МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЗОРИЧ СЕРГІЙ ДМИТРОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
доктор техн. наук, доцент
_____ Т. В. Нескородєва
«_____» _____ 2022р.

РОЗРОБКА СИСТЕМИ ІНФОРМАЦІЙНОЇ ПІДТРИМКИ ПРОЦЕСУ ЗАХИСТУ КУРСОВИХ ТА КВАЛІФІКАЦІЙНИХ РОБІТ

Спеціальність 122 «Комп'ютерні науки»
Кваліфікаційна (магістерська) робота

Керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій
к. ф-м. н, доцент
_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2022

ЗМІСТ

АНОТАЦІЯ

Зорич С.Д. «Розробка системи інформаційної підтримки процесу захисту курсових та кваліфікаційних робіт» Спеціальність 122 «Комп'ютерні науки» Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційній (магістерській) роботі досліджено технологію розробки мобільних додатків, а саме клієнтської частини та розробку серверної частини. За допомогою мови програмування Kotlin та Android SDK було розроблено мобільний додаток та за допомогою мови програмування Kotlin та бібліотеки Ktor було розроблено серверну частину системи інформаційної підтримки захисту курсових та кваліфікаційних робіт, яка дозволяє користувачеві додавати роботи, задавати питання, виставляти оцінки, тощо. Передбачено декілька видів користувачів з різним функціоналом. Розглянуто види нормалізації баз даних. Розроблено базу даних що відповідає необхідним функціям, наведено схему бази даних. Також було розроблено механізм перевірки робіт на плагіат.

Ключові слова: мобільний додаток, Figma, сервер, Ktor, дизайн, компоненти, Android SDK, HTTP, бази даних, нормалізація баз даних, перевірка на плагіат.

112с., 38 рис., 63 джерела.

ANNOTATION

Zorych S.D. "Development of an information support system for the process of defending course and qualification papers" Specialty 122 "Computer Science" Vasyl Stus Donetsk National University, Vinnytsia, 2022.

In the qualification (master's) work, the technology of developing mobile applications, namely the client part and the development of the server part, was investigated. A mobile application was developed using the Kotlin programming language and the Android SDK, and the server part of the information support system for the protection of coursework and qualification papers was developed using the Kotlin programming language and the Ktor library, which allows the user to add works, ask questions, set grades, etc. Several types of users with different functionality are provided. Types of database normalization are considered. A database that corresponds to the necessary functions has been developed, and the database scheme is given. A mechanism for checking works for plagiarism was also developed.

Keywords: mobile application, Figma, server, Ktor, design, components, Android SDK, HTTP, databases, database normalization, plagiarism checker.

112 pages, 38 pictures, 63 sources.

ВСТУП

Смартфони значно спростили та прискорили життя людини. З розвитком технологій, що лежать в основі смартфонів, розвивається і індустрія мобільних додатків. З більш ніж 2,96 мільйонами програм у магазині Google Play (і продовжує рости), Android набирає обертів з моменту свого запуску, випередивши своїх конкурентів з часткою ринку понад 85%. Рік за роком Android продовжує приносити нові ідеї, інновації, методи та інструменти для розробки мобільних додатків. Майбутнє розробки додатків для Android виглядає дуже перспективним і допомагає розробникам залишатися попереду конкурентів.

Впровадження Android зростало семимильними кроками, в основному завдяки покращеному інтерфейсу. Прийняття тенденцій завтрашнього дня необхідно, щоб залишатися попереду в індустрії, що постійно змінюється, розробки мобільних додатків. Перш ніж ви вирішите найняти професійного розробника мобільних програм для Android, ви повинні ознайомитися з останніми тенденціями розробки програм для Android.

Навіщо мобільні додатки:

- Організація продажів і стимулювання повторних покупок.
- Підвищення лояльності покупців і клієнтів.
- Автоматизація бізнес-процесів.
- Аналіз аудиторії на основі різних даних.
- Прийом платежів.
- Надання сервісу нового рівня і т.д.
- Спрощення освіти

Через карантин та пандемію COVID-19, рекордно зросла кількість користувачів мобільних додатків. Тижні та місяці в ізоляції й карантині змусили бізнесу та навчання перейти в онлайн, що сильно вплинули на звички та спосіб життя людей. Люди частіше купують в інтернеті, замовляють їжу в інтернеті, проводять більше часу зі своїми смартфонами і витрачають більше часу граючи в ігри. Саме це призвело до зростання ринку мобільних додатків. Порівнюючи з 2019 роком, у 2020 році на 40% зріс час, який люди витрачають на мобільні додатки щомісяця. Ці тенденція зберігається й навіть зростає у 2021 році. Також зросла кількість завантажень мобільних додатків, а саме: мобільні ігри (43%), додатки для бізнесу (105%) та доставка їжі (73%). Також у 2020-й зросла кількість відеоконференцій. Завантаження додатку Zoom для відеоконференцій зросла на 2000% порівнюючи з 2019 роком. Мобільні додатки перетворилися в найбільшу споживчу систему що існує на планеті. За прогнозами, у 2021 році світова виручка від мобільних додатків має досягнути $6,3 трлн.[1]

Створення платформи для навчання в умовах COVID-19 є дуже актуальною темою. За допомогою цієї платформи люди можуть не виходячи з дому проходити курси для саморозвитку, поліпшенню професійних навиків та навпаки, люди – що можуть надати корисну інформацію шляхом публікування своїх курсів на платформі.

Переваги онлайн освіти:

- Доступ до програм найкращих викладачів світу: онлайн-освіта дає можливість навчатися скрізь, де є інтернет.
- Найновіша інформація, технології, теорії: матеріал онлайн-курсів оновлюється динамічніше, ніж офлайнові програми.
- Безкоштовне чи доступніше за ціною навчання, ніж звичайне денне навчання в університеті.
- Можливість навчатись у будь-якому місці та у бідь-який час.

- Можливість поєднувати різні формати та підходи до навчання.

Недоліки онлайн освіти:

- Відсутність живого контакту з викладачем та групою.
- Самомотивація та самоорганізації є досить складною. Саме це є наслідком низького відсотку завершення курсів (у світі 7%, в Україні – 15%).
- Недостатня кількість спеціалізованих матеріалів вищого рівня складності: більшість матеріалів створені для вступного рівня – щоб охопити якомога більшу аудиторію.
- Закінчення курсів не гарантує наявності потрібних знань і навичок.

Актуальність теми дослідження. Сьогодні ринок мобільних додатків є найпопулярнішою тенденцією IT-ринку. Сучасні мобільні додатки маю кілька вирішальних переваг: це надзвичайна швидкість та зручність під час користування. Кількість технологій та варіацій реалізації постійно зростає і саме це робить непростим створення мобільного додатку. Отже саме ця тема найактуальнішою на сьогодні.

Мета дослідження. Розробка клієнтської та серверної частини системи інформаційної підтримки процесу захисту курсових та кваліфікаційних робіт. Додаток має бути візуально приємним, інтуїтивно зрозумілим та багато функціональним.

Завдання дослідження:

- Вивчити та з’ясувати, як працюють Figma, Kotlin, Android SDK, Android Studio, Android Emulator, Firebase Authentication, Firebase Storage, Ktor, Rest API, WebSocket.
- Визначити як розробляти мобільні додатки для Android використовуючи Android SDK та мову програмування Kotlin.

- Визначити як розробляти серверну частину додатку виростовуючи Ktor та мову програмування Kotlin.
- Розробити серверну частину системи інформаційної підтримки процесу захисту курсових та кваліфікаційних робіт
- Розробити клієнтську частину системи інформаційної підтримки процесу захисту курсових та кваліфікаційних робіт

Об’єкт дослідження. Методи проектування, використання технології для створення серверної частини та мобільних додатків для Android.

Предмет дослідження. Популярні та сучасні інструменти, бібліотеки та фреймворки, що дозволяють створювати серверну частину. Популярні та сучасні інструменти, бібліотеки та фреймворки, що дозволяють створювати мобільні додатки під Android.

Досягнення поставленої мети потребує вирішення таких **задач дослідження**:

- Розглянути поняття, що таке мобільний додаток.
- Проаналізувати та обрати інструменти.
- Розробити прототип та дизайн додатку.
- Розробити структуру бази даних
- Визначити моделі даних та інтегрувати бізнес логіку у проект.
- Створити серверну частину, що буде задовольняти меті роботи.
- Створити мобільний додаток для Android, що буде задовольняти меті роботи.

Робота складається зі вступу, трьох розділів зі своїми підрозділами, висновку та списку літературних джерел кількістю 63 найменування та 38 рисунків. Загальний об’єм роботи 112 сторінок.

РОЗДІЛ 1

ПОНЯТТЯ МОБІЛЬНОГО ДОДАТКУ

Даний розділ містить опис поняття «мобільний додаток» та серверна розробка, що таке API і його види, окреслені їх переваги та недоліки.

1.1 Поняття мобільного додатку

Мобільний додатком є програмним забезпечення, призначене для роботи на телефонах, планшетах та інших мобільних пристроях. Багато мобільних додатків встановлено на самому пристрої або їх можна завантажити з онлайн магазинів мобільних додатків: Google Play, App Store, Windows Phone Store та інших, безкоштовно або за окремуплату.

Насамперед мобільні додатки використовували для швидкої та зручної перевірки електронної пошти, тощо. Про високий попит призвів до розширення їх функціоналу та призначень зовсім в інших галузях, приклад ігри, GPS, спілкування, перегляд відео та доступ до інтернету.

Мобільні додатки переслідують, щонайменше дві явні цілі: не дати користувачу нудьгувати та спростити побут.

Необхідність мобільного додатку – недоречно навіть обговорювати, через те, що поширення доступних мобільних пристроїв, зручного сенсорного екрану, вільний, або взагалі безкоштовний, вихід у мережу інтернет, легкий та швидкий обмін інформацією – збільшило аудиторію потенційних клієнтів будь-якого бізнесу.

1.2 Сучасні мобільні додатки

Всі мобільні пристроїв продаються з уже встановленим набором додатків, наприклад набір додатків від Google, а саме Gmail, Google Drive, Youtube, Google Chrome та інші. Попередньо встановлені додатки, можуть бути вилучені з

мобільного пристрою, але не завжди. За це відповідає компанія, що виробляє мобільний пристрій.

Користувач може сам встановлювати необхідні йому додатки через платформи їх розповсюдження. Платформою розповсюдження є магазин додатків. Перші магазини з'явилися у 2008 році, та зазвичай, керуються компанією, що розробляє та володіє мобільною операційною системою, наприклад: Apple - App Store, Google - Gooogle Play, Windows - Windows Phone Store. Але існують незалежні магазини додатків, такі як Cydia на iOS, Get Jar або F-Droid на Android. Існують платні та безкоштовні мобільні додатку. Більшість додатків є безкоштовними для користувачів.

Переваги мобільних додатків:

- Взаємодія з користувачами. У мобільному додатку є можливість відправляти Push-повідомлення, на відміну від сайтів. Недолік сайтів полягає у тому що, користувачі дізнаються про щось нове (наприклад новий товар) лише тоді, коли зайдуть на сайт. У мобільних додатках цю проблему вирішують push – повідомлення. Вони надають змогу відправляти усім користувачам безкоштовні повідомлення, без обмеження по символам.

- Локальні сповіщення працюють однаково, лише встановлюються на самому пристрої. Наприклад, якщо користувач зробив якусь дію у додатку, яка очікує від нього реакції через певний час, додаток може нагадати йому про це.

- Широкі можливості зворотного зв'язку – користувачі можуть залишити повідомлення, відгуки у магазинах додатків, відправляти повідомлення через додаток або у соцмережах і вони автоматично транслюватимуться на всі ресурси.

- Інтерфейс. Всі елементи управління: кнопки, текстові поля, посилання мають бути зручними для натискання на мобільному. Мобільні операційні системи мають для цього свою спеціальну логіку.

- Навігація. Мобільна операційна система має свою логіку переходів між екранами у додатках. Користувач кожної мобільної операційної системи звик до тієї ж поведінки та логіки переходів у кожному додатку. Проте навігація ж на кожному сайті зроблена по-своєму, і потративши на черговий сайт, потрібно кожного разу шукати кнопки «Ок», «Назад», «Скасування», тощо, яких взагалі може й не бути.

- Рівень персоналізації. Однією з ключових причин бурхливого зростання мобільних додатків э те, що мобільні телефони знають про свого власника дуже багато, і використовую цю інформації для покращення рівня сервісу. Мобільні додатки мають можливість запам’ятовувати дані користувача і змінювати інтерфейс додатку в залежності від його потреб.

- Дозволи. Користувач сам вирішує якому додатку надати дозволи, а якому ні. Наприклад дозволи до доступу мережі інтернету, камери, акселерометра, компаса, барометра, тощо.

- Локація. Мобільний додаток може визначити місце розташування користувача і, наприклад, відобразити список ресторанів, до яких користувач має найменшу відстань.

- Робота в офлайн режимі. Користувачі не завжди мають доступ до якісної та швидкої мережі інтернет. Навіть при наявності мобільного інтернету, його якість в середньому гірша, ніж якість домашньої чи офісної мережі інтернет. Якщо поставлено завдання, щоб користувач не втрачав зв’язок з додатком відразу, як тільки обірвався інтернет-зв’язок і міг далі користуватись додатком, то розробка саме мобільного додатку з режимом роботи, який не потребує доступ до мережі інтернет – це єдине можливе рішення;

- Ознака престижу. Розробка мобільного додатку компанією є ознакою того, що компанія дбає про своїх клієнтів та користувачів, даючи їм свободу вибору платформи користування та вищий рівень комфорту, наприклад, коли потрібно побудувати тривалий і якісний зв’язок з користувачами.

Недоліки мобільних додатків:

- Встановлення мобільних додатків. Якщо розмістити QR-коди на сайт і на мобільний додаток, то переходів на сайт буде завжди більше ніж встановлень мобільного додатку. Більшість користувачів після переходу за посиланням на додаток у магазин, не встановлюють його.

- Платформа. Кожний мобільний додаток створюється під певну мобільну операційну систему і може бути запущений тільки на цій операційній системі, на відміну від сайту, який може бути запущений на будь-якій платформі, або пристрої.

- Оновлення. Оновлення мобільного додатку в магазинах завжди займає деякий час на їх перевірку адміністрацією магазину на відміну від сайту. Після оновлення сайту користувачі образу можуть їм користуватись

- Вартість. Вартість розробки й підтримки мобільного додатку майже завжди вище, ніж розробка та підтримка роботи сайту.

- Маркетинг. Мобільні додатки просувати, значно, дорожче аніж сайт. Ціна одного нового користувача безпосередньо залежить від порогу його входження (наприклад вік, користування іншими додатків, країна проживання), чим він вищий - тим вища ціна.

Спершу мобільні додатки були інструментами для оперування та контролю загальних потоків інформації, враховуючи календар, електронну пошту, контакти, інформацію про погоду та фондовий ринок. Так чи інакше, доступність інструментів та попит з боку розробників призвели до швидкого розповсюдження програм для інших категорій електронних пристроїв, що функціонують за допомогою настільних програм. Різноманітність та багатство мобільних додатків породили величезну кількість ресурсів знань про них. Зокрема, блог, журнал та спеціалізований онлайн-сервіс пошуку додатків з оглядами, рекомендаціями та відгуками.

Мобільні програми стають все більш популярними серед мобільних користувачів. За даними App Annie, у 2017 році кількість завантажень додатків зросла на 60%, а споживчі витрати збільшилися більш ніж удвічі. Кожен користувач проводить у додатку близько 43 днів на рік. Загальна кількість завантажень додатків у 2019 році збільшилась до 115 мільярдів. 84 мільярди в Google Play і 31 мільярд в App Store. Дослідники виявили, що використання мобільного додатка позитивно корелює з його змістом і залежить від точного розташування користувача та часу доби.

1.3 Поняття серверної розробки

Серверна-розробка - це набір програмно-апаратних засобів, що реалізують логіку вашого сайту. Простіше кажучи, це приховано від користувача і відбувається поза браузером або комп'ютером.

Наприклад, введення запиту на сторінці пошукової системи та натискання Enter призведе до виходу із зовнішнього інтерфейсу та запуску внутрішнього інтерфейсу. Запити надсилаються на сервери Google або іншої пошукової системи, де розташовані алгоритми пошуку. Тут відбуваються всі "дива". Як тільки потрібна інформація з'явиться на моніторі, знову за логіку роботи відповідає fronted зона.

За великим рахунком, сервер це той самий комп'ютер, тільки потужніший. Зберігає дані та відповідає на запити користувачів.

Backend-розробники використовують інструменти, доступні на сервері. Він має право вибрати одну з універсальних мов програмування, таких як Ruby, PHP, Python, Java та інші. Все залежить від конкретного проекту та завдань замовника.

Системи управління базами даних також використовують для серверної розробки:

1. MySQL

2. PostgreSQL
3. SQLite
4. MongoDB

Залежно від продукту обов'язки бекенд-розробника сильно різняться. Ці фахівці можуть створювати та консолідувати бази даних, забезпечувати безпеку або налаштовувати технології резервного копіювання та відновлення.

Фронтенд та бекенд працюють по колу:

frontend відправляє інформацію користувача на backend сервер;

сервер обробляю цю інформацію;

сервер надсилаю відповідь fronted’у в зрозумілій, для нього, формі.

У цьому беруть участь різні професіонали, і бажано, щоб кожен із них розумів принципи роботи свого колеги. Навіть якщо ви дизайнер інтерфейсів (дизайнер інтерфейсу користувача), принаймні в цілому, важливо знати, що таке бекенд для проекту, над яким ви працюєте. Це допомагає правильно оцінити технічні можливості сайту чи програми.

Взаємодії між клієнтами та серверами створюються за допомогою HTTP-запитів, що надсилаються безпосередньо на сервер, який витягує та вбудовує інформацію у шаблони та повертає її у вигляді HTML-сторінок.

Обов'язки front-end і back-end розробників зазвичай розділені, але інколи програмісти вирішують проблеми як із боку сервера, так і з боку фронтенду. Обидва типи розробки включають як технічні, так і творчі елементи. На ринку часто є фахівці, які впевнені як у фронтенді, так і в бекенді та вміють їх поєднувати.

1.4 Що таке API ?

API (Application Programming Interface) - інтерфейс прикладного програмування, набір методів, класів, бібліотек та функцій, що забезпечують взаємодію програм.

Ця розробка призначена для спрощення роботи програмістів. Намагаючись визначити його простими словами, API - це корисний інструмент, що є набором класів, функцій, процедур і стандартів, які дозволяють додаткам ефективно взаємодіяти один з одним. Програмісти використовують цей механізм під час створення різних систем.

Де використовують API?

Швидка реєстрація в додатку через обліковий запис у соціальній мережі. За допомогою спеціальних протоколів API соціальних мереж Facebook, Telegram та інших платформ, користувачі можуть легко отримати можливість користуватися продуктами компанії, швидко зареєструвавшись на сайті через авторизацію облікового запису.

Google. Система використовує інтерфейси програмування, щоб надати розробникам різних програм доступ до інформації та можливість інтеграції різних сервісів. Наприклад, ви можете знайти та переглянути відео з платформи YouTube у своїй програмі.

API у вигляді готового продукту. Розробник надає доступ до програми для отримання оперативних даних зведення погоди в будь-який момент часу на Землі.

Основний функціонал API

Популярність API обумовлена їх простотою використання та функціональністю. Програмістам не потрібно вивчати внутрішню роботу цього програмного інтерфейсу. З його допомогою дуже легко поєднувати програми в одну систему.

Принцип роботи механізму API полягає у організації багаторівневої ієрархії, у якій створюються підкомпоненти з однаковою структурою. Побудовано стандартну модель мережі OSI з певною кількістю кроків (не менше 7). Внутрішні рівні розділені на категорії, що різняться між додатками HTTP, IMAP, фізичними широкомовними рівнями тощо. Ця структура дозволяє використовувати функціональні можливості нижчих API в інтерфейсі для операцій більш високих API.

Бібліотеки функцій та класів з сигнатурами та семантичними описами створені для ефективної організації вашої роботи. Сигнатура у разі є частиною оголошення функції, яка ідентифікує елемент. Для представлення та визначення ймовірності перезавантаження можна використовувати різні мови програмування. Написавши мову виклику, експерт поділяє сигнатуру виклику та реалізацію заданого функціоналу. Підпис визначається з урахуванням сфери дії та послідовності фактичних типів аргументів. Такі компоненти дозволяють компілятор розпізнавати функції при роботі з мовою C++. Якщо це метод певного класу, підпис буде включено до імені зазначеного класу.

Семантика функції визначає її роботу та принципи роботи. Описує результат обчислення та властивості, які залежать від його отримання. Тобто в таких моделях результат залежить не лише від аргументів, а й від фактичного стану. У той же час, можливість отримувати інформацію через API-з'єднання менш важлива.

Бібліотеки використовуються для створення програм та програм, створення сервісів для обслуговування клієнтів і так далі.

Типи API

При розробці сайту з використанням API або інших продуктів вибираються типи інтерфейсів, які підходять для вирішення конкретного завдання. Програмні інтерфейси класифікуються за функціями, призначенням, виконуваним

завданням та переліком функцій. Існують стандартні продукти та альтернативні рішення, які можуть вирішити ту саму проблему іншими методами.

Існують глобальні продукти з окремими мовами програмування, призначені для вирішення локальних завдань. Модулі додатків (wxWidgets, Qt, GTK тощо), операційні системи (Amiga ROM Kernel, POSIX, Linux Kernel API ruen, Cocoa, OS/2 API, Windows API), звуки (DirectMusic/DirectSound, OpenAL), віконні інтерфейси і т.д. . Графічний API призначений для підвищення яскравості та різкості зображень у комп'ютерних іграх та різних програмах з якісною візуалізацією. Чим складніша система інтерфейсу, тим більша ймовірність технічних складнощів при роботі з додатком. Можуть виникнути такі проблеми:

- складність перенесення програмного коду при зміні API у системі. Такі проблеми можуть виникати під час перенесення модулів на іншу ОС.;
- сменшення списку функцій системи при застосуванні в іншому додатку (наприклад, при переході в більш керовану систему). Полегшуються типові завдання певного класу, але губляться деякі можливості (доступ до контролю над іншими регуляторами та ін.). Тобто управління базовими елементами стає зручнішим і легким, але частина опцій залишається недоступною.

У API існує політика випуску, яка визначає ступінь доступності використання технології різними типами користувачів. Виділяють делька політик API:

- Private – доступний лише для внутрішнього користування;
- «Партнер» - доступ до технології мають лише окремы діловы партнере;
- Public - API є публічним і відкритим для будь-якого користувача.

API пошукових систем та веб-спеціалістів

Майстри, які займаються програмуванням і дизайном сайтів і їх просуванням, використовують спеціальний Web API. Це інтерфейси, які містять

набір конкретних запитів HTTP. Отримавши такий запит, модуль генерує HTTP-відповідь із певною структурою. Для передачі інформації між ними використовуються формати XML або JSON. У цій області застосування Web API є синонімом веб-сервісу, конкретної програми з відповідним інтерфейсом. Для доступу до цих модулів потрібна процедура ідентифікації за онлайн-адресою в Інтернеті. Іншими словами, якщо вам потрібно передати дані на сервер, вам слід використовувати модуль взаємодії Server API.

Коли розробники вибудовують програмні системи на базі сервіс-орієнтованої структури, веб-служба виступає рівнем, де формуються модулі. Це звичні для кожного користувача онлайн-сервіси - електронна пошта, файлообмінник, закладки соціальних мереж тощо. Для перевірки ефективності роботи програми розробники надають тестовий механізм інтерфейсу. Такі програмні системи можуть виконувати своє призначення незалежно від типу десктопного або мобільного пристрою, виду браузера.

Одним із прикладів API в інтернет-рекламі є додаток, яке використовує «Яндекс.Директ». Для настройки і більш ефективного управління рекламними кампаніями створюються спеціальні модулі, які дозволяють поліпшити параметри пошукової оптимізації (SEO) шляхом інформаційної взаємодії.

Щоб спростити вибір інтерфейсу, розробники намагаються включити його призначення та основну функцію в назву (наприклад, API під назвою syngestureapisampleapp додаток було створено для роботи одного користувача).

Вибираючи найкращу програму, веб-майстри повинні враховувати зміни, які спостерігаються після масового впровадження стандартів Web 2.0. Інновації в протоколі обміну структурованими даними (розподіл в обчислювальних середовищах, пов’язаних з доступом до об’єктів) у SOAP. Ці протоколи зведені до спрощеного архітектурного стилю. Це дозволило прискорити процес виконання зазначених дій на інтернет-магазинах та інших інтернет-ресурсах з великими обсягами інформації. Тому розробники при виборі програми

вирішують, який інтерфейс використовувати для автоматизації всіх ключових процесів.

Окремі компоненти системи взаємодіють один з одним за аналогією з підключенням між серверами та користувачами Інтернету. Незважаючи на відсутність єдиного стандарту, системи на основі архітектури REST реалізуються за допомогою традиційних моделей HTTP, URL, JSON і XML. Такий підхід дозволяє додавати та розширювати функціональність вашої програми.

1.4 Платформа Android

Android — це операційна система та платформа для мобільних телефонів, планшетних комп'ютерів, електронних книг, цифрових плеєрів, годинників, фітнес-браслетів, ігрових приставок, ноутбуків, нетбуків, Google Glass, телевізорів та інших пристроїв.

Він заснований на ядрі Linux і власній реалізації віртуальної машини Java від Google. Спочатку розроблено Android Inc., а пізніше придбано Google. Android дозволяє писати програми Java, які керують пристроєм за допомогою бібліотек, розроблених Google.

В даний час Android є найпопулярнішою операційною системою у світі. Згідно з офіційними даними агрегатора статистичних даних Statcounter, до кінця 2020 року на операційну систему Android припадатиме 39,77% всіх пристроїв, як настільних, так і мобільних, а на Windows - 32,31%. %, IOS – 17,66 %, OS X – 7,98 %. Це не повинно викликати подиву. Тому що в середньому у світі вони дешеві, доступні, прості у використанні і їх вистачає багатьом користувачам, тому вони можуть дозволити собі купувати пристрої з ОС Android більше, ніж персональні комп'ютери та ноутбуки. Функціонал мобільних гаджетів, що призначені для повсякденної роботи. У мобільних телефонах Android має значну частку – 70,43%, а IOS – 29,06%. [6]

Android є простим та зручним і з боку розробки програмного забезпечення для мобільних пристроїв. Існує безліч зручних середовищ розробки для

програмування під Android, включаючи Android Studio, IntelliJ, NetBeans IDE та Visual Studio, які підтримують такі популярні мови програмування, як Java, Kotlin та C#. Кожен розробник може опублікувати свою програму в Play Market, а завдяки популярності платформи потенційні клієнти досить великі і приносять більший прибуток.

До основних переваг Android відносяться:

- Відкритий вихідний код. Будь-який кваліфікований програміст може створювати програми та завантажувати їх до офіційного магазину Android Play Market.

- Зручність передачі даних - коли користувачі хочуть завантажити улюблені пісні, відео або інші файли на згадку про свій смартфон, вони можуть підключити гаджет до комп'ютера за допомогою USB-кабелю і переносити файли з однієї папки в іншу, просто переносити. Не потрібно "воювати" з додатковими драйверами або іншими програмами.

- Ціна – знайдіть недорогі Android-смартфони для всіх. Завдяки тому, що Android доступний як пенсіонерам, так і студентам компанія повністю домінує на ринку ОС.

- Зручна та швидка синхронізація даних з іншими пристроями.

- Play Market – офіційний магазин додатків, більша частина яких є повністю безкоштовною. Магазин є надійним джерелом, оскільки всі програми проходять перевірку на злоякісний код та віруси, тому це дає майже стовідсоткову гарантію безпеки пристрою та особистих даних. Хоча і були випадки проникнення шкідливих програм і оновлень, проте в загальному випадку ризик зведений до мінімуму.

- Можливість встановлення сторонніх програм, яких немає у офіційному магазині Play Market. Операційна система Android має можливість завантаження програм із сторонніх джерел. Наприклад, операційна система

iOS постійно забороняє користувачам встановлювати програми з невідомих джерел.

- Налаштування. Пристосовуючи продукт до індивідуальних уподобань користувача, пристрій можна використовувати зручніше. Ви завжди можете змінити тему, тип клавіатури, встановити лаунчери та віджети на робочий стіл під час роботи на смартфоні.Поряд із численними перевагами, існують і певні недоліки, до числа яких можна віднести:

- Швидке втрачання заряду батареї. Таке явище спричинене постійним використання інтернету та роботі фонових додатків та процесів.
- Якість заліза. Через велику кількість пристроїв, які випускаються на платформі Android, виробникам важко підтримувати контроль якості.
- Підтримка оновлень. Пристрої на системі Android дуже рідко отримують оновлення чи нову версію, а бюджетні смартфони взагалі можуть залишитись без уваги.

Таким чином, операційна система Android має як переваги, так і недоліки, що дозволяє покупцям смартфонів вибирати пристрої на базі операційної системи, виходячи з особистих вимог та бюджету. Але як показує статистика, більшість віддає перевагу ОС Android.

1.5 Android Studio

Android Studio – це інтегроване середовище розробки (IDE) для роботи з платформою Android. Android Studio заснована на програмному забезпеченні IntelliJ IDEA від JetBrains, офіційному інструменті для розробки програм для Android [9].

Ця програма розробки мобільних додатків підтримує кілька мов програмування, таких як Kotlin, C/C++ та Java, з вбудованим емулятором та

різними шаблонами та компонентами, які значно спрощують та прискорюють процес розробки програмного продукту, має велику бібліотеку.

Android Studio дозволяє створювати програми для останніх версій Android. Ви можете відразу ж перевірити свою програму на наявність помилок, протестувати кожен елемент вашої програми різними інструментами і заздалегідь виявити всі можливі гріхи в роботі. Завдяки вбудованому емулятору ми можемо запускати тести продуктивності та коректності програм, розроблених на різних системах, та за необхідності проводити оптимізації. Емулятори також дозволяють тестувати на різних екранах та пристроях з різним співвідношенням сторін. Ця функція стала особливо важливою відколи ми увійшли в тренд смартфонів з екранами зі співвідношенням сторін 18:9. Відмінною рисою емуляторів є можливість відображати зразкові показники продуктивності під час запуску програм на найпопулярніших пристроях.

Розробники-початківці часто вибирають Android Studio, тому що це середовище простіше для розуміння і дозволяє їм працювати без великого досвіду програмування. Локалізувати вашу програму набагато простіше за допомогою функцій SDK, які також включені до списку переваг Android Studio[10].

Отже, Android Studio має такі особливості та переваги.:

a) середовище розробки підтримує роботу з декількома мовами програмування, до яких відносяться найпопулярніші – C / C ++, Java.

b) редактор коду, з яким зручно працювати;

c) дозволяє розробляти програми не тільки для смартфонів / планшетів, а й для портативних ПК, приставок для телевізорів Android TV, пристроїв Android Wear, новомодних мобільних пристроїв з незвичайним співвідношенням сторін екрану;

d) тестування коректності роботи нових ігор, утиліт, їх продуктивності в тій чи іншій системі відбувається безпосередньо в емуляторі;

e) рефакторинг вже готового коду;

f) розробка програми для останньої версії операційної системи Android;

g) попередня перевірка вже створеного додатку на предмет помилок в ньому;

h) великий набір засобів інструментів для тестування кожного елемента програми, ігри;

i) для недосвідчених / початківців розробників спеціально створено посібник з використання Android Studio, який розміщено на офіційному сайті;

j) базування на Gradle;

k) шаблони для створення поширених Android дизайнів та компонентів.

Недоліки

Android Studio насправді є ідеальною IDE для створення мобільних програм для операційної системи Android, але для комфортної роботи в цьому середовищі потрібен потужний комп'ютер. Сучасні ПК часто ідеально відповідають усім необхідним вимогам, але з старішими машинами можуть виникнути проблеми. Незважаючи на наявність вбудованого емулятора Android у самому середовищі розробки, тестування вашої програми може бути проблематичним. Тому для його запуску дуже важлива апаратна база ПК, що тестується з точки зору продуктивності. Іншим недоліком є неможливість створення серверних проектів на Java для ПК і Android-пристроїв.

1.6 IntelliJ IDEA

IntelliJ IDEA – це інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, включаючи Java, JavaScript та Python, розроблене JetBrains.

Перша версія з'явилася в січні 2001 року і швидко завоювала популярність як перше середовище Java з широким набором інтегрованих рефакторингових інструментів [6], які дозволяли програмістам швидко реорганізовувати вихідний текст своїх програм. Дизайн цього середовища орієнтований на продуктивність програміста, дозволяючи програмістам зосередитись на функціональних завданнях та дозволяючи IntelliJ IDEA виконувати рутинні операції.

Починаючи з 6-ї версії продукту IntelliJ IDEA надає інтегрований інструментарій для розробки графічних інтерфейсів. Крім інших функцій, середовище повністю сумісне з багатьма популярними безкоштовними інструментами розробки, такими як CVS, Subversion, Apache Ant, Maven та JUnit. У лютому 2007 року розробники IntelliJ анонсували ранню версію модуля, що підключається, що підтримує програмування мовою Ruby.

Починаючи з версії 9.0, це середовище доступне в двох редакціях: Community Edition і Ultimate Edition. Community Edition – повністю безкоштовна версія, доступна за ліцензією Apache 2.0, в якій реалізовано повну підтримку Java SE, Kotlin, Groovy, Scala та інтеграцію з найбільш популярними системами контролю версій. Версія Ultimate Edition, доступна за комерційною ліцензією, реалізує підтримку Java EE, діаграм UML, обчислень покриття коду та підтримку інших систем керування версіями, мов та середовищ.

Переваги IntelliJ IDEA:

- **Дизайн**

Кожен аспект IntelliJ IDEA розроблено для максимального підвищення продуктивності розробника. Разом інтелектуальна допомога в кодуванні та ергономічний дизайн роблять розробку не тільки продуктивною, але й приємною.

- **Глибокий інтелект**

Як тільки IntelliJ IDEA індексує ваш вихідний код, він забезпечує блискавично швидку та інтелектуальну роботу, надаючи актуальні пропозиції у будь-якому контексті: миттєве та інтелектуальне завершення коду, оперативний аналіз коду та надійні інструменти рефакторингу.

- **Нестандартний досвід**

Основні інструменти, такі як інтегрована система контролю версій і широкий спектр мов і фреймворків, що підтримуються, завжди у вас під рукою - ніяких проблем з плагінами.

- **Розумне завершення коду**

Базове завершення надає імена класів, методів, полів та ключових слів у межах видимості, тоді як інтелектуальне завершення надає лише типи, які очікують у поточному контексті.

- **Спеціальна рамкова допомога**

IntelliJ IDEA - це IDE для Java, але вона розуміє багато інших мов, таких як SQL, JPQL, HTML, JavaScript, і забезпечує інтелектуальну допомогу в написанні коду, навіть коли мовні вирази вбудовані в рядкові літерали в коді Java.

- **Прискорювачі продуктивності**

IDE передбачає ваші потреби та автоматизує стомлюючі та повторювані завдання розробки, щоб ви могли зосередитися на загальній картині.

- **Ергономіка розробника**

Приймаючи кожне рішення щодо дизайну та впровадження, ми пам’ятаємо про ризик переривання роботи розробника та робимо все можливе, щоб усунути або мінімізувати його.

IDE стежить за вашим контекстом і автоматично викликає відповідні інструменти.

- **Ненав'язливий інтелект**

Допомога в написанні коду IntelliJ IDEA стосується не лише редактора. Це також допомагає вам залишатися продуктивним під час роботи з іншими аспектами. Введіть поле для пошуку у списку елементів. Доступ до вікон інструментів. Або перемикати налаштування і т.д..

Висновок до розділу 1

У цьому розділі було розглянуто загальні поняття мобільних додатків та сучасних мобільних додатків. Сучасні реалії визначають необхідність використання мобільних додатків, виділяючи їх переваги та недоліки. Також було визначено перспективу розвитку ринку мобільних додатків. Було розглянуто загальне поняття серверної частини на Android платформи. Розглянуто що таке API , де і як його використовують , основні функції та його типи. Було розглянуто інструменти, які застосовуються для розробки серверної частини та Android платформи.

РОЗДІЛ 2

ПОСТАНОВКА ЗАДАЧІ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ

Даний розділ присвячено постановці задачі дослідження, а також опису необхідних інструментів для її реалізації.

2.1 Постановка задачі

Необхідно розробити структуру та прототип мобільного додатку під платформу Android й структуру бази даних. Розробити моделі даних, Rest API серверної частина та можливість підключення до WebSocket. Інтегрувати бізнес-логіку та реалізувати логіку спілкування додатку із сервером, застосовуючи мову програмування Kotlin та Android SDK.

Завданням даної роботи є розробка клієнтської та серверної частини системи інформаційної підтримки процесу захисту курсових та кваліфікаційних робіт. Мобільний додаток буде надавати можливість користувачам створювати факультети, курси, додавати студентів відповідного курсу, відображати чергу доповідачів кваліфікаційних робіт, задавати питання доповідачу кваліфікаційної роботи, відображати час доповіді, оцінку студента.

Завдяки цій системі студенти та викладачі зможуть, спростити процес захисту кваліфікаційних робіт.

Користувач має авторизуватися у системі, після чого йому буде надано відповідний функціонал, в залежності від його типу: потрібен «Студент» або ж «Викладач». У цих 2 типів є свої відмінності.

Функціонал, що надається учню:

- Реєстрація.
- Перегляд дати та часу захисту кваліфікаційної роботи.
- Перегляд черги захисту кваліфікаційної роботи.

- Завантаження кваліфікаційної роботи
- Завантаження презентації
- Додавання посилань
- Можливість задати питання під час захисту іншої кваліфікаційної роботи
- Перегляд питань, які були задані під час захисту

Функціонал , що надається викладачу:

- Додавання факультетів.
- Додавання курсів.
- Додавання студентів до курсу.
- Визначення дати та часу захисту кваліфікаційних робіт
- Визначення черги захисту кваліфікаційних робіт
- Перегляд кваліфікаційних робіт
- Перегляд презентацій
- Перегляд посилань
- Можливість задати питання під час захисту іншої кваліфікаційної роботи
- Перегляд питань, які були задані під час захисту
- Визначення оцінки

2.2 Огляд існуючих аналогів

Під час огляду існуючих аналогів було знайдено системи які лише частково реалізують визначений функціонал:

Mentimeter - простий та доступний інструмент для голосування. Забезпечити миттєвий зворотний зв'язок від вашої аудиторії. Доступний як на

мобільних пристроях, так і в електронному середовищі, він зручний для проведення опитувань учнів у класі як реального часу.

Онлайн-опитування може містити серію з декількох питань та мати різні типи відповідей:

- множинний вибір (один або кілька з кількох);
- відкриту відповідь;
- оцінка за шкалою;
- ранжування відповідей у межах 100%;
- введення відповіді у вигляді точки на плоскій координатній площині.

Slido – це просте у використанні додаток для проведення нарад запитання та відповіді проведення опитувань, що підвищує залученість під час будь-якого типу нарад, у прямому чи віртуальному вигляді. Вона допомагає вести значні бесіди, дає аудиторії можливість ставити запитання та говорити свою думку під час опитувань у прямому ефірі. 2 грудня 2021 р. організатори нарад можуть натиснути кнопку Програми в Webex Meetings для доступу Slidoк.

Після аналізу існуючих систем було вирішено створити власну систему із більшим функціоналом для надання можливості спростити процес захисту кваліфікаційних робіт.

2.3 Огляд дизайнерських інструментів

Figma - це графічний онлайн редактор для сумісної роботи декількох користувачів. Її можна використовувати для створення прототипу сайту, інтерфейсу програми та обговорити правки з колегами в реальному часі.

Чому була обрана саме Figma:

1. Екосистема. Створюючи веб-сайт або мобільний додаток, дизайнерам часто доводиться завантажувати великі файли на хмарні сховища,

переслати їх поштою, тощо. Це забирає багато часу, крім того, файли можуть займати досить багато місця в пам’яті комп’ютера. Саме цю проблему вирішує Figma, через те що усі робочі файли знаходяться на власному хмарному сховищі, дизайнер може просто відправити посилання на файл замовнику або розробнику. Крім того, якщо на проекті змінюється фахівець з дизайну чи розробки, не виникне ніякої проблем з тим, де знаходяться всі вихідні дані.

2. Спільне та одночасне редагування. Набір додатків Google Docs відмінно показали, що одночасна робота декількох людей над одними файлами полегшує комунікацію і покращує результат. Використовуючи Figma дизайнери, розробники, менеджери і замовники можуть одночасно ставити запитання, писати коментарі та редагувати дизайн.

3. Багатозадачність. Якщо потрібно працювати одночасно з декількома робочими областями то Sketch та Photoshop це не дуже оптимальний вибір, оскільки розробники досить часто скаржаться на їх продуктивність. Figma дозволяє дизайнерам та розробникам працювати з більш, ніж відкритими десятьма проектами одночасно.

4. Прототипування. Переваги цього продукту найкраще розкриваються при розробці дизайну мобільних додатків. Якщо проект складається з більш, ніж 30 екранів, досить складно переходити між ними. Але завдяки Figma можна зібрати всі екрани в одному місці і керувати ними. Figma досить корисна і для розробників, вона дозволяє безпосередньо в програмі подивитися інформацію про компоненти, яка потрібні необхідно розробити [2].

2.4 Android SDK та Android API

SDK (Software Development Kit) – це набір інструментів для створення програмного забезпечення для конкретної платформи. Загалом це можуть бути компілятори, налагоджувачі, емулятори, різні бібліотеки, інструменти,

документація та приклади. Android SDK - це модуль коду Java. Програмісти можуть отримати доступ до функцій мобільних пристроїв, сторонніх служб та багато іншого.

Одним із основних компонентів Android SDK є бібліотека Gradle. Щоб інтегрувати платформу соціальних мереж (наприклад, Facebook) у свою програму, вам необхідно завантажити бібліотеку коду (або SDK) з Facebook та повідомити про це Gradle. Середній мобільний додаток Android використовує близько 15 окремих SDK. Найпопулярніші категорії - аналітика та реклама.

Удосконалення Android SDK тісно пов'язані із загальною розробкою платформи Android. SDK також підтримує старіші версії Android, якщо розробники хочуть зосередити свої програми на старіших пристроях. Засоби розробки є компонентами, що завантажуються, тому після завантаження останньої версії та платформи можна використовувати більш старі платформи та інструменти для тестування сумісності. [3]

SDK зазвичай включають API-інтерфейси фреймворку, реалізації API і різні системні образи для емуляторів Android.

API (інтерфейс прикладного програмування) - це інтерфейс, рівень абстракції, який забезпечує зв'язок між двома різними частинами програмного забезпечення. Він діє як контракт між постачальником (наприклад, бібліотекою) та споживачем (наприклад, додатком).

API надає формальний набір визначень класів, методів, функцій, модулів, констант тощо, які інші розробники можуть використовувати для написання свого коду. При цьому API не містить жодної реалізації.

Платформа Android надає API-інтерфейси фреймворку, які програми можуть використовувати для взаємодії з базовою системою Android. Фреймворк API:

- Набір основних пакетів і класів;

- Набір стандратних XML-елементів та атрибутів для оголошення файлу manifest;
- Набір XML-елементів та атрибутів для заданння доступу до ресурсів;
- Набір стандартний й найбільш поширених інтентів;
- Набір системниих дозволів, які можуть використовуватися програмою, а також засоби отримання дозволів, вбудовані у систему.

Оновлення Framework API призначено для сумісності нових API з попередніми версіями API. Тобто більшість змін API є адитивними та вводять нові функції чи функції, які їх замінюють. Коли оновлені частини API, старі замінені частини старіють, але не видаляються, тому існуючі програми можуть продовжувати їх використовувати. У дуже невеликій кількості випадків частини API можуть бути змінені або видалені, але зазвичай такі зміни необхідні тільки для забезпечення надійності API та безпеки вашої програми або системи. Усі інші частини API з попередніх версій переносяться без змін.

API фреймворку, який забезпечує платформа Android, визначається за допомогою рівня API. Рівень API - це ціле число, яке однозначно ідентифікує версію API фреймворку, пропоновану платформою Android. Кожна версія платформи Android підтримує рівно один рівень API, хоча підтримка передбачається для всіх попередніх рівнів API.

Android Nougat	New York Cheesecake	7.0	24	August 22, 2016
		7.1 – 7.1.2	25	October 4, 2016
Android Oreo	Oatmeal Cookie	8.0	26	August 21, 2017
		8.1	27	December 5, 2017
Android Pie	Pistachio Ice Cream[20]	9	28	August 6, 2018
Android 10	Quince Tart[21]	10	29	September 3, 2019
Android 11	Red Velvet Cake[21]	11	30	September 8, 2020
Android 12	Snow Cone	12	31	October 4, 2021
Android 12L	Snow Cone v2	12.1[a]	32	March 7, 2022
Android 13	Tiramisu[23]	**13**	33	August 15, 2022
Android 14	Upside Down Cake[24]	14	34	Q3 2023
Legend: Old version	Older version, still maintained	**Latest version**	Future release	

Рисунок 2.1 – Відповідність версії Android та рівня API.

Ідентифікатор рівня API виконує ключову роль у забезпеченні найкращого досвіду для користувачів та розробників цих додатків:

- Це дозволяє описати максимальну версію API для платформі Android, яку вона підтримує.
- Це дозволяє додаткам описувати необхідну версію API фреймворку.
- Це дозволяє системі узгоджувати встановлення програм на пристрої користувача, так що програми, несумісні з версіями, не встановлюються.

Кожна нова версія операційної системи Android зберігає свій ідентифікатор рівня API внутрішньо, в самій системі Android.

Додатки використовують елемент маніфесту <uses-sdk>, що надається інфраструктурою API, для опису мінімального та максимального рівнів API, з якими може працювати програма, та бажаних рівнів API, для підтримки яких призначено програму. Цей елемент має три основні атрибути:

- android:minSdkVersion – ціле число, що позначає мінімальний рівень API, на якому програма може працювати. Система Android не дозволяє користувачеві

встановити програму, якщо рівень API системи нижчий за значення, вказане в цьому атрибуті. За замовчуванням атрибут приймає значення «1», у випадку якщо про цей атрибут не заявлено. Це означає, що програма сумісна з будь-якою версією Android. Якщо версія API системи менша, ніж заявлена мінімальна версія програми, то будуть виникати аварійні ситуації в роботі програми при спробі отримати доступ до недоступних API.

- android:targetSdkVersion – вказує рівень API, на якому призначена робота програми. У деяких випадках це дозволяє додатку використовувати елементи маніфесту або поведінку, визначені на цільовому рівні API, а не обмежуватись використанням лише тих, що визначені для мінімального рівня API. Якщо значення атрибуту не вказано, то воно приймає значення за замовчуванням, яке рівне значенню атрибуту android:minSdkVersion.
- android:maxSdkVersion – вказує максимальний рівень API, на якому програма може працювати. В Android 1.5, 1.6, 2.0 та 2.0.1 система перевіряє значення цього атрибута під час встановлення програми та при повторній перевірці програми після оновлення системи. У будь-якому випадку, якщо maxSdkVersion атрибут програми нижчий за рівень API, який використовує сама система, система не дозволяє встановлювати програму. У разі повторної перевірки після оновлення системи це ефективно видаляє вашу програму з пристрою. Якщо не оголошено, система припускає, що програма не має максимального рівня API. [12]

При оголошенні в маніфесті програми <uses-sdk> елемент може виглядати так:

```
<manifest>
  <uses-sdk android:minSdkVersion="5" />
  ...
</manifest>
```

Також в середовищі розробки Android Studio при створенні нового проекту є можливість відразу обрати мінімальний рівень API.

Вибір рівня API для розробки програми має враховувати як мінімум дві речі:

1. Поточний дистрибутив. Скільки пристроїв може реально підтримувати мій додаток, якщо він був розроблений для рівня API 23, він не може працювати на рівні API 22 і нижче, тоді тільки близько 85% пристроїв можуть його запускати.
2. Вибір більш низького рівня API може підтримувати більше пристроїв, але для додатку передбачено менше функціональності.

Як правило, проект під платформу Android складається з коду, написаного розробниками з обов'язковим використанням Android API, а також деяких інших бібліотек, залежностей і ресурсів.[13]

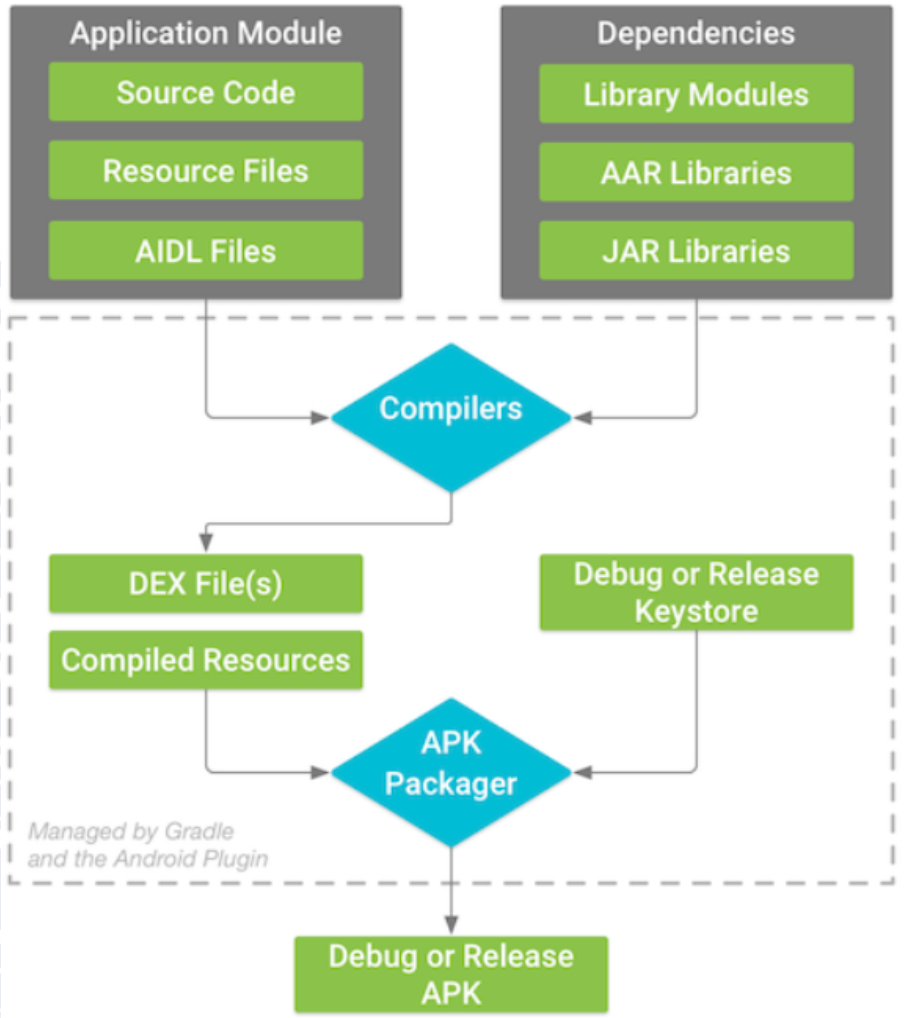

Рисунок 2.2 - Процес збірки файлу APK

Процес побудови типового gradle-модуля програми під Android, як показано на рисунку 2.2, виконує такі кроки:

1. Компілятор перетворює вихідний код, написаний на Java або Kotlin (включаючи залежності), на файли DEX, що містять байт-код, який працює на пристроях Android, а все інше — на скомпіловані ресурси.

2. Пакувальник APK об'єднує файли DEX та пов'язані ресурси в один файл .apk. Однак, перш ніж ви зможете встановити та розгорнути свою програму на пристрої Android, ви повинні підписати свій APK.

3. Пакетувальник підписує APK за допомогою ключа налагодження або випуску.:

- При створенні версії налагодження програми, тобто програми, призначеного тільки для цілей тестування та профілювання, пакувальник підписує програму, використовуючи сховище ключів налагодження. Android Studio автоматично налаштовує нові проекти за допомогою репозиторію налагодження..
- При складанні версії випуску програми, яка має бути випущена ззовні, пакувальник підписує програму, використовуючи сховище ключів випуску.

4. Перед створенням остаточного файлу .apk пакувальник використовує інструмент zipalign для оптимізації програми та використання меншого об'єму пам'яті під час роботи на пристрої.

Наприкінці процесу складання створюється APK для налагодження або випуску, який можна використовувати для розгортання, тестування або випуску зовнішніх користувачів. [12]

Android Runtime

Android Runtime (ART) – це кероване середовище виконання, використовуване програмами Android та деякими системними службами. ART та його попередник Dalvik спочатку створювалися спеціально для Android-проектів. ART виконує формат виконуваного файлу Dalvik та специфікації байт-коду Dex під час виконання.

ART та Dalvik – це сумісні середовища виконання, які запускають байт-код Dex, тому програми, розроблені для Dalvik, повинні працювати при запуску ART. Однак деякі методи, що працюють у Dalvik, не працюють у ART. [12]

Gradle

Gradle – це автоматизована система збирання, побудована на принципах Apache Ant та Apache Maven. В Android Studio Gradle завжди супроводжує

розробників. Gradle не є винаходом Android Studio. Ця система раніше розроблялася і використовувалася в програмах Java, Scala та інших мовах. [14]

Gradle та плагін Android допомагають налаштувати такі аспекти вашої збірки:

- Типи збірки

Типи збірки визначають певні властивості, які Gradle використовує під час створення та упакування додатка, і зазвичай налаштовуються на різні етапи життєвого циклу розробки. Наприклад, тип збірки налагодження включає параметри налагодження та підписує файл .apk ключем налагодження, тоді як тип збірки випуску може стискатися, затуманювати та підписувати APK ключем випуску для розповсюдження. Потрібно визначити принаймні один тип збірки, щоб створити свій додаток - Android Studio за замовчуванням створює типи налагодження та випуску.

- Product flavors

Product flavors представляють різні версії додатку, які можна випустити для користувачів, наприклад безкоштовну та платну версії додатка. Є можливість налаштувати дану функцію для використання іншого коду та ресурсів, одночасно використовуючи та повторно використовуючи частини, загальні для всіх версій додатка. Product flavors є необов'язковими, і потрібно створювати їх вручну.

- Build variants

Build variants – це перехресний продукт типу збірки та Product flavors, і це конфігурація, яку Gradle використовує для створення додатка. Використовуючи Build variants, можна створювати налагоджувальну версію Product flavors під час розробки або підписані версії Product flavors для розповсюдження. Хоча Build variants не налаштовуються безпосередньо, можна конфігурувати типи збірки та Product flavors, які їх формують.

- Записи маніфесту

Можна вказати значення для деяких властивостей файлу маніфесту в конфігурації Build variants. Ці значення побудови замінюють наявні значення у файлі маніфесту. Це корисно, при створені декілька файлів .apk для своїх модулів, де кожен з файлів apk має різну назву програми, мінімальну версію SDK або цільову версію SDK. Коли присутні кілька маніфестів, Gradle об’єднує налаштування маніфесту.

- Залежності

Система збірки керує залежностями проекту від локальної файлової системи та від віддалених сховищ. Це запобігає необхідності вручну шукати, завантажувати та копіювати двійкові пакети залежностей у каталог проекту.

- Підписання

Система збірки дозволяє вказати параметри підписання в конфігурації збірки, і вона може автоматично підписувати файли .apk під час процесу збірки. Система збірки підписує версію налагодження за замовчуванням ключем та сертифікатом, використовуючи відомі облікові дані, щоб уникнути запиту пароля під час збірки. Система збірки не підписує версію випуску, якщо чітко не визначити конфігурацію підпису для цієї збірки.

- Зменшення коду та ресурсів

Система збірки дозволяє вказати інший файл правил ProGuard для кожного Build variants. Створюючи додаток, система складання застосовує відповідний набір правил для зменшення коду та ресурсів за допомогою вбудованих інструментів, що стискаються , таких як R8.

- Підтримка декількох файлів .apk

Система збірки дозволяє вам автоматично створювати різні файли .apk, які містять лише код та ресурси, необхідні для певної щільності екрану або двійкового інтерфейсу додатків (ABI).

2.5 Мова програмування Kotlin

Kotlin - це мова програмування, створена в компанії JetBrains. Його розробили у 2011 році на заміну Java, який у компанії вважали надто багатослівним. Нова мова вийшла на 40% компактнішою за попередника, що допомогло прискорити роботу над основним продуктом JetBrains — середовищем розробки IntelliJ IDEA. При цьому Kotlin повністю сумісний із Java, тому що запускається на його віртуальній машині (JVM).

Переваги програмування на Kotlin

На Google I/O 2019 було оголошено, що мова програмування Kotlin стала пріорітетом для Android-розробки.

- Об'єктно-орієнтоване програмування

Kotlin включає об'єктно-орієнтоване програмування (ООП). Це концепція, що визначає як тип даних та його структуру, а й набір функцій, які до них застосовуються. Таким чином, структури даних стають об'єктами, якими можна маніпулювати створення відносин між різними об'єктами.

- Kotlin - мова високого рівня з простим синтаксисом і плавною кривою навчання

Kotlin - це мова високого рівня, схожа на людську мову. Він відрізняється від низькорівневих мов, що нагадують машинну мову. Мови високого рівня перекладаються компіляторами або інтерпретаторами. Синтаксис Kotlin простий, тому навіть новачки можуть швидко та ефективно використовувати код для досягнення певних результатів. Він простий, типізований, передбачуваний і дозволяє навчитися думати у правильному напрямі.

- Безпека

Є думка, що Kotlin є безпечною мовою, але це не зовсім так. Сама мова не захищає від уразливостей, але деякі її функції усувають поширені вразливості. По-перше, на відміну від C, у Kotlin немає вказівників. Вказівник — це об'єкт, який зберігає адресу пам'яті іншого значення та може призвести до нелегального доступу до пам'яті. По-друге, у Kotlin є Security Manager - політика безпеки, створена для кожної програми, де можна вказати правила доступу. Це дозволяє програмам Java працювати в «пісочниці», що усуває вразливості.

- Крос-платформність

"Напиши один раз, використовуй скрізь" - популярна фраза в IT, і Sun Microsystems описує кросплатформні можливості Kotlin. Можливо створювати програми Kotlin у Windows, компілювати їх у байт-код і запускати їх на інших платформах, що підтримують віртуальну машину Java (JVM). Таким чином, JVM діє як рівень абстракції між вашим кодом та обладнанням.

- Автоматичне управління пам'яттю

Завдяки автоматичному управлінню пам'яттю (AMM) розробникам Kotlin не потрібно вручну писати код для керування пам'яттю. Ефективність програми пов'язана з пам'яттю. При цьому обсяг пам'яті обмежений. При написанні програм на мовах, що управляють пам'яттю вручну, розробники ризикують забути виділити пам'ять, збільшивши обсяг пам'яті, яку займає програма, і викликати проблеми з продуктивністю. Програма очищення пам'яті шукає та видаляє об'єкти, які більше не використовуються програмою. Це впливає на продуктивність процесора, але інтелектуальна оптимізація та налаштування можуть пом'якшити цей вплив..

- Багатопоточність

Потік це найменша одиниця обробки в програмуванні. Щоб максимально ефективно використовувати процесорний час Kotlin дозволяє одночасно запускати кілька потоків, що називається багатопоточністю.

Потоки використовують один і той же простір пам'яті, тому ви можете швидко перемикатися між ними. Потоки є незалежними один від одного. Один потік неспроможна проводити поведінка інших потоків. Це особливо корисно для ігор та програм з великою кількістю анімації.

Мінуси програмування на Kotlin

- Низька продуктивність

Мови високого рівня працюють дуже погано через компіляцію та абстракцію з використанням віртуальних машин. Однак, це не єдина причина повільності Kotlin. Наприклад, додаток для очищення пам'яті – корисна функція, але якщо він споживає більше 20% процесорного часу, на жаль, він викликає серйозні проблеми з продуктивністю. Неправильні установки кешу можуть призвести до надмірного використання пам'яті. Існують також взаємоблокування потоків. Це відбувається, коли кілька потоків намагаються отримати доступ до одного і того ж ресурсу. У цьому випадку кожен Java-розробник боїться. Це помилка через брак пам'яті.

2.6 Сучасні архітектурні рішення

MVC

Модель-представлення-контролер (MVC, Model-view-controller) - це архітектурний шаблон, який використовується при проектуванні та розробці програмного забезпечення.

Цей шаблон ділить систему на три взаємопов'язані частини: модель даних, уявлення (інтерфейс користувача) і модуль управління. Він використовується для відокремлення даних (моделі) від інтерфейсу

користувача (уявлення). Це дозволяє змінам в інтерфейсі користувача мінімальний вплив на маніпулювання даними і дозволяє зберігати зміни в даних моделі без зміни користувальницького інтерфейсу.

Метою створення шаблонів є ефективний дизайн програмного забезпечення, який не тільки забезпечує можливість повторного використання окремих компонентів програми, але й полегшує подальшу модифікацію та розширення програми. Крім того, використання цього шаблону у великих системах призводить до нерегулярних структур, які менш складні та легші для розуміння.

В рамках архітектурного шаблону MVC додаток ділиться на три незалежні та взаємопов'язані частини, розділяючи функціональність між компонентами. Моделі відповідають за зберігання даних та їх структуру. Подання хороші для представлення цих даних або програмних інтерфейсів. Контролер управляє компонентом, отримує сигнали як дій користувача (зміна становища курсору миші, натискання кнопки, відображення даних у текстовому полі тощо. буд.) і надсилає дані у модель.

Моделі — це центральний компонент шаблону MVC, що представляє поведінку вашої програми незалежно від інтерфейсу користувача. Ця модель вимагає прямого управління даними, логікою та правилами додатків.

- Активна модель - вигляд відстежує зміни в моделі та реагує на них.
- Пасивна модель - вигляд оновлюється через контролер.

Поданням може бути будь-яке подання вихідної інформації, таке як графік чи діаграма. У цьому можуть співіснувати кілька образів (уявлень) однієї й тієї інформації. Наприклад, гістограми для управління компанією та таблиці для бухгалтерії.

Контролери приймають вхідні дані та перетворюють їх на команди для моделі або подання.

Модель містить ядро даних та основні функції їх обробки і не залежить від процесу введення або виведення даних.

Уявлення можуть містити декілька взаємопов'язаних областей, таких як різні таблиці та поля форм, у яких відображаються дані.

Функціональність контролера включає відстеження зумовлених подій, які надають результати дій користувача. Контролери дозволяють структурувати код, групуючи пов'язані дії окремі класи. Наприклад, типовий проект MVC може мати контролер користувача, який містить групу методів, пов'язаних з управлінням обліковими записами користувачів, таких як реєстрація, авторизація, редагування профілю, зміна пароля і т. д.

Зареєстровані події транслюються іншим запитам на компонент моделі або об'єкт, придатний для відображення даних. Відділення моделі від представлення даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Отже, якщо користувач вносить зміни до даних моделей через контролер, інформація, що надається одним або декількома візуальними компонентами, автоматично коригуватиметься відповідно до змін..

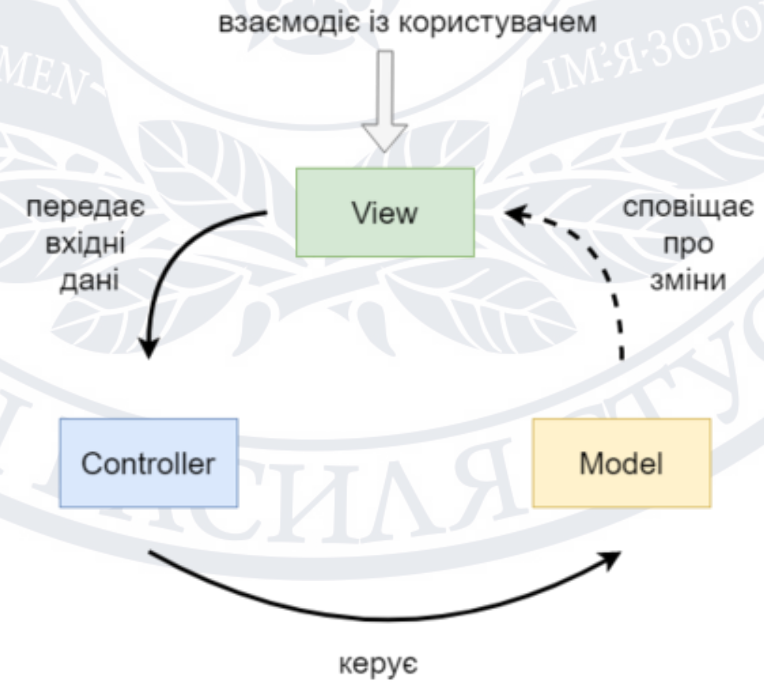

Рисунок 2.3 - Взаємодія у MVC

MVP

Model-View-Presenter (англ. Model-View-Presenter, MVP) - це патерн проектування, похідний від MVC, який поділяє візуальне подання та поведінку обробки подій на різні класи: View та Presenter (Presenter).

Цей підхід дозволяє нам створювати репрезентативні абстракції. Для цього нам потрібно призначити інтерфейс уявлення з певним набором властивостей та методів. Потім презентатор отримує посилання на реалізацію інтерфейсу, підписується на події презентації та модифікує модель відповідно до запиту.

Ознаки презентера:

- Двостороння комунікація з поданням;
- Подання взаємодіє безпосередньо з презентером шляхом виклику відповідних функцій або подій екземпляра презентера;
- Презентер взаємодіє з View шляхом використання спеціального інтерфейсу, реалізованого уявленням;
- Один екземпляр презентера пов'язаний із одним відображенням.

Реалізація:

Кожне представлення має реалізовувати відповідний інтерфейс. Інтерфейс подання визначає набір функцій та подій, необхідних для взаємодії з користувачем (наприклад, IView.ShowErrorMessage(string msg)). У презентатора має бути посилання на реалізацію відповідного інтерфейсу. Зазвичай це передається у конструкторі.

Для логіки показу потрібне посилання на екземпляр презентатора. Усі події презентації відправляються на обробку доповідачеві і фактично не

обробляються логікою презентації (включаючи створення інших презентацій).

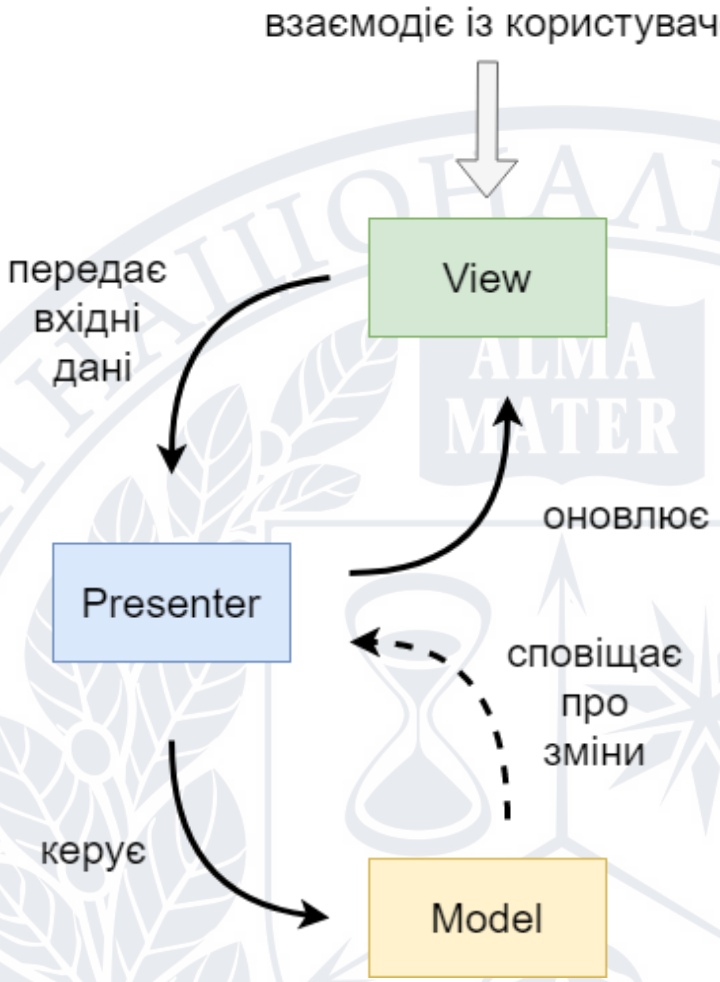

Рисунок 2.4 - Взаємодія у MVP

MVVM

Model-View-ViewModel — це шаблон проектування, який використовується під час проектування архітектури програми. Вперше він був опублікований Джоном Госсманом у 2005 році як модифікована версія шаблону моделі презентації. MVVM фокусується на сучасних платформах розробки, таких як Windows Presentation Foundation та Microsoft Silverlight.

MVVM дозволяє легко відокремити розробку GUI від розробки бізнес-логіки (внутрішньої логіки), яка називається моделлю (також відома як відділення подання від моделі). Модель представлення – це частина, яка відповідає за перетворення даних для подальшої підтримки та використання. З

цього погляду модель уявлення більше схожа на модель, ніж уявлення, і обробляє більшу частину, а то й всю логіку подання даних. Моделі уявлень також можуть реалізовувати посередники шаблонів, які організують доступ до логіки серверної частини на основі набору правил використання підтримуваних поданням.

Цей підхід дозволяє підключати елементи вашого уявлення до властивостей та подій моделі View. Імовірно, кожен шар у цьому шаблоні не знає про існування іншого шару.

Ознаки View-моделі:

- Двостороння комунікація з поданням;
- View-модель – це абстракція уявлення. Зазвичай означає, що властивості подання збігаються з властивостями View-моделі/моделі
- View-модель не має посилання на інтерфейс уявлення (IView). Зміна стану View-моделі автоматично змінює уявлення та навпаки, оскільки використовується механізм зв'язування даних (Bindings)
- Один екземпляр View-моделі пов'язаний з одним відображенням.

Реалізація:

У разі використання цього шаблону подання не реалізує відповідний інтерфейс (IView).

Подання необхідне посилання джерело даних (дані). У разі це модель представлення. Елементи уявлення прив'язані (прив'язані) до відповідних властивостей та подій у моделі уявлення.

Потім модель вистави реалізує спеціальний інтерфейс, який використовується для автоматичного оновлення елементів представлення.

Прикладом такого інтерфейсу WPF є INotifyPropertyChanged.

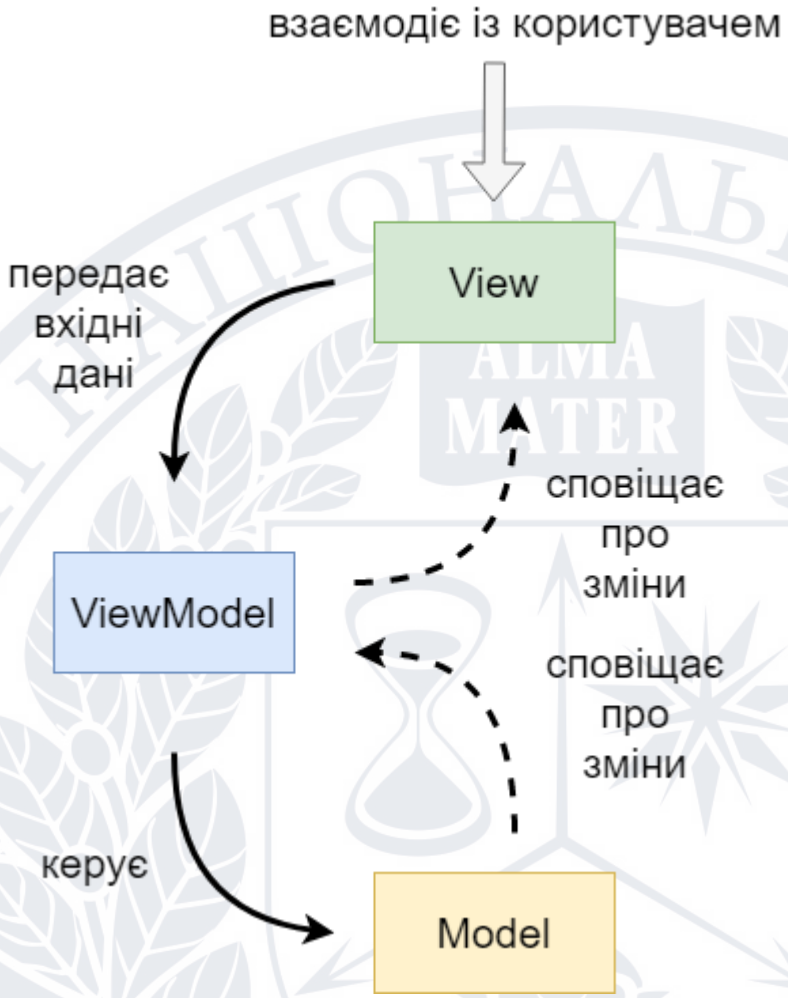

Рисунок 2.5 - Взаємодія у MVVM

MVI

MVI (Model-View-Intent) оптимізує процес створення та розробки програм, запроваджуючи реактивний підхід. У певному сенсі цей шаблон є сумішшю MVP і MVVM, адаптованих до реактивного програмування. Це виключає використання зворотного виклику та значно зменшує кількість методів введення/виведення. Це також чудове рішення для синхронізації станів між переглядом і рівнем бізнес-логіки.

У міру розширення програми або додавання незапланованих заздалегідь функціональних можливостей – без чіткого керування станом візуалізація перегляду разом з бізнес-логікою може стати дещо безладною.

Чим масштабованіший код програми, тим гнучкіший він для нових ідей і оновлень. Масштабованість, гнучкість і легкість тестування - ось що пропонує нам архітектура MVI.

Ключові переваги MVI:

- єдине джерело істини - одне незмінне стан, загальне для всіх верств, як єдине джерело істини
- односпрямований і циклічний потік даних
- легкість виявлення та виправлення помилок
- легкість тестування коду
- можливість тестувати всі рівні програми за допомогою модульних тестів

Представлення (View) – рівень перегляду спостерігає за діями користувача та системними подіями. У результаті він встановлює намір для ініційованої події. Крім того, він слухає і реагує на зміну стану моделі.

Модель (Model). Модель є представленням стану перегляду. Він містить всю інформацію, необхідну для правильного відтворення перегляду.

Намір (Intent) - представлення майбутньої дії, яка змінює стан моделі.

Користувач* - багато людей також включають користувача програми як частину архітектури MVI. Він або вона спостерігає та реагує на зміни стану перегляду, працюючи з програмою.

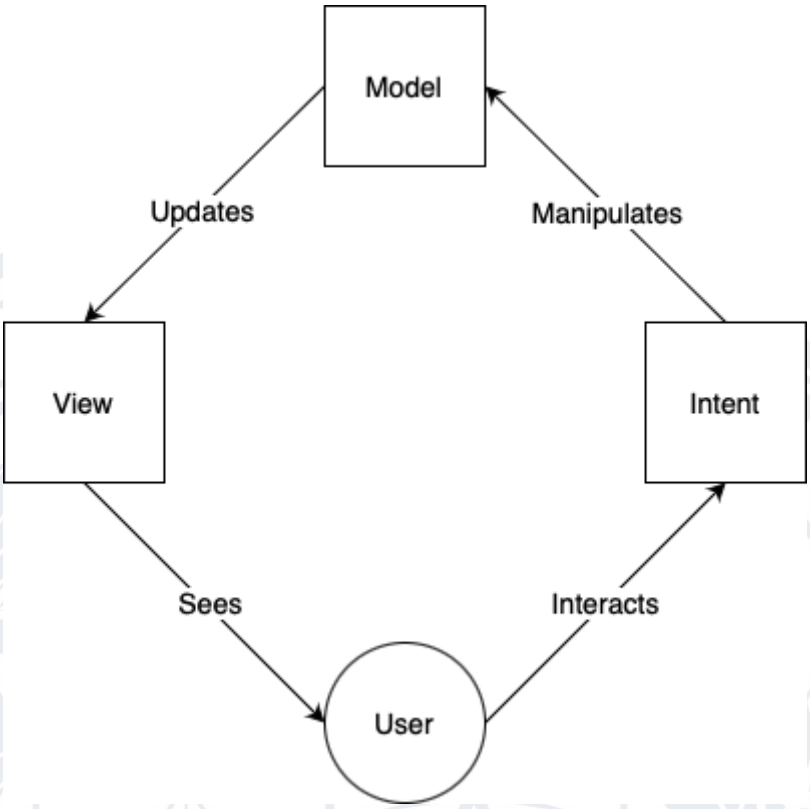

Рисунок 2.6 - Зваємодія у MVI

Приклад реалізації MVI у клієнтській частині за допомогою Android Studio та мови програмування Kotlin: Додаток “Лічильник”

Рисунок 2.7 - Додаток “Лічильник”

Наступний клас, який містить стан лічильника, є представленням стану перегляду для цього екрана:

Лістинг 1:

```
data class CounterViewState(
    val counterValue: Int = 0
) : ViewState {
    val counterValueText: String = "$counterValue"
}
```

Потім на цьому екрані єдиним ефектом, який може виникнути, є навігація до наступного екрана, тому viewEffect виглядатиме так

```
sealed class CounterViewEffect : ViewEffect {
```

```
    object NavigateToSecondScreen : CounterViewEffect()
}
```

Дії, які може виконувати користувач. Користувач може натиснути кнопку «збільшити», натиснути кнопку «зменшити» або перейти до наступного екрана. Клас з відповідними намірами:

```
sealed class CounterIntent : Intent {
    object Increase : CounterIntent()
    object Decrease : CounterIntent()
    object NavigateToSecondScreen : CounterIntent()
}
```

Потрібно реалізувати часткові класи стану, які знатимуть, як змінити viewstate залежно від випадку:

Лістинг 2:

```
sealed class CounterPartialState : PartialState<CounterViewState,
CounterViewEffect> {
    object Increase : CounterPartialState() {
        override fun reduce(previousState: CounterViewState): CounterViewState {
            return previousState.copy(counterValue = previousState.counterValue + 1)
        }
    }
    object Decrease : CounterPartialState() {
        override fun reduce(previousState: CounterViewState): CounterViewState {
            return previousState.copy(counterValue = previousState.counterValue - 1)
        }
    }
    object NavigateToSecondScreen : CounterPartialState() {
        override fun mapToViewEffect(): CounterViewEffect {
            return CounterViewEffect.NavigateToSecondScreen
        }
```

```
    }
}
```

Створення презентора, який відображає наміри в часткових станах:

Лістинг 3:

```
class CounterPresenter @Inject constructor(
    @Named(MAIN_THREAD) mainThread: Scheduler
) : Presenter<CounterViewState, CounterView, CounterPartialState, CounterIntent,
CounterViewEffect>(mainThread) {
    override val defaultViewState: CounterViewState
        get() = CounterViewState()

    override fun intentToPartialState(intent: CounterIntent):
Observable<CounterPartialState> =
        when (intent) {
            is CounterIntent.Increase -> Observable.just(CounterPartialState.Increase)
            is CounterIntent.Decrease ->
Observable.just(CounterPartialState.Decrease)
            is CounterIntent.NavigateToSecondScreen ->
Observable.just(CounterPartialState.NavigateToSecondScreen)
        }
}
```

Реалізація представлення цього екрану:

Лістинг 4:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
    <data>
        <variable
            name="viewState"
```

```
type="com.bonacode.modernmvi.sample.presentation.feature.counter.CounterViewState"
/>

        </data>

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="16dp"
            android:orientation="vertical">
            <androidx.appcompat.widget.AppCompatTextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginBottom="32dp"
                android:gravity="center"
                android:text="@{viewState.counterValueText}"
                android:textColor="@color/colorBlack"
                android:textSize="80sp"
                tools:text="43" />
            <com.google.android.material.button.MaterialButton
                android:id="@+id/increaseButton"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Increase"
                android:textSize="50sp" />
            <com.google.android.material.button.MaterialButton
                android:id="@+id/decreaseButton"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="Decrease"
                android:textSize="50sp" />
        </LinearLayout>
```

```
</layout>
```

Останнім кроком є реалізація класу активності/фрагмента, який збирає наміри та відтворює стан на екрані:

Лістинг 5:

```
class CounterActivity :
    MviActivity<CounterViewState, CounterViewEffect, CounterView,
CounterPresenter, ActivityCounterBinding>(),
    CounterView {
    override val binding: ActivityCounterBinding by
viewBinding(ActivityCounterBinding::inflate)
    override val presenter: CounterPresenter by viewModels()
    override fun getMviView(): CounterView = this
    override fun render(viewState: CounterViewState) {
      binding.viewState = viewState
      binding.executePendingBindings()
    }
    override fun handleViewEffect(event: CounterViewEffect) {
      when (event) {
        is CounterViewEffect.NavigateToSecondScreen ->
navigateToSecondScreen()
      }
    }
    override fun emitIntents(): Observable<CounterIntent> = Observable.merge(
    listOf(
      binding.increaseButton clicksTo CounterIntent.Increase,
      binding.decreaseButton clicksTo CounterIntent.Decrease,
      binding.navigateForwardButton clicksTo
CounterIntent.NavigateToSecondScreen
    )
  )
    private fun navigateToSecondScreen() {
```

```
        startActivity(
            Intent(
                this,
                DogsActivity::class.java
            )
        )
    }
}
```

2.6 Ktor

Ktor – це платформа для простого створення підключених додатків (веб-додатків, HTTP-сервісів, мобільних та браузерних додатків). Сучасні підключені програми повинні бути асинхронними, щоб забезпечити найкращу взаємодію користувача. Програми для спільної роботи на Kotlin надають чудовий спосіб зробити це простим та прямим способом. Мета Ktor — надати наскрізну структуру кроссплатформи для підключених додатків.

Створений з нуля з використанням Kotlin та співпрограм, Ktor забезпечує потужність асинхронного програмування з коротким крос-платформною мовою та інтуїтивно зрозумілим імперативним потоком.

Ktor дозволяє вам використовувати тільки те, що вам потрібно, і налаштувати вашу програму так, як ви цього хочете. Крім того, також можливо дуже легко розширити Ktor за допомогою власного плагіна.

Приклад реалізацію серверної частини у IntelliJ IDEA за допомогою мови програмування Kotlin та платформи Ktor:

1. Створення проекту:

    1.1. На екрані привітання натисніть «Новий проект». В іншому випадку в головному меню виберіть Файл | Новий | Проект.

1.2. У майстрі нового проекту виберіть Ktor зі списку ліворуч. На правій панелі вкажіть такі параметри:

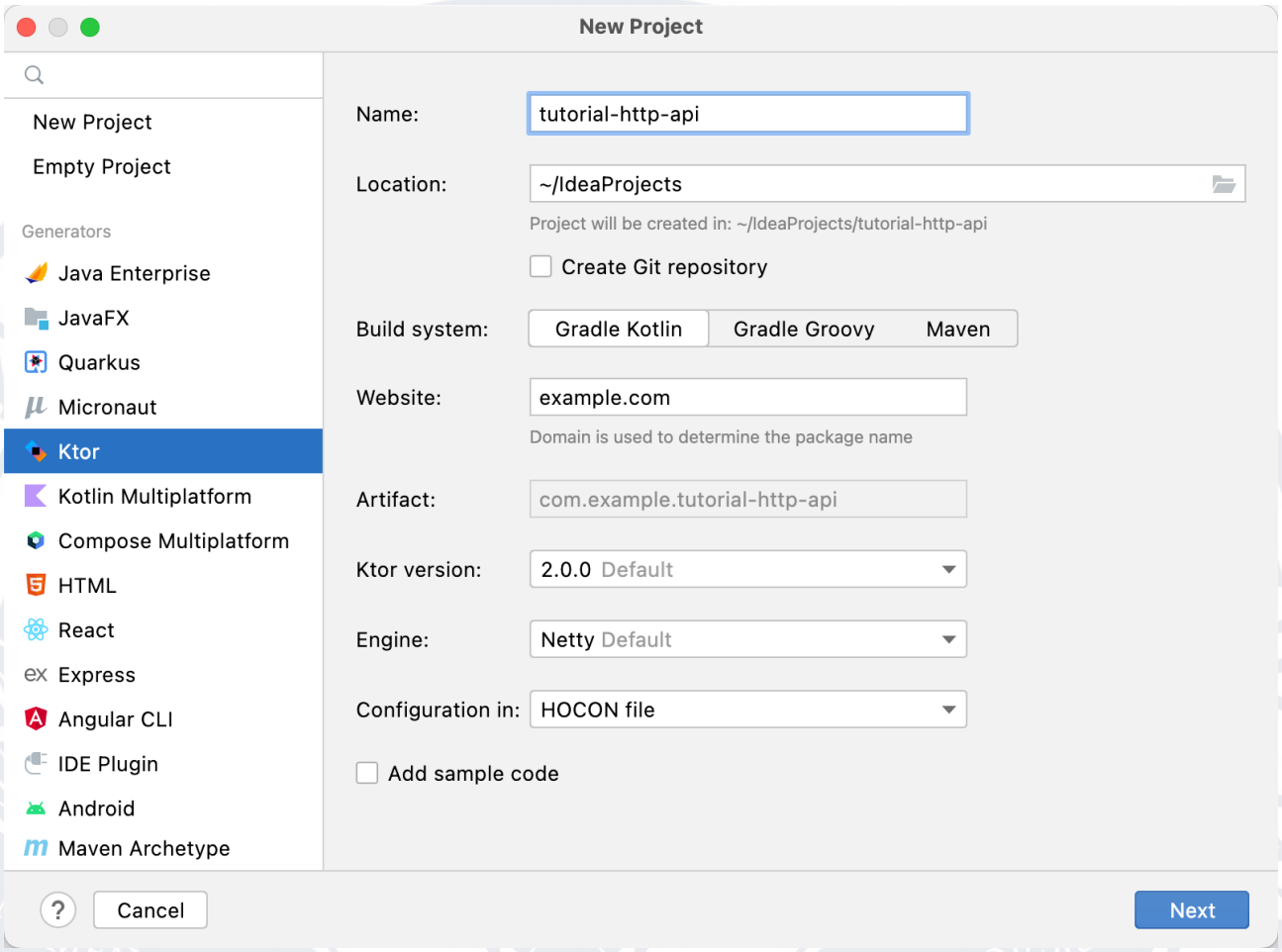

Рисунок 2.8 - Створення Ktor проекту

1.2.1. Назва: вкажіть назву проекту.

1.2.2. Розташування: вкажіть каталог для вашого проекту.

1.2.3. Система збірки: переконайтеся, що як систему збірки вибрано Gradle Kotlin.

1.2.4. Веб-сайт: залиште значення за замовчуванням example.com як домен, який використовується для створення імені пакета.

1.2.5. Артефакт: у цьому полі відображається назва згенерованого артефакту.

1.2.6. Версія Ktor: виберіть останню версію Ktor.

1.2.7. Механізм: залиште механізм Netty за умовчанням.

1.2.8. Конфігурація в: виберіть файл HOCON, щоб указати параметри сервера у спеціальному файлі конфігурації.

1.2.9. Додати зразок коду: вимкніть цей параметр, щоб пропустити додавання зразка коду для плагінів.

Натисніть “Далі”.

1.3. На наступній сторінці додайте плагіни Routing, ContentNegotiation і kotlinx.serialization:

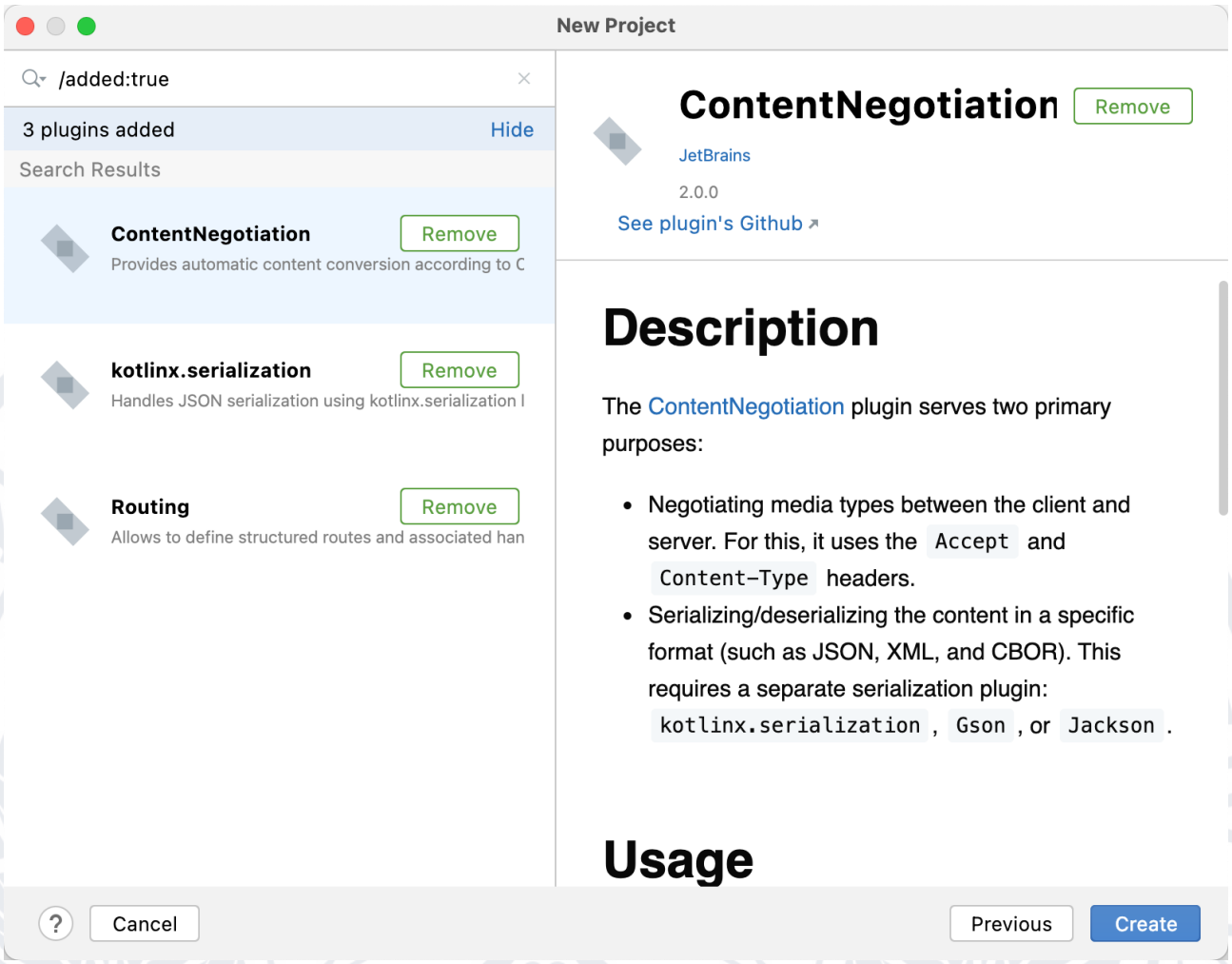

Рисунок 2.9 - Конфігурація плагінів

Натисніть “Створити”.

Після створення проект вже має згенерований код який можна переглянути у “Project View”:

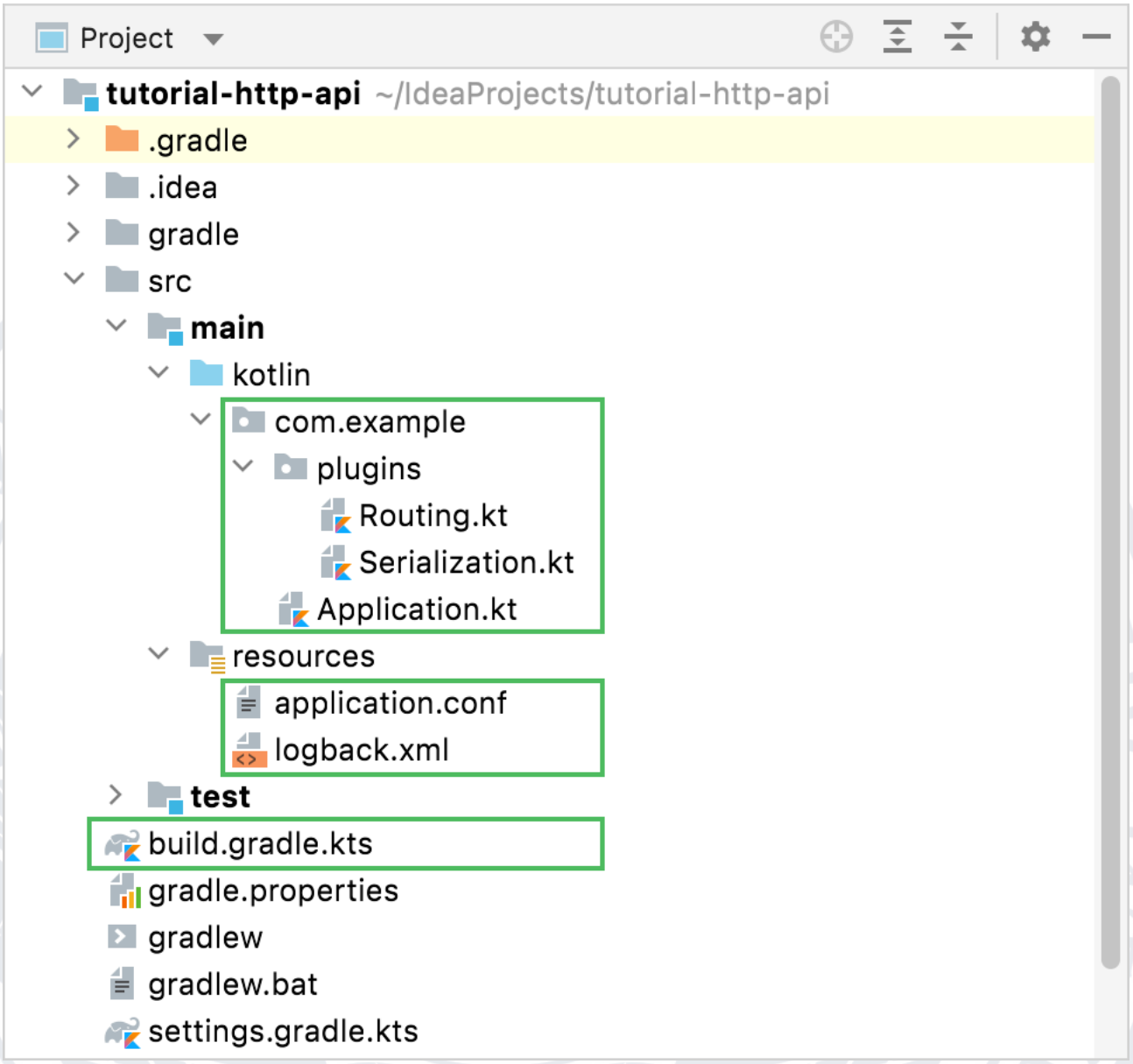

Рисунок 2.10 - Project View

- Файл build.gradle.kts містить залежності, необхідні для сервера Ktor і плагінів.
- Папка main/resources містить конфігураційні файли.
- Папка main/kotlin містить згенерований вихідний код.

Залежності у згенерованому проекті:

```
dependencies {
    implementation("io.ktor:ktor-server-core:$ktor_version")
    реалізація ("io.ktor:ktor-server-netty:$ktor_version")
    implementation("io.ktor:ktor-server-content-negotiation:$ktor_version")
    implementation("io.ktor:ktor-serialization-kotlinx-json:$ktor_version")
    implementation("ch.qos.logback:logback-classic:$logback_version")
    testImplementation("io.ktor:ktor-server-test-host:$ktor_version")
    testImplementation("org.jetbrains.kotlin:kotlin-test-junit:$kotlin_version")
}
```

- ktor-server-core додає основні компоненти Ktor до проекту.
- ktor-server-netty додає механізм Netty до проекту, дозволяючи використовувати функціональні можливості сервера без необхідності покладатися на зовнішній контейнер програми.
- ktor-server-content-negotiation і ktor-serialization-kotlinx-json забезпечують зручний механізм для перетворення об’єктів Kotlin у серіалізовану форму, наприклад JSON, і навпаки. Застосовується для форматування виводів API та використання введених даних користувача, які структуровані в JSON. Щоб використовувати ktor-serialization-kotlinx-json, також потрібно застосувати плагін plugin.serialization.

```
plugins {
    id("org.jetbrains.kotlin.plugin.serialization") version "1.7.10"
}
```

- logback-classic забезпечує реалізацію SLF4J, що дозволяє бачити добре відформатовані журнали в консолі.
- ktor-server-test-host і kotlin-test-junit дозволяють тестувати частини програми Ktor без використання всього стеку HTTP в процесі. Використається для визначення модульних тестів для нашого проекту.

2.7 Проектування бази даних

З одного боку, процес проектування структури БД є творчим, неоднозначним, з іншого боку, його вузлові моменти можуть бути формалізовані. У процесі розробки логічні моделі даних постійно тестуються та перевіряються на відповідність вимогам користувачів. Коректність логічної моделі даних забезпечується процедурою нормалізації.

Процедура нормалізації бази даних усуває надмірність даних та виявляє функціональні залежності. Усунення надмірності даних забезпечує компактність наборів даних, уникаючи непотрібного дублювання та запобігаючи аномалії при вставці, видаленні та оновленні кортежів після фізичної реалізації бази даних. Функціональна залежність пов'язує атрибути в одному відношенні з єдиним значенням в іншому.Функціональну залежність для відношень A та B прийнято позначати як A→B. Це поняття підводить “на один крок” до спорідненої концепції об'єднання відношень зв'язками типу один до одного (1:1) або один до багатьох (1:M).

Правила нормалізації або правила Кодда, як їх тепер називають, дуже прості й нечисленні, але досить суворі. У разі застосування до відношень кожне правило описує наступний рівень відповідності вимогам теорії реляційних БД і різні ступені нормалізації.

Розрізняють такі рівні нормалізації: перша нормальна форма (1НФ), 2НФ, 3НФ, нормальна форма Бойса-Кодда (БКНФ), 4НФ, 5НФ. Проте на сьогоднішній день жодна із систем управління реляційними базами даних

(СУБД) не підтримує належним чином усі п'ять нормальних форм. Це з жорсткими вимогами до продуктивності. Суть у тому, що повністю нормалізована база даних вимагатиме підключення такої кількості таблиць для виконання запиту, що продуктивність такої системи не задовольнятиме користувачів. Тому на практиці використовуються лише перші три рівні нормалізації (1НФ, 2НФ, 3НФ).

Перша нормальна форма

Відношення буде зведено до першої нормальної форми (1НФ) тоді й тільки тоді, коли всі його атрибути містять тільки неподільні (атомарні) значення й у ньому відсутні групи атрибутів з однаковими за змістом значеннями, які повторюються у межах одного кортежу.

Неподільність значення атрибута говорить про те, що його не можна розбити на більш дрібні частини. Наприклад, якщо у атрибуті «Прізвище, ім'я, по батькові» міститься прізвище, ім'я та по батькові читача, вимога неподільності не виконується. Тут необхідно виділити в окремі атрибути ім'я та по батькові. У результаті вийде три атрибути відношення «Читачі»: «Прізвище», «Ім'я» і «По батькові».

а

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Перв.
2.	Прізвище, Ім'я, По батькові	
3.	Серія паспорта	По-тенц.
4.	№ паспорта	
5.	Дата народження	
6.	Місце народження	
7.	Стать	
8.	Місце видачі паспорта	
9.	Дата видачі паспорта	
10.	Місце проживання	
11.	Місце основної роботи	
12.	Посада	
13.	№ телефону	

б

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Перв.
2.	Прізвище	
3.	Ім'я	
4.	По батькові	
5.	Серія паспорта	По-тенц.
6.	№ паспорта	
7.	Дата народження	
8.	Країна місця народження	
9.	Адмін. утворення місця народження	
10.	Населений пункт місця народження	
11.	Стать	
12.	Країна місця видачі паспорта	
13.	Адмін. утворення місця видачі паспорта	
14.	Населений пункт місця видачі паспорта	
15.	Дата видачі паспорта	
16.	Країна місця проживання	
17.	Адмін. утворення місця проживання	
18.	Населений пункт місця проживання	
19.	Ж.м./Просп./Вул./Пров.	
20.	Будинок	
21.	Корпус	
22.	Квартира	
23.	Місце основної роботи	
24.	Тип підприємства	
25.	Назва підприємства	
26.	Посада	
27.	Домашній телефон	
28.	Робочий телефон	
29.	Мобільний телефон	

Рисунок 2.11 (а,б) - Приклад таблиці бази даних

На більш дрібні частини можна розбити атрибути: «Місце народження» («Країна», «Адміністративне утворення», «Населений пункт»); «Місце видачі паспорта» («Країна», «Адміністративне утворення», «Населений пункт»);

«Місце основної роботи» («Тип підприємства», «Назва підприємства»), «Місце проживання» («Країна», «Адміністративне утворення», «Населений пункт», «Житловий масив / Проспект / Вулиця / Провулок», «Будинок», «Корпус», «Квартира»).

Для контакту читач може визначити один, декілька або жодного номеру телефону. Таким чином у загальному випадку інформація у атрибуті «Номер телефону» може бути розділена на декілька частин, кожна з яких є окремим телефонним номером. На перший погляд, цю проблему можна вирішити так само, як і для прізвища, імені та по батькові, виділивши для найпоширеніших типів телефонів окремі атрибути. Однак у цьому випадку ми зіткнемося з групою атрибутів, що мають однакові за змістом значення в межах одного кортежу, наприклад: «Домашній телефон», «Робочій телефон», «Мобільний телефон».

Щоб відношення «ЧИТАЧІ» відповідало 1НФ, треба вилучити з нього групу атрибутів з номерами телефонів, які повторюються в межах одного кортежу, в інше відношення разом з копією ключового атрибута «№ квитка читача». Причому, для подання номера й типу телефона виділимо окремі атрибути. Це дозволяє: по-перше, урахувати не тільки три зазначені типи телефона, але й додати нові і по-друге, зазначити для кожного читача тільки ті типи телефонів, які в нього є, і по-третє, можна указувати для будь-якого читача кілька однотипних телефонів або не указувати жодного номера телефону.

ЧИТАЧІ

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Перв.
2.	Прізвище	
3.	Ім'я	
4.	По батькові	
5.	Серія паспорта	По-тенц.
6.	№ паспорта	
7.	Дата народження	
8.	Країна місця народження	
9.	Адмін. утворення місця народження	
10.	Населений пункт місця народження	
11.	Стать	
12.	Країна місця видачі паспорта	
13.	Адмін. утворення місця видачі паспорта	
14.	Населений пункт місця видачі паспорта	
15.	Дата видачі паспорта	
16.	Країна місця проживання	
17.	Адмін. утворення місця проживання	
18.	Населений пункт місця проживання	
19.	Ж.м./Просп./Вул./Пров.	
20.	Будинок	
21.	Корпус	
22.	Квартира	
23.	Місце основної роботи	
24.	Тип підприємства	
25.	Назва підприємства	
26.	Посада	

ТЕЛЕФОНИ

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Зовн.
2.	Тип телефону	
3.	№ телефону	

Рисунок 2.12 - Приклад таблиі бази даних

Друга нормальна форма

Відношення буде зведено до другої нормальної форми (2НФ) тоді й тільки тоді, коли воно є в першій нормальній формі, і кожний неключовий атрибут повністю визначається первинним ключем, тобто щоб первинний ключ

однозначно визначав кортеж і не був надлишковим (збігався із суперключем). Ті атрибути, які залежать тільки від частини суперключа, мають бути виділені в окремі таблиці.

Відношення «ЧИТАЧІ» не зведено до форми 2НФ. У ньому кожен кортеж однозначно ідентифікується такими атрибутами: «№ квитка читача», «Серія паспорта» і «№ паспорта». Сукупність цих атрибутів є суперключем цього відношення. Він складається з двох потенційних ключів, кожен з яких окремо може ідентифікувати кортежі відношення. Із двох потенційних ключів за первинний вибирається той, довжина якого мінімальна. Наявність обох потенційних ключів обумовлена вимогами користувачів.

Звести відношення «ЧИТАЧІ» до 2НФ можна, якщо винести в окреме відношення атрибути 2 – 22, які стосуються паспортних даних, і копію первинного ключа «№ квитка читача». Однак у результаті ми одержимо відношення «ПАСПОРТНІ ДАНІ», яке має такий самий суперключ, як і відношення «ЧИТАЧІ» до його зведення до 2НФ. У нашому випадку подальша нормалізація відношення «ПАСПОРТНІ ДАНІ» неможлива.

ПАСПОРТНІ ДАНІ

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Зовн.
2.	Прізвище	
3.	Ім'я	
4.	По батькові	
5.	Серія паспорта	По-тенц.
6.	№ паспорта	
7.	Дата народження	
8.	Країна місця народження	
9.	Адмін. утворення місця народження	
10.	Населений пункт місця народження	
11.	Стать	
12.	Країна місця видачі паспорта	
13.	Адмін. утворення місця видачі паспорта	
14.	Населений пункт місця видачі паспорта	
15.	Дата видачі паспорта	
16.	Країна місця проживання	
17.	Адмін. утворення місця проживання	
18.	Населений пункт місця проживання	
19.	Ж.м./Просп./Вул./Пров.	
20.	Будинок	
21.	Корпус	
22.	Квартира	

ЧИТАЧІ

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Перв.
2.	Місце основної роботи	
3.	Тип підприємства	
4.	Назва підприємства	

ТЕЛЕФОНИ

№ п.п.	Атрибути	Ключ
1.	№ квитка читача	Зовн.
2.	Тип телефону	
3.	№ телефону	

Рисунок 2.13 - Приклад таблиці бази даних

У відношенні «ТЕЛЕФОНИ» на перший погляд здається, якщо в опис логічної моделі додати номер телефону читача, включаючи код країни, оператора або населеного пункту, то потенційним буде ключ, у складі якого всього один атрибут – «№ телефона». Однак у двох різних читачів може бути однаковий однаковим номер його домашнього або робочого телефона. Отже, однозначно, ідентифікувати кортежі можливо за складеним суперключем, до якого входять атрибути «№ квитка читача» та «№ телефону». Ми бачимо, що суперключ не є надлишковим. Його зміст збігається з первинним ключем. Виходить, що відношення «ТЕЛЕФОНИ» презентовано у другій нормальній формі.

Третя нормальна форма

1НФ та 2НФ зазвичай вважаються проміжними етапами в процесі нормалізації бази даних. Більшість СУБД спрямовані на досягнення наступного рівня нормалізації, третьої нормальної форми (3НФ). Це з тим, що приведення відносин до 3НФ чудово відповідає майже всім практичним завданням. При розробці великих систем на швидких комп'ютерах бажано додатково нормалізувати відносини, якщо обсяг даних необхідно зменшити до максимуму.

Відношення буде зведено до третьої нормальної форми (3НФ) тоді й тільки тоді, коли воно є у другій нормальній формі і у ньому немає транзитивних залежностей між неключовими атрибутами, тобто значення будь-якого атрибута відношення, що не входить до первинного ключа, не залежить від значення іншого атрибута, що не входить до первинного ключа.

Це визначення – усього лише оригінальний спосіб відобразити необхідність подання системи пов'язаних відношень у такому вигляді, щоб значення атрибутів кожного відношення безпосередньо визначалися або суперключем, або потенційним ключем цього відношення.

Існує метод розрахунків мінімального числа відношень, яке необхідно для зведення всіх відношень БД до 3НФ. У тому випадку, якщо Ви склали список усіх функціональних залежностей для Ваших даних, то можна застосувати алгоритм Бернштейна, який описаний у будь-якому підручнику реляційної алгебри.

Розглянемо простий приклад з обліку товару на складі. Здається логічним, щоб у відношення з оперативною інформацією про товар на складі входили у число інших такі атрибути: «Кількість товару», «Ціна за одиницю товару» та «Загальна вартість товару». Вони не є ключовими. Для одержання значення атрибута «Загальна вартості товару» необхідно перемножити між собою значення, що перебувають у атрибутах: «Кількість товару» та «Ціна за одиницю товару», тобто значення атрибута «Загальної вартості товару» залежить від

величин двох атрибутів, що не входять до складу первинного ключа. Це суперечить визначенню 3НФ. Щоб дане відношення відповідало третій нормальній формі з нього треба вилучити атрибут «Загальна вартість товару».

Відзначимо, що відношення «ТЕЛЕФОНИ» і «ЧИТАЧІ» (рис. 3.3) відповідають третій нормальній формі. Це випливає з того, що вони зведені до другої нормальної форми й у них немає транзитивних залежностей.

Нижче наведений варіант визначення 3НФ, яка називається нормальною формою Бойса-Кодда (Boyce – Codd) – БКНФ, де встановлюються більш суворі вимоги.

Відношення X буде зведено до нормальної форми Бойса-Кодда тоді, коли в кожній нетривіальній функціональній залежності B→A B є суперключем.

Четверта та п’ята нормальні форми

У відношенні X існує багатозначна залежність A→B тоді, коли в ньому можна виявити ситуації, де пара кортежів містить значення A, які дублюються, і одночасно існують інші пари кортежів, що отримані шляхом перестановки значень B, які присутні у першій парі.

Насамперед, для існування багатозначної залежності потрібна наявність пар кортежів. A й B можуть бути як окремими атрибутами, так і об'єднанням деякого набору атрибутів. Тривіальна багатозначна залежність для A→B існує тоді, і тільки тоді, коли B є підмножиною A або Aпоєднує B = XS (більше відношення містить вихідне відношення).

Існування багатозначної залежності породжує аномалію оновлення. 4НФ усуває нетривіальну багатозначну залежність у відношенні за допомогою створення менших відношень. Процес нормалізації полягає в створенні як можна більшого числа дрібних відношень з метою зменшення надмірності даних.

Відношення X буде зведено до четвертої нормольної форми (4НФ) тоді й тільки тоді, коли воно є у БКНФ і для будь-якої багатозначної залежності A→B у цьому відношенні можна сказати, що вона є або тривіальною, або A є суперключем таблиці X.

П’ята нормальна форма (5НФ) досягається в тому випадку, коли відношення не може далі розбиватися на більш малі відношення за допомогою операції проектування.

Під операцією проектування розуміється декомпозиція даних (без їх утрати), при якій відношення розбивається на частини (кожна з них є окремим відношенням) таким чином, щоб залишалася можливість об'єднати ці частини у вихідне відношення.

Нормалізація - за й проти

Метою нормалізації відношень БД є усунення надлишкової інформації. Як видно з наведених вище прикладів, нормалізовані відношення БД містять тільки один елемент надлишкових даних – це атрибути зв'язку, які присутні одночасно в батьківському та дочірньому відношеннях. Оскільки надлишкові дані у відношеннях не зберігаються, то заощаджується місце на носіях інформації. Однак у нормалізованій БД є й недоліки, насамперед, практичного характеру.

Чим більше суб'єктів (сутностей) охоплюється предметною областю, тим з більшого числа відношень буде складатися нормалізована БД. Бази даних у складі великих систем, які задіяні у життєдіяльності великих організацій та підприємств, можуть містити сотні зв'язаних між собою відношень. Оскільки поріг людського сприйняття не дозволяє одночасно охопити велику кількість об'єктів з урахуванням їх взаємозв'язків, то можна стверджувати, що зі збільшенням числа нормалізованих відношень цілісне сприйняття БД як системи взаємозалежних даних зменшується. Тому при розробці й експлуатації великих систем існують ситуації, коли кожний співробітник уявляє собі процеси, що відбуваються тільки в частині системи. Відомі випадки

еволюційного створення таких систем, коли принципи їх функціонування згодом виходили за межі розуміння.

Іншим недоліком нормалізованої БД є необхідність зчитування з відношень зв'язані дані при виконанні складних запитів, які надають інформацію про взаємодію сутностей технологічного процесу між собою. При великих обсягах це призводить до збільшення часу доступу до даних.

Третя нормальна форма та нормальна форма Бойса-Кодда є теоретичними конструкціями, у той час як більшість розроблювачів БД працюють у реальному світі. Тому доречно зробити кілька зауважень про недоліки, що властиві відношенням, які зведені до 3НФ. Існують варіанти, коли має сенс поділити відношення на декілька дрібних, якщо частина наведених у ньому даних непостійна й часто оновлюється (оперативна інформація), а інші дані пасивні та змінюються не так часто (довідкова інформація). Також є сенс у разі необхідності об'єднати відношення, щоб забезпечити високу швидкість реакції на запит. Можна навіть піти на дублювання даних у відношеннях, якщо це дозволить зменшити витрати на обробку запитів, хоча формально не варто було б цього робити.

Необхідно також відзначити, що відношення між собою доцільно з’єднувати по атрибутах, які зовсім звільнені від семантичної залежності, що породжується особливостями реалізації дійсних процесів у БД. Для цього використовують спеціально виділені інкрементні атрибути однозначної ідентифікації кортежів усередині відношень. Такий підхід дозволяє позбавитися необхідності процесу перевизначення зв'язків між відношеннями, що виникає в реальному житті при зміні правил обліку різних суб'єктів діяльності організації.

Наведені вище міркування не слід сприймати як заклик зовсім не нормалізувати дані. Вони лише мають довести, що при роботі з даними великого обсягу доводиться шукати компроміс між вимогами нормалізації (тобто "логічності" даних і економії місця на носіях інформації) та необхідністю

збільшення швидкодії. При цьому треба звернути увагу на вимоги користувачів, щоб уникнути зайвої деталізації суб'єктів реальних процесів, які відбуваються на підприємстві.

Якщо користувачу немає сенсу деталізувати атрибути «Місце народження», «Місце видачі паспорта», «Місце основної роботи», «Місце проживання», то пошук компромісу між вимогами нормалізації й швидкодією для відношення «ЧИТАЧІ» (рис. 3.1, а) приведе до того, що, по суті, це відношення може стати не нормалізованим (рис. 3.4). Необхідно звернути увагу, що паспортні дані читачів також не винесені в окреме відношення. Це дозволяє скоротити час виконання запиту, який видає всі наявні дані про читачів, оскільки немає сенсу з'єднувати відношення «ПАСПОРТНІ ДАНІ» та «ЧИТАЧІ».

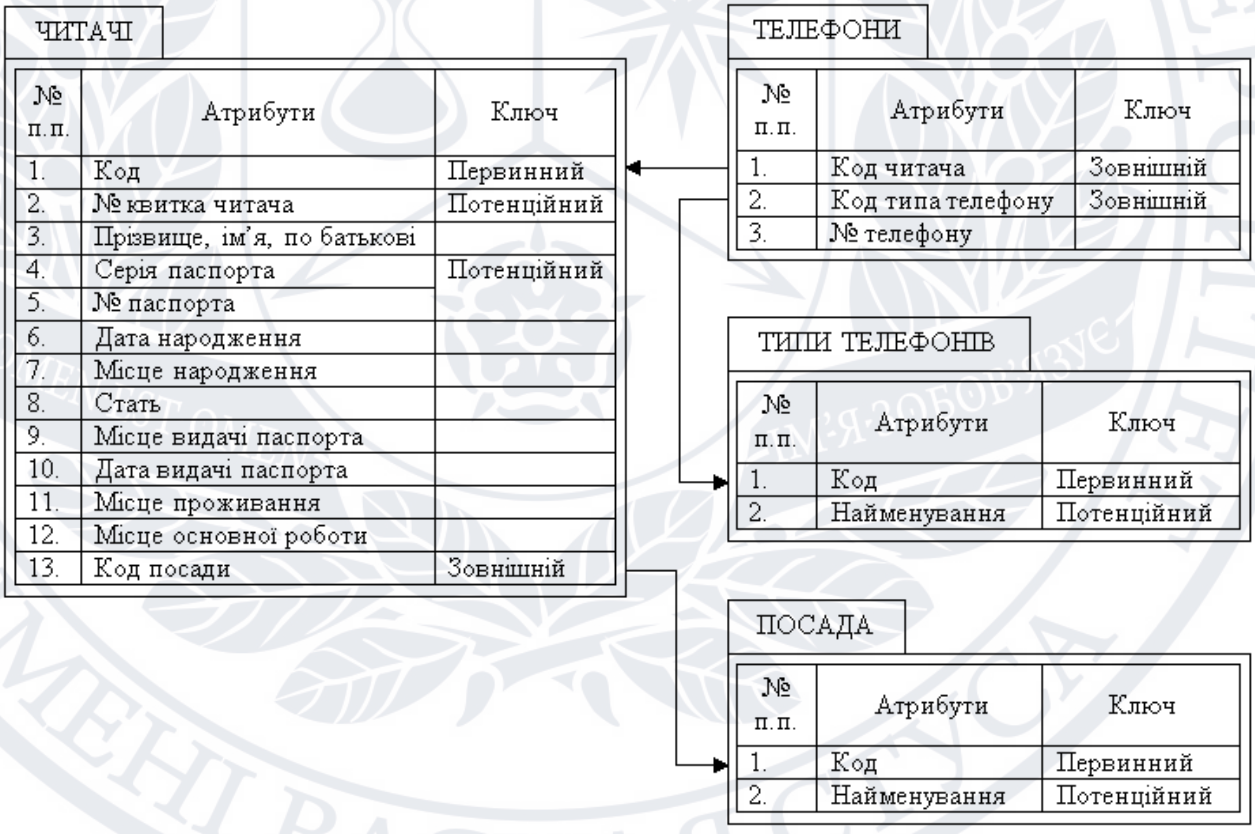

Рисунок 2.14 - Результат нормалізації відношення «ЧИТАЧІ»

На відміну від паспортних даних відомості про телефонні номери читачів винесені в окреме відношення. Це пов'язане з тим, що у читача може бути як

кілька контактних телефонів, так і не бути їх взагалі. Інакше кажучи, зв'язок 1:M між суб'єктами обліку та діяльності організації у більшості випадків слід реалізувати за допомогою двох відношень. Зв'язок 1:1 найчастіше реалізується в одному відношенні.

На користь реалізації зв'язку 1:1 між суб'єктами обліку більш ніж в одному відношенні говорить наявність у вимогах користувачів необхідності одержання узагальнюючої інформації відносно якого-небудь атрибута. Саме це виправдовує винесення даних про тип телефона та посади читачів в окремі таблиці. У цьому разі посада та типа телефона наводяться один раз.

Якщо користувачі будуть вносити назву посади або тип телефона у відповідні атрибути відношень «ЧИТАЧІ» і «ТЕЛЕФОНИ», то розроблювачі не зможуть гарантувати, що із БД запити будуть вибирати коректну інформацію. Це пов'язане з тим, що назву однієї й тієї ж посади або типу телефона оператор може внести із синтаксичною помилкою. У цьому разі стандартні алгоритми обробки запитів користувачів до БД будуть інтерпретувати однакові по суті значення відповідного атрибута як різні.

Запит користувача на одержання узагальнюючої інформації може бути сформульований так: «вивести на екран робочі номери телефонів читачів із вказівкою їх прізвищ, імен, по батькові, місця роботи та посади», або так: «вивести на екран контактні номери телефонів всіх асистентів із вказівкою їх прізвищ, імен, по батькові».

Уведення інкрементних атрибутів «Код» дозволить цілком відмовитися від необхідності зміни властивостей ключових атрибутів у зв'язку зі змінами правил обліку читачів у бібліотеці. Вони дозволили трохи компенсувати збільшення необхідного обсягу пам'яті для реалізації БД за рахунок зменшення місця для атрибута відношення, що пов'язує «ЧИТАЧІ» та «ТЕЛЕФОНИ» з відношеннями «ПОСАДА» та «ТИПИ ТЕЛЕФОНІВ» відповідно. Ці відношення можна було б зв'язати з атрибутами «Найменування посади» та

«Найменування типу телефона» замість атрибутів «Код посади» та «Код типу телефона». Економія буде залежить тільки в кількості знаків цих назв, яка закладена згідно з вимогами користувачів. Результат нормалізації відношення «ЧИТАЧІ» може бути іншим. Він залежить від вимог користувачів, досвіду та погляду розробника на процедуру нормалізації відношень, необхідних для вирішення поставлених задач. У нашому прикладі ми одержали з одного відношення «ЧИТАЧІ» чотири відношення: «ЧИТАЧІ», «ТЕЛЕФОНИ», «ТИПИ ТЕЛЕФОНІВ» і «ПОСАДА». Відношення «ЧИТАЧІ» ненормалізоване. Відношення «ТЕЛЕФОНИ» зведено до 3НФ. Відношення «ТИПИ ТЕЛЕФОНІВ» і «ПОСАДА» зведені до 1НФ.

2.8 Методи перевірки на плагіат

Плагіат – видача чужого наукового тексту за власний або використання такого тексту в своїх роботах, не вказуючи посилання на автора. Також розрізняють термін запозичення — використання у тексті роботи фрагментів другої роботи. У Законі України "Про авторське право та суміжні права" плагіат визначається як оприлюднення (опублікування), повністю або частково, чужого твору під іменем особи, яка не є автором цього твору.

Алгоритми пошуку плагіату у текстах можна розділити на дві основні категорії: глобальні та локальні. Глобальними називають ті, які використовують певні знання про усі документи, що розглядаються, а інші — локальні. Основна ідея локальних алгоритмів зводиться до синтаксичного аналізу документа. Наприклад, можна обчислити хеш-функцію від конкатенації двох найдовших речень, знайдених у тексті. Перевагами такого підходу є його швидкодія та те, що можна зробити висновок про наявність плагіату у документі. До недоліків можна віднести його негнучкість, а також відсутність можливості перевірки усіх речень в тексті при використанні оригінальної варіації алгоритму.

Більш ефективним можна вважати метод частоти слова (term frequency). Обраховується відношення числа входжень певного слова до усієї кількості слів

у тексті. Недостатньо точний, але дозволяє виявити плагіат у випадках, коли текст скопійованої роботи не змінювався кардинально.

У системах перевірки плагіату використовується також алгоритм Moodle Crot. Його суть полягяє у видаленні з тексту документа усіх слів, що мають довжину менше трьох символів, небуквені знаки, пробіли, дефіси, крапки, коми тощо. Таким чином, отримуємо ланцюжок букв. Потім він з кроком n ділиться на частини по N символів у кожній. Далі потрібно обчислити хеш-функцію від кожної частини і результат обчислення зберігається в набір. Далі порівнюються два різних набори хеш-сум. Найбільша точність алгоритму буде при n=1, але при цьому він буде працювати найменш продуктивно. Перевагою даного алгоритму можна назвати точність та гнучкість. Недоліком є те, що при збільшенні точності до n=1, його продуктивність значно падає. Тому Moodle Crot не доцільно використовувати при швидкій перевірці текстів.

Метод шинглів найпопулярніший та найшвидший при пошуку плагіату в довільних текстах. Саме його варіації використовують пошукові машини та сервіси аналізу текстів на плагіат.

Шингли − послідовності слів. Спочатку з тексту видаляються усі сполучники, спеціальні символи, короткі слова, крім пробілів. Далі з отриманих слів складаються самі шингли — послідовності. Плагіатом можна вважати збіг у 6 слів, тому коротші послідовності не враховуються. Від кожного шингла обраховується хеш-сума, потім створюється набір хеш-сум документа. Аналогічна операція виконується для другого документа. Таким чином, порівнюються шингли, і якщо у документах співпадають один або більше шинглів, тексти можуть вважатися подібними.

Існує ряд модифікацій алгоритма на основі шинглів. Наприклад, одна із модифікацій базується на сортуванні, тобто перед обчисленням хеш-суми слова у шинглі будуть відсортовані. Це покращує пошук запозичень у випадках, коли

слова у тексті проаналізованої роботи були переставлені місцями для обману системи анти-плагіату.

До переваг простого алгоритму шинглів можна віднести гнучкість, точність, швидкість та популярність. Проте недоліком є те, що в оригінальній версії порівнюється довільна кількість хеш- сум шинглів, а не всіх, тому нема можливості вказати всі конкретні місця плагіату в документі. На рис. 2.15 представлені етапи роботи системи анти-плагіату, яка використовує алгоритм шинглів.

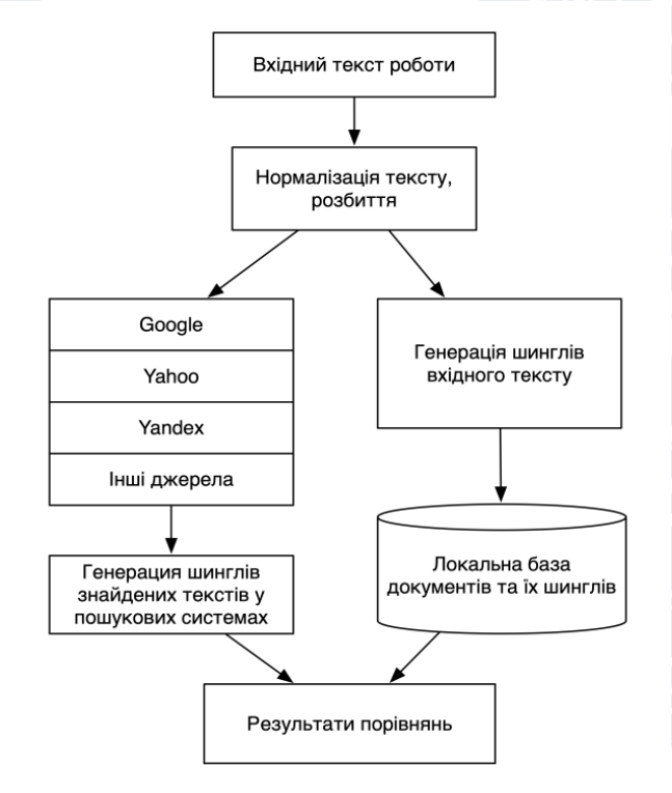

Рисунок 2.15 - Схема алгоритму шинглів

Етап нормалізації тексту призначений для виявлення та видалення прийомів обману систем-анти плагіату. Прийоми можна розділити на 2 основні категорії: обман людини та обман системи.

Для приховування факту плагіату від людини використовують перестановки слів, абзаців, додавання слів, що не несуть змісту, але візуально дозволяють тексту стати унікальним. Знаючи про можливу перевірку робіт

автоматизованими алгоритмами, плагіатори використовують прийоми, спрямовані саме на обман програми.

Найпростіший варіант — заміна літер з кирилиці на латиницю. Цей прийом досить очевидний, як і перестановка слів. Тому на етапі нормалізації тексту (рис. 2.15), виконуються перетворення та сортування. Відомий також метод синонімізації слів. Але програми-синонімайзери майже завжди не враховують контекст вживання слова, тому початковий зміст втрачається. Важливий недолік даного способу − мала кількість українських словників у загальному доступі.

Найбільш реальний спосіб приховати плагіат на сьогодні – рерайт. Тобто переписування оригінальної роботи так, щоб структурно вона відрізнялась, але зміст був той самий. Якість рерайту може бути низькою, наприклад, проста перестановка слів, заміна прямої мови на непряму, використання синонімів.

У випадку якісного рерайту, зміст передається той ж самий, під іншим кутом зору, але маніпулює тими ж фактами, що і в джерелі. Саме такий рерайт іноді неможливо визнати плагіатом, тому що потрібно довести, що при різних на вигляд текстах не було додано нових досліджень і фактів з теми. Рерайт вимагає багато часу та праці людини — це основна відмінність від наведених вище прийомів приховування плагіату у текстах. Реалізувати його автоматичним шляхом наразі неможливо.

Після виконання етапу нормалізації та розбиття тексту на речення, залежно від налаштувать система антиплагіату шукає тексти за реченнями у пошукових системах і параллельно генерує їх шингли для порівняння з локальною базою. Отримані від пошукових систем короткі відповіді (сніпети) також проходять процес генерації шинглів. Потім шингли документа, що перевіряється, порівнюються із шинглами знайдених в Інтернеті текстів і в локальній базі шинглів документів. Якщо шингли співпадають, робиться висновок про наявність плагіату та розраховується його відсоток.

Висновок до розділу 2

В цьому розділі розглянуто сучасні архітектурні рішення і обрано MVI для реалізації клієнтської частини та MVC для реалізації серверної частини. Було розглянуто принципи проектування баз даних. Обрання правильних інструментів значно спрощує розробку та робить програмний код легко розширюваним та зрозумілим, що дуже важливо для підтримання продукту.

РОЗДІЛ 3

РОЗРОБК СЕРВЕРНОЇ ТА КЛІЄНТСЬКОЇ ЧАСТИНИ СИСТЕМИ

3.1 Розробка серверної частини

Розробка бази даних

Перш за все було спроектовано базу даних яка відповідає необхідним вимогам. База даних налічує 18 таблиць, а саме:

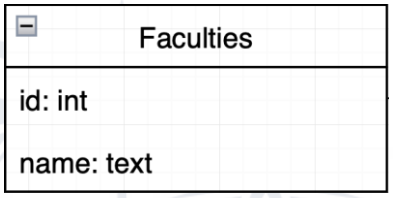

Рисунок 3.1 - Таблиця факультетів

Таблиця факультетів зберігає унікальний ідентифікатор факультету й назву факультету.

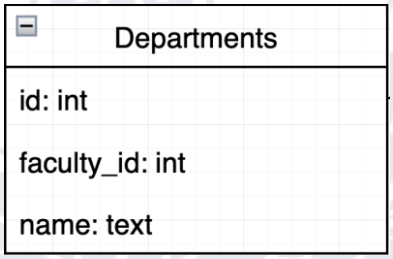

Рисунок 3.2 - Таблиця кафедр

Таблиця кафедр зберігає унікальний ідентифікатор кафедри, ідентифікатор факультету (оскільки кожна кафедра відноситься до певного факультету) й назву кафедри.

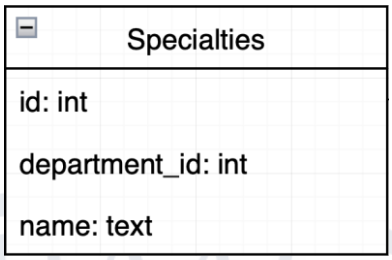

Рисунок 3.3 - Таблиця спеціальностей

Таблиця спеціальностей зберігає унікальний ідентифікатор спеціальності, ідентифікатор кафедри (оскільки кожна спеціальності відноситься до певної кафедри) й назву спеціальності.

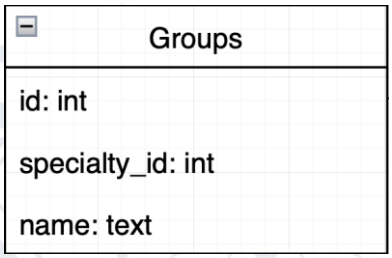

Рисунок 3.4 - Таблиця груп

Таблиця груп зберігає інформацію про групу студентів. Має унікальний ідентифікатор, ідентифікатор спеціальності (оскільки кожна групу відноситься до певної кафедри).

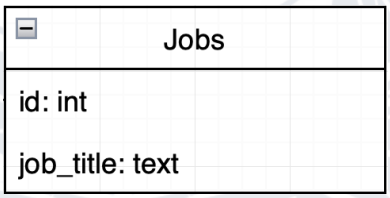

Рисунок 3.5 - Таблиця посад

Таблиця посад зберігає інформацію про посади які доступні працівникам закладу. Має унікальний ідентифікатор й назву посади.

Рисунок 3.6 - Таблиця ролей

Таблиця ролей зберігає інформацію про ролі які доступні користувачам. Має унікальний ідентифікатор й назву ролі.

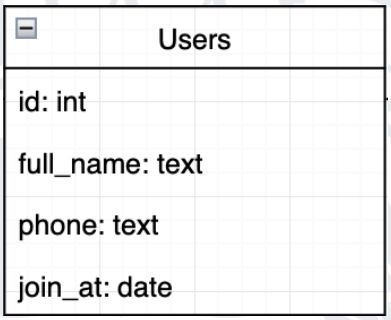

Рисунок 3.7 - Таблиця користувачів

Таблиця користувачів зберігає інформацію про певного користувача, а саме: його ім’я, номер телефону і коли він був зареєстрований у системі.

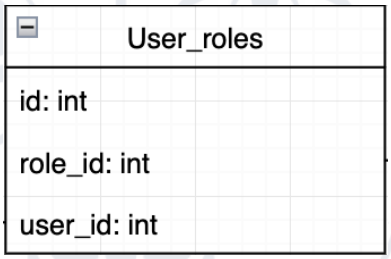

Рисунок 3.8 - Таблиця ролей користувача

Було додано таблицю ролей користувача оскільки кожен користувач може мати декілька ролей у системи. Має унікальний ідентифікатор, ідентифікатор ролі, ідентифікатор користувача.

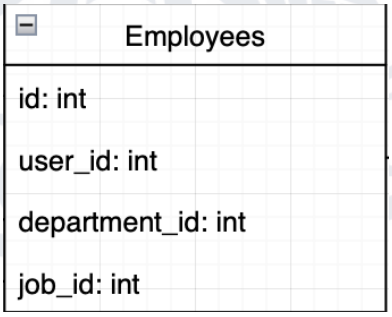

Рисунок 3.9 - Таблиця співробітників

Таблиця співробітників зберігає інформацію про первого співробітника установи, а саме: унікальний ідентифікатор, ідентифікатор користувача, ідентифікатор кафедри, ідентифікатор посади.

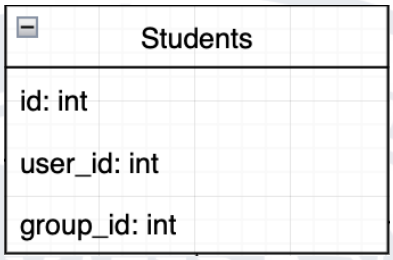

Рисунок 3.10 - Таблиця учнів

Таблиця учнів зберігає інформацію про первого учня установи, а саме: унікальний ідентифікатор, ідентифікатор користувача, ідентифікатор посади.

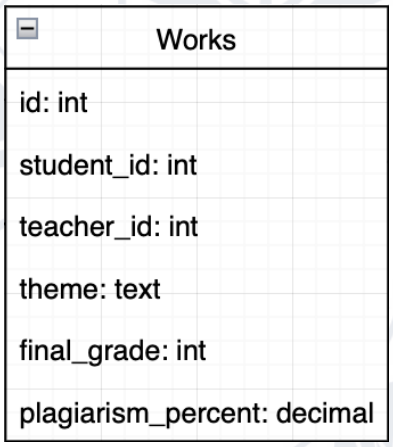

Рисунок 3.10 - Таблиця робіт

Таблиця робіт зберігає інформацію про бакалаврські та кваліфікаційні роботи, а саме: унікальний ідентифікатор, ідентифікатор учня, ідентифікатор викладача, тема роботи, кінцева оцінка роботи, процент плагіату.

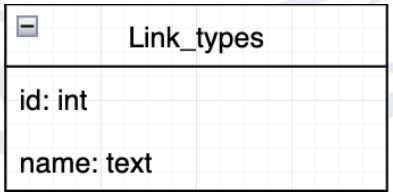

Рисунок 3.11 - Таблиця типів посилань у роботі

Таблиця типів посилань зберігає інформацію про тип посилання у певній роботі (наприклад посилання на текстову частину, посилання на презентацію, посилання на код програмної частини роботи, тощо). Має унікальний ідентифікатор, назву посилання.

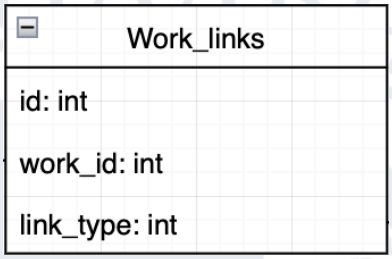

Рисунок 3.12 - Таблиця посилань роботи

Таблиця посилань роботи має унікальний ідентифікатор, ідентифікатор роботи, ідентифікатор типу посилання.

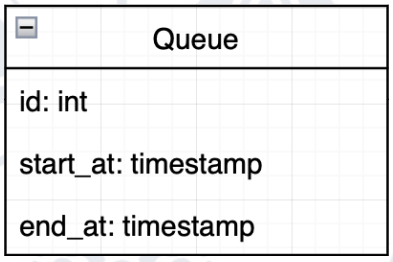

Рисунок 3.13 - Таблиця черг робіт

Таблиця черг зберігає унікальний ідентифікатор, дату та час початку захисту бакалаврських чи кваліфікаційних робіт, дату та час закінчення захисту бакалаврських чи кваліфікаційних робіт.

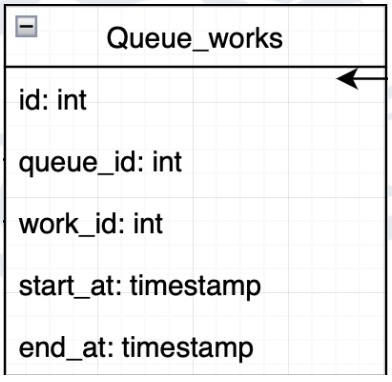

Рисунок 3.14 - Таблиця робіт у черзі

Таблиця робіт у черзі зберігає унікальний ідентифікатор, ідентифікатор черги, ідентифікатор роботи, дату та час початку захисту бакалаврської чи кваліфікаційної роботи, дату та час закінчення захисту бакалаврської чи кваліфікаційної роботи.

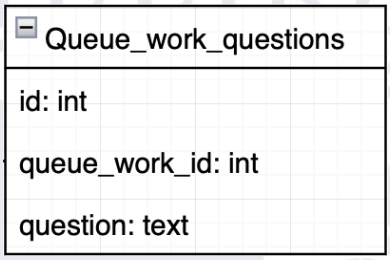

Рисунок 3.15 - Таблиця запитань до роботи

Таблиця запитань до роботи зберігає питання які були задані під час захисту бакалаврської чи кваліфікаційної роботи, а саме: унікальний ідентифікатор, ідентифікатор роботи у черзі, текст запитання.

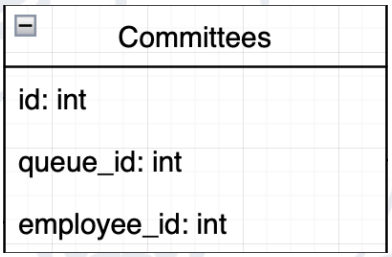

Рисунок 3.16 - Таблиця комісії

Таблиця комісії зберігає інформацію стосовно членів комісії які присутні на захисті бакалаврських чи кваліфікаційних робіт. Має унікальний ідентифікатор, ідентифікатор черги, ідентифікатор співробітника.

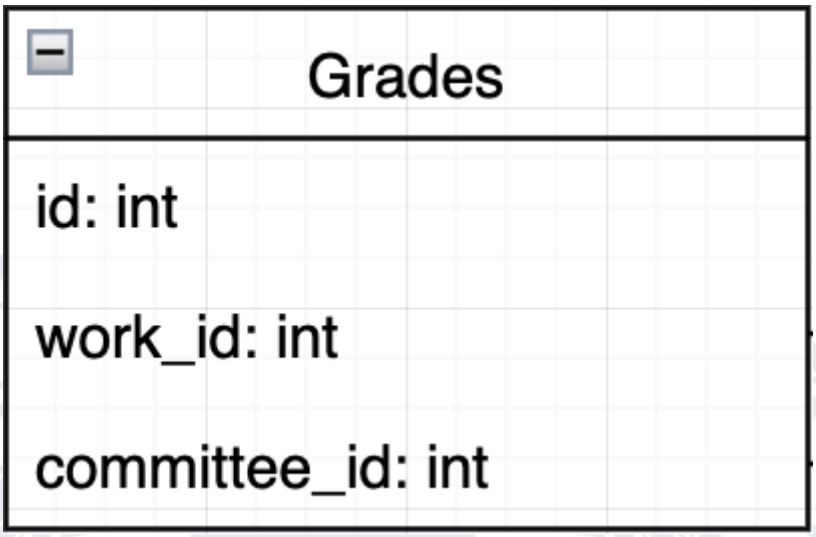

Рисунок 3.17 - Таблиця балів

Таблиця балів зберігає інформацію про бали які були виставлені членами комісії відповідній бакалаврській чи кваліфікаційній роботі.

Розробка REST API

Після розробки бази даних було розроблено серверну частину з відповідним REST API:

Взаємодія з таблицею Faculties (факультети) за допомогою відповідних запитів:

- POST /api/faculties - додавання нового факультету
- PUT /api/faculties/{id} - редагування відповідного факультету
- DELETE /api/faculties/{id} - видалення факультету
- GET /api/faculties/ - отримання всіх факультетів
- GET /api/faculties/{id} - отримання відповідного факультету

Взаємодія з таблицею Departments (кафедри) за допомогою відповідних запитів:

- POST /api/departments - додавання нової кафедри

- PUT /api/departments/{id} - редагування відповідної кафедри
- DELETE /api/departments/{id} - видалення кафедри
- GET /api/departments/ - отримання всіх кафедр
- GET /api/departments/{id} - отримання відповідної кафедри

Взаємодія з таблицею Specialties (спеціальності) за допомогою відповідних запитів:

- POST /api/specialties - додавання нової спеціальності
- PUT /api/specialties/{id} - редагування відповідної спеціальності
- DELETE /api/specialties/{id} - видалення спеціальності
- GET /api/specialties/ - отримання всіх спеціальностей
- GET /api/specialties/{id} - отримання відповідної спеціальності

Взаємодія з таблицею Groups (групи студентів) за допомогою відповідних запитів:

- POST /api/groups - додавання нової групи
- PUT /api/groups/{id} - редагування відповідної групи
- DELETE /api/groups/{id} - видалення групи
- GET /api/groups/ - отримання всіх груп
- GET /api/groups/{id} - отримання відповідної групи

Взаємодія з таблицею Jobs (посади) за допомогою відповідних запитів:

- POST /api/jobs - додавання нової посади
- PUT /api/jobs/{id} - редагування відповідної посади
- DELETE /api/jobs/{id} - видалення посади

- GET /api/jobs/ - отримання всіх посад
- GET /api/jobs/{id} - отримання відповідної посади

Взаємодія з таблицею Roles (ролі) за допомогою відповідних запитів:

- POST /api/roles - додавання нової ролі
- PUT /api/roles/id - редагування відповідної ролі
- DELETE /api/roles/{id} - видалення ролі
- GET /api/roles/ - отримання всіх ролей
- GET /api/roles/{id} - отримання відповідної ролі

Взаємодія з таблицями Users (користувачі), Employees (співробітники), Students (учні) за допомогою відповідних запитів:

- POST /api/employees - додавання нового співробітника
- POST /api/students - додавання нового учня
- PUT /api/employees/{id} - редагування відповідного співробітника
- PUT /api/students/{id} - редагування відповідного учня
- DELETE /api/employee/{id} - видалення співробітника
- DELETE /api/students/{id} - видалення учня
- GET /api/employees/ - отримання всіх співробітників
- GET /api/students/ - отримання всіх учнів
- GET /api/employees/{id} - отримання відповідного співробітника
- GET /api/students/{id} - отримання відповідного учня

Взаємодія з таблицями Link_types (типи посилань у роботі) за допомогою відповідних запитів:

- POST /api/link-types - додавання нового типу посилань
- PUT /api/link-types/{id} - редагування відповідного типу посилань
- DELETE /api/link-types/{id} - видалення типу посилань
- GET /api/link-types/ - отримання всіх типів посилань
- GET /api/link-types/{id} - отримання відповідного типу посилань

Взаємодія з таблицями Works (роботи), Work_links (посилання у роботі), за допомогою відповідних запитів:

- POST /api/works - додавання нової роботи
- PUT /api/works/{id} - редагування відповідної роботи
- DELETE /api/works/{id} - видалення роботи
- GET /api/works/ - отримання всіх робіт
- GET /api/works/{id} - отримання відповідної роботи

Взаємодія з таблицями Queue (черги), Queue_works (роботи у черзі), Committees (члени комісії), за допомогою відповідних запитів:

- POST /api/queue - додавання нової черги
- PUT /api/queue/{id} - редагування відповідної черги
- DELETE /api/queue/{id} - видалення черги
- GET /api/queue/ - отримання всіх черг
- GET /api/queue/{id} - отримання відповідної черги
- POST /api/queue/start - розпочати захисті робіт
- POST /api/queue/start/{id} - розпочати захист відповідної роботи
- POST /api/queue/finish/{id} - завершити захист відповідної роботи

Взаємодія з таблицями Queue_work_questions (запитання до роботи) відбувається за допомогою відповідних запитів:

- POST /api/queue/{id}/questions - додавання нового запитання
- PUT /api/queue/questions/{id} - редагування відповідного запитання
- DELETE /api/queue/questions/{id} - видалення роботи
- GET /api/queue/{id}/questions/ - отримання всіх запитань відповідної роботи

Взаємодія з таблицями Grades (бали), Works (роботи) відбувається за допомогою відповідних запитів:

- POST /api/queue/{id}/grades - виставлення бала за роботуу
- GET /api/queue/{id}/grades - отримання всіх балів відповідної роботи

Розробка механізму перевірки на плагіат за допомогою метода шинглів

Абстрактний клас для подібностей рядків (лістинг 6), які спираються на операції з наборами (наприклад, косинусну подібність або індекс жаккарда).

k-shingling — це операція перетворення рядка (або текстового документа) у набір n-грамів, який можна використовувати для вимірювання подібності між двома рядками або документами.

Загалом, k-грама - це будь-яка послідовність з k токенів. Тут ми використовуємо визначення з Leskovec, Rajaraman & Ullman (2014), «Видобуток масивних наборів даних», Cambridge University Press: кілька наступних пробілів замінюються одним пробілом, а k-грама — це послідовність із k символів.

Значення k за замовчуванням дорівнює 3. Хорошим емпіричним правилом є уявити, що є лише 20 символів, і оцінити кількість k-шинглів як 20^k. Для невеликих документів, таких як електронні листи, рекомендоване значення k =

5. Для великих документів, таких як дослідницькі статті, k = 9 вважається безпечним вибором.

Лістинг 6:

```
abstract class ShingleBased(val k: Int) {
    init {
        require(k > 0) { "k should be positive!" }
    }
    internal constructor() : this(DEFAULT_K)
    fun getProfile(string: String?): Map<String, Int> {
        val shingles = HashMap<String, Int>()
        val string_no_space: String = SPACE_REG.matcher(string).replaceAll(" ")
        for (i in 0 until string_no_space.length - k + 1) {
            val shingle = string_no_space.substring(i, i + k)
            val old = shingles[shingle]
            if (old != null) {
                shingles[shingle] = old + 1
            } else {
                shingles[shingle] = 1
            }
        }
        return Collections.unmodifiableMap(shingles)
    }
    companion object {
        private const val DEFAULT_K = 3
        private val SPACE_REG: Pattern = Pattern.compile("\\s+")
    }
}
```

Подібністю між двома рядками є косинус кута між цими двома представленнями векторів (Лістинг 7). Він обчислюється як V1. V2 / (|V1| * |V2|). Косинусна відстань обчислюється як 1 – косинус подібності.

Лістинг 7:

```
class Cosin(k: Int) : ShingleBased(), NormalizedStringDistance, NormalizedStringSimilarity {
    fun similarity(s1: String?, s2: String?): Double {
        if (s1 == null) {
            throw NullPointerException("s1 must not be null")
        }
        if (s2 == null) {
            throw NullPointerException("s2 must not be null")
        }
        if (s1 == s2) {
            return 1
        }
        if (s1.length < k || s2.length < k) {
            return 0
        }
        val profile1 = getProfile(s1)
        val profile2 = getProfile(s2)
        return (dotProduct(profile1, profile2)
            / (norm(profile1) * norm(profile2)))
    }
    fun distance(s1: String?, s2: String?): Double {
        return 1.0 - similarity(s1, s2)
    }
    fun similarity(
        profile1: Map<String, Int>,
        profile2: Map<String, Int>
    ): Double {
        return (dotProduct(profile1, profile2)
            / (norm(profile1) * norm(profile2)))
```

```
    }
    companion object {
        private fun norm(profile: Map<String, Int>): Double {
            var agg = 0.0
            for ((_, value): Map.Entry<String, Int> in profile) {
                agg += 1.0 * value * value
            }
            return Math.sqrt(agg)
        }
        private fun dotProduct(
            profile1: Map<String, Int>,
            profile2: Map<String, Int>
        ): Double {

            // Loop over the smallest map
            var small_profile = profile2
            var large_profile = profile1
            if (profile1.size < profile2.size) {
                small_profile = profile1
                large_profile = profile2
            }
            var agg = 0.0
            for ((key, value): Map.Entry<String, Int> in small_profile) {
                val i = large_profile[key] ?: continue
                agg += 1.0 * value * i
            }
            return agg
        }
    }
}
```

3.2 Розробка клієнтської частини

Після розробки й тестування серверної частини було розроблено два клієнтських додатка, а саме додаток для адміністрування і заповнення даних і додаток для користувачів системи. Мобільний додаток розділення функціоналу який доступний учня і який доступний співробітникам. Учні мають доступ до перегляду черги до якої вони відносяться, створення роботи, редагування роботи, перегляд запитань до роботи, створення нового запитання до роботи, перегляд часу початку на завершення захисту роботи, перегляду кінцевого балу роботи, перегляд відсотка плагіату роботи. Співробітники мають доступ до перегляду всіх створених черг захисту роботі, перегляд детальної інформації про чергу, виставлення балів за роботу, початок захисту роботи, завершення захисту роботи, перегляд запитань до роботи, створення нового запитання до роботи, перегляд відсотка плагіату роботи.

Додаток для адміністрування має окремий функціонал:

- Перегляд, створення, редагування, видалення факультету
- Перегляд, створення, редагування, видалення кафедри
- Перегляд, створення, редагування, видалення спеціальність
- Перегляд, створення, редагування, видалення групи учнів
- Перегляд, створення, редагування, видалення учнів
- Перегляд, створення, редагування, видалення співробітників
- Перегляд, створення, редагування, видалення черги
- Перегляд, створення, редагування, видалення типів посилань
- Перегляд, створення, редагування, видалення посад
- Перегляд, створення, редагування, видалення ролей
- Перегляд робіт
- Перегляд балів, які виставили співробітники від час захисту роботи
- Перегляд запитань
- Перегляд, створення, редагування, видалення ролей користувача

- Перегляд, створення, редагування, видалення комісії яка присутня під час захисту робіт
- Зміна посади співробітника
- Зміна групи учня
- Запуск перевірки певної роботи на плагіат
- Запуск перевірки всіх робіт на плагіат

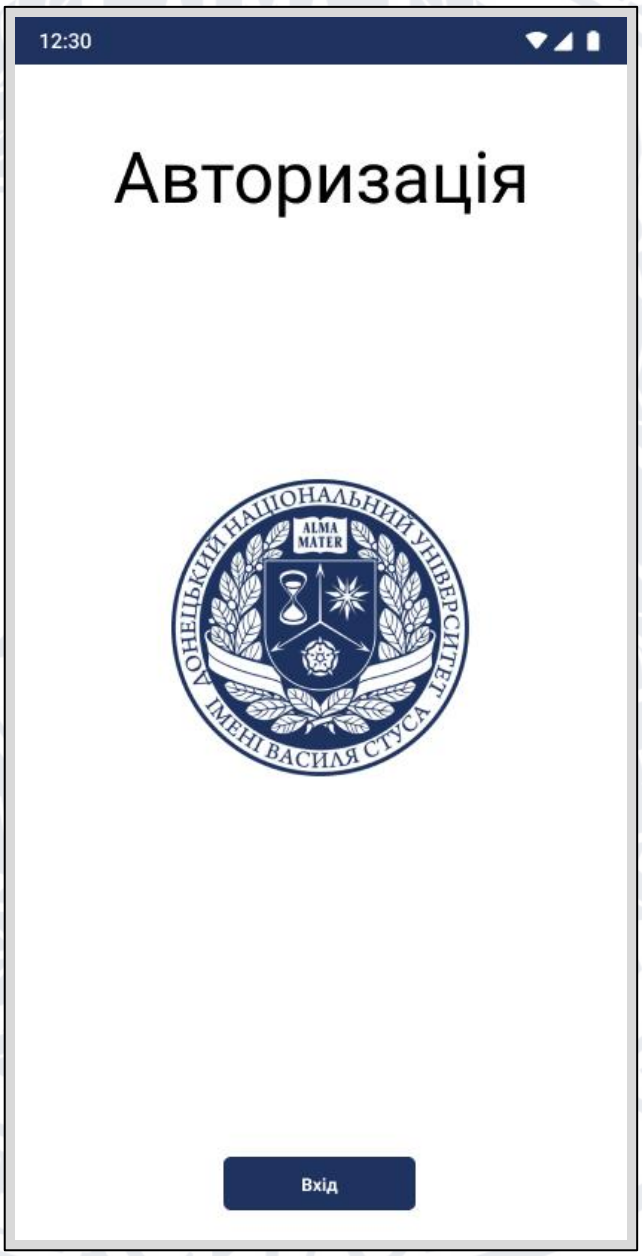

Рисунок 3.18 - Екран авторизації

Взаємодія з додатком починається з екрану авторизації, екран виглядає однаково для авторизації учнів та співробітників. При натисканні на кнопку

“Вхід” відбувається перехід на екран авторизації за номером телефону. Якщо користувач вже авторизований у системі то він одразу буде перенаправлений на головний екран з інформацією про чергу.

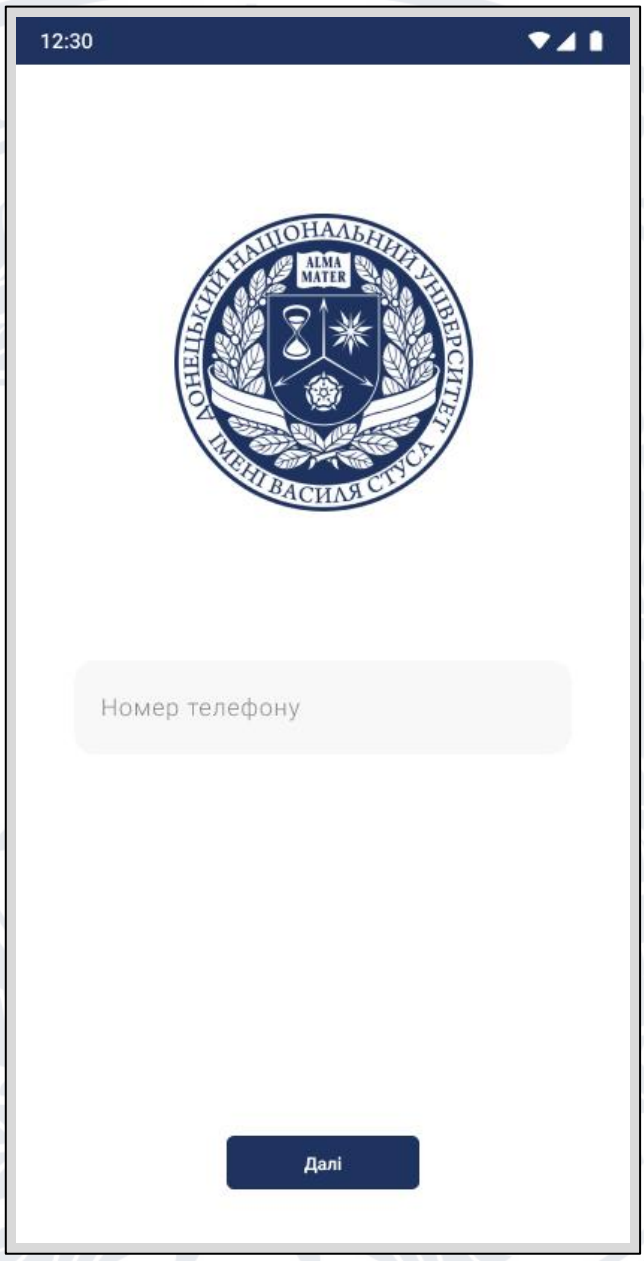

Рисунок 3.19 - Екран вводу номеру телефону

Після вводу коректного номеру телефону і натисканні кнопки “Далі” відбувається відправка СМС на введений номер телефону з кодом для авторизації, після чого відбувається перехід на екран вводу номеру телефону.

Рисунок 3.20 - Екран вводу коду з смс

Після вводу коду з смс відбувається авторизація користувача у системі і перехід на головний екран з чергою робіт.

Якщо авторизований користувач це співробітник закладу то відбувається перехід на екран з усі сформованими чергами захисту робіт. Якщо авторизований користувач це учень закладу то відбувається перехід на екран з детальною інформацією про чергу до якої відноситься учень закладу.

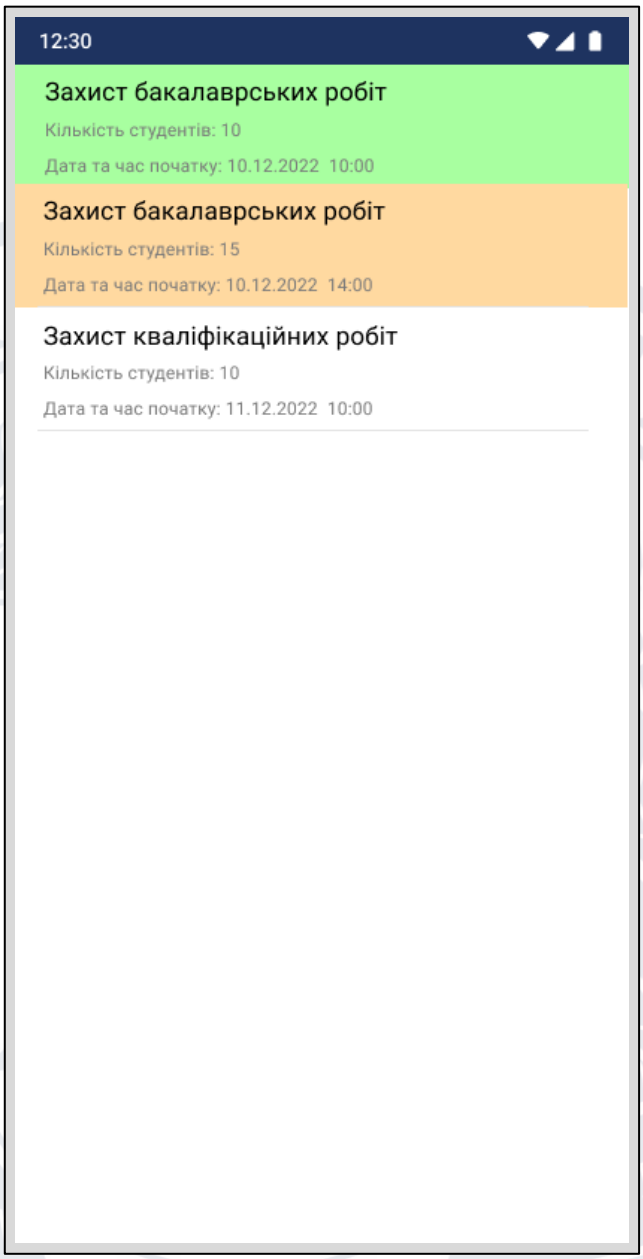

Рисунок 3.21 - Екран всіх черг на захист робіт

Екран з усіми чергами доступний лише співробітникам. Екран показує інформацію про чергу, а саме кількість учнів, дату на час початку захисту. Зеленим кольором показано чергу в якій вже завершився захист всіх робіт. Помаранчовим кольором показано чергу в якій зараз проходить захист робіт. Білим кольором показано чергу в якій ще не розпочався захист робіт. При натисканні на елемент списку відбувається перехід на екран з детальною інформацію про чергу захисту робіт.

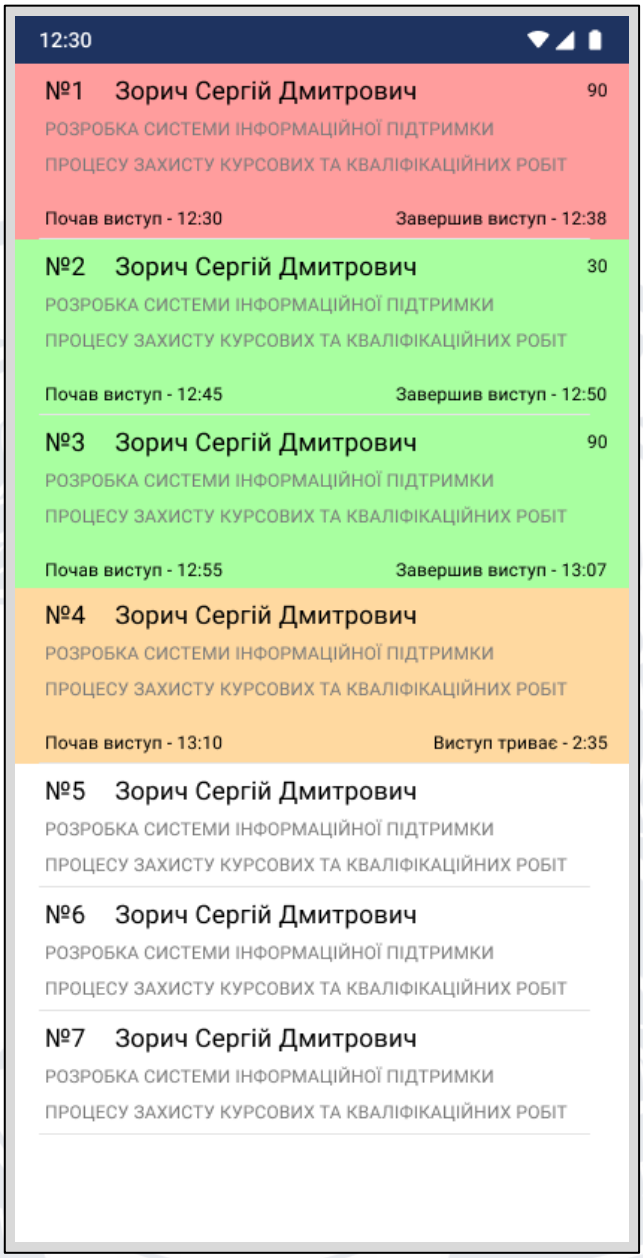

Рисунок 3.22 - Екран з детальною інформацію про чергу захисту робіт

Екран з детальною інформацією про чергу захисту робіт доступний як співробітникам так і учням закладу. Екран відображає послідовність захисту робіт, ФІО учня, тему роботи, час початку та закінчення захисту, кінцевий бал за захист роботи. Червоним відображаються роботи які не були захищені, або роботи в яких кінцевий бал меньше 60. Зеленим відображаються роботи які були успішно захищені і в яких бал більше 60. Помаранчевим відображається робота яку зараз захищають. Одночасно не можливо захищати більше однієї роботи в однії черзі. Білим кольором відображається роботи в яких ще не почався захист.

При натисканні на елемент списку відбувається перехід на екран детальної інформації про роботу.

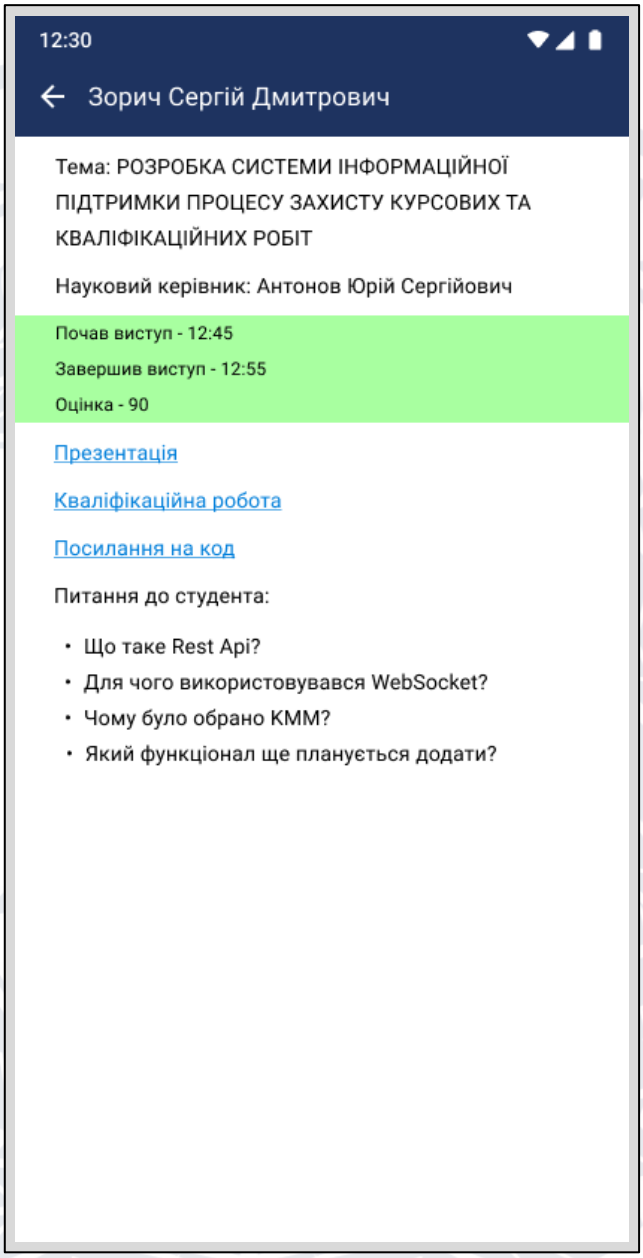

Рисунок 3.23 - Екран з детальною інформацією про роботу

На екрані відображається інформацію про вибрану роботу:

- Тема роботи
- Науковий керівник
- Дата та час початку захисту роботи
- Дата та час завершення захисту роботи
- Кінцевий бал роботи
- Посилання на роботу

- Питання які були задані під час виступу
- Відображається результат захисту роботи:
    - Зеленим кольором відображається робота з кінцевим балом більше 60
    - Помаранчевим кольором відображається робота під час захисту
    - Червоним кольором відображається робота яка не була захищена або з кінцевим балом меньше 60
- Для членів комісії доступний функціонал виставлення балу за роботу
- Процент плагіату у роботі
- Для учні і членів комісії доступний функціонал створення питань під час захисту роботи

Висновок до розділу 3

В цьому розділі було розглянуто розробку інформаційної системи захисту бакалаврських та кваліфікаційних робіт. Було спроектовано та розроблену базу даних для зберігання необхідної інформації. Також було розроблено серверну частину за допомогую Ktor, яка взаємодіє з базою даних. Розроблено мобільний додаток для користувачів і мобільний додаток для адміністрування, які використовують відповідні Rest API серверної частини. Наведено схему бази даних й відповідне Rest API. Також наведено частину екранів мобільного додатку для користувачів.

ВИСНОВКИ

Сьогодні ринок мобільних додатків є найпопулярнішою тенденцією IT-ринку. Сучасні мобільні додатки маю кілька вирішальних переваг: це надзвичайна швидкість та зручність під час користування.

Метою кваліфікаційної роботи було розробити інформаційну систему підтримки захисту кваліфікаційних та бакалаврських робіт.

Для того, щоб розпочати розробку системи було проаналізовано необхідність та актуальність розробки.

Перед розробкою було досліджено, вивчено і обрано багато інструментів для створення сучасного мобільного додатку та серверної частини. Було вирішено розробляти мобільний додаток під Android. Для розробки серверної частини було обрано Ktor. В якості мови програмування було обрано мову Kotlin. Kotlin - мова програмування, яка створена компанією JetBrains та позиціонується, як сучасна альтернатива Java.

У результаті виконання кваліфікаційної роботи детально вивчено технологію створення серверної частини та мобільних додатків, тобто методи та засоби програмної реалізації продукту. У результаті було обрано створення програми для платформи Android та використання Ktor для реалізації серверної частини. Для цього було досліджено роботу інтегрованих середовищ розробки Android Studio та IntelliJ IDEA. Також використання мови програмування Kotlin покращило вже набуті навички та дало нові знання. Також було розглянуто подібні системи та виявлено їх сильні та слабкі сторони.

Android Studio - одне з провідних середовищ розробки, вона дивовижна тим, що містить всі функції та можливості, необхідні для зручного програмування Android-додатків, а також відповідає всім останнім вимогам та тенденціям розробки мобільних додатків.

Програма Figma була обрана для створення макету та прототипу майбутнього мобільного додатка. Розробляючи UX/UI дизайн було взято до уваги та враховано ефект «естетика-юзабіліті». З цієї причини кольори та

графічні компоненти підібрані таким чином, щоб привертати увагу, а дизайн був доступний та інтуїтивно зрозумілий користувачу.

Архітектура MVI (Model – View – Intent) була використана у мобільному додатку для користувачів системи.

Архітектура MVC (Model – View – Controller) була використана у серверній частині системи.

Було розгляду основні методи перевірки робіт на плагіат і обрано метод шинглів для реалізації перевірки на плагіат у системи.

Також було спроектовано та розроблену базу даних.

Тестування даного програмного продукту показало, що система виконує покладені на неї функції, стабільно працює та загалом придатний для використання.

Отже, за час виконання поставленої задачі, була пророблена велика та об’ємна робота по створенню сучасної системи інформаційної підтримки захисту бакалаврських та кваліфікаційних робіт у вигляді двох мобільних додатків, написаних під Android. Розроблено серверну частину для комунікації з клієнтською частиною та базою даних за допомогою Ktor. Реалізовано усі заплановані функції системи.

СПИСОК ЛІТЕРАТУРИ

1. Мобільні додатки у період пандемії COVID - 19. URL: https://www.appannie.com/en/insights/market-data/mobile-app-usage-surged-40-during-covid-19-pandemic/
2. 8 причин, щоб полюбити Figma. URL: https://artjoker.ua/ru/blog/8-prichin-chtoby-polyubit-figma-tak-zhe-silno-kak-my/
3. Android Studio. URL: https://developer.android.com/studio
4. Guide to mobile App design. URL: https://buildfire.com/mobile-app-design/
5. Guide to components in Figma. URL: https://help.figma.com/hc/en-us/articles/360038662654-Guide-to-Components-in-Figma
6. Какая операционная система лучше для смартфона? URL: https://setphone.ru/stati/kakaya-operacionnaya-sistema-luchshe-dlyasmartfona/
7. Все приложения в Google Play безопасны: правда или миф? URL: https://www.kaspersky.ru/blog/google-play-malware/23629/
8. Что всё-таки лучше: Android или iOS? URL: https://lifehacker.ru/chto-luchshe-android-ili-ios/
9. Android Studio: среда разработки мобильных приложений. URL: https://arduinoplus.ru/android-studio/#_Android_Studio
10. Android for Developers. URL: https://developer.android.com/guide/
11. Что означает уровень API? URL: http://www.ohandroid.com/api.html
12. Gradle. URL: http://developer.alexanderklimov.ru/android/theory/gradle.php

13. Develop Android apps with Kotlin. URL: https://developer.android.com/kotlin

14. Ktor. URL: https://ktor.io/

15. Creatings HTTP APIs. URL: https://ktor.io/docs/creating-http-apis.html

16. Modern Android Architecture with MVI design pattern. URL: https://amsterdamstandard.com/en/post/modern-android-architecture-with-mvi-design-pattern

17. MVI Architecture with Android. URL: https://medium.com/swlh/mvi-architecture-with-android-fcde123e3c4a

18. MVI Architecture for Android Tutorial: Getting Started. URL: https://www.kodeco.com/817602-mvi-architecture-for-android-tutorial-getting-started

19. Model-VIew-Controller. URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

20. Everything you need to know about MVC. URL: https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1

21. The Model View Controller Pattern – MVC Architecture and Frameworks Explained. URL: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

22. Android version history. URL: https://en.wikipedia.org/wiki/Android_version_history

23. Чиркин Е.С. Системы автоматизированной проверки на неправомерные заимствования // Вестник ТГУ. – 2013. − No12. − С. 164-171.

24. Петренко В.С. Поняття плагіату. URL: http://www.clj.nuoua.od.ua/archive/14/29.pdf
25. Закон України “Про авторське право і суміжні права”. URL: https://zakon.rada.gov.ua/laws/show/3792-12
26. Зеленков Ю.Г, Сегалович И.В. Сравнительный анализ методов определения нечетких дубликатов для Web-документов. URL: http://rcdl2007.pereslavl.ru/papers/paper_65_v1.pdf
27. Білощицький А.О., Діхтяренко О.В. Ефективність методів пошуку збігів у текстах // Управління розвитком складних систем. − 2013. −No 14. − С. 144-147.
28. Методы обхода антиплагиата. URL: http://antiplag.ru/blog/neskolko-effektivnyx-metodov-obxoda-proverki-na-antiplagiat
29. Simon Brown. Software Architecture for Developers: Technical leadership and the balance with agility. - 2022
30. Neil Smyth. Android Studio Bumble Bee Essentials - Kotlin Edition. - 2022
31. Pierre-Olivier Laurence, Amanda Hinchman-Dominguez, Mike Dunn. Programming Android with Kotlin: Achieving Structured Concurrency with Coroutines.
32. Brett McLaughlin. Programming Kotlin Applications: Building Mobile and Server-Side Applications with Kotlin. - 2021
33. Alexander Aronowitz. Programming Kotlin: Enhance your skills for Android development using Kotlin. - 2022
34. Venkat Subramaniam. Programming Kotlin: Create Elegant, Expressive, and Performant JVM and Android Applications. - 2019
35. Prateek Prasad. App Design Apprentice (2nd Edition). - 2022

36. Bryan Sills, Brian Gardner, Kristin Marsicano and Chris Stewart. Android Programming: The Big Nerd Ranch Guide, 5th Edition - 2022

37. Chet Haase. Androids: The Team That Built the Android Operating System. - 2022

38. Melvin K. Briscoe. Figma 101: Complete beginner's guide to UI/UX app and web design. Create interactive components with real projects. - 2022

39. Dario Calonaci. Designing User Interfaces: Exploring User Interfaces, UI Elements, Design Prototypes and the Figma UI Design Tool. - 2021

40. Luke Chambers. UX: Essential Tools. - 2019

41. Окунев А. Руководство по Figma - 2019

42. Mike Amundsen. Restful Web API Patterns and Practices Cookbook. - 2022

43. James Gough, Daniel Bryant, Matthew Auburn. Mastering API Architecture. - 2022

44. MVP (Model View Presenter) Architecture Pattern in Android with Example. URL: https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/

45. Difference Between MVC and MVP Patterns. URL: https://www.baeldung.com/mvc-vs-mvp-pattern

46. MVP for Android: how to organize the presentation layer. URL: https://antonioleiva.com/mvp-android/

47. Guide to app architecture. URL: https://developer.android.com/topic/architecture

48. Better Android Apps Using MVVM With Clean Architecture. URL: https://www.toptal.com/android/android-apps-mvvm-with-clean-architecture

49. Kotlin for server side. URL: https://kotlinlang.org/docs/server-overview.html

50. Is MVC the right architectural parent for server side web frameworks? URL: https://www.quora.com/Is-MVC-the-right-architectural-pattern-for-server-side-web-frameworks

51. MVC Architecture in Java. URL: https://www.javatpoint.com/mvc-architecture-in-java

52. REST API Architecture Constraints. URL: https://www.geeksforgeeks.org/rest-api-architectural-constraints/

53. Building a REST API with Ktor. URL: https://medium.com/@billwixted/building-a-rest-api-with-ktor-4c322d31eb31

54. Ktor REST Apis - Part 1 (Project Set up). URL: https://proandroiddev.com/build-rest-apis-using-ktor-framework-i-dbbf36b332bb

55. Ktor REST Apis - Part 2 (Create Routes). URL: https://proandroiddev.com/build-rest-apis-using-ktor-framework-ii-47948e89f1d6

56. Ktor REST Apis - Part 3 (Testing Routes). URL: https://proandroiddev.com/build-rest-apis-using-ktor-framework-iii-87e579a7258e

57. Gerardus Blokdyk. Database Normalization A Complete Guide. - 2021

58. Christoper J. Date. Database Design and Relational Theory: Normal Forms and All That Jazz. - 2012

59. Toby J. Teorey, Sam S. Lightstone, Tom Nadeau, H.V. Jagadish. Database Modeling and Design. - 2011

60. Philip J. Pratt, Mary Z. Last. Concepts of Database Management. - 2014

61. David M. Kroenke, David J. Auer. Database Concepts. - 2008

62. Richard T. Snodgrass. Developing Time-oriented Database Applications in SQL. - 2000

63. S. K. Singh. Database Systems: Concepts, Design and Applications. - 2009

Зорич Сергій Дмитрович
Прізвище, ім'я по батькові

Інформаційних і прикладних технологій
Факультет

122 «Комп'ютерні науки»
Шифр і назва спеціальності

«Комп'ютерні технології обробки даних (Data Science)»
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему: «Розробка системи інформаційної підтримки процесу захисту курсових та кваліфікаційних робіт» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

(дата) (підпис здобувача)