунікації, навчання та організації. Зокрема, для студентів університетів, оперативний доступ до актуального розкладу занять є критично важливим для ефективного планування навчального процесу та особистого часу.

Однак, незважаючи на загальну доступність технологій, значна частина існуючих інформаційних систем та мобільних застосунків, які надають освітні установи, часто не враховують потреби всіх категорій користувачів. Особливо це стосується студентів з особливими освітніми потребами, до яких можуть належати люди з порушеннями зору, слуху, рухового апарату, дислексією або іншими когнітивними особливостями. Для таких студентів, неадаптовані інтерфейси, дрібний шрифт, відсутність голосового супроводу або несумісність з допоміжними технологіями стають значною перешкодою на шляху до отримання рівного доступу до інформації та повноцінної участі в освітньому процесі.

Метою цієї роботи є створення мобільного застосунку, який забезпечить легкий та зручний доступ до розкладу університету для всіх студентів, мінімізуючи бар'єри та сприяючи їхній інтеграції в освітнє середовище. Це включатиме глибокий аналіз існуючих стандартів та рекомендацій з інклюзивного дизайну, проектування інтуїтивно зрозумілого інтерфейсу з урахуванням потреб користувачів з обмеженими можливостями, розробку функціоналу, що підтримує адаптивні можливості (наприклад, зміну розміру шрифту, контрастності), та тестування застосунку.

Практична цінність даної роботи полягає у розробці інструменту, який покращить щоденне життя студентів, а також стане зразком для подальших ініціатив з впровадження інклюзивного дизайну в університетських інформаційних системах. Це не лише підкреслить соціальну відповідальність освітніх закладів, але й сприятиме формуванню більш відкритого та доступного освітнього простору.

1. АНАЛІЗ ТА ХАРАКТЕРИСТИКА МОБІЛЬНИХ ЗАСТОСУНКІВ

1.1 Сучасні тенденції розвитку мобільних додатків

У сучасному цифровому середовищі мобільні додатки стали невід'ємною частиною повсякденного життя, поєднуючи розваги, продуктивність та комунікацію в одному пристрої. Сьогодні у світі налічується понад 6,5 мільярди користувачів смартфонів, а середньостатистичний американський користувач перевіряє свій телефон 262 рази на добу, проводячи в мобільних додатках по декілька годин на день. Така постійна взаємодія стимулює безперервне вдосконалення функціональності та дизайну у мобільній індустрії.

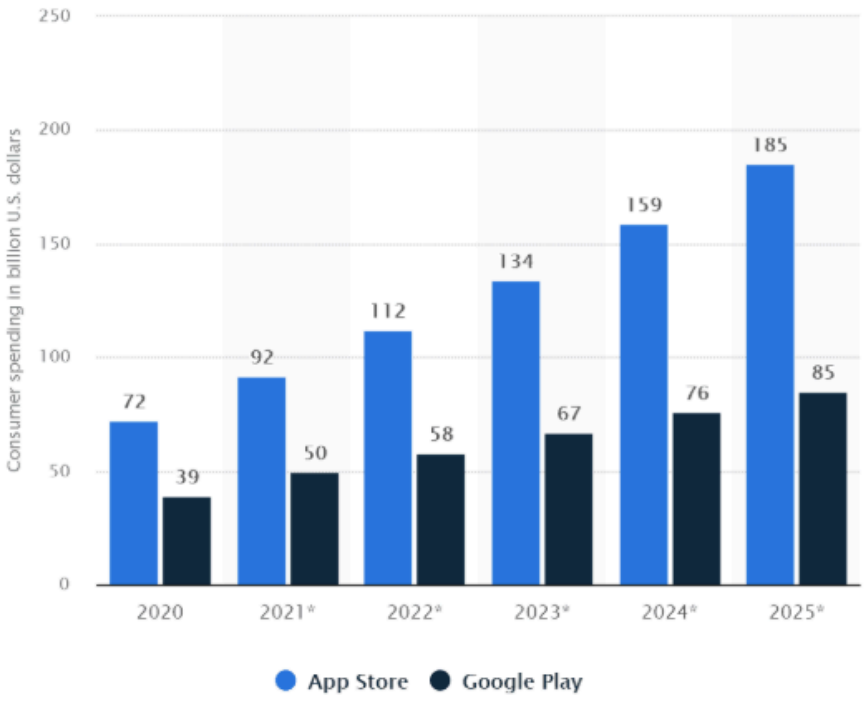


Рис. 1.1 Споживчі витрати на мобільні додатки

Згідно даними Statista, у 2022 році було завантажено близько 255 мільярдів мобільних додатків, а у 2023 році цей показник виріс до 270 мільярдів. Споживчі витрати на мобільні додатки також демонструють стабільне зростання: у 2023 році

користувачі витратили близько 201 мільярда доларів, а у 2023 році цей показник оцінюється в 235 мільярдів.

Величезний обсяг застосунків та висока залученість користувачів, поєднані зі швидким зростанням ринку, формують критичну потребу в диференціації та стратегіях утримання користувачів. Це означає, що розробники не можуть покладатися лише на базову функціональність; для успіху на сучасному ринку застосунки повинні пропонувати винятковий користувацький досвід та унікальну цінність. У міру того, як ринок мобільних застосунків продовжує дозрівати та ставати все більш конкурентоспроможним, нові застосунки, особливо в насичених нішах, повинні виділятися завдяки передовій функціональності, інтуїтивному дизайну та здатності задовольняти очікування користувачів.

Актуальність дослідження тенденцій для розробки інноваційних рішень

Постійна еволюція технологій та зростання вимог користувачів до безперебійного, захоплюючого та візуально привабливого досвіду активно переформатовують підходи до дизайну та розробки мобільних застосунків. Користувачі більше не задовольняються стандартними функціями; вони очікують, що застосунки будуть адаптивними, інтелектуальними та інтуїтивно зрозумілими. Ця динаміка ринку стимулює безперервні інновації, де інтеграція передових технологій стає не перевагою, а необхідністю.

Інновації, такі як інтеграція штучного інтелекту, доповненої реальності та покращених біометричних заходів безпеки, вже переосмислили стандарти продуктивності та функціональності мобільних застосунків. Ці технологічні прориви не лише розширюють можливості застосунків, але й підвищують планку для користувацького досвіду. Швидка технологічна еволюція створює як значні можливості для розробників, так і нові виклики.

Інтеграція 5G технологій

Технологія 5G є фундаментальним фактором, що забезпечує розвиток багатьох інших передових мобільних тенденцій, таких як доповнена та віртуальна реальність та IoT. 5G пропонує надшвидкий інтернет, значно низьку затримку та більшу пропускну здатність, що кардинально покращує продуктивність мобільних

застосунків. Ці характеристики є вирішальними для обробки даних у реальному часі та забезпечення насиченого мультимедійного досвіду, що було неможливим з попередніми поколіннями мобільних мереж.

Можливості 5G дозволяють здійснювати потокову передачу даних у реальному часі без затримок, підтримувати хмарні ігри з мінімальним лагом та забезпечувати захоплюючі AR/VR досвіди, які вимагають високої швидкості передачі даних та низької затримки. Крім того, 5G значно розширює можливості підтримки пристроїв IoT, сприяючи розвитку розумних будинків, розумних міст та підключених транспортних засобів. Швидкість 5G до 100 разів перевищує швидкість 4G, а ультранадійна комунікація з низькою затримкою (URLLC) забезпечує обробку даних у реальному часі, що є критично важливим для інтерактивних застосунків.

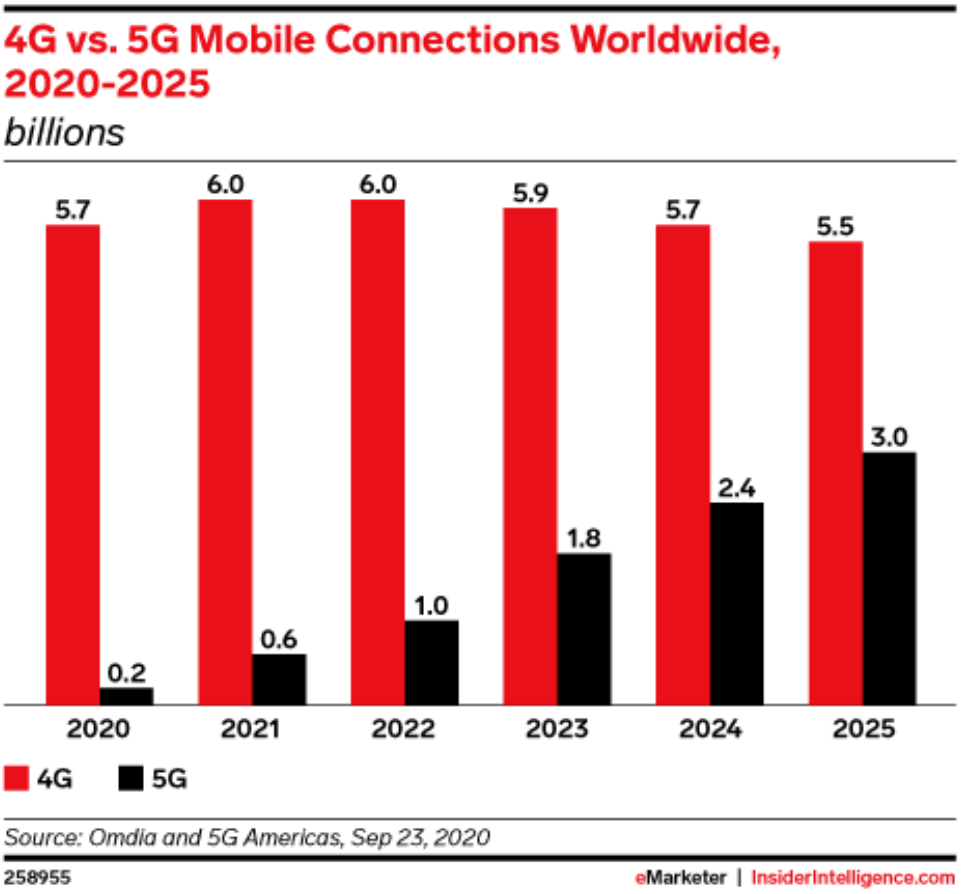


Рис 1.2 Порівняння кількості підключень до 4G та 5G мереж з 2020 р. до 2025 р.

Мінімалізм та Контент-орієнтовані Інтерфейси

Мінімалізм революціонізує дизайн мобільних застосунків, зосереджуючись на простоті та чіткості для покращення користувацького досвіду. Цей підхід передбачає пріоритетність основних елементів та усунення зайвого безладу, що робить застосунки більш інтуїтивно зрозумілими та ефективними. Мета полягає в тому, щоб зменшити когнітивне навантаження на користувача, дозволяючи йому швидше обробляти інформацію та легше знаходити потрібні функції. Чисті макети та ефективне використання вільного простору сприяють цьому, а менша кількість анімацій та візуальних відволікань покращує продуктивність застосунку, призводячи до швидшого завантаження.

1.2 Варіанти розробки мобільних додатків

У сучасному середовищі розробки мобільних додатків існує кілька основних підходів, кожен з яких має свої сильні та слабкі сторони залежно від цілей проєкту, ресурсів команди та очікувань користувачів. Нижче наводиться детальний опис варіантів розробки та їхніх ключових характеристик.

Нативний підхід передбачає створення окремих версій додатку для кожної платформи з використанням рідних мов та SDK. Для iOS це насамперед Swift або Objective-C у сердовищі Xcode, для Android - Kotlin чи Java в Android Studio. Такий підхід забезпечує максимальну продуктивність, повний доступ до можливостей операційної системи та найкращий досвід користувача завдяки дотриманню вимог платформи. Нативний код зазвичай дає змогу швидше реагувати на апаратні події та використовувати новітні фічі ОС одразу після їхнього релізу. Основним недоліком є необхідність підтримувати дві окремі кодові бази, що може суттєво збільшувати витрати часу й коштів на розробку та подальше обслуговування.

Щоб уникнути подвійного написання коду, дедалі популярнішими стають рішення, які дозволяють створювати єдину кодову базу для iOS і Android. Найвідоміші серед них - Flutter від Google (мовою Dart) та React Native від

Facebook/Meta (на JavaScript/TypeScript). Flutter використовує власний движок рендерингу, що гарантує схожість інтерфейсу на різних пристроях, тоді як React Native переводить JavaScript-компоненти в рідні елементи платформи. Обидва фреймворки підтримують «гарячий перезапуск» , що значно пришвидшує ітерації розробки, а також мають розвинуті екосистеми плагінів для роботи з апаратурою пристрою. Серед недоліків - певні обмеження в доступі до новітніх API та необхідність час від часу писати «нативні мости» для розширених функцій.

Підхід, що використовує веб-технології (HTML, CSS, JavaScript) з обгорткою в нативному shell, реалізовану за допомогою таких рішень, як Apache, Cordova чи Ionic. Фактично логіка та інтерфейс рендеряться через вбудований веб-вигляд (WebView), а доступ до апаратних можливостей здійснюється через плагіни. Така архітектура дозволяє розробникам із досвідом у веб-проєктах швидко перенести свої знання на мобільні платформи. Головними перевагами є мінімальні терміни виходу на ринок і можливість легко оновлювати контент. Однак продуктивність таких додатків найчастіше поступається нативним та кросплатформним рішенням, особливо в анімаціях та роботі зі складною графікою.

PWA комбінує переваги веб-додатків та мобільного досвіду: вони запускаються в браузері, але можуть встановлюватися на домашній екран, працювати офлайн і отримувати push-сповіщення. Оскільки PWA не потребують проходити модерацію у маркетах, оновлення доступні миттєво, а сам підхід суттєво знижує витрати на розробку, адже достатньо підтримувати одну веб-версію. Проте PWA обмежені у роботі з низькорівневими функціями пристрою, тож для випадків із високими вимогами до апаратної інтеграції вони не підходять.

Для швидкого прототипування або проєктів із невеликими бюджетами дедалі частіше застосовують платформи типу AppGyver, Adaalo, Bubble, які дозволяють створювати мобільні інтерфейси й бізнес логіку через візуальні конструкції без глибоких знань програмування. Такі інструменти особливо корисні для стартапів та внутрішніх корпоративних рішень, де критично важливо

перевірити концепцію якнайшвидше. Водночас обмеження цих систем полягають у вузькому наборі підтримуваних інтеграцій і в тому, що складні алгоритми або унікальні UI-завдання реалізувати складно або неможливо без «вилітань» аз середовища платформ.

Вибір оптимального підходу залежить від кількох чинників: цільової аудиторії (користувачів iOS, Android чи обох), необхідного рівня продуктивності та доступу до фічпакету, пристрою наявних ресурсів у команді (час, бюджет, кваліфікація розробників) та потреб у швидкому виведенні MVP на ринок. Нативна розробка залишається еталоном у проєктах з високими вимогами до продуктивності та UX, кросплатформені фреймворки об’єднують зручність і гнучкість, а веб– і low-code рішення забезпечують максимальну швидкість і мінімальні витрати на старт. Успіх продукту багато в чому залежить від того, наскільки чітко поставлені бізнес-цілі та наскільки обраний варіант відповідає їхнім вимогам.

1.3 Мови програмування для розробки на Android

Розробка мобільних застосунків для платформи Android вимагає глибокого розуміння архітектури операційної системи, принципів взаємодії з користувачем, а також володіння відповідними інструментами та мовами програмування. Вибір мови є одним з ключових рішень на етапі проєктування, оскільки він суттєво впливає на ефективність розробки, продуктивність застосунку, легкість підтримки та масштабованість проєкту. В екосистемі Android домінуючими мовами є Java та Kotlin, кожна з яких має свої архітектурні особливості та надає розробникам унікальний набір переваг та недоліків. Історично Java була основною для розробки Android-застосунків, проте останніми роками Kotlin набуває більшої популярності, отримавши офіційну підтримку від Google. Крім того, існують альтернативні підходи, такі як кросплатформна розробка, що дозволяє використовувати інші мови, але для нативної розробки на Android фокус залишається на двох.

Для розробки застосунків використовується офіційний набір інструментів Android SDK, який є ключовим компонентом для створення програмного забезпечення. Цей комплект інструментів дозволяє компілювати програмний код, а також файли даних та ресурсів, в єдиний архівний файл формату APK або сучасніший Android App Bundle. Ці формати є стандартними для розповсюдження Android-застосунків.

Java

Мова програмування Java, розроблена компанією Sun Microsystems у 1991 році, щовидко здобула статус однієї з найпопулярніших мов та найпотужніших мов у світі. Вона має одну з найбільших спільнот підтримки, що сприяє її стабільності та поширеності. Java є об’єктно-орієнтованою та класово-орієнтованою мовою, спроєктованою з мінімальною кількістю залежностей від реалізації, що забезпечує її високу портативність.

Фундаментальною особливістю Java є принцип “Write Once, Run Anywhere”, що дозволяє розробникам створювати код, який може виконуватися на різних платформах без необхідності модифікації. Ця універсальність зробила Java надзвичайно гнучкою, дозволяючи її застосування у широкому спектрі галузей - від банківських систем до відеоігор. Вона довгий час була основною мовою для розробки Android застосунків, і значна кількість популярних мобільних програм були написані з її використанням.

Переваги Java

Однією з ключових переваг є її об’єктно-орієнтована парадигма, яка сприяє модульному дизайну, повторному використанню коду та легшій підтримці проєктів. ООП допомагає запобігати помилкам, інкапсулюючи інформацію в об’єктах.

Масивна спільнота розробників Java є ще однією вагомою перевагою. Ця велика та активна спільнота забезпечує постійну підтримку, швидкі відповіді на запитання та доступ до величезної кількості спільних інструментів та ресурсів. Наявність великого пулу кваліфікованих розробників також спрощує пошук та

наймання відповідних фахівців, що зможе заощадити час та кошта в довгостроковій перспективі.

Java маж багатий набір API та відкритих бібліотек, які значно прискорюють розробку застосунків та підвищують їхню якість. Мова також відома своїми надійними функціями безпеки, що є критично важливим для захисту конфіденційної інформації та запобігання поширеним вразливостям.

Масштабованість та продуктивність є ще одними сильними сторонами Java. Завдяки підтримці багатопоточності, Java може ефективно обробляти тисячі запитів, що є важливим для високопродуктивних корпоративних застосунків. Хоча існують певні накладні витрати на JVM, інструменти, такі як JIT-компіляція та налаштування JVM, можуть значно покращити продуктивність, наближаючи її до низькорівневих мов. Автоматичне керування пам’яттю за допомогою збирача сміття спрощує кодування, автоматично звільняючи невикористану пам’ять та зменшуючи риз витоків пам’яті, що є поширеною проблемою в таких мовах, як C або C++.

Java також вирізняється своєю платформонезалежністю, дозволяючи коду працювати на різних пристроях без змін. Це забезпечує універсальність та портативність для компаній. Крім того, вона має вражаючі можливості інтеграції, оскільки більшість хостинг-провайдерів підтримують Java. Вона також є відносно недорогою в обслуговуванні, оскільки не вимагає залежності від конкретної інфраструктури та можу працювати на будь-якому типі машини.

Недоліки Java

Одним з основних недоліків є накладні витрати на віртуальну машину Java, що може призводити до вищого споживання пам’яті та більш повільного часу запуску порівняно з деякими іншими мовами.

Синтаксис Java часто вважається багатослівним, що може уповільнять розробку, особливо в порівнянні з більш лаконічними мовами, такими як Kotlin. Це може призводити до написання більшої кількості шаблонного коду для досягнення тієї ж функціональності. Хоча Java є потужною, вона може бути менш придатною для швидкого прототипування або скриптингу.

Існують також обмеження щодо низькорівневого програмування та складності керування пам’яттю для високопродуктивних застосунків, незважаючи на автоматичний збирач сміття. Для дуже чутливих до затримок застосунків, таких як високочастотна торгівля або ігрові рушії, накладні витрати на JVM можуть бути небажаними.

Важливо також зазначити, що Java може мати проблеми з фрагментацією, а також з підтримкою застарілих шаблонів або бібліотек, що може призвести до накопичення технічного боргу та уповільнення інновацій. Хоча Java є безкоштовною для більшості випадків, комерційне використання Oracle JDK може вимагати платної ліцензії, що може бути недоліком для деяких підприємств.

Kotlin

Kotlin - сучасна, статично типізована мова програмування, розроблена компанією JetBrains, яка вперше була випущена у 2011 році. Вона працює на віртуальній машині Java і є повністю сумісною з Java, що означає можливість взаємного виклику коду, написаного на обох мовах. Це робить Kotlin ідеальним вибором для розробників, які працюють в екосистемі Java і бажають скористатися перевагами сучасних функцій, не відмовляючись від існуючої кодової бази.

У травні 2019 року Google офіційно оголосив Kotlin своє пріоритетною мовою для розробників Android-застосунків. Це рішення мало значний вплив на індустрію, оскільки воно не просто рекомендувало Kotlin, а чітко вказало майбутній напрямок розвитку платформи. Така стратегічна перевага, надана Google, є критичним фактором для довгострокової життєздатності і стабільності розроблених застосунків. Це означає, що нові функції, інтеграція інструментів, офіційна документація та основні бібліотеки будуть, ймовірно, оптимізовані або першочергово розроблені для Kotlin.

Хоча Kotlin спочатку був орієнтований на розробку під Android, він швидко здобув популярність в інших галузях, таких як бекенд-розробка, веб-розробка та навіть наука про дані. Його кросплатформні можливості, особливо з Kotlin Multiplatform, дозволяють розробникам писати спільний код, який може працювати на Android, iOS, веб- та десктопних застосунках. Ця універсальність

зробила Kotlin однією з найбільших динамічно зростаючих мов програмування десятиліття.

Переваги Koltin

Однією з найважливіших є її лаконічний та виразний синтаксис, який дозволяє писати менше коду для досягнення тієї ж функціональності порівняно з Java. Це не тільки максимізує продуктивність команд, але й робить код більш читабельним та легким для підтримки.

Ключовою особливістю є вбудована безпека від нульових посилань (Null Safety). На відміну від Java, де NullPointerException є поширеною проблемою під час виконання, Kotlin вимагає явного визначення, чи може змінна бути нульовою. Це значно зменшує ймовірність помилок під час виконання та підвищує стабільність застосунків.

Повна сумісність з Java є величезною перевагою. Розробники можуть використовувати Kotlin разом з Java в одному проєкті, поступово мігруючи код або використовуючи Java-бібліотеки та фреймворки без необхідності їх переписування.

Kotlin також відрізняється підтримкою корутин для асинхронного програмування. Корутини спрощують написанян асинхронного коду, роблячи його більш читабельним та ефективним, ніж традиційне багатопотокове програмування в Java.

Недоліки Kotlin

Незважаючи на численні переваги, Kotlin має певні недоліки, які слід враховувати. Одним з них є потенційно повільніша швидкість компіляції порівняно з Java, особливо під час чистої збірки. Хоча для інкрементальних збірок Kotlin може бути швидшим, Java все ще може домінувати в цьому аспекті.

Екосистема Kotlin, хоча і зростає, все ще є менш зрілою, ніж Java, і може мати більш обмежені ресурси та навчальні матеріали, що може призвести до труднощів у пошуку досвідчених фахівців з Kotlin. Крім тогож ,можуть виникати випадкові проблеми сумісності, хоча взаємодія з Java є загалом безкоштовною.

Flutter

Flutter - це відкрити UI-інструментарій, розроблений Google, який був вперше випущений у 2017 році. Він призначений для створення нативно скомпільованих багатоплатформних застосунків з єдиної кодової бази.

Flutter використовує мову програмування Dart, яка також розроблена Google. Dart є сучасною, об’єктно-орієнтованою мовою, що підтримує як JIT компіляцію для швидкої розробки, так і AOT компіляція для досягнення нативної продуктивності у випущених застосунках.

Ключовою особливістю Flutter є його архітектура, яка дозволяє йому “контролювати кожен піксель” на екрані. Це досягається завдяки використанню власного механізму рендерингу Skia, який дозволяє малювати інтерфейси користувача безпосередньо на полотні пристрою з високою частотою кадрів, забезпечуючи плавність та чіткість інтерфейсу на різних платформах.

Переваги Flutter

Однією з найважливіших переваг є можливість кросплатформної розробки з єдиною кодовою базою. Це дозволяє створювати застосунки для iOS, Android, вебу та десктопу, використовуючи 90% одного й того ж коду, що значно підвищує ефективність та швидкість розробки.

Функція “гарячого перезавантаження” (Hot Reload) є однією з найпопулярніших переваг Flutter. Вона дозволяє розробникам миттєво бачити зміни в коді без перезапуску всього застосунку, що значно прискорює налагодження та налаштування інтерфейсу користувача.

Flutter забезпечує високу продуктивність завдяки компіляції коду Dart в нативний машинний код для iOS та Android. Це усуває потребу в “мості” або інтерпретаційному шарі, як у деяких інших кросплатформних фреймворках, що дозволяє коду виконуватися безпосередньо та забезпечує швидкий та плавний користувацький досвід.

Фреймворк надає багатий каталог попередньо розроблених віджетів, які дозволяють створювати візуально привабливі та високоінтерактивні інтерфейси

користувача. Ці віджети відповідають конкретним рекомендаціям платформ, тому застосунок може виглядати та відчуватися як нативний на будь-якому пристрої.

Недоліки Flutter

Flutter також має свої недоліки. Одним з них є більший розмір застосунків у порівнянні з нативними. Це пов’язана з тим, що Flutter включає власний рушій рендерингу та необхідні компоненти фреймворку в APK-файл.

Хоча Flutter прагне забезпечити нативний вигляд та відчуття, іноді застосунок може не повністю відповідати естетиці iOS, оскільки Flutter використовує власний рушій рендерингу для забезпечення послідовного інтерфейсу користувача на обох платформах.

Крім того, Flutter може мати обмежену вбудовану підтримку для деяких нативних функцій пристрою або функціональних можливостей, що може вимагати залежності від сторонніх плагінів або написання платформоспецифічного коду.

React Native

React Native - відкритий мобільний фреймворк, розроблений Facebook, який був вперше представлений у 2015 році. Його основна мета - дозволити розробникам створювати нативні застосунки для кількох платформ з використанням єдиної кодової бази на JavaScript. Фреймворк базуєтсья на React, що робить його особливо привабливим для фронтенд-розробників.

React Native досягає нативного вигляд та продуктивності, використовуючи “міст”, який полегшує зв’язок між JavaScript-кодом та нативними компонентами платформи. Коли користувач ініціює подію, React Native перетворює нативну подію на JavaScript-подію через цей міст, дозволяючи React ефективно обробляти та оновлювати інтерфейс користувача, зберігаючи при цьому продуктивність, близьку до нативної.

Переваги React Native

Однією з найважливіших переваг є можливість кросплатформної розробки з єдиною кодовою базою. Це дозволяє розробникам писати код один раз і розгортати його як на Android, так і на iOS, значно скорочуючи зусилля на розробку, час виходу на ринок та витрати на обслуговування.

React Native забезпечує продуктивність, близьку до нативної. На відміну від традиційних гібридних фреймворків, він рендерить нативні компоненти замість веб-переглядів, забезпечуючи плавніший користувацький досвід та анімації.

Функція "гарячої перезавантаження" (Hot Reloading) або "швидкого оновлення" (Fast Refresh) значно прискорює процес розробки. Розробники можуть миттєво бачити зміни в коді без необхідності перекомпіляції всього застосунку.

Фреймворк відносно легкий у вивченні, особливо для розробників, які вже знайомі з JavaScript та React. Це дозволяє фронтенд-розробникам швидко адаптуватися до мобільної розробки без вивчення нової мови. Можливість інтеграції з існуючим нативним кодом є ще однією перевагою. Для функцій, критичних до продуктивності, React Native дозволяє безшовну інтеграцію з нативними модулями Java, Swift або Kotlin.

Недоліки React Native

Однією з головних недоліків є потенційні проблеми з продуктивністю, особливо для складних анімацій або обробки великих обсягів даних. Хоча для більшості застосунків продуктивність близька до нативної, для високопродуктивних або інтенсивних з точки зору ОС випадків використання React Native може бути не найкращим вибором. Це пов'язано з використанням "мосту" між JavaScript та нативними потоками, який може стати вузьким місцем, особливо при швидкому прокручуванні або виконанні складних обчислень.

Хоча React Native сприяє повторному використанню коду, складні функції (наприклад, розширені елементи керування камерою або фонова обробка) часто вимагають нативних модулів. Це означає необхідність підтримки окремих кодових баз для iOS/Android, що нівелює деякі переваги "написати один раз" та збільшує довгострокове обслуговування. Екосистема, хоча й багата, може не мати готових рішень для нішевих або передових функцій, що змушує розробників писати власні нативні модулі або чекати на підтримку спільноти.

Налагодження в React Native може бути складнішим, ніж у нативних мовах. Розробники часто стикаються з проблемами налагодження, особливо при взаємодії між JavaScript та нативним кодом через міст. Крива навчання, хоча й легша для

веб-розробників, може бути крутішою для тих, хто звик до чисто нативної розробки, оскільки вона вимагає знання JavaScript, React та концепцій мобільних мостів.

Таблиця 1.1

Порівняння мов програмування та фреймворків для Android

	Java	Kotlin	Flutter (Dart)	React Native (Java Script)
Тип розробки	Нативна	Нативна	Кросплатформна	Кросплатформна
Рік випуску	1991	2011	2017	2015
Пріоритет Google	Ні	Так	Так (для кросплатформи)	Ні
Продуктивність	Висока	Висока	Висока	Близька до нативності
Швидкість розробки	Помірна	Висока	Дуже висока	Дуже висока
Розмір застосунку	Менший	Менший	Більший (включає рушій)	Більший (включає JS-пакет)
Синтаксис	Багатослівний	Лаконічний, виразний	Лаконічний	Гнучкий
Спільнота	Дуже велика	Зростаюча	Велика	Дуже велика
Доступ до нативних API	Прямий	Прямий	Через плагіни/модулі	Через міст\нативні модулі
Сумісність з Java	-	Повна	Обмежена (через плагіни)	Обмежена (через міст)
Безпека від NPE	Ні	Так	Так	Ні

1.4 Середовища розробки Android застосунків

Найпопулярніші середовища розробки здебільшого базуються на платформі IntelliJ IDEA й забезпечують повний набір інструментів для кодування, емуляції та управління залежностями. Водночас існують і більше легковагові або спеціалізовані рішення, які можуть стати в нагоді для окремих задач, наприклад, розробки ігор або кросплатформенних проєктів.

Найбільш розповсюдженим та рекомендований офіційним інструментом є Android Studio, яке регулярно оновлюється разом із випуском нових версій Android SDK. Воно надає редактор XML-розмітки інтерфейсу, інтегрований емулятор пристроїв різних конфігурацій, вбудовані засоби профілювання продуктивності та глибоку підтримку Gradle для побудови проєкту. Крім того, Android Studio міє автодопонювач коду, уміння рефакторингу та систему шаблонів, які пришвидшують написання часто використовуваних фрагментів.

Іншим популярним варіантом є повна і платна версія IntelliJ IDEA Ultimate. Вона пропонує аналогічний до Android Studio функціонал із додатковими інструментами для веб- та серверної розробки, підтримкою баз даних, Docker, Kubernets та інших технологій.

Для розробників, які вже працюють у середовищі Microsoft або орієнтуються на кросплатформені фреймворки Xamarin, існує можливість створювати Android-додатки у Visual Studio. У поєднанні з плагіном Xamarin.Android ця IDE дозволяє написати код на C# із доступом до Android API, а також надає власний емулятор і налагоджувач.

У випадках, коли потрібен надлегкий редактор коду або ж розробник віддає переваги редакторам, орієнтованим на веб-технології, часто використовують Visual Studio Code із встановленими розширеннями для Java/Kotlin, Android SDK Adapter та Gradle. Такий підхід дозволяє мати в одному вікні й мобільні, й веб-проєкти, а також інтегруватися з численними плагінами - від контролю версій до Docker.

Окрім традиційних рішень є середовища, орієнтовані на конкретні типи проєктів, зокрема Unity та Unreal Engine для розробки ігорю Вони надають власні редактори сцен, системи скриптів та оптимізовані конвеєри збірки для мобільних платформ. Для бажаючих спробувати онлайн-IDE без локального встановлення доступні такі сервіси, як GitPod або Codeanywhere, де можна розгорнути Android SDK у контейнері і вести розробку прямо у браузері.

1.5 Задачі кваліфікаційної роботи

1. Провести аналіз існуючих застосунків для відображення розкладу університету.
2. Розробити архітектуру мобільного застосунку та визначити перелік вимог до нього.
3. Проєктування інтерфейсу застосунку з елементами інклюзивного дизайну.
4. Розробити мобільний застосунок.
5. Провести тестування та налагодження розробленої системи.

Функціонал:

1. Відображення розкладу занять обраної групи, викладача та аудиторії.
2. Пошук розкладу за групою, викладачем та аудиторією.
3. Зберігання обраного розкладу для автоматичного відображення при наступному запуску застосунку.
4. Експорт розкладу у соціальні мережі.
5. Надання інклюзивного режиму для користувачів з особливими потребами.

2 КОНЦЕПЦІЯ

2.1 Розробка концепції

Цільова аудиторія та жанр.

Мобільний застосунок орієнтований на студентів, викладачів та адміністративний персонал університету. Особлива увага приділяється користувачам з порушенням зору, для яких реалізовано елементи інклюзивного дизайну. Жанр застосунку можна визначити як «освіта».

Основна вікова категорія користувачів - люди від 17 років.

Короткий опис застосунку.

Застосунок розроблений для спрощення доступу до актуального розкладу занять, полегшуючи навчальний процес як для студентів, так і для викладачів. Інтерфейс реалізовано з урахуванням принципів доступності: адаптивна типографіка, контрастність елементів та підтримка жестів.

Ключові функції застосунку.

Розроблений мобільний застосунок включає наступні функціональні можливості:

- відображення розкладу занять відповідно до обраної навчальної групи, аудиторії або викладача;
- інклюзивний режим, що забезпечує адаптацію інтерфейсу для користувачів з порушенням зору: передбачено темну тему, збільшені розмір і контрастність елементів інтерфейсу та підтримку жестової навігації;
- збереження розкладу з можливістю швидкого доступу при наступному запуску програми:
- експорт розкладу до соціальних мереж або месенджерів, що дозволяє ділитися розкладом з іншими користувачами.

Оновлення

Важливо зазначити, що дані розкладу отримуються з віддаленого сервера, тому будь-які зміни в ньому автоматично синхронізуються із застосунком, що

дозволяє уникнути потреби у випуску оновлень лише для внесення змін у контент, що значно спрощує супровід проєкту.

Таким чином, післярелізна підтримка зосереджена насамперед на покращенні функціональних та нефункціональних характеристик застосунку, а також на забезпеченні зручного доступу до актуального розкладу без залучення користувача до процесу оновлення даних.

2.2 Завдання мобільного застосунку

Головними завданнями проєкту є:

- забезпечення доступності для користувачів з обмеженими можливостями через інтеграцію режиму високої контрастності;
- створення зрозумілого інтерфейсу для перегляду розкладу з підтримкою темного/світлого режимів та адаптацію під різні діагоналі екранів;
- оптимізація продуктивності застосунку для швидкого завантаження даних та стабільної роботи на різних версіях Android.

Забезпечення доступності для користувачів з обмеженими можливостями через інтеграцію режиму високої контрастності.

Дане завдання передбачає реалізація функціональності, яка дозволить користувачам із вадами зору комфортно користуватись мобільним застосунком. Основною метою є створення інтерфейсу, що відповідає стандартам доступності, закрема WCAG 2.1 (рівень AA). Одним із ключових рішень є впровадження режиму високої контрастності, що забезпечить достатній рівень різниці між фоном і текстовими елементами, там самим полегшуючи сприйняття інформації.

Для реалізації цього завдання враховуються наступні аспекти:

- використання чітких шрифтів, достатнього розміру тексту;
- застосування кольорової палітри, яка відповідає мінімальному контрастному співвідношенню 4.5 : 1;
- можливість перемикання між звичайним режимом і режимом високої контрастності через кнопку у застосунку;

- Забезпечення сумісності з екранними зчитувачами, такими як TalkBack, включаючи додавання опису до візуальних елементів інтерфейсу.

Створення зрозумілого інтерфейсу для перегляду розкладу з підтримкою темного/світлого режимів та адаптацію під різні діагоналі екранів

Завдання зосереджене на забезпеченні зрозумілого інтерфейсу для всіх категорій користувачів. Основна мета - надати можливість швидко й без зусиль переглядати розклад занять незалежно від моделі пристрою чи налаштувань користувача. Для досягнення цього передбачено реалізацію адаптивного дизайну з дотриманням принципів UI/UX та рекомендацій Material Design 3.

У межах цього завдання реалізуються такі технічні рішення:

- зрозуміла навігація: розміщення елементів керування у звичних для Android-застосунків місцях, використання іконок та надписів, а також логічна структура вкладок;
- підтримка темного та світлого режимів: застосунок автоматично підлаштовується під тему системи;
- адаптивність до різних екранів: інтерфейс масштабовано відповідно до розміру дисплея. Застосовано компоненти Jetpack Compose, які підтримують адаптивні сітки, гнучку верстку та автоматичне розміщення елементів.

Оптимізація продуктивності застосунку для швидкого завантаження даних та стабільної роботи на різних версіях Android

Оскільки розклад оновлюється з віддаленого сервера, важливо мінімізувати час очікування при завантаженні даних, забезпечуючи при цьому безперебійну роботу навіть на пристроях із середніми або слабкими технічними характеристиками.

Для реалізації завдання було враховано такі технічні аспекти:

- Асинхронне завантаження даних з використанням корутин, що дозволяє виконувати запити до сервера без блокування основного потоку користувацького інтерфейсу;
- Використання сучасних бібліотек;
- Тестування продуктивності на різних емуляторах і реальних пристроях із різними характеристиками для виявлення й усунення потенційних вузьких місць.

3 ОСОБЛИВОСТІ ТА МЕТОДИ РОЗРОБКИ

3.1 Архітектури Android

Архітектура програмного забезпечення - це структурований підхід до проєктування програм, який визначає компоненти системи, їхні функції та способи взаємодії між ними. У контексті мобільної розробки архітектури допомагає створювати масштабовані, підтримувані та тестовані застосунки, які легко адаптуються до змін вимог або функціоналу.

Дослідження архітектури є важливим етапом при проєкутванні застосунку, оскільки:

- воно дозволяє чітко розмежувати логіку, інтерфейс та доступ до даних;
- забезпечує модульність і повторне використання коду;
- спрощує тестування окремих компонентів;
- полегшує розширення функціоналу та підтримку у довгостроковій перспективі.

Загалом, архітектура Android-застосунків базується на принципі відповідальностей та часто реалізується з використанням шаблонів проєктування (MVC, MVP, MVVM тощо). Базова структура Android-платформи містить кілька рівнів, які взаємодіють між собою (див. рисунок 3.2):

Applications

Верхній шар архітектури. Попередньо встановлені програми, такі як контакти, камера, галерея тощо, та сторонні програми, завантаженні з Play Store, будуть встановлені на цьому шарі. Вони працюють у середовищі виконання Android за допомогою класів і служб;

Application Framework

Основна частина Android, яка надає розробникам інструменти та служби, необхідні для створення програм. Він забезпечує доступ до функцій пристрою, таких як апаратне забезпечення, дисплей екрану та системні ресурси. Він містить

декілька важливих служб, які полегшують розробку потужних програм Android, не створюючи все з нуля. Application Framework складається з наступних служб (див. рисунок 3.1):

- Activity Manager - керує активностями програми та їхнім життєвим циклом (наприклад, відриттям, призупиненням або закриттям екранів);
- Notification Manager - дозволяє програмам показувати сповіщення або оновлення користувачеві;
- Package Manager - відстежує всі програми, встановлені на пристрої;
- Window Manager - обробляє розміщення та вигляд вікон на екрані;
- Content Providers - допомагають програмам обмінюватись даними з іншими програмами (наприклад, контактами або фотографіями);
- View System - контролює, як речі розташовані на екрані.

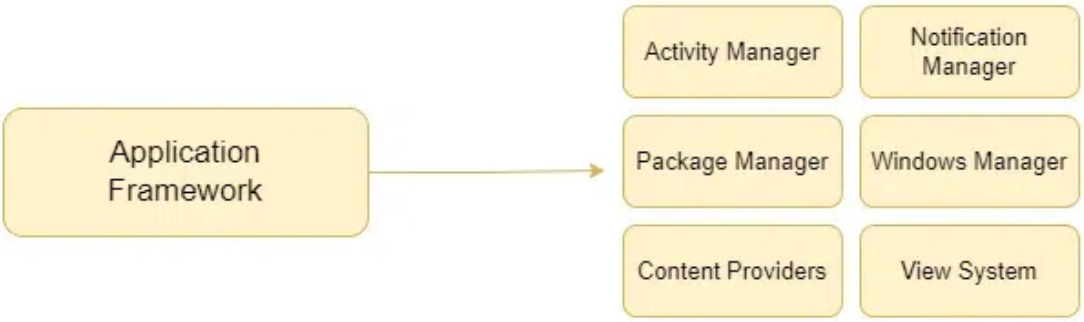


Рис 3.1 Компоненти Application Framework

Android Runtime

Є однією з найважливіших частин Android. Воно містить такі компоненти, як основні бібліотеки та віртуальну машину Dalvik (DVM). В основному, воно забезпечує основу для структури додатків і забезпечує роботу наших додатків за допомогою основних бібліотек. Як і віртуальна машина Java (JVM), віртуальна машина Dalvik (DVM) є віртуальною машиною на основі регістрів, спеціально розробленою та оптимізованою для Android, щоб забезпечити одночасну роботу декількох екземплярів.

Platform Libraries

Включає в себе різні бібліотеки C/C++ та Java-бібліотеки, такі як Media, Graphics, Surface Manager, OpenGL тощо, щоб забезпечити підтримку розробки Android.

Бібліотека Media забезпечує підтримку відтворення та запису аудіо- та відеоформатів.

Surface Manager відповідає за управління доступом до підсистеми дисплея.

SGL та OpenGL - це міжмовні, міжплатформні інтерфейси прикладних програм (API), які використовуються для 2D- та 3D-комп'ютерної графіки.

Linux Kernel

Є серцем архітектури Android. Воно управляє всіма доступними драйверами, такими як драйвери дисплея, камери, Bluetooth, аудіо, пам’яті тощо, які необхідні під час роботи системи.

Ядро Linux забезпечує абстрактний рівень між апаратним забезпеченням пристрою та іншими компонентами архітектури Android. Воно відповідає за управління пам’яттю, живленням, пристроями тощо. Її особливостями є:

- забезпечення безпеки між додатком і системою;
- ефективне управління пам’яттю, надаючи свободу для розробки наших додатків;
- управління процесами, розподілення ресурсів між процесами, коли вони цього потребують;
- управління мережевим зв’язком;
- забезпечення належної роботи додатка на пристрої, а виробники апаратного забезпечення відповідають за вбудовування своїх драйверів у збірку Linux.

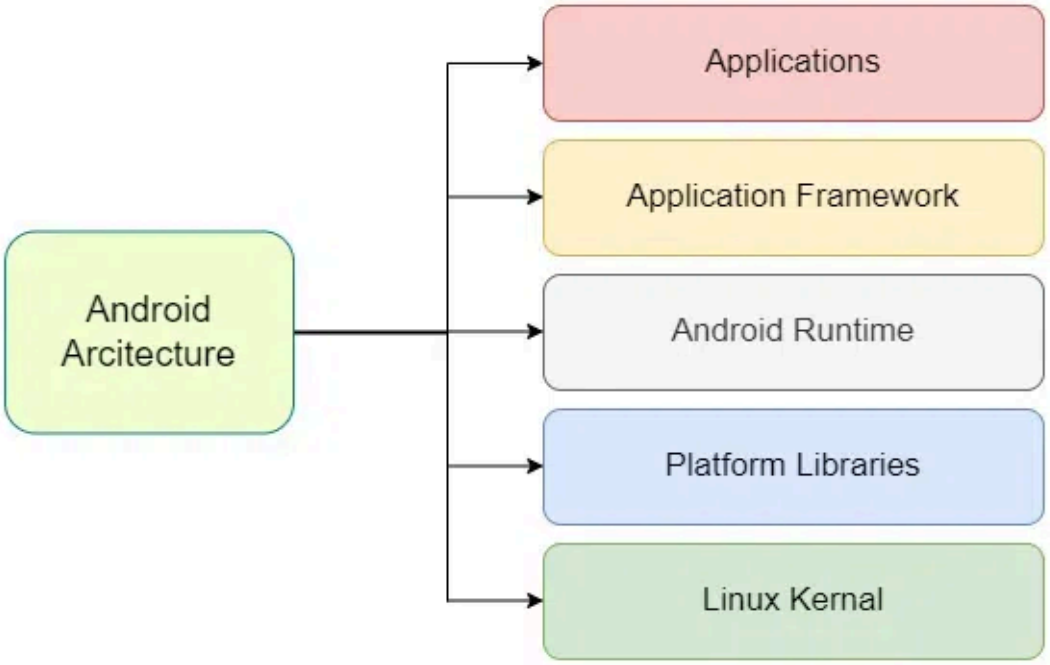


Рис 3.2 Компоненти архітектури Android

3.1.1 MVC

Model-View-Controller - класичний архітектурний шаблон, який був одним із перших, що застосовувалися для організації коду в Android-додатках. Його основна ідея полягає у розділенні програми на три незалежні компоненти: модель, представлення та контролел, кожен з яких відповідає за окрему частину функціональності (див. рисунок 3.3).

Модель (Model)

Відповідає за зберігання, обробку та управління даними додатку. Вона містить бізнес-логіку, працює з локальними чи віддаленими джерелами та передає дані контролеру. Це можуть бути сервіси, бази даних або репозиторії.

Представлення (View)

Відповідає за відображення інформації користувачу. У контексті Android - це XML-розмітка та UI-компоненти, які не мають доступу до бізнес-логіки напряму, а лише показують те, що їм передає контролер.

Контролер (Controller)

Посередник між моделлю та представленням. У Android-архітектурі роль контролера найчастіше виконують Activity або Fragment, які приймають дії користувача, оновлюють модель і змінюють стан представлення.

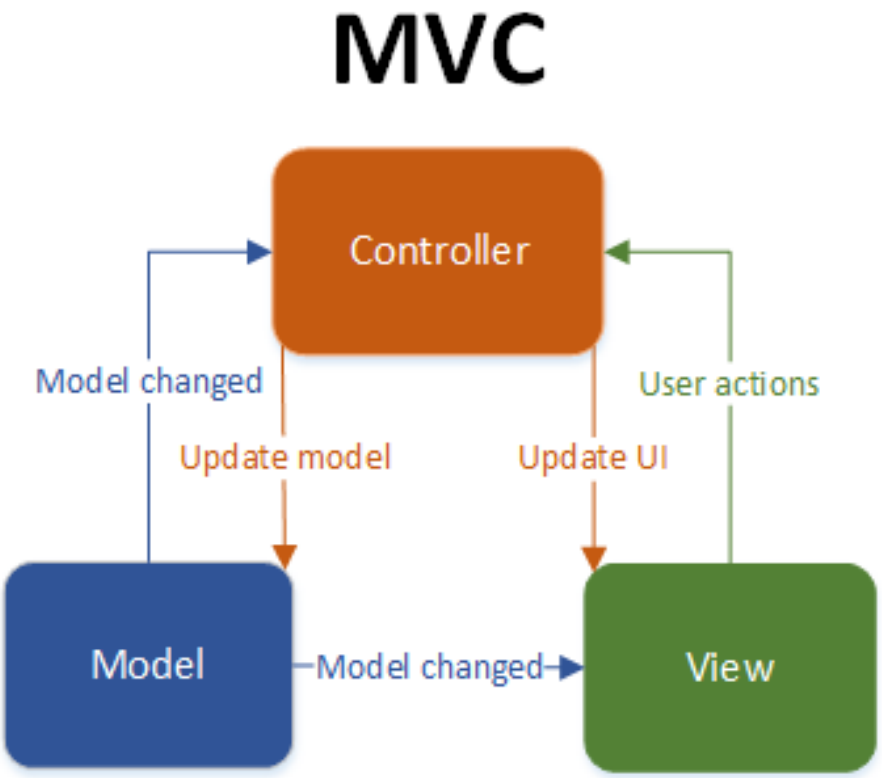


Рис 3.3 Архітектура MVC

MVC є фундаментальною для розуміння принципів побудови програм із чітким розділенням логіки. Її важливість полягає в тому, що вона:

- Дає змогу початківцям усвідомити чому не слід змішувати бізнес-логіку та інтерфейс;
- Підвищує читабельність та підтримуваність коду, оскільки різні частини застосунку відповідають лише за свою функцію;

- Сприяє повторному використанню компонентів, наприклад, модель може використовуватись у декількох інтерфейсах.
- Дає змогу простіше тестувати окремі частини застосунку, особливо бізнес-логіку, яка ізольована від інтерфейсу.

Однак попри ці переваги, архітектура має низку обмежень. Основна проблема полягає в тому, що контроле із часом починає накопичувати надто багато відповідальностей, що призводить до так званого “God Object” - надто великого та складного компонента, який важко підтримувати та тестувати. Крім того, традиційна модель MVC не враховує особливості життєвого циклу Android-компонентів, що ускладнює інтеграцію з платформою та призводить до помилок при зміні станів Activity або Fragment. При розробці масштабного застосунку така архітектура втрачає свою ефективність: вона не забезпечує гнучкої модульності, і розділення між компонентами стає розмитим.

3.1.2 MVP

Model-View-Presenter є логічним розвитком підходу MVC, адаптованим до особливостей розробки мобільних застосунків, зокрема середовищі Android. Її основною метою є більш чітке розділення відповідальностей між компонентами застосунку для забезпечення кращою модульності, тестованості та підтримуваності коду. В основі архітектури MVP також лежить трикомпонентна структура, зображена на рисунку 3.4.

Модель (Model)

Відповідає за бізнес-логіку та обробку даних.

Представлення (View)

Відображає інтерфейс користувача та приймає взаємодії.

Пред’явник (Presenter)

Координує обмін інформацією між моделлю та уявленням, при цьому не маючи прямої залежності від конкретних елементів Android UI.

Відмінності

У MVP пред’явник не взаємодіє безпосередньо з системними подіями, а є повністю ізольованим посередником. Такий підхід дозволяє легко замінити інтерфейс або протестувати бізнес-логіку без необхідності ініціалізації Android-компонентів. В Android-розробці MVP забезпечує хорошу масштабованість: кожне представлення має свого пред’явника, який містить лише логіку, необхідну для керування цією конкретною частиною інтерфейсу. Це дає змогу ефективно розділяти відповідальності між командами розробників або підтримувати складні застосунки, де багато взаємопов'язаних екранів.

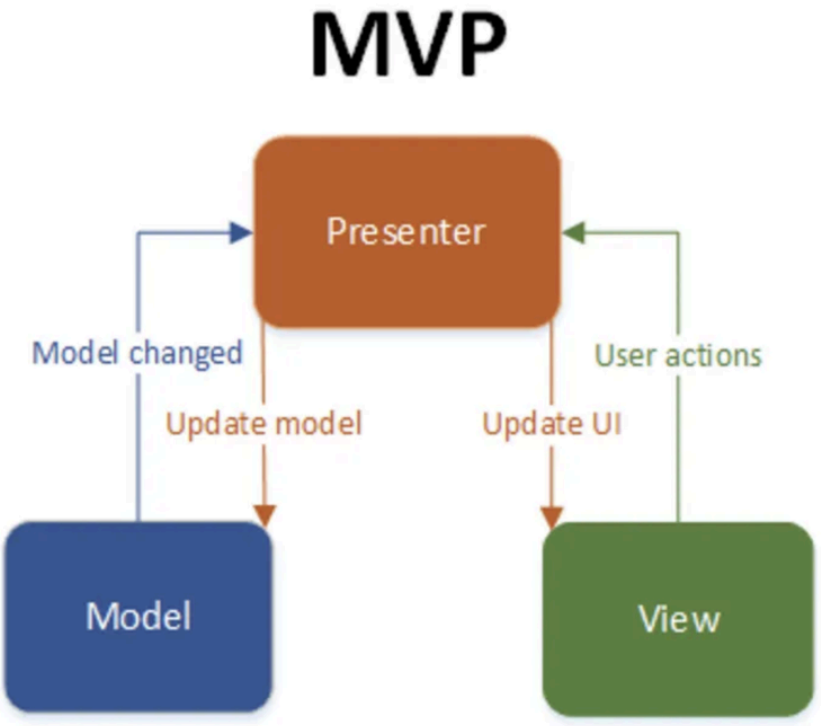


Рис 3.4 Архітектура MVP

Перевагою MVP також є простота модульного тестування, адже пред’явник не залежить від Android SDK, тому його можна перевіряти за допомогою юніт-тестів у звичайному JVM-середовищі. Крім того, архітектувра сприяє кращій структурованості коду - кожна частина системи відповідає за свою чітко визначену функцію. Водночас інтерфейс View найчастіше реалізується у вигляді інтерфейсу, що полегшує заміну або оновлення окремих компонентів.

MVP має і певні недоліки. Вона потребує додаткових зусиль на початковому етапі проєктування, оскільки вимагає великої кількості шаблонного коду. У невеликих або простих проєктах це може створити зайву складність і перевантажити кодову базу.

3.1.3 MVVM

Є сучасним і широко застосовуваним підходом у розробці Android-застосунків. Вона була розроблена з урахуванням потреб платформ із багатим інтерфейсом користувача де важливо досягти максимальної гнучкості, масштабованості та зручності при тестуванні. В основі архітектури також лежить три компоненти (див. рисунок 3.5).

Модель (Model)

Відповідає за роботу з даними та бізнес-логіку.

Представлення (View)

Забезпечує відображення інформації й отримує взаємодію від користувача.

Модель представлення (ViewModel)

Виконує роль посередника, який оперує даними та надає їх View у зручному вигляді.

На відміну від пред’явника у MVP, ViewModel не має прямого посилання на View - замість того, вона використовує механізми спостереження, такі як LiveData або StateFlow, щоб передавати зміни даних у View автоматично. Це дозволяє досягти слабкої зв’язаності компонентів і значно полегшує підтримку проєкту.

Однією з головних переваг є можливість створення реактивного інтерфейсу, який автоматично оновлюється при зміні стану даних у ViewModel. Завдяки використанню двостороннього зв’язку через Data Binding, розробники можуть мінімізувати кількість коду, що обробляє події інтерфейсу вручну.

MVVM також значно полегшує тестування бізнес-логіки, оскільки ViewModel не залежить від AndroidSDK і може бути протестована в ізоляції. Крім цього, повторне використання ViewModel у різних представленнях дозволяє зекономити ресурси та час на реалізацію нових функцій.

Втім, архітектура має і свої складнощі. По-перше, вона потребує високого рівня дисципліни та досвіду від розробника, особливо у випадках з великою кількістю подій, які потрібно обробляти в інтерфейсі. По-друге, активне використання двостороннього зв’язку може призвести до складнощів у відстеженні змін стану, якщо не контролювати залежності між компонентами.

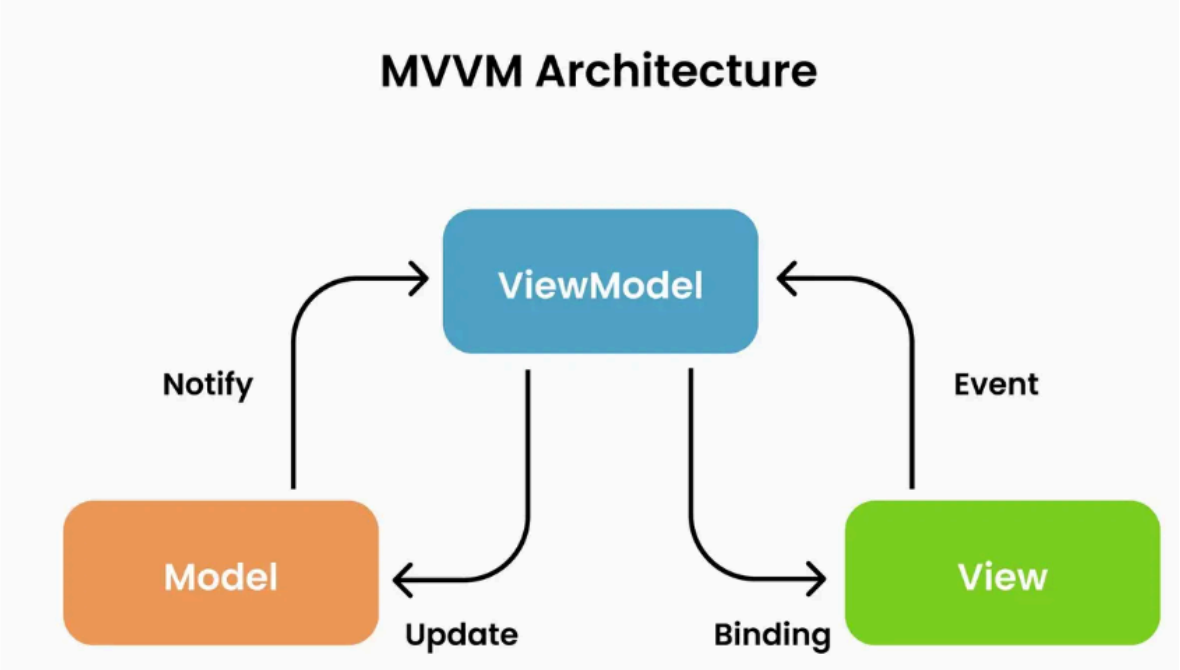


Рис 3.5 Архітектура MVVM

3.2 Особливості управління пам’яттю

Через обмежені ресурси мобільних пристроїв, а також особливості роботи Android-системи, розробник повинен розуміти, як керувати пам’яттю, уникати витоків і ефективно працювати з ресурсами.

Android побудовано на базі віртуальної машини, яка використовує автоматичне збирання сміття (Garbage Collector) для вивільнення непотрібних об’єктів з оперативної пам’яті. Завдяки GC, розробнику не потрібно вручну звільняти пам’ять, однак потрібно дотримуватись певних практик, щоб уникнути утримання посилань на об’єкти, які вже не використовуються. Неправильна роботи зі збиранням сміття може призводити до витоків пам'яті.

Одним із поширених джерел витоків є неправильне використання Context. Наприклад, якщо зберігати посилання на Activity або Fragment у статичних змінних, це перешкоджатиме їх знищенню після завершення життєвого циклу.

Однією з найкращих практик є використання ViewModel, оскільки вона зберігає стан навіть при зміні конфігурації, але автоматично знищується, коли зникає відповідна Activity або Fragment.

3.3 Життєвий цикл застосунку

Життєвий цикл Android-застосунків визначає, як система керує їх створенням, виконаннаям, призупиненням і завершенням. Основним компонентом, що управляє інтерфейсом користувача, є Activity. Кожна Activity проходить через низку станів, які визначають її поведінку та взаємодію із системою та користувачем.

Основні етапи життєвого циклу Activity:

onCreate(Bundle)

Викликається при створенні Activity. На даному етапі відбувається ініціалізація інтерфейсу, завантаження ресурсів і відновлення збереженого стану, якщо він був.

onStart()

Activity стає видимою для користувача, але ще не готова до взаємодії.

onResume()

Activity отримує фокус і стає готовою до взаємодії з користувачем. Це активний стан.

onPause()

Викликається, коли Activity частково втрачає фокус (наприклад, при появі діалогового вікна). Тут слід зберігати важливі дані або зупиняти анімації.

onStop()

Activity більше не видно користувачу (наприклад, при переході до іншої Activity або додатку).

onRestart()

Викликається перед onStart(), коли Activity повертається на передній план після зупинки.

onDestroy()

Останній етап життєвого циклу Activity, коли вона знищується й вивільняє всі ресурси.

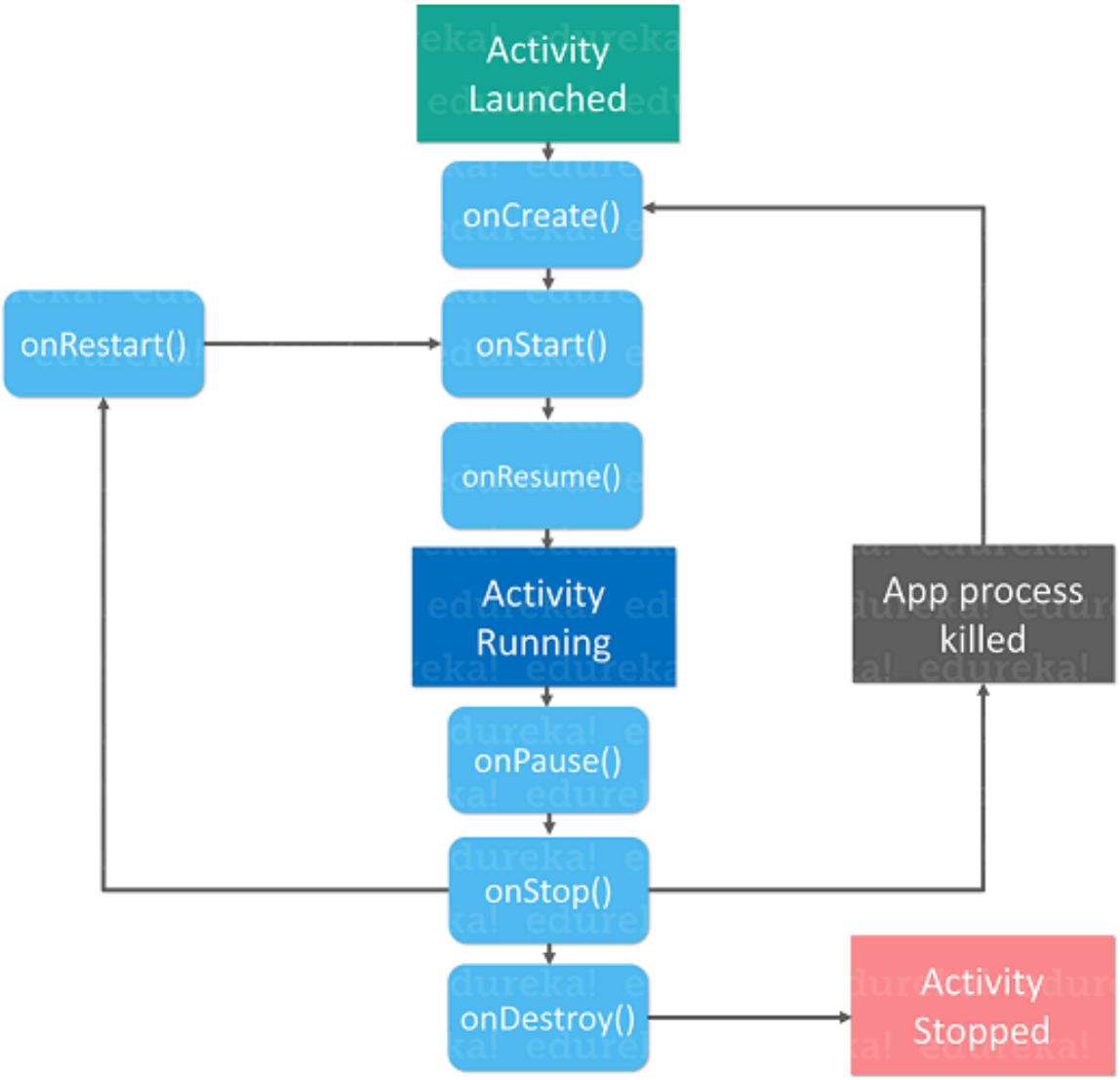


Рис 3.6 Життєвий цикл Activity

3.4 Принципи SOLID

SOLID - це набір п’яти основних принципів об’єктно орієнтованого програмування, які спрямовані на підвищення якості, гнучкості та підтримуваності програмного коду.

1. **Single Responsibility Principle (SRP) - Принцип єдиної відповідальності**

Клас повинен мати лише одну причину для зміни, тобто виконувати лише одну задачі або відповідати за одну функціональність, що зменшує зв’язність і підвищує модульність коду.

2. **Open/Close Principle (OCP) - Принцип відкритості/закритості**

Програмні сутності (класи, модулі, функції) повинні бути відкриті для розширення, але закриті для модифікації. Це означає, що поведінку системи можна розширювати без зміни існуючого коду, що знижує ризик появи помилок.

3. **Liskov Substitution Principle (LSP) - Принцип підстановкки Барбари Лісков**

Об’єкти підкласів повинні бути замінними на об’єкти базового класу без порушення коректості роботи програми. Іншими словами, підкласи повинні зберігати поведінку базового класу.

4. **Interface Segregation Principle (ISP) - Принцип розділення інтерфейсів**

Клієнти не повинні залежати від інтерфейсів, які вони не використовують, що зменшує зв’язність і підвищує гнучкість. Тому краще мати багато спеціалізованих класів, ніж один загальний.

5. **Dependency Inversion Principle (DIP) - Принцип інверсії залежностей**

Високорівневі модулі не повинні залежати від низькорівневих; обидва повинні залежати абстракцій. Абстракції не повинні залежати від деталей, а деталі - від абстракцій, що сприятиме слабкому зв’язуванню між компонентами.

3.5 XML

XML - це розширювана мова розмітки, яка використовується для структурованого зберігання даних у вигляді текстових файлів з тегами.

В Android XML широко застосовується для опису ресурсів і користувацького інтерфейсу. Наприклад, макети екранів створюються у вигляді XML-файлів, які визначають ієрархію елементів інтерфейсу, їх розташування, властивості та стилі. Окрім макетів, XML також використовується для опису

файлу маніфесту, який містить інформацію про компоненти додатку, дозволи, версії SDK та інші налаштування. Перевагами використання XML у Android є:

Структурованість та читабельність

XML має чітку ієрархічну структуру, що полегшує розуміння і модифікацію ресурсів.

Відокремлення UI від логіки

Розділення опису інтерфейсу і коду підвищує гнучкість розробки і дозволяє дизайнерам працювати незалежно від програмістів.

Підтримка локалізації

Текстові ресурси можна зберігати у XML-файлах, що дозволяє легко додавати переклади на різні мови.

Підтримка різних конфігурацій

XML-файли можна розміщувати у спеціальних папках (наприклад, res/layout, res/values), що дозволяє адаптувати додаток під різні розміри екранів, орієнтації, мови тощо.

Однак, XML має своє обмеження:

Імперативність і дублювання

При складних інтерфейсах XML-файли можуть ставати громіздкими, а зміни часто потребують багато ручної роботи, що ускладнює підтримку.

Відсутність логіки

XML описує лише структуру і властивості, але не може містити логіку поведінку, що змушує розробників додатково писати код на Java/Kotlin.

Обмежена гнучкість

Для динамічного створення або зміни UI під час виконання XML не підходить, що іноді вимагає додаткових рішень.

3.6 Jetpack Compose

Jetpack Compose - сучасний інструмент для створення користувацького інтерфейсу в Android-додатках, розроблений компанією Google і базований на

декларативному підході. На відміну від традиційної розробки з використанням XML-розмітки, Compose дозволяє описувати UI у вигляді composable-функцій на мові Kotlin, які автоматично оновлюються при зміні стану даних.

Jetpack Compose має декілька ключових переваг:

Інтерактивний та динамічний UI

Compose забезпечує автоматичне оновлення інтерфейсу в реальному часі при зміні стану даних, що робить додатки більш привабливими та чуйними. Це досягається через механізм recomposition, який оновлює лише ті частини інтерфейсу, які залежать від змінених даних.

Повторне використання компонентів

Composable-функції можна легко комбінувати і повторно використовувати в різних частинах додатку або навіть в різних проєктах.

Повна інтеграція з Kotlin

Оскільки Jetpack Compose побудований на Kotlin, він повністю використовує всі переваги мови, включаючи строгий контроль типів, що дозволяє виявляти помилки на етапі компіляції.

Адаптивність і підтримка тем

Compose спрощує реалізація світлого і темного режимів, підтримку різних діагоналей екранів і динамічне налаштування стилів. Завдяки системі тем і модифікаторів, інтерфейс легко адаптується до будь-яких змін без дублювання.

Попередній перегляд

Android Studio підтримує прев'ю композицій у реальному часі, що дає змогу переглядати інтерфейс без компіляції всього проєкту. Це значно прискорює тестування окремих компонентів і зменшує час на внесення змін.

Проте варто зазначити, що Jetpack Compose має і свої недоліки:

Сирість технології і нестача функціоналу

Jetpack Compose - відносно новий фреймворк, який на початкових етапах містив багато багів і не мав повного набору функціональних можливостей. Розробникам доводиться дописувати відсутній функціонал самостійно або використовувати старі технології для окремих екранів.

Обмеження навігації

Стандартні засоби навігації Compose мають обмеження, наприклад, параметри, що передаються між екранами, мають бути примітивними типами. Це змушує розробників застосовувати складні інженерні рішення для передачі об’єктів між екранами.

3.7 Інклюзивність

В екосистемі Android концепція інклюзивності є невід’ємною частиною філософії розробки, що відображається у численних інструментах, рекомендація та функціональних можливостях, спрямованих на створення доступних інтерфейсів. Інклюзивний дизайн є набагато більше, ніж просто набір технічних рекомендацій; це фундаментальний підхід, що спрямований на створення цифрових продуктів, які можуть бути ефективно використані всіма людьми. Доступність забезпечує можливість користування застосунком для осіб з різними вадами, такими як порушення зору, слуху, моторики, когнітивні розлади тощо.

Переваги інклюзивнох розробки для Android-застосунків є багатогранними:

- Доступний застосунок може використовуватись більшою кількістю людей, включаючи населення, які мають певні форми інвалідності.
- Принципи інклюзивного дизайну часто покращують зручність використання для всіх користувачів. Наприклад, чіткі контрасти, великі елементи управління та інтуїтивно зрозуміла навігація є корисними для кожного, а не лише для людей з особливими потребами.
- Розв’язання завдань доступності часто вимагає творчих підходів, що може призвести до розробки інноваційних рішень, які покращують застосунок в цілому.

Реалізація інклюзивності в Andoid-застосунках грунтуєтсья на чотирьох ключових принципах доступності, визначених в WCAG , які успішно адаптуються до мобільних платформ:

Сприйнятність

Інформація та компоненти інтерфейсу повинні бути представлені у спосіб, що може бути сприйнятий користувачами. Для Android це означає:

- надання осмислених текстових описів для всіх нетекстових елементів для зчитування зчитувачами екрану;
- забезпечення достатнього контрасту між текстом та фоном, а також між елементами інтерфейсу, щоб зробити їх читабельними для людей зі слабким зором або дальтонізмом;
- застосунок повинен коректно відображатися при збільшенні розміру шрифту, встановленого користувачем у системних налаштуваннях;

Керованість

Елементи інтерфейсу та навігація повинні бути керованими, що включає:

- забезпечення можливості керування застосунком без використання сенсорного екрана, тобто за допомогою клавіатури, трекболу, зовнішніх перемикачів та спеціальних символів;
- Інтерактивні елементи повинні мати достатній розмір (мінімум 48dp), щоб їх було легко торкнутися, особливо для людей з порушенням моторики.
- при навігації за допомогою клавіатури або інших методів, елемент під фокусом повинен бути чітко візуально виділений.

Зрозумілість

Інтерфейс та інформація повинні бути зрозумілими. Це означає:

- застосунок повинен мати логічну та передбачувану структуру, де елементи управління розташовані послідовно та інтуїтивно зрозуміло;
- використання простої та зрозумілої мови, уникнення складних термінів та жаргону;

Надійність

Контент повинен бути надійним і достатньо стійким, щоб його могли інтерпретувати різні користувацькі агенти, включаючи допоміжні технології.

4. КРОКИ РОЗРОБКИ ДОДАТКА

4.1 Головний екран додатка

Головний екран - це екран, у який користувач потрапляє одразу після входу у застосунок. Цей екран водночас є і головним меню усього додатку. На цьому екрані у користувача є можливість потрапити на екрани, які розділені на певні категорії для доступного знаходження інформації, яка цікавить користувача додатку.

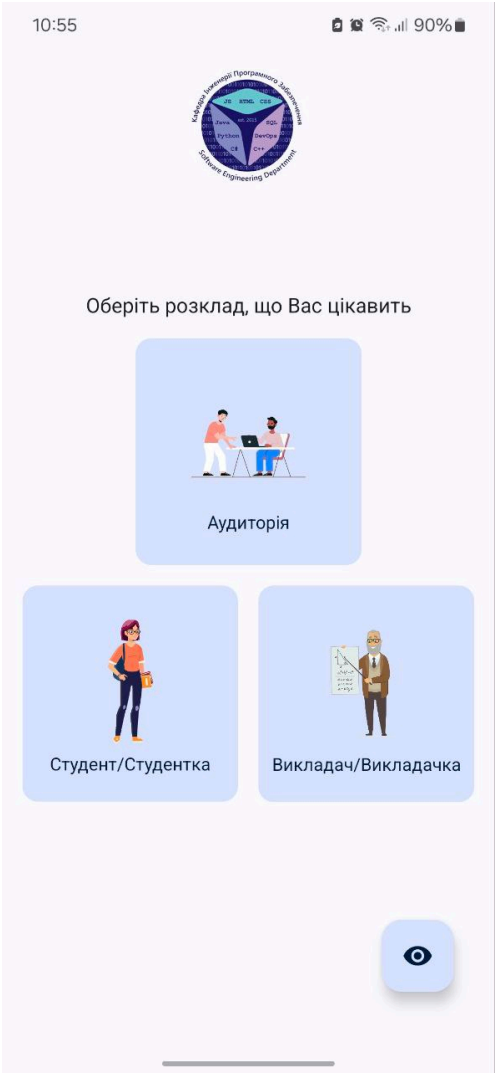


Рис 4.1 - Головний екран застосунку

На рисунку 4.1 можна побачити головний екран, який має простий та інтуїтивно зрозумілий дизайн, що дозволяє користувачам швидко орієнтуватись у

додатку. Більшу частину екрану займають інтерактивні елементи для вибору необхідного типу розкладу.

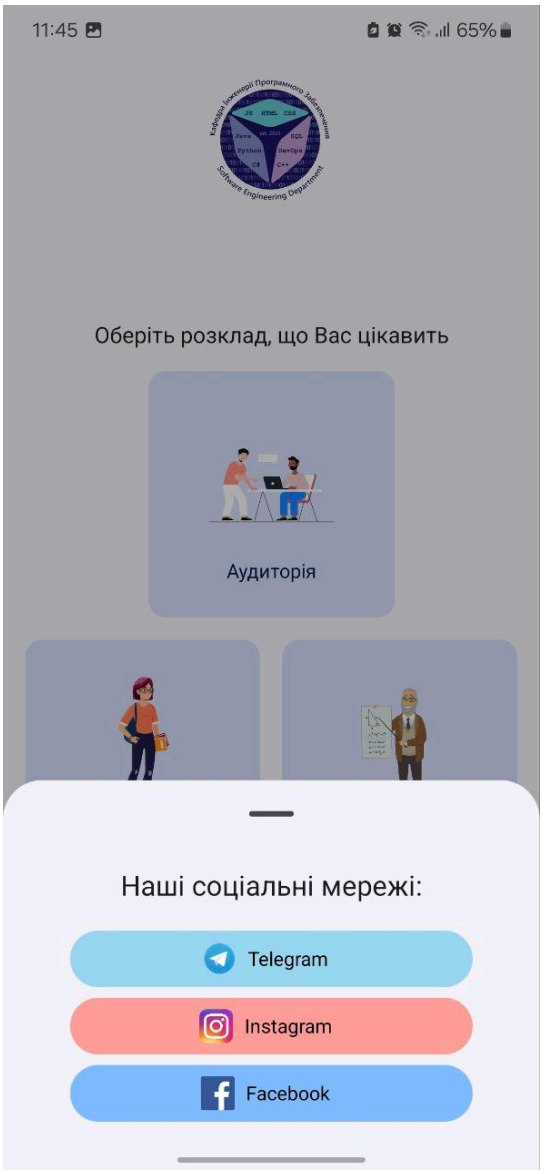


Рис 4.2 Модальне вікно з посиланнями на соціальні мережі

У верхній частині екрану розташований логотип університету, який виконує як естетичну, так і функціональну роль.

При натисканні на нього знизу відкривається модальне вікно з посиланнями на соціальні мережі кафедри, що дозволяє користувачам отримати актуальні новини або додаткову інформацію.

```
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@Composable
fun EntryScreen(
    navController: NavController,
    viewModel: InclusiveModeViewModel,
    modifier: Modifier = Modifier
) {
    val sheetState = rememberModalBottomSheetState()
    var showBottomSheet by remember { mutableStateOf( value: false) }
    ScaffoldWithInclusiveMode(
        viewModel = viewModel,
        modifier = modifier
    ) {
        Box(modifier = modifier.fillMaxSize()) {
            IpzLogo(
                ipzLogoId = R.drawable.ipz_logo,
                onLogoClick = { showBottomSheet = true }
            )

            Column (
                modifier = Modifier
                    .align(Alignment.Center)
                    .padding(dimensionResource(16.dp))
                    .fillMaxSize(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                InclusiveText(
                    text = stringResource("Оберіть розклад, що Вас цікавить"),
                    viewModel = viewModel,
                    modifier = Modifier
                        .padding(dimensionResource(16.dp)),
                    style = TextStyle(
                        fontSize = 16.sp,
                        textAlign = TextAlign.Center
                    )
                )
                Column (
                    modifier = Modifier.fillMaxWidth(),
                    horizontalAlignment = Alignment.CenterHorizontally,
                    verticalArrangement = Arrangement.spacedBy(16.dp)
                ){
                    Box(modifier = Modifier.fillMaxWidth( fraction: 0.5f)) {
                        ScheduleOption(
                            textId = "Аудиторія",
```

Рис 4.3 Частина коду головного екрану

Кнопки для вибору розкладу були розроблені таким чином, щоб їхні розміри динамічно змінювалися у залежності від розміру екрану на якому функціонує додаток, ще дозволяє зберігати чітку композицію інтерфейсу.

4.2 Екран «Аудиторія»

Екран вибору аудиторії створений для пошуку розкладу занять у конкретному кабінеті.

Рис 4.4 Екран вибору аудиторії

Інтерфейс складається з заголовку, кнопки для повернення до попереднього екрану, кнопки для переходу до розкладу, та основного елементу - інтерактивне поле для вибору аудиторії. При натисканні на неї відкривається випадаючий список із номерами аудиторії, серед яких користувач може обрати необхідний варіант. Вибір аудиторії робить кнопку доступною для натискання, що дозволить користувачу перейти до бажаного розкладу.

4.3 Екран «Студент/Студентка»

Даний екран є ще однією опцією вибору розкладу, який призначений для отримання розкладу відповідно до академічної групи.

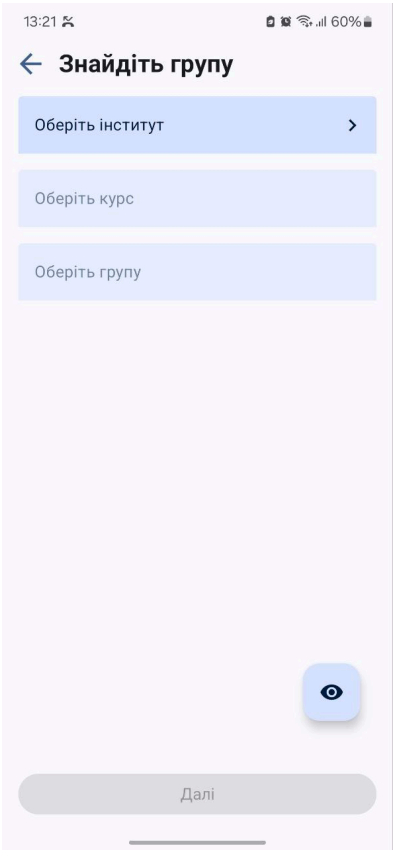


Рис 4.5 Екран вибору групи

Основна структура екрана включає три взаємопов’язані поля з випадаючими списками. Вибір у кожному з них впливає на опції у наступному полі. При відкритті випадаючого списку виконується анімація, яка зміщує нижчі елементи екрана, щоб опції не перекривали інші елементи інтерфейсу.

Як і в екрані вибору аудиторії, у ньому наявні кнопки для повернення для на попередній екран та переходу на екран розкладу, яка залишається неактивною, доки користувач не заповнить усі три поля, що виключає можливість переходу без вибору усіх необхідний даних для запиту.

4.4 Екран «Викладач/Викладачка»

Екран вибору викладача надає два способи пошуку, між якими користувач може перемикатись за допомогою перемикача.

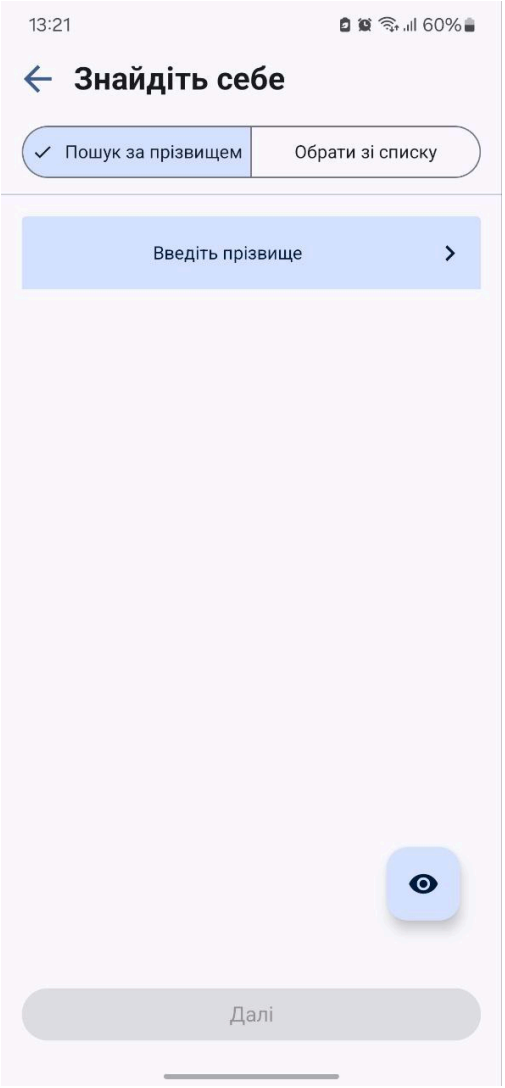


Рис 4.6 - Екран вибору викладача

Перший спосіб передбачає пошук за ім’ям викладача. Пошук виконується за допомогою введення імені у текстове поле. Результати пошуку відображаються у випадаючому списку після введення трьох символів.

Другий спосіб побудований на використанні трьох взаємопов’язаних випадаючих списків. Даний варіант передбачає користувача обрати факультет, кафедру та самого викладача.

Після вибору викладача одним з наведених способів, стає активною кнопка переходу до розкладу, дозволяючи користувачеві отримати необхідну інформацію. Завдяки зручному перемиканню між двома методами пошуку екран адаптується під різні сценарії використання, що забезпечує максимальну ефективність та комфорт для користувача.

4.5 Екран «Розклад»

Екран розкладу є ключовим елементом додатку, оскільки саме тут користувач отримує основну інформацію - розклад занять, який обирався за допомогою одного з трьох запропонованих додатком способів. Інтерфей організований таким чином, щоб забезпечити швидкий доступ до потрібних даних, зручність навігації та можливість взаємодії з розкладом.

Рис 4.7 Екран з розкладом

У верхній частині екрану розташовано дві кнопки:

Кнопка «Поділитись розкладом»

Дана кнопка дозволяє користувачеві швидко поділитись розкладом на вибраний день через соцмережі, месенджери або інші доступні способи. Натискання на кнопку відкриває системне меню обміну, де можна обрати зручний спосіб відправки.

Кнопка «Запам'ятати розклад»

Дана кнопка надає можливість додатку відкривати збережений розклад при наступному запуску, що значно економить час. Реалізація цієї функції реалізується за допомогою Jetpack DataStore, що дозволяє зберігати вибрані параметри у вигляді Preferences безпосередньо на пристрої, що забезпечує швидкий доступ до вибраного розкладу після перезапуску додатку. У DataStore зберігається інформація про тип розкладу, ідентифікатор групи або викладача та назва збереженого розкладу у вигляді імені викладача або назви групи.

```
private val Application.dataStore: DataStore<Preferences> by preferencesDataStore(name = "schedule_preferences")
class RememberScheduleViewModel(
    context: Application
) : AndroidViewModel(context) {
    private val dataStore = context.dataStore

    private val _rememberedSchedule = MutableStateFlow<RememberedScheduleState?>( value: null)
    val rememberedSchedule: StateFlow<RememberedScheduleState?> = _rememberedSchedule.asStateFlow()

    private val _showDialog = MutableStateFlow( value: false)
    val showDialog: StateFlow<Boolean> = _showDialog.asStateFlow()

    init {
        viewModelScope.launch {
            loadRememberedSchedule()
        }
    }

    private suspend fun loadRememberedSchedule() {
        dataStore.data.firstOrNull()?.let { preferences ->
            val type = when(preferences[PreferenceKeys.SCHEDULE_TYPE]) {
                "group" -> preferences[PreferenceKeys.GROUP_ID]?.let { ScheduleType.Group(it) }
                "teacher" -> preferences[PreferenceKeys.TEACHER_ID]?.let { ScheduleType.Teacher(it) }
                else -> null
            }
            val label = preferences[PreferenceKeys.LABEL] ?: ""
            _rememberedSchedule.value = RememberedScheduleState(type, label)
        }
    }
```

Рис 4.8 Частина коду збереження розкладу

Під верхньою панеллю розташований ряд із днями тижня, який використовується для перемикання між датами. Кожен день представлений у вигляді окремої клітинки, натиснувши на яку змінюється відображуваний розклад. Обраний день візуально виділяється, що забезпечує зручність користування, дозволяючи швидко орієнтуватися в розкладі. При наявності пар у певному, клітинка матиме відповідний круг-індикатор, що дозволяє користувачеві заздалегідь бачити, які дні мають заняття.

Основна частина екрану містить розклад занять для обраного дня. Кожна пара представлена у вигляді окремого блоку, що відображає основну інформацію про заняття, а саме його назву, тип, викладача та місце проведення. Для полегшення візуального сприйняття різні типи занять мають власне колірне позначення. При натисканні на пару відкривається модальне вікно, яке містить повну інформацію про заняття.

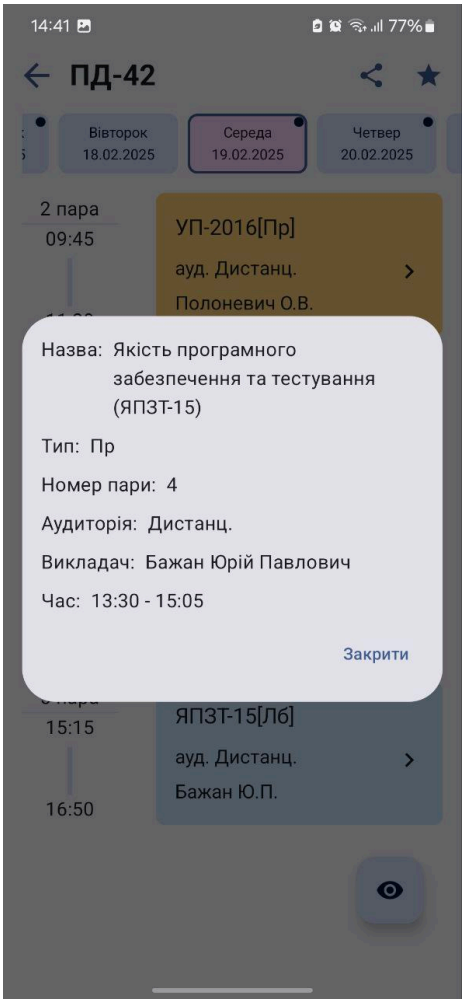


Рис 4.9 Модальне вікно з повною інформацією про заняття

При відсутності пар в обраний день, замість розкладу відображається відповідне повідомлення. Крім того, для покращення користувацького досвіду та зменшення відчуття порожності но екрані демонструється випадкова анімація, яка додає інтерактивності та естетичної привабливості.

4.6 Інклюзивний режим

Ключовою функціональністю розробленого застосунку є реалізація інклюзивного режиму, який дозволяє користувачам адаптувати інтерфейс для покращеного візуального сприйняття. Суть цього режиму полягає у цілеспрямованому збільшенні контрастності елементів інтерфейсу та масштабуванні розміру шрифту, що є критично важливим для користувачів з різними особливостями зору, включаючи людей з низьким зором, дислексією, астигматизмом, або тих, хто просто віддає перевагу більшому та тексту та чіткішим візуальним елементам.

Перехід між стандартним та інклюзивним режимом здійснюється за допомогою спеціального перемикача у вигляді кнопки в нижньому правому куті на всіх екранах застосунку. Такий підхід надає користувачеві повний контроль над візуальним представленням інформації, дозволяючи обирати найбільш комфортний для себе варіант.

При активації інклюзивного режиму, застосунок змінює колірну палітру та розміри шрифтів, перетворюючи візуальне оформлення з метою максимальної доступності.

В інклюзивному режимі основна увага приділяється забезпеченню високого коефіцієнта контрастності між текстовими та графічними елементами та їхнім фоном. Реалізується це наступним чином:

- Застосовується колірна схема з чіткими та насиченими кольорами, які забезпечують максимальну візуальну розбірливість. Використовується домінування світлого фону у поєднанні з темними, насиченими кольорами для тексту та інтерактивних елементів, що відповідає рекомендаціям WCAG

2.1 щодо мінімального співвідношення контрастності (4.5:1 для звичайного тексту).

- Елементи, що знаходяться в активному стані, виділяються за допомогою більш контрастного кольору фону та тексту, що інформує користувача про поточний вибір.

Окрім підвищеної контрастності, активація інклюзивного режиму призводить до збільшення розміру шрифту у всьому застосунку. Інтерфейс застосунку розроблено з урахуванням гнучкості макетів. При збільшенні шрифту, елементи перекомпонуються, щоб текст не виходив за межі контейнерів, не накладався на інші елементи та залишався повністю видимим.

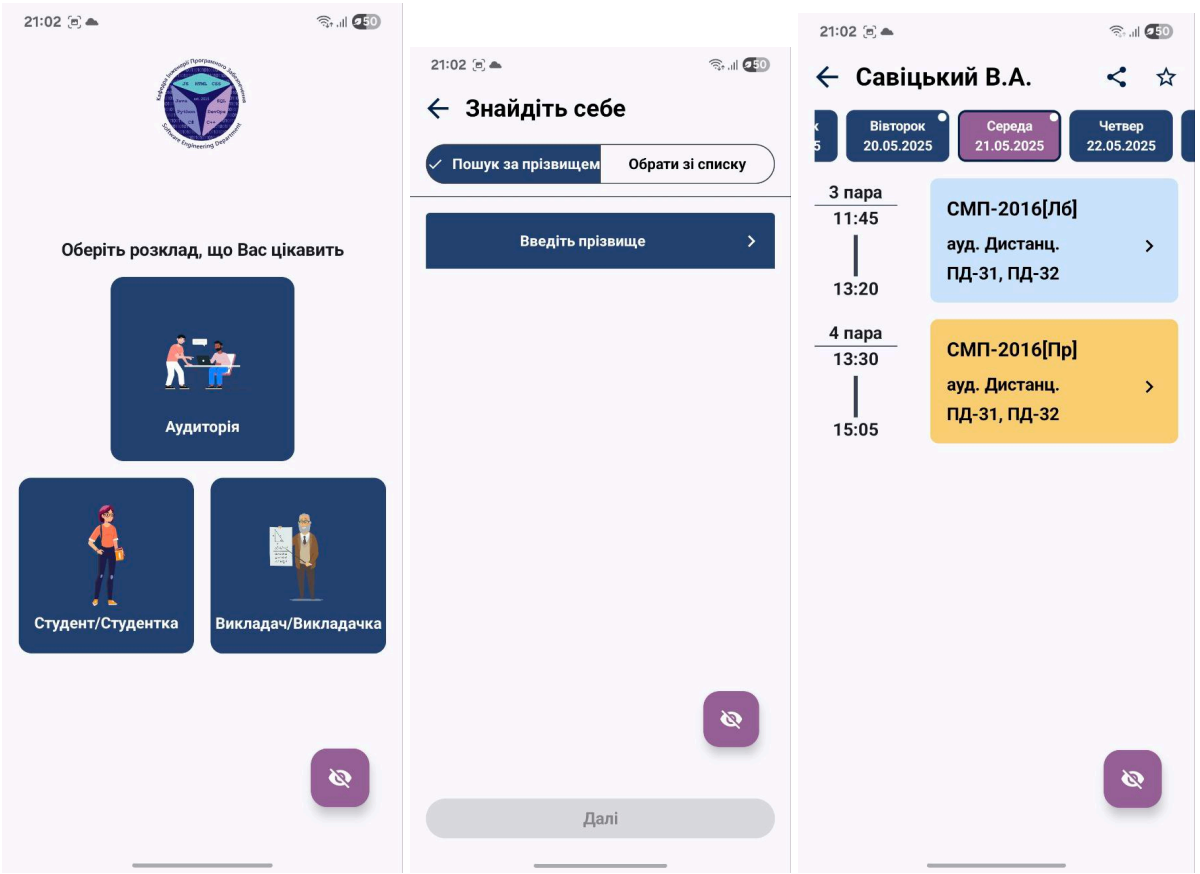


Рис 4.10 Приклади екранних форм застосунку в інклюзивному режимі

5 ТЕСТУВАННЯ ЗАСТОСУНКУ

Тестування розробленого застосунку було виконано з особливою увагою. Метою було не лише перевірити коректність функціонування всіх заявлених можливостей, а й оцінити загальну стабільність роботи застосунку, а також відповідність реалізації принципам інклюзивного дизайну, що є критично важливим для забезпечення доступності для широкого кола користувачів.

Процес тестування тестування був структурований таким чином, щоб охопити максимально широкий спектр можливих сценаріїв використання та конфігурацій. Тестування проводилось як на реальних фізичних пристроях з різними версіями Android та апаратними характеристиками, так і за допомогою емуляторів.

Тестування охоплювало наступні аспекти:

- Перевірка відповідності розробленого функціоналу заявленим вимогам технічного завдання, що включало перевірку всіх інтерактивних елементів, логіки роботи основних функцій, таких як коректне завантаження та відображення розкладу, перемикання між звичайним та інклюзивним режимами.
- Оцінка візуального відображення елементів, їхньої інтерактивності та адаптивності на різних формах екранів. Була перевірена коректність відображення шрифтів, іконок, кольорових схем та загального дизайну.

Для первинної оцінки стабільності та працездатності основних функцій застсоунку було проведено димове тестування. Результати тестування наведено у таблиці 5.1.

У ході тестування було виявлено та усунено низку незначних недоліків, які переважно стосувалися візуального відображення окремих елементів інтерфейсу на специфічних розмірах екранів та оптимізації швидкості завантаження. Виявлення та виправлення цих недоліків дозволило значно підвищити загальну стабільність та зручність використання застосунку.

Таблиця 5.1

Результати димого тестування застосунку

№	Test case summary	Goal	Pre-conditions	Test Case Steps	Expected results	Actual Results
1	Відкриття застосунку	Перевірити, чи запускається застосунок без помилок	Встановлений застосунок	1. Натиснути іконку застосунку 2. Дочекатись завантаження головного екрана	Головний екран відкривається без помилок	Головний екран з’явився через 1.5 сек, помилок немає
2	Завантаження розкладу	Перевірити коректне завантаження даних з сервера	Підключення до інтернету	1. Запустити застосунок 2. Обрати групу 3. Дочекатись завантаження розкладу	Відображається розклад для обраної групи	Відображається розклад для обраної групи
3	Пеермикання теми	Перевірити перемикання між темною та світлою темою	Встановлена версія Android 10+	1. Перейти в налаштування 2. Увімкнути темну тему	Інтерфейс змінює кольорову схему	Інтерфейс змінює кольорову схему
4	Увімкнення інклюзивного режиму	Перевірити роботу режиму високої контрастності	Встановлений застосунок	1. Відкрити застосунок 2. Натиснути на іконку інклюзивного режиму	Застосунок переходить у висококонтрастний режим	Застосунок переходить у висококонтрастний режим

ВИСНОВКИ

Під час розробки мобільного застосунку у дипломній роботі було пройдено всі етапи, а саме:

1. Досліджено рішення для відображення розкладу, що дозволило визначити їх переваги, недоліки та ключові функціональні можливості.
2. Сформовано функціональні та нефункціональні вимоги, які забезпечать швидкодію та адаптивність системи.
3. Розроблено архітектуру мобільного застосунку, яка забезпечує модульність, масштабованість та надійну взаємодію між компонентами системи.
4. Розроблено інтерфейс, який враховує потреби людей з порушенням зору.
5. Розроблено клієнтську частину застосунку з використанням сучасних технологій, що забезпечило швидкодію та стабільність роботи.
6. Проведено тестування, яке підтвердило працездатність застосунку, його адаптивність до різних пристроїв та відповідність сформованим вимогам.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Фам Х.В., Савіцький В.А. Визначення вимог до мобільного застосунку з елементами інклюзивного дизайну для відображення розкладу університету: Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2025, ДУІКТ, м. Київ. К.: ДУІКТ, 2025, С. 601-602.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kotlin Documentation [Електронний ресурс] - Режим доступу до ресурсу: https://kotlinlang.org/docs/getting-started.html
2. GeeksForGeeks. Android Architecture [Електронний ресурс] - Режим доступу до ресурсу: https://www.geeksforgeeks.org/android-architecture/
3. Web Content Accessibility Guidelines [Електронний ресурс] - Режим доступу до ресурсу: https://www.w3.org/TR/WCAG21/
4. Application Fundamentals [Електронний ресурс] - Режим доступу: https://developer.android.com/guide/components/fundamentals
5. Native Android Development on Windows [Електронний ресурс] - Режим доступу: https://learn.microsoft.com/en-us/windows/android/native-android
6. The Six Most Popular Cross-Platform App Development Frameworks https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html
7. The Kotlin Multiplatform Developer Roadmap [Електронний ресурс] - Режим доступу: https://github.com/skydoves/kmp-developer-roadmap
8. Native, hybrid , or cross-platform apps? [Електронний ресурс] - Режим доступу: https://www.microsoft.com/en-us/power-platform/products/power-apps/topics/app-development/native-vs-cross-platform-apps
9. The Ultimate Guid to Mobile Accessibility in 2025 [Електронний ресурс] - Режим доступу: https://www.accessibilitychecker.org/guides/mobile-apps-accessibility/
10. Digital Ocean. The First 5 Principles of Object Oriented Design [Електронний ресурс] - Режим доступу: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

11. Design Gurus. What is XML in Android? [Електронний ресурс] - Режим доступу: https://www.designgurus.io/answers/detail/what-is-xml-in-android
12. Jetpack Compose Documentation [Електронний ресурс] - Режим доступу: https://developer.android.com/develop/ui/compose/documentation
13. ProAndroidDev. Understanding Low Memory Management in Android [Електронний ресурс] - Режим доступу: https://proandroiddev.com/understanding-low-memory-management-in-android-kswapd-lmk-9ff694c98f7e
14. Regine M.Gilbert. Inclusive Design for a Digital World, 2019, p. 83 - Режим доступу: https://oscqr.suny.edu/wp-content/uploads/2024/04/dokumen.pub_inclusive-design-for-a-digital-world-designing-with-accessibility-in-mind-1st-ed-978-1-4842-5015-0-978-1-4842-5016-7.pdf
15. Business of Apps. App market trends 2025 [Електронний ресурс] - Режим доступу: https://www.businessofapps.com/guide/app-market-trends-2025/
16. Mobiloud. 57 Mobile App Download, Usage and Revenue Statistics for 2024 [Електронний ресурс] - Режим доступу: https://www.mobiloud.com/blog/mobile-app-statistics?utm_source=chatgpt.com
17. Medium. Modern Android App Architecture with Clean Code Principles [Електронний ресурс] - Режим доступу: https://medium.com/design-bootcamp/modern-android-app-architecture-with-clean-code-principles-2025-edition-95f4c2afeadb
18. An Overview of 5G Technology Worldwide 2021 [Електронний ресурс] - Режим доступу: https://www.emarketer.com/content/overview-of-5g-technology-worldwide-2021

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку з елементами інклюзивного дизайну для відображення розкладу університету

Виконав студент 4 курсу
групи ПД-42
Фам Хоанг Ву
Керівник роботи
Викладач кафедри ІПЗ Савіцький Вячеслав Андрійович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення сприйняття розкладу занять для студентів із порушенням зору за рахунок використання принципів інклюзивного дизайну.

Об’єкт дослідження: процес відображення розкладу занять та його доступність для користувачів із порушенням зору.

Предмет дослідження: мобільний застосунок з елементами інклюзивного дизайну для відображення розкладу університету.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз існуючих застосунків для відображення розкладу університету.
2. Визначити функціональні та нефункціональні вимоги.
3. Розробити архітектуру мобільного застосунку.
4. Проєктування інтерфейсу застосунку з елементами інклюзивного дизайну.
5. Розробити мобільний застосунок.
6. Провести тестування та налагодження розробленої системи.

3

АНАЛІЗ АНАЛОГІВ

Показник	Розклад ХНУРЕ	Розклад ЛП	Розклад СумДПУ	Розклад ДУІКТ
Інтеграція з іншими додатками	-	-	+	+
Сповіщення про наступні заняття	+	+	+	+
Відображення повної інформації про заняття	+	-	-	+
Автоматичне оновлення розкладу	+	+	+	+
Підтримка темної теми	+	+	+	+
Інклюзивний режим	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Відображення розкладу занять обраної групи/викладача .
2. Надання інклюзивного режиму.
3. Зберігання розкладу для його відкриття при наступному запуску.
4. Експортування розкладу у соціальні мережі.

Нефункціональні вимоги:

1. Завантаження розкладу менше ніж за 2 секунди.
2. Система повинна мати інтерфейс відповідно до принципів Material Design 3.
3. Відповідність стандартам доступності WCAG 2.1 (рівень AA).
4. Коректне відображення на пристроях з різними діагоналями екрана.
5. Сумісність зі зчитувачами екрану.
6. Підтримка версії Android 10 та новіші.
7. Підтримка темної теми.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Kotlin

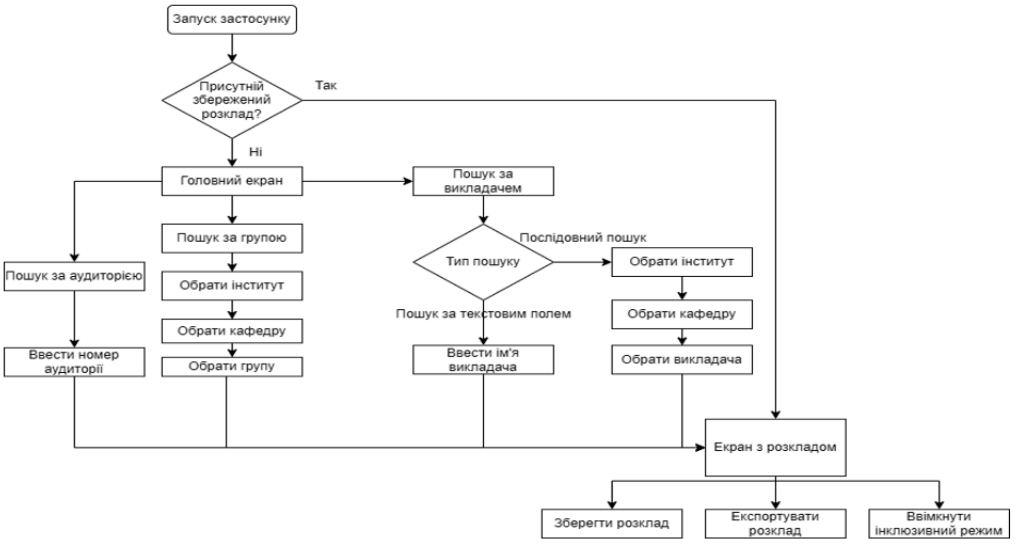
android studio

Jetpack Compose

Retrofit

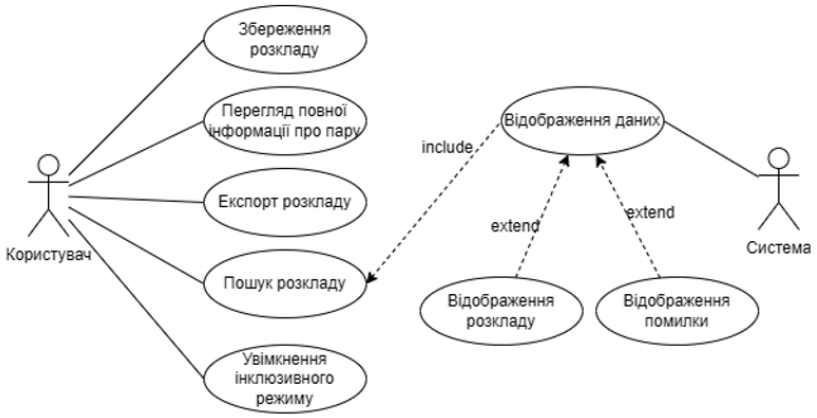
6

Блок схема роботи застосунку



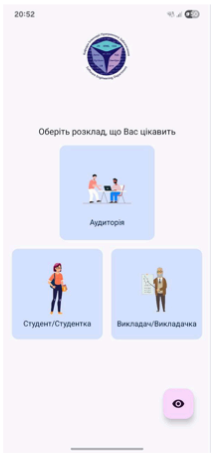
7

Діаграма варіантів використання

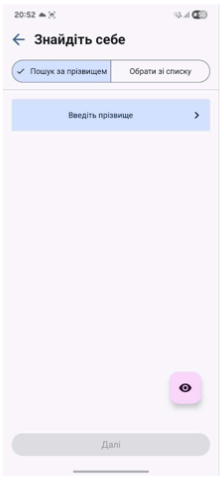


8

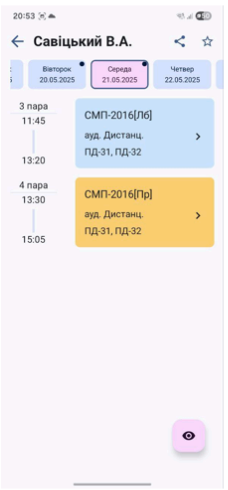
ЕКРАННІ ФОРМИ



Головний екран застосунку



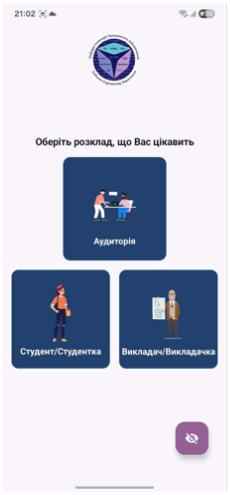
Екран вибору розкладу викладача



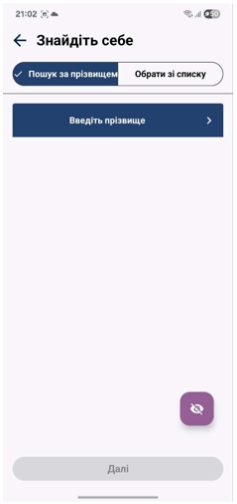
Екран з розкладом

9

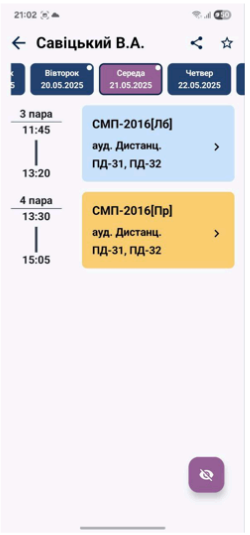
ЕКРАННІ ФОРМИ



Головний екран застосунку
в інклюзивному режимі



Екран вибору розкладу викладача
в інклюзивному режимі



Екран з розкладом в
інклюзивному режимі

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Фам Х. В., Савіцький В.А. Визначення вимог до мобільного застосунку з елементами інклюзивного дизайну для відображення розкладу університету: Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2025, ДУІКТ, м. Київ. К.: ДУІКТ, 2025. С. 601

11

ВИСНОВКИ

1. Досліджено рішення для відображення розкладу, що дозволило визначити їх переваги, недоліки та ключові функціональні можливості.
2. Сформовано функціональні та нефункціональні вимоги, які забезпечать швидкодію та адаптивність системи.
3. Розроблено архітектуру мобільного застосунку, яка забезпечує модульність, масштабованість та надійну взаємодію між компонентами системи.
4. Розроблено інтерфейс, який враховує потреби людей з порушенням зору.
5. Розроблено клієнтську частину застосунку з використанням сучасних технологій, що забезпечило швидкодію та стабільність роботи.
6. Проведено тестування, яке підтвердило працездатність застосунку, його адаптивність до різних пристроїв та відповідність сформованим вимогам.

12

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
private const val BASE_URL = "https://api.e-rozklad.dut.edu.ua/"

private val retrofit = Retrofit.Builder()
    .baseUrl(BASE_URL)
    .addConverterFactory(GsonConverterFactory.create())
    .build()

object ScheduleApi {
    val scheduleApi : ScheduleApiService by lazy {
        retrofit.create(ScheduleApiService::class.java)
    }
}

interface ScheduleApiService {
    @POST("list/faculties")
    suspend fun getFaculties(): List<Faculty>

    @POST("list/courses")
    suspend fun getCourses(@Body request: CourseRequest): List<CourseResponse>

    @POST("list/groups")
    suspend fun getGroups(@Body request: GroupRequest): List<GroupResponse>

    @POST("time-table/group")
    suspend fun getGroupSchedule(@Body request: ScheduleGroupRequest):
List<ScheduleResponse>

    @POST("other/search-teachers")
    suspend fun getTeachersByName(@Body request: SearchTeacherRequest):
List<SearchTeacherResponse>

    @POST("list/chairs")
    suspend fun getChairs(@Body request: ChairRequest): List<ChairResponse>

    @POST("list/teachers-by-chair")
    suspend fun getTeachers(@Body request: TeacherRequest): List<TeacherResponse>

    @POST("time-table/teacher")
    suspend fun getTeacherSchedule(@Body request: ScheduleTeacherRequest):
List<ScheduleResponse>
}

@RequiresApi(Build.VERSION_CODES.O)
fun handleShareClick(
    context: Context,
    scheduleType: ScheduleType,
    label: String,
    selectedDay: LocalDate,
```

```
    scheduleState: LoadState<List<ScheduleResponse>>
) {
    if (scheduleState is LoadState.Success) {
        val scheduleData = scheduleState.data
        val selectedDaySchedule = scheduleData.find {
            it.date == selectedDay.format(DateTimeFormatter.ISO_LOCAL_DATE)
        }
        shareSchedule(
            context = context,
            scheduleType = scheduleType,
            label = label,
            selectedDate = selectedDay,
            schedule = selectedDaySchedule
        )
    }
}

@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun rememberCalendarState(
    today: LocalDate = LocalDate.now()
) : Pair<CalendarState, CalendarStateHandlers> {
    val firstDayOfWeek = remember(today) { today.with(DayOfWeek.MONDAY) }
    var selectedDay by remember { mutableStateOf(today) }

    val daysList = remember {
        generateSequence(firstDayOfWeek) { it.plusDays(1) }
            .take(30)
            .toList()
    }

    val state = CalendarState(
        today = today,
        firstDayOfWeek = firstDayOfWeek,
        selectedDay = selectedDay,
        daysList = daysList
    )

    val handlers = CalendarStateHandlers(
        onDaySelected = { selectedDay = it }
    )

    return state to handlers
}

@Composable
fun rememberDialogState(): Pair<DialogState, DialogStateHandlers> {
    var selectedLesson by remember { mutableStateOf<Pair<Lesson, Period>?>(null) }

    val state = DialogState(selectedLesson = selectedLesson)
```

```
    val handlers = DialogStateHandlers(
        onLessonSelected = { lesson, period -> selectedLesson = lesson to period },
        onDismiss = { selectedLesson = null }
    )

    return state to handlers
}

@RequiresApi(Build.VERSION_CODES.O)
fun formatScheduleToShare(
    scheduleType: ScheduleType,
    label: String,
    date: LocalDate,
    schedule: ScheduleResponse?
): String {
    val dateFormatter = DateTimeFormatter.ofPattern("dd.MM.yyyy")
    val formattedDate = date.format(dateFormatter)

    val builder = StringBuilder()
    if (scheduleType is ScheduleType.Group) {
        builder.append("Група: $label\n")
    } else {
        builder.append("Викладач: $label\n")
    }
    builder.append("Дата: $formattedDate\n\n")

    if (schedule == null || schedule.lessons.isEmpty()) {
        builder.append("Немає пар")
        return builder.toString()
    }

    schedule.lessons.sortedBy { it.number }.forEach { lesson ->
        lesson.periods.forEach { period ->
            val lessonType = LessonType.fromString(period.typeStr)
            builder.append("${lesson.number} пара (${period.timeStart}-${period.timeEnd})\n")
            builder.append("${period.disciplineShortName} [${lessonType.shortName}]\n")
            if (scheduleType is ScheduleType.Group) {
                builder.append("${period.teachersName}\n")
            } else {
                builder.append("${period.groups}\n")
            }
            builder.append("ауд. ${period.classroom}\n\n")
        }
    }
    return builder.toString()
}

@RequiresApi(Build.VERSION_CODES.O)
fun shareSchedule(
    context: Context,
```

```
    scheduleType: ScheduleType,
    label: String,
    selectedDate: LocalDate,
    schedule: ScheduleResponse?
) {
    val shareText = formatScheduleToShare(
        scheduleType = scheduleType,
        label = label,
        date = selectedDate,
        schedule = schedule
    )

    val sendIntent = Intent().apply {
        action = Intent.ACTION_SEND
        putExtra(Intent.EXTRA_TEXT, shareText)
        type = "text/plain"
    }

    val shareIntent = Intent.createChooser(sendIntent, "Поділитись розкладом")
    context.startActivity(shareIntent)
}

@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun ScheduleApp() {
    val context = LocalContext.current
    val navController = rememberNavController()
    val studentViewModel: StudentViewModel = viewModel()
    val teacherViewModel: TeacherHierarchyViewModel = viewModel()
    val searchViewModel: TeacherSearchViewModel = viewModel()
    val screenViewModel: TeacherSelectionViewModel = viewModel()
    val rememberViewModel: RememberScheduleViewModel = viewModel(
        factory = RememberScheduleViewModelFactory(context.applicationContext as Application)
    )
    val inclusiveModeViewModel: InclusiveModeViewModel = viewModel(
        factory = InclusiveModeViewModelFactory(context.applicationContext as Application)
    )

    val rememberedSchedule by rememberViewModel.rememberedSchedule.collectAsState()

    LaunchedEffect(rememberedSchedule) {
        rememberedSchedule?.let { state ->
            state.type?.let { type ->
                val route = when(type) {
                    is ScheduleType.Group -> "schedule/${type.groupId}/${state.label}"
                    is ScheduleType.Teacher -> "schedule/teacher/${type.teacherId}/${state.label}"
                }
                if (navController.currentDestination?.route != route) {
                    navController.navigate(route) {
                        popUpTo("entry_screen") { inclusive = true }
```

```
                }
            }
        }
    }
}

NavHost(navController = navController, startDestination = "entry_screen") {
    composable("entry_screen") {
        EntryScreen(navController, inclusiveModeViewModel, Modifier.fillMaxSize())
    }
    composable("classroom_selection_screen") { ClassroomSelectionScreen(navController,
inclusiveModeViewModel) }
    composable("student_selection_screen") {
        StudentSelectionScreen(studentViewModel, inclusiveModeViewModel, navController)
    }
    composable("teacher_selection_screen") {
        TeacherSelectionScreen(
            navController, searchViewModel, teacherViewModel, screenViewModel,
inclusiveModeViewModel
        )
    }
    composable(
        route = "schedule/{groupId}/{groupName}",
        arguments = listOf(
            navArgument("groupId") { type = NavType.IntType},
            navArgument("groupName") { type = NavType.StringType}
        )
        ) { navBackStackEntry ->
        val groupId = navBackStackEntry.arguments?.getInt("groupId") ?: 0
        val groupName = navBackStackEntry.arguments?.getString("groupName") ?: ""

        val scheduleViewModel: ScheduleViewModel = viewModel(
            key = "schedule_$groupId"
        )
        ScheduleScreen(
            scheduleType = ScheduleType.Group(groupId),
            label = groupName,
            viewModel = scheduleViewModel,
            rememberViewModel = rememberViewModel,
            navController = navController,
            inclusiveViewModel = inclusiveModeViewModel
        )
    }
    composable("schedule/teacher/{teacherId}/{teacherName}",
        arguments = listOf(
            navArgument("teacherId") { type = NavType.IntType },
            navArgument("teacherName") { type = NavType.StringType}
        )
    ) { navBackStackEntry ->
        val teacherId = navBackStackEntry.arguments?.getInt("teacherId") ?: 0
```

```
            val teacherName = navBackStackEntry.arguments?.getString("teacherName") ?: ""

            val scheduleViewModel: ScheduleViewModel = viewModel(
                key = "schedule_teacher_$teacherId"
            )

            ScheduleScreen(
                scheduleType = ScheduleType.Teacher(teacherId),
                label = teacherName,
                viewModel = scheduleViewModel,
                rememberViewModel = rememberViewModel,
                navController = navController,
                inclusiveViewModel = inclusiveModeViewModel
            )
        }
    }
}

@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun CalendarRow (
    calendarState: CalendarState,
    scheduleState: LoadState<List<ScheduleResponse>>,
    onDaySelected: (LocalDate) -> Unit,
    inclusiveViewModel: InclusiveModeViewModel
) {
    val lazyListState = rememberLazyListState()
    val density = LocalDensity.current
    val selectedIndex = calendarState.daysList.indexOf(calendarState.selectedDay)

    LaunchedEffect(calendarState.selectedDay, lazyListState.layoutInfo.viewportSize.width) {
        if (lazyListState.layoutInfo.viewportSize.width > 0) {
            delay(50)
            lazyListState.animateScrollToItem(
                index = selectedIndex,
                scrollOffset = -(lazyListState.layoutInfo.viewportSize.width - with(density) { 90.dp.toPx()
}).toInt() / 2
            )
        }
    }

    LazyRow(
        state = lazyListState,
        modifier = Modifier
            .fillMaxWidth()
            .padding(
                start = dimensionResource(R.dimen.medium_padding),
                bottom = dimensionResource(R.dimen.medium_padding)
            ),
        horizontalArrangement = Arrangement.spacedBy(dimensionResource(R.dimen.small_padding))
```

```
    ) {
        items(calendarState.daysList.size) { index ->
            val day = calendarState.daysList[index]
            val isSelected = day == calendarState.selectedDay
            val isToday = day == calendarState.today

            val hasLessons = if (scheduleState is LoadState.Success) {
                scheduleState.data.any { schedule ->
                    schedule.date == day.toString()
                }
            } else false

            DayItem(
                date = day,
                isSelected = isSelected,
                isToday = isToday,
                hasLessons = hasLessons,
                onClick = { onDaySelected(day) },
                inclusiveViewModel = inclusiveViewModel
            )
        }
    }
}

@Composable
fun LessonDetailDialogWrapper(
    dialogState: DialogState,
    scheduleType: ScheduleType,
    onDismiss: () -> Unit,
    inclusiveViewModel: InclusiveModeViewModel
) {
    dialogState.selectedLesson?.let { (lesson, period) ->
        val lessonType = LessonType.fromString(period.typeStr)
        LessonDetailDialog(
            scheduleType = scheduleType,
            lessonFullName = period.disciplineFullName,
            lessonShortName = period.disciplineShortName,
            type = lessonType.shortName,
            number = lesson.number.toString(),
            classroom = period.classroom,
            teacherOrGroup = if (scheduleType is ScheduleType.Group) {
                period.teachersNameFull
            } else {
                period.groups
            },
            startTime = period.timeStart,
            endTime = period.timeEnd,
            onDismiss = onDismiss,
            inclusiveViewModel = inclusiveViewModel
        )
```

```
    }
}

@Composable
fun NoClassesScreen(
    modifier: Modifier = Modifier
) {
    val animations = listOf(
        R.raw.no_classes_animation_1,
        R.raw.no_classes_animation_2,
        R.raw.no_classes_animation_3,
        R.raw.no_classes_animation_4,
        R.raw.no_classes_animation_5,
        R.raw.no_classes_animation_6,
        R.raw.no_classes_animation_7,
        R.raw.no_classes_animation_8,
        R.raw.no_classes_animation_9,
        R.raw.no_classes_animation_10,
        R.raw.no_classes_animation_11
    )

    val selectedAnimation = remember {
        animations.random()
    }

    val composition by rememberLottieComposition(
        spec = LottieCompositionSpec.RawRes(selectedAnimation)
    )

    val progress by animateLottieCompositionAsState(
        composition = composition,
        iterations = LottieConstants.IterateForever
    )

    Column(
        modifier = modifier
            .fillMaxSize()
            .clip(RoundedCornerShape(dimensionResource(R.dimen.medium_padding)))
            .background(color = MaterialTheme.colorScheme.primaryContainer),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = stringResource(R.string.no_classes),
            fontSize = 28.sp,
            fontWeight = FontWeight.W500,
            color = MaterialTheme.colorScheme.onPrimaryContainer
        )
        Spacer(modifier = Modifier.height(dimensionResource(R.dimen.medium_padding)))
```

```
        Box(modifier = Modifier
            .clip(RoundedCornerShape(dimensionResource(R.dimen.medium_padding)))
            .background(color = MaterialTheme.colorScheme.surface)
            .padding(dimensionResource(R.dimen.medium_padding))
        ) {
            LottieAnimation(
                composition = composition,
                progress = { progress },
                modifier = Modifier.size(250.dp)
            )
        }
    }
}

@Composable
fun SegmentedButton(
    modifier: Modifier = Modifier,
    selectedScreen: Int,
    onOptionSelected: (Int) -> Unit,
    inclusiveViewModel: InclusiveModeViewModel
) {
    var selectedIndex by remember { mutableIntStateOf(selectedScreen) }
    val options = listOf("Пошук за прізвищем", "Обрати зі списку")

    SingleChoiceSegmentedButtonRow(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = dimensionResource(R.dimen.medium_padding))
    ) {
        options.forEachIndexed { index, option ->
            SegmentedButton(
                selected = index == selectedIndex,
                onClick = {
                    selectedIndex = index
                    onOptionSelected(index)
                },
                colors = SegmentedButtonDefaults.colors(
                    activeContainerColor = MaterialTheme.colorScheme.primaryContainer
                ),
                shape = SegmentedButtonDefaults.itemShape(index = index, count = options.size)
            ) {
                InclusiveText(
                    text = option,
                    viewModel = inclusiveViewModel,
                    style = TextStyle(fontSize = 14.sp)
                )
            }
        }
    }
}
```

```
class InclusiveModeViewModel(
    context: Application
) : AndroidViewModel(context) {
    private val dataStore = context.dataStore
    private val INCLUSIVE_MODE = booleanPreferencesKey("inclusive_mode_enabled")

    private val _isInclusiveModeEnabled = MutableStateFlow(false)
    val isInclusiveModeEnabled: StateFlow<Boolean> = _isInclusiveModeEnabled.asStateFlow()

    init {
        viewModelScope.launch {
            dataStore.data.collect { preferences ->
                _isInclusiveModeEnabled.value = preferences[INCLUSIVE_MODE] ?: false
            }
        }
    }

    fun toggleInclusiveMode() {
        viewModelScope.launch {
            val newValue = !_isInclusiveModeEnabled.value
            dataStore.edit { preferences ->
                preferences[INCLUSIVE_MODE] = newValue
            }
        }
    }
}

private val Application.dataStore: DataStore<Preferences> by preferencesDataStore(name =
"schedule_preferences")
class RememberScheduleViewModel(
    context: Application
) : AndroidViewModel(context) {
    private val dataStore = context.dataStore

    private val _rememberedSchedule = MutableStateFlow<RememberedScheduleState?>(null)
    val rememberedSchedule: StateFlow<RememberedScheduleState?> =
_rememberedSchedule.asStateFlow()

    private val _showDialog = MutableStateFlow(false)
    val showDialog: StateFlow<Boolean> = _showDialog.asStateFlow()

    init {
        viewModelScope.launch {
            loadRememberedSchedule()
        }
    }

    private suspend fun loadRememberedSchedule() {
        dataStore.data.firstOrNull()?.let { preferences ->
```

```
            val type = when(preferences[PreferenceKeys.SCHEDULE_TYPE]) {
                "group" -> preferences[PreferenceKeys.GROUP_ID]?.let { ScheduleType.Group(it) }
                "teacher" -> preferences[PreferenceKeys.TEACHER_ID]?.let { ScheduleType.Teacher(it) }
                else -> null
            }
            val label = preferences[PreferenceKeys.LABEL] ?: ""
            _rememberedSchedule.value = RememberedScheduleState(type, label)
        }
    }

    fun showRememberDialog() {
        _showDialog.value = true
    }

    fun hideDialog() {
        _showDialog.value = false
    }

    fun rememberSchedule(type: ScheduleType, label: String) {
        viewModelScope.launch {
            dataStore.edit { preferences ->
                when (type) {
                    is ScheduleType.Group -> {
                        preferences[PreferenceKeys.SCHEDULE_TYPE] = "group"
                        preferences[PreferenceKeys.GROUP_ID] = type.groupId
                    }
                    is ScheduleType.Teacher -> {
                        preferences[PreferenceKeys.SCHEDULE_TYPE] = "teacher"
                        preferences[PreferenceKeys.TEACHER_ID] = type.teacherId
                    }
                }
                preferences[PreferenceKeys.LABEL] = label
            }
            _rememberedSchedule.value = RememberedScheduleState(type, label)
        }
        hideDialog()
    }

    fun forgetSchedule() {
        viewModelScope.launch {
            dataStore.edit { preferences ->
                preferences.clear()
            }
            _rememberedSchedule.value = null
        }
        hideDialog()
    }

    private object PreferenceKeys {
        val SCHEDULE_TYPE = stringPreferencesKey("schedule_type")
```

```
        val GROUP_ID = intPreferencesKey("group_id")
        val TEACHER_ID = intPreferencesKey("teacher_id")
        val LABEL = stringPreferencesKey("label")
    }
}

class RememberScheduleViewModelFactory(
    private val application: Application
) : ViewModelProvider.Factory {
    override fun <T: ViewModel> create(modelClass: Class<T>): T {
        if(modelClass.isAssignableFrom(RememberScheduleViewModel::class.java)) {
            @Suppress("UNCHECKED_CAST")
            return RememberScheduleViewModel(application) as T
        }
        throw IllegalArgumentException("Unknown ViewModel class")
    }
}

@RequiresApi(Build.VERSION_CODES.O)
class ScheduleViewModel : ViewModel() {
    private val _state =
MutableStateFlow<LoadState<List<ScheduleResponse>>>(LoadState.Loading)
    val state: StateFlow<LoadState<List<ScheduleResponse>>> = _state.asStateFlow()

    private var currentScheduleType: ScheduleType? = null
    fun loadSchedule(scheduleType: ScheduleType, startDate: LocalDate, endDate: LocalDate) {
        viewModelScope.launch {
            _state.value = LoadState.Loading
            _state.value = try {
                currentScheduleType = scheduleType
                val schedule = when (scheduleType) {
                    is ScheduleType.Group -> {
                        val request = ScheduleGroupRequest(
                            groupId = scheduleType.groupId,
                            dateStart = startDate.format(DateTimeFormatter.ISO_LOCAL_DATE),
                            dateEnd = endDate.format(DateTimeFormatter.ISO_LOCAL_DATE)
                        )
                        scheduleApi.getGroupSchedule(request)
                    } is ScheduleType.Teacher -> {
                        val request = ScheduleTeacherRequest(
                            teacherId = scheduleType.teacherId,
                            dateStart = startDate.format(DateTimeFormatter.ISO_LOCAL_DATE),
                            dateEnd = endDate.format(DateTimeFormatter.ISO_LOCAL_DATE)
                        )
                        scheduleApi.getTeacherSchedule(request)
                    }
                }
                LoadState.Success(schedule)
            } catch (e: Exception) {
                LoadState.Error
```

```
            }
        }
    }

    override fun onCleared() {
        super.onCleared()
        _state.value = LoadState.Loading
    }
}

class StudentViewModel : ViewModel() {
    private val _state = MutableStateFlow<LoadState<StudentSelectionData>>(LoadState.Loading)
    val state: StateFlow<LoadState<StudentSelectionData>> = _state.asStateFlow()

    private var isInitialized = false
    fun initialize() {
        if(!isInitialized) {
            isInitialized = true
            loadFaculties()
        }
    }

    private fun loadFaculties() {
        viewModelScope.launch {
            _state.value = LoadState.Loading
            _state.value = try {
                val faculties = scheduleApi.getFaculties()
                LoadState.Success(StudentSelectionData(faculties = faculties))
            } catch(e: Exception) {
                isInitialized = false
                LoadState.Error
            }
        }
    }

    fun onFacultySelected(faculty: Faculty) {
        viewModelScope.launch {
            _state.value = try {
                val currentState =
                    (_state.value as? LoadState.Success)?.data ?: return@launch
                val courses = scheduleApi.getCourses(CourseRequest(faculty.id))
                LoadState.Success(
                    currentState.copy(
                        selectedFaculty = faculty,
                        courses = courses,
                        groups = emptyList(),
                        selectedCourse = null,
                        selectedGroup = null
                    )
                )
```

```
            } catch(e: Exception) {
                isInitialized = false
                LoadState.Error
            }
        }
    }

    fun onCourseSelected(course: CourseResponse) {
        viewModelScope.launch {
            _state.value = try {
                val currentState =
                    (_state.value as? LoadState.Success)?.data ?: return@launch
                val groups = scheduleApi.getGroups(
                    GroupRequest(currentState.selectedFaculty!!.id, course.course)
                )
                LoadState.Success(
                    currentState.copy(
                        selectedCourse = course,
                        groups = groups,
                        selectedGroup = null
                    )
                )
            } catch(e: Exception) {
                isInitialized = false
                LoadState.Error
            }
        }
    }

    fun onGroupSelected(group: GroupResponse) {
        val currentState = (_state.value as? LoadState.Success)?.data ?: return
        _state.value = LoadState.Success(currentState.copy(selectedGroup = group))
    }

    override fun onCleared() {
        super.onCleared()
        isInitialized = false
        _state.value = LoadState.Loading
    }
}
```