КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІД ОС ANDROID ДЛЯ СТВОРЕННЯ ВІРТУАЛЬНИХ ЕКСПОЗИЦІЙ

Виконала студентка 4-го курсу
Тетяна ПАНЧЕНКО _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Оксана ШКІЛЬНЯК

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № 11
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 53 сторінки, 15 ілюстрацій, 5 таблиць, 12 джерел посилань.

ВЕБЗАСТОСУНОК, ВЕБКЛІЄНТ, ВЕБСЕРВЕР, ANDROID-ДОДАТОК, БАЗА ДАНИХ, ВІДДАЛЕНИЙ СЕРВЕР.

Об’єктом роботи є інформаційна система для архівації, публікації та перегляду культурних пам’яток та артефактів.

Предметом роботи є додаток для ОС Android та супроводжуючий вебсервіс.

Метою роботи є розробка та розгортання вебсервісу, а також створення мобільного додатку, що з ним взаємодіє.

Інструменти розробки: мова програмування Java, мова програмування JavaScript, мова програмування Kotlin, фреймворк Spring, фреймворк Hilt, хмарна платформа AWS, система керування реляційною базою даних MySQL, бібліотека Simple Java Mail, набір бібліотек Android Architecture Components, набір засобів розробки Jetpack Compose.

Результат роботи: розроблено та розгорнуто вебсервіс для архівації та публікації даних, а також мобільний додаток для ОС Android для перегляду публічних записів.

Результат роботи може використовуватись як інформаційна система для архівації та популяризації даних про культурні пам’ятки та артефакти. Вебсервіс був розроблений з оглядом на можливість в потребі розширення, аби установи, що публікують в ньому дані, могли використовувати його як основу для створення власних застосунків.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

JSON – JavaScript Object Notation (запис об'єктів JavaScript)

XML – Extensible Markup Language (розширювана мова розмітки)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

БД – База даних

СУБД – Система управління базами даних

ОС – Операційна система

UI – User interface (інтерфейс користувача)

API – Application programming interface (прикладний програмний інтерфейс)

ПЗ – Програмне забезпечення

3D – Тривимірний

SQL – Structured Query Language (мова структурованих запитів)

SSO – Single Sign-On (технологія єдиного входу)

ВСТУП

Оцінка сучасного стану об’єкта розробки. Культурна спадщина народу — це свідчення людського духу та нашої здатності творити, впроваджувати інновації та надихати. В Україні однією з актуальних проблем в цій сфері є недостатнє фінансування: багато установ та організацій не мають необхідних ресурсів для виконання вимог по догляду за об’єктами в повній мірі. Окрім того, не слід забувати про вплив політичної нестабільності: конфлікти та війни безпосередньо завдають шкоди об’єктам культурної спадщини та артефактам, а також перешкоджають зусиллям щодо їх збереження. Варто також зазначити погану доступність багатьох об’єктів культурної спадщини та музеїв для певних осіб або громад через відстань, вартість або фізичні обмеження. Одним з факторів, що сповільнюють розвиток даної сфери є відсутність єдиного зручного достовірного джерела. Це обмежує можливість широкого загального доступу до інформації про культурну спадщину, адже її популяризація здатна не лише підвищити відвідуваність та доходи відповідних установ, а й вплинути на їх підтримку з боку держави.

Актуальність роботи та підстави для її виконання. В Україні станом на 2012 рік під охороною держави перебувало 140 тис. культурних об’єктів. Стан 50-70% з них класифіковано, як незадовільний: щороку понад 200 пам’яток національного значення потребують проведення невідкладних протиаварійних та консерваційних робіт [1]. Вторгнення Російської Федерації в Україну поставило під загрозу значну кількість культурної спадщини країни. Під час бойових дій було серйозно пошкоджено або повністю зруйновано багато церков, музеїв, архітектурних та скульптурних споруд, бібліотек, інших історичних і культурних пам’яток. Наразі на сайті Міністерства культури та інформаційної політики України [2] зафіксовано

553 пошкоджених та зруйнованих об’єктів культурної спадщини і культурних установ України.

Мета й завдання роботи. Метою роботи є створення мобільного додатку для найбільш поширеної ОС в Україні [3], а також розробка та розгортання супроводжуючого вебсервісу. Для досягнення цієї мети було поставлено наступні завдання:

1. Визначити ролі та відповідні можливості користувачів системи.
2. Обрати технології для розробки вебсервісу.
3. Обрати технології для розробки мобільного додатку.
4. Обрати архітектурний підхід для розробки мобільного додатку.
5. Спроєктувати структури даних, які зберігатимуться в БД.
6. Розробити простий та інтуїтивний дизайн програмних продуктів.
7. Забезпечити безпеку та обмеження в доступі відповідно ролі користувача в системі.
8. Забезпечити можливість адміністрування.
9. Розробити вебклієнт та вебсервер з використанням обраних технологій.
10. Розробити мобільний додаток з використанням обраних технологій.

Об’єкт, методи й засоби розробки. Об’єктом роботи є інформаційна система для архівації, публікації та перегляду культурних пам’яток та артефактів. Система складається з двох окремих проєктів: вебсервісу та мобільного додатку. Такий поділ був виконаний з метою підвищення рівня безпеки та забезпечення кращої розширюваності системи.

При створенні серверної частини вебсервісу було використано мову програмування Java, СУБД MySQL, фреймворк Spring, засіб для автоматизації збірки Maven, препроцесор Lombok та ряд допоміжних бібліотек для збірки проєкту, надсилання листів електронною поштою, генерації JWT-токенів і парсингу JSON-рядків.

При створенні клієнтської частини вебсервісу було використано механізм шаблонів Thymeleaf, фреймворк Bootstrap та JavaScript бібліотеку Three.js.

При створенні Android-додатку було використано мову програмування Kotlin, набір бібліотек Android Architecture Components, засіб для автоматизації збірки Gradle, фреймворк Hilt, набір засобів розробки Jetpack Compose.

Можливі сфери застосування. Розроблений вебсервіс може використовуватись установами, що займаються збереженням пам’яток для організації, архівації та поширення інформації про об’єктів в своїй юрисдикції. Розроблений застосунок використовує публічну інформацію, отриману від вебсервісу і може використовуватись усіма, хто бажає більше дізнатись про культурну спадщину своєї країни.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 British Museum Audio Buddy

Android-додаток «British Museum Audio Buddy» був створений для Британського музею [4]. Його задача полягає в тому, щоб полегшити навігацію будівлею.

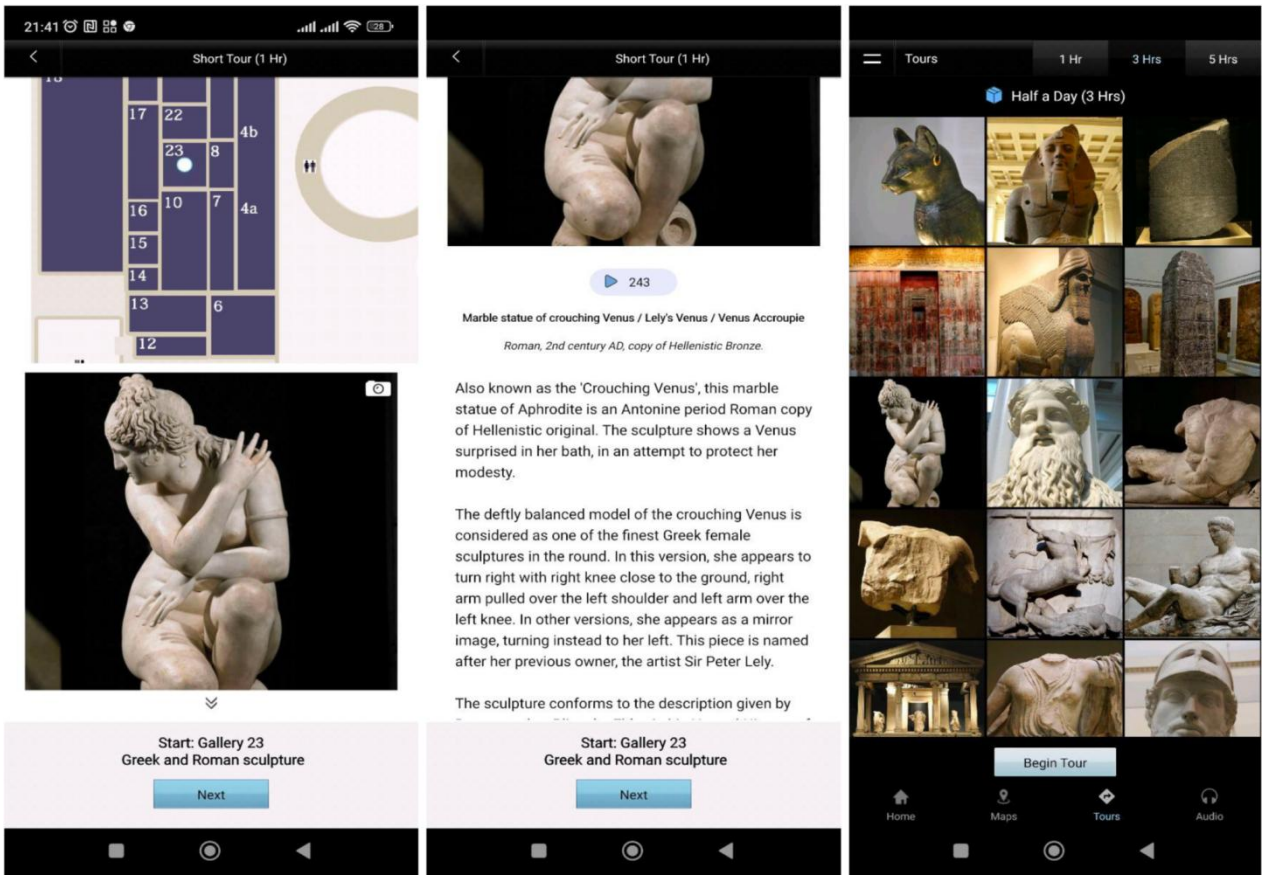

Рисунок 1 – Інтерфейс додатку «British Museum Audio Buddy»

Повна версія пропонує 2 основні функції:

1. Взаємодія з інтерактивною мапою для навігації між кімнатами та перегляду експонатів в конкретній кімнаті.
2. Перегляд трьох типів турів: на 1 годину, на 3 години, на 5 годин. Кожен тур представлений галереєю об’єктів.

Експонати представлені назвою, місцем походження, датою, описом та хоча б одним фото. Фотографію можна відкрити на повний екран та

приблизити. На екрані з повною інформацією про експонат також доступне прослуховування аудіодоріжки, яка озвучує опис.

В додатку є 2 недоліки:

1. Навігація в додатку непослідовна: якщо переглядати експонати через кімнату на інтерактивній мапі, то між експонатами можна переміщатись за допомогою свайпу – як до попереднього, так і до наступного, але якщо їх переглядати через тури, то перейти можна тільки до наступного експонату при натисканні кнопки “Next”.
2. Безкоштовна версія доволі обмежена: для перегляду доступно лише 2 кімнати, а серед турів опції “3 години” та “5 годин” є недоступними. Це одна з причин здебільшого негативних відгуків.

1.2 Asian Art Museum SF

Android-додаток «Asian Art Museum SF» був створений для музею мистецтва Азії в Сан-Франциско [5]. В ньому також міститься інтерактивна мапа з аналогічним функціоналом.

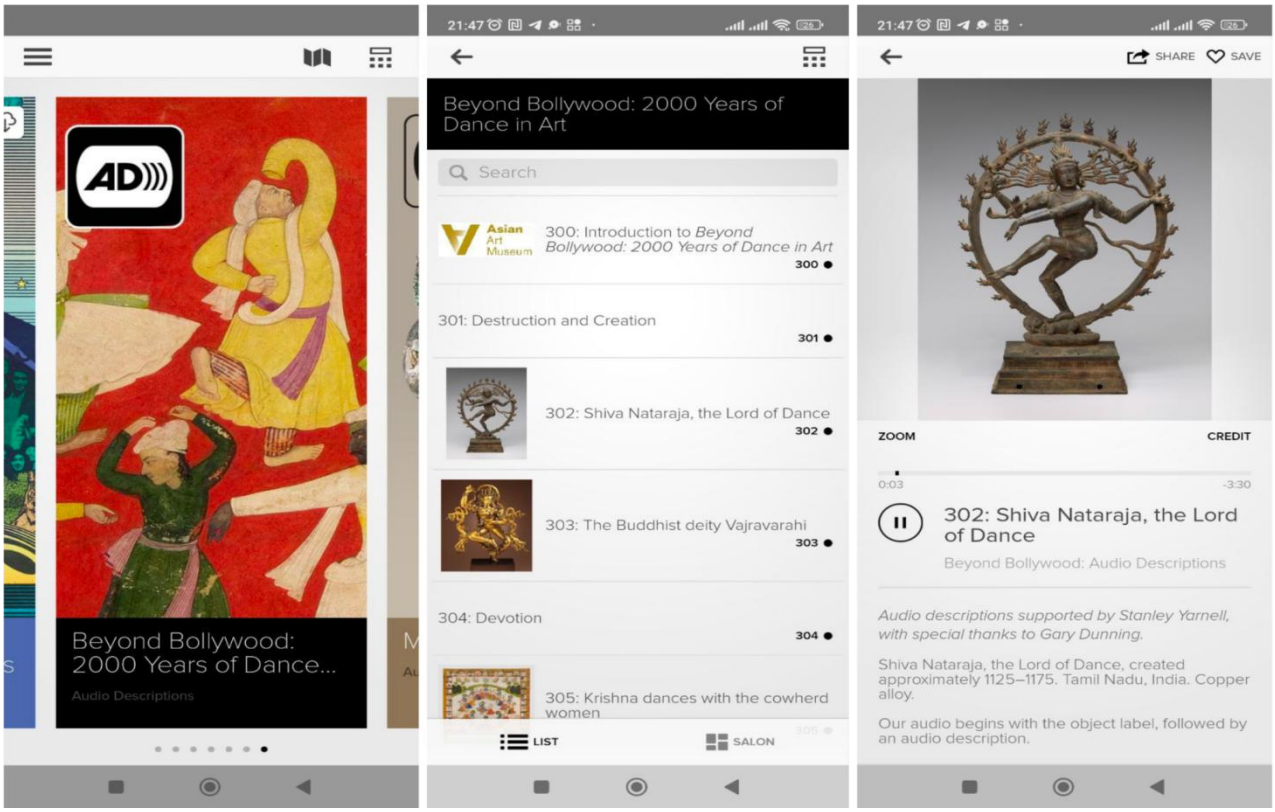

Рисунок 2 – Інтерфейс додатку «Asian Art Museum SF»

На відміну від попереднього додатку, замість турів «Asian Art Museum SF» пропонує колекції, розділені на підгрупи експонатів.

Експонати представлені назвою, описом та фото. Фотографію можна відкрити на повний екран та приблизити. На екрані з повною інформацією про експонат також доступне прослуховування аудіодоріжки, яка озвучує опис.

Переваги:

1. В додатку існує доповнення до однієї з колекцій, що дозволяє, замість прослуховування опису, переглянути відео, де та ж інформація передається мовою жестів.

Недоліки:

1. Кожну колекцію необхідно завантажувати додатково, їх розмір сягає від 43 МБ до 221 МБ. Таким чином, загальний розмір додатку, якщо завантажена хоча б одна колекція, є доволі великим.
2. Більшість колекцій необхідно попередньо завантажувати перед переглядом, але доповнення з відео, де інформація подається мовою жестів, не завантажується, а програється, як потік даних. Це непослідовність інтерфейсу, тому незрозуміло, чи додаток орієнтований на офлайн перегляд, чи для його використання варто бути в мережі.

1.3 Europeana

Web-портал «Europeana» – це цифрова бібліотека, музей та архів, яка надає доступ до мільйонів цифрових об'єктів, пов'язаних з культурною спадщиною Європи [6]. Вона створена з метою збереження та доступу до цінних культурних ресурсів Європи та підтримки їх використання у різних сферах життя.

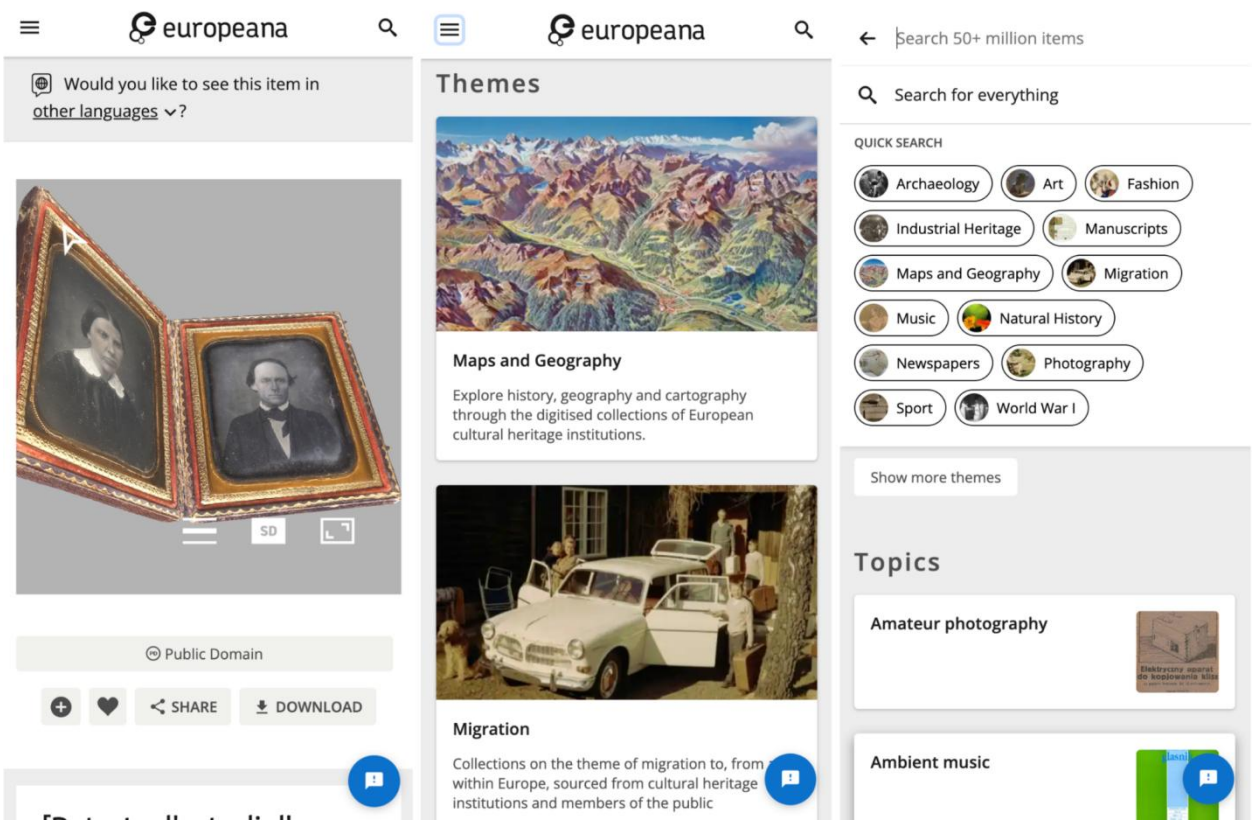

Рисунок 3 – Інтерфейс Web-порталу «Europeana»

Переваги:

1. Europeana надає доступ до колекцій з більш ніж 4000 культурних установ з усіх куточків Європи, включаючи музеї, бібліотеки, архіви та галереї. Сьогодні портал містить більше 50 мільйонів об’єктів.
2. Цифрові об'єкти, які можна знайти на порталі, представлені різноманітними типами медіа: фото, текст, відео, аудіо, 3D моделі.
3. Портал дозволяє здійснювати пошук об'єктів за різними параметрами, наприклад: тема, тип медіа, мова, колір, авторське право, агрегатор тощо.

Цей сервіс не зберігає на власних серверах медіафайли, надані інституціями або агрегаторами. Для публікації культурного ресурсу необхідно надати посилання на його діджиталізовану версію. Вочевидь, це зроблено, в першу чергу, для дотримання авторських прав.

Такий підхід має один недолік: ресурси не знаходяться під контролем сервісу, тому не знадобилось багато часу, аби знайти непослідовність в способі демонстрації певного типу медіа, неробочі посилання та несправні файли. Найчастіше це проявляється у ресурсах, що мають 3D модель серед медіафайлів: іноді доступно лише завантаження файлу(що потребує спеціального ПЗ для його відкривання), іноді надається посилання, що перенаправляє на зовнішній переглядач агрегатора, іноді використовується вбудований переглядач.

1.4 Порівняння наявних рішень

Розглянуті застосунків для ОС Android були створені для конкретного музею і насамперед орієнтовані на полегшення навігації в закладі. Така концепція не зовсім підходить під мету даної роботи, але ці застосунки продемонстрували доволі зручний для користувача підхід щодо організації експонатів: експонати варто розподіляти в колекції.

Натомість, Europeana дозволяє дослідити експонати тисяч музеїв та інституцій Європи, що є ближчим до мети даної роботи. Цей портал надає зручні інструменти пошуку для звичайного користувача і засоби для публікації експонатів установам.

Дослідження наявних рішень допомогло сформувати певні вимоги розроблюваної до інформаційної системи. Застосунок та вебсервіс повинні мати послідовний інтерфейс і забезпечувати інструменти для ефективного пошуку експонатів. Крім того, з метою оптимальної організації об'єктів на мобільних екранах, вони мають бути розміщені в колекціях.

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Spring-фреймворк

Spring Framework – це популярний фреймворк на основі Java для створення корпоративних додатків. Він надає широкий набір інструментів для створення надійних, масштабованих і високопродуктивних сервісів. Його модульна архітектура, механізм впровадження залежностей, підтримка AOP та інтеграційні можливості роблять його популярним вибором для розробників.

Серед засобів розробки, запропонованих Spring, в проєкті використовуються Spring Boot, Spring Security та Spring JPA.

Spring Boot надає три основні можливості: спрощене управління залежностями, спрощене розгортання та автоконфігурація [7].

Управління залежностями може стати справжнім викликом, особливо, коли виникають конфлікти. Spring Boot розв'язує цю проблему за допомогою механізму стартерів (starters). Стартери є готовими наборами залежностей, які надають певні функціональні можливості у стандартизованому форматі. Додавання одного стартера, наприклад "spring-boot-starter-web", автоматично включає всі необхідні залежності як єдиний компонент програми. Всі залежності у стартері синхронізовані за версіями та пройшли спільні тестування, що забезпечує сумісність та мінімізує можливі конфлікти.

Java-програми та їхні залежності раніше упаковувалися як окремі JAR-файли в одному архіві. Spring Boot реалізував новий підхід: розміщення класів і ресурсів залежностей безпосередньо в JAR-файлі програми (створення uber-JAR або fat JAR.) Це дозволяє зберегти оригінальну структуру, уникнути конфліктів версій та юридичних проблем, пов'язаних з перепакуванням програмного забезпечення з різними ліцензіями. Використання єдиного JAR-файлу спрощує процес розгортання, що особливо актуально в умовах зростаючої популярності контейнеризації. Завдяки цій

функції, не потрібно вручну збирати та перевіряти, чи всі залежності були розгорнуті, оскільки плагін Spring Boot автоматично упаковує їх у вихідний JAR-файл.

Завдяки автоконфігурації, Spring Boot може самостійно виявляти, налаштовувати та інтегрувати різноманітні компоненти, такі як бази даних, веб-сервери, сервіси безпеки і багато іншого. При запуску додатка Spring Boot сканує класи та конфігураційні файли, аналізує їх та автоматично встановлює необхідні залежності та параметри. Це дозволяє розробникам уникнути великої кількості рутинних налаштувань і сконцентруватися на написанні бізнес-логіки.

При створення вебдодатків Spring Boot допомагає реалізувати ієрархічну архітектуру, в якій кожен рівень взаємодіє з рівнем, що знаходиться безпосередньо під ним. Для реалізації цієї архітектури використовуються наступні компоненти:

- Контролери (Controllers) – це класи, що представляють рівень презентації (Presentation Layer). Вони отримують запити від клієнтів та повертають відповіді. Контроллери позначаються анотаціями @RestController або @Controller.
- Сервіси (Services) – це класи, що визначають бізнес-логіку додатка (Business Layer). Вони використовуються контролерами для обробки запитів. Сервіси позначаються анотацією @Service.
- Репозиторії (Repositories) – це класи, які забезпечують доступ до бази даних та перетворюють бізнес-моделі на моделі, що розуміє СУБД і навпаки (Persistence Layer/DAO Layer). Вони використовуються сервісами для роботи з даними. Репозиторії позначаються анотацією @Repository.

Spring Security є набором сервлет-фільтрів, що дозволяють додати автентифікацію та авторизацію до вебдодатку. Він автоматично створює

сторінки для входу та виходу, а також надійно захищає від розповсюджених експлойтів, таких як CSRF.

Для роботи з Spring Security необхідно перш за все оволодіти трьома ключовими поняттями, а саме: Аутентифікація, Авторизація та Фільтри сервлетів.

Аутентифікація передбачає перевірку імені користувача та пароля, щоб упевнитися, що користувач є тим, за кого він себе видає.

Авторизація передбачає перевірку дозволів автентифікованого користувача, щоб дозволити або заборонити йому доступ до певних частин програми. Це особливо важливо в більш складних програмах, які мають концепцію дозволів або ролей.

Фільтри сервлетів дозволяють здійснити певні дії над запитом перед тим, як він буде надісланий до додатку, або з відповіддю, що повертається з додатку. Фільтри сервлетів можуть бути використані для забезпечення автентифікації та авторизації, здійснення перевірки правильності запитів, а також для обробки винятків та помилок.

Spring JPA – це модуль у фреймворку Spring, який надає спрощений доступ до бази даних. JPA - це стандартна специфікація для роботи з об'єктно-реляційним відображенням (ORM) в Java. Spring JPA забезпечує високорівневий API для роботи з БД, що дозволяє зосередитись на бізнес-логіці, а не на деталях доступу до даних. Цей модуль автоматично генерує SQL-запити на основі визначених моделей даних, що дозволяє уникнути помилок.

2.2 Стандарт JWT

JWT (JSON Web Token) – це популярний стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об'єкту. Цей стандарт часто використовується для надійної та безпечної аутентифікації в клієнт-серверних програмах. Процес починається зі створення токену сервером,

який після цього підписується секретним криптографічним ключем та передається клієнту. Після отримання токена, клієнт може використати його для підтвердження своєї особи.

Основною перевагою використання JWT-токенів є відсутність потреби зберігання стану (Statelessness): JWT-токени є самодостатніми та містять усю необхідну інформацію, тому їх можна легко передавати без необхідності зберігати будь-який стан на стороні сервера. Серверу потрібно лише зіставити токен та криптографічний підпис із інформацією у файлі, тобто він може виконати набагато менше роботи з пошуку інформації про постачальника ідентифікаційної інформації (IdP). Це зменшує витрати на сервер, а також робить автентифікацію більш масштабованою та сумісною з SSO.

JWT токен складається з заголовка, корисного навантаження та підпису. Заголовок (Header) зазвичай складається з двох частин: тип токену (“jwt”) та алгоритм підпису, що використовується. Корисне навантаження (Payload) містить твердження (claims) про, в випадку аутентифікації, користувача. Ці твердження поділяються на типи:

- Зарезервовані: визначені в стандарті, містять інформацію про джерело токену, дату закінчення терміну дійсності, тощо
- Публічні: додаткові поля, які повинні відповідати IANA JSON Web Token Registry задля уникнення колізій, можуть містити інформацію про роль користувача, його ім’я чи прізвище, тощо
- Приватні: нестандартизована додаткова інформація

Для формування підпису, необхідно закодувати заголовок та корисне навантаження і підписати їх за допомогою алгоритму підпису із заголовка. Підпис використовується для підтвердження того, що емітент JWT є тим, за кого він себе заявляє, і щоб переконатися, що повідомлення не було пошкоджено чи змінено в процесі передачі.

2.3 Simple Java Mail

Для здійснення відправки електронних листів використовується бібліотека Simple Java Mail, яка є однією з найбільш зручних Java-бібліотек для роботи з протоколом SMTP. Simple Java Mail є обгорткою над потужною бібліотекою JavaMail API, яка дозволяє значно скоротити кількість коду, необхідного для простої відправки листа.

Перевагами цієї бібліотеки є надійність і невеликий розмір. Ця бібліотека підтримує відправку HTML, зображень і вкладень, а також можливість відправляти листи одночасно кільком отримувачам.

З недоліків можна виділити меншу підтримку спільноти порівняно з JavaMail API.

2.4 СУБД MySQL

MySQL — це система керування реляційною базою даних з відкритим кодом. Її легко встановити та налаштувати, а документація вичерпна та зручна для користувача. З точки зору продуктивності, MySQL відома своєю швидкістю та ефективністю. Ця СУБД призначена для обробки великого обсягу транзакцій і може масштабуватися відповідно до потреб програми.

При виборі СУБД розглядались й інші рішення: Oracle, Microsoft SQL Server і PostgreSQL.

Oracle і Microsoft SQL Server — комерційні системи керування базами даних, відомі своєю масштабованістю та надійністю. Вони використовується в основному для великих корпоративних систем і призначені для обробки складних транзакцій. Ці варіанти було відкинуто, адже вони не є безкоштовними та більш складні в налаштуванні.

PostgreSQL — це система керування об’єктно-реляційними базами даних з відкритим кодом, багато в чому схожа на MySQL. PostgreSQL підтримує розширені типи даних та дозволяє успадковувати таблиці, що призводить до створення складніших систем баз даних. Це робить її відмінним вибором для проєкту, в якому доводиться виконувати складні

процеси читання-запису, використовуючи при цьому інформацію, яка потребує перевірки. Проте, якщо така система займатиметься здебільшого читанням, то вона може працювати повільніше, ніж MySQL.

Отже, для даного проєкту було обрано MySQL, адже пріоритетами є надійність, простота, швидке читання та масштабованість.

2.5 Amazon Web Services

Amazon Web Services (AWS) є хмарною платформою, що надає набір послуг для забезпечення інфраструктури, розробки програмного забезпечення та зберігання даних віддалено.

AWS працює на основі розподілених серверів у дата-центрах по всьому світу, що дозволяє користувачам отримувати доступ до ресурсів в будь-який час і з будь-якого місця за умови присутності доступу до Інтернету. Платформа пропонує гнучку модель оплати за використані послуги, що дозволяє користувачам платити лише за те, що вони використовують, без необхідності вкладати в кошти на покупку та обслуговування власної обладнання.

Серед послуг AWS, для реалізації даного проєкту було використано Elastic Beanstalk, Relational Database Service (RDS), Simple Storage Service (S3), Elastic Compute Cloud (EC2).

- Elastic Beanstalk – це послуга AWS, яка дозволяє розгортати та масштабувати вебдодатки та сервіси. Elastic Beanstalk автоматично налаштовує інфраструктуру, включаючи вебсервери, бази даних та балансувальники навантаження, щоб забезпечити високу доступність та продуктивність вебдодатків.
- Amazon Elastic Compute Cloud (EC2) – послуга обчислювальної потужності в хмарі.

- Amazon Simple Storage Service (S3) – послуга зберігання об'єктів в хмарі, що дозволяє користувачам зберігати інформацію в безпечному та доступному місці.
- Amazon Relational Database Service (RDS) – послуга управління реляційними базами даних в хмарі, що дозволяє користувачам легко налаштовувати, керувати та масштабувати бази даних.

2.6 Архітектурний підхід MVVM

Очевидно, що використання конкретного архітектурного шаблону робить код простішим для сприйняття для більшості розробників, тому ігнорувати етап дослідження доступних варіантів було б недоцільно. Для найбільш ефективного використання будь-якої технології, варто розуміти, які проблеми вона вирішує, адже бездумне копіювання популярної опції навряд дозволить розкрити її справжній потенціал.

Model-View-Controller (MVC) був найпершим з широко застосовуваних архітектурних шаблонів при розробці Android-додатків. Це не дивно, оскільки на той час він вже успішно зарекомендував себе в розробці програмного забезпечення різного призначення. Розробникам, які вже мали досвід в інших сферах, було легше застосовувати знайому архітектуру, а новачкам він подобався в силу легкості освоєння.

Суть шаблону полягає в тому, щоб розділити додаток на 3 шари:

- Модель (Model) - це шар, де знаходиться бізнес-логіка та стан даних, відповідає за отримання та маніпулювання даними, взаємодію з контролером, базою даних та мережею, іноді оновлює представлення.
- Представлення (View) - це шар, що описує візуальний інтерфейс (часто в вигляді XML-файлу), взаємодіє з моделлю та передає інформацію про дії користувача контроллеру.

- Контроллер (Controller) - шар, що відповідає за комунікацію між моделлю та представленням, в Android-розробці роль контроллера бере на себе компонент Activity або Fragment.

Використання цього шаблону дозволило розділити бізнес-логіку та логіку інтерфейсу, але згодом багато розробників звернули увагу на негативні наслідки сильної залежності шару представлення від контролера та моделі. З розвитком і популяризацією платформи, інтерфейси ставали складнішими, а майже кожна зміна на рівні представлення потребувала оновлень одразу в декількох класах, що значно ускладнювало підтримку, масштабування коду, а також написання тестів.

Підхід **Model-View-Presenter** (MVP) з’явився, як покращення MVC. Першочергово, він націлений на зменшення залежності шару представлення та інших компонент. Пропонується поділ коду на 3 шари: Модель, Представлення, Презентер.

Модель (Model) – це шар, де знаходиться бізнес-логіка та стан даних, відповідає за отримання та маніпулювання даними, взаємодію з презентером, базою даних та мережею. Не взаємодіє з шаром представлення.

Представлення (View) – це шар, що складається з опису візуального інтерфейсу (часто в вигляді XML-файлу), та компонентів ОС Android, таких як Activity та Fragment. Він взаємодіє презентером.

Презентер (Presenter) – шар, що реалізує поведінку додатку, відповідно діям користувача, працює з даними, отриманими від моделі.

Шар представлення та презентер спілкуються через інтерфейс, тобто презентер не знає про деталі представлення і не містить прямих посилань на елементи View. Цей інтерфейс декларує усі можливі дії, які може виконати користувач. Це дає більшу гнучкість та можливість тестування Presenter незалежно від View.

Такий підхід дозволяє легше розширювати та змінювати візуальний інтерфейс, не чіпаючи при цьому логіку презентера, адже він знає лише про

те, що йому необхідно реалізувати певні дії. Та все ж, цей підхід має два фундаментальних недоліків:

1. Життєвий цикл презентера дуже пов’язаний з життєвим циклом View, що, за неуважності розробника, відкриває додаткові шляхи для витоку пам’яті, збоїв програми. Найпростіші дії користувача, наприклад, поворот чи блокування екрану, можуть призвести до зміни конфігурації і, відповідно, знищення View, а разом з ним і будь-яких даних, які Presenter не встиг передати моделі для запису.
2. Велика кількість шаблонного коду.

Model-View-ViewModel (MVVM) є більш новим підходом порівняно з MVP, але він вже здобув значну популярність серед розробників.

В MVVM роль представлення та моделі така ж, як і в MVP. Однак тут з’являється новий компонент – View Model. Цей компонент також обробляє логіку та використовує модель для отримання даних, але він не так сильно прив’язаний до життєвого циклу дії і витримує зміни конфігурації. Це дозволяє уникнути проблем зі збоями та витоками пам'яті, які можуть виникнути в MVP. У MVVM взаємодія між компонентами відбувається за допомогою шаблону Observer, який дозволяє компонентам сповіщати один одного про зміни даних. Наприклад, якщо дані в моделі змінюються, вона повідомляє про це ViewModel, яка в свою чергу сповіщає View про зміни, які потрібно відобразити на екрані. Це значно спрощує процес налагодження та написання тестів.

Серед недоліків можна зазначити дещо вищий поріг входу, адже цей підхід побудований на використанні Android Architecture Components (AAC) – набору бібліотек, що надає інструменти для створення lifecycle-aware компонентів (компонентів, які реагують на зміни життєвого циклу Activity або Fragment). З іншого боку, засвоєння Android Architecture Components стимулює до написання чистого та зрозумілого коду, а також сприяє більшій стійкості до помилок та легшій масштабованості додатків.

Після MVVM з'явився підхід **Model-View-Intent** (MVI) [8]. В MVI, модель та View залишаються такими ж, але контролер (Presenter або ViewModel) замінюється на інтент (Intent), який відповідає за перетворення подій користувача на команди для моделі. Інтент забезпечує однонаправлений потік даних, що дозволяє робити додаток більш детермінованим та простішим для тестування. MVI є доволі новим підходом для розробки додатків на платформі Android та поки не набув широкої популярності. Хоч цей підхід і додає модульності та повинен полегшувати довгострокову підтримку, він не має стандартів, які були б визначені компанією Google, у той час як MVVM був офіційно рекомендований для використання на Android.

Отже, після детального аналізу різних архітектур, було обрано архітектуру MVVM, оскільки вона вирішує всі необхідні проблеми і має широку підтримку спільноти. Крім того, вона має чітко визначені стандарти та готові інструменти для її реалізації з детально прописаною документацією.

2.7 Android Architecture Components

На Google I/O 2017 був представлений набір бібліотек під назвою Android Architecture Components [9]. Його впровадження дозволило формалізувати та офіційно визначити рекомендований підхід до написання коду. Використання цього набору бібліотек дозволяє зменшити кількість шаблоного коду при роботі з локальною базою даних, запобігти витоку пам’яті та проблем зі змінами конфігурації. Окрім того, він спрощує роботу з асинхронними операціями.

З цього набору при розробці застосунку використовувались компоненти LiveData, ViewModel та Room Persistence.

LiveData є одним із компонентів Android Architecture Components, який надає реактивну програмну модель для спостереження за змінами даних. Він

використовується для забезпечення сполучення між джерелами даних (наприклад, базою даних, репозиторієм або мережевим запитом) та компонентами користувацького інтерфейсу (наприклад, Activity або Fragment) у зручний та безпечний спосіб. LiveData дозволяє оголосити об'єкт, який можна спостерігати, та надає методи для оновлення даних. Компоненти користувацького інтерфейсу можуть підписатися на LiveData та отримувати сповіщення про зміни даних. Це дозволяє оновлювати інтерфейс користувача автоматично, коли дані змінюються.

ViewModel є одним із компонентів Android Architecture Components, призначеним для збереження та управління даними, пов'язаними з візуалізацією (UI) користувацького інтерфейсу. Він призначений для вирішення проблеми збереження даних під час поворотів екрану або рекреації Activity або Fragment, а також для відокремлення бізнес-логіки від інтерфейсу користувача. ViewModel розуміє життєвий цикл компонентів Android, таких як Activity або Fragment, і відповідає за збереження та відновлення даних відповідно до стану життєвого циклу.

Room Persistence є одним із компонентів Android Architecture Components, призначеним для спрощення роботи з базою даних SQLite. Room надає високорівневий інтерфейс для взаємодії з базою даних, який допомагає виконувати операції з базою даних швидко, безпечно та зрозуміло.

2.8 Hilt

Hilt – це фреймворк розроблений Google для впровадження залежностей (dependency injection) в Android-додатках. Він базується на функціональності іншого відомого фреймворку для впровадження залежностей – Dagger. Коли додаток стає складним і має багато класів, які взаємодіють один з одним, ручне впровадження залежностей може стати проблемою. Це може призвести до надмірної зв'язності між класами,

ускладнювати тестування і робити код менш масштабованим та повторно використовуваним.

Основні переваги використання Hilt:

- Простота налаштування: Hilt надає простий API для впровадження залежностей.
- Зменшення зв'язності: Впровадження залежностей допомагає зменшити прямі зв'язки між класами.
- Покращення масштабованості: Hilt полегшує масштабування додатків, дозволяючи додавати нові залежності і модулі.
- Покращення тестування: Hilt спрощує тестування, дозволяючи легко замінювати реальні залежності на підроблені або імітовані залежності.
- Інтеграція з Android Jetpack: Hilt інтегрується з бібліотеками Android Jetpack, спрощуючи впровадження залежностей у компонентах Jetpack.

2.9 Jetpack Compose

На Google I/O 2019 було представлено Android Jetpack [10]. Це набір засобів розробки, що має на меті спростити та прискорити розробку візуального інтерфейсу для Android додатків завдяки впровадженню декларативного стилю. Це дозволяє розробникам писати менше коду та зосереджуватися на тому, як має виглядати інтерфейс.

Традиційним для Android є імперативний стиль побудови візуального інтерфейсу: використовуючи комбінацію XML-файлів та Java або Kotlin класів. XML-файли використовуються для опису компонентів і екранів, які будуть відображені. У цих файлах вказуються розміщення, властивості та унікальні ідентифікатори елементів. Java або Kotlin класи відповідають за прив'язку до XML-файлів і містять логіку обробки дій користувача, таких як натискання кнопок або введення тексту. Таким чином, імперативний підхід передбачає явне програмування кожного елемента і керування їхнім станом та поведінкою. Результатом є ієрархічна структура коду, де кожен компонент відповідає за свою частину інтерфейсу. Складність підтримки додатку за

використанням такого підходу швидко збільшується при масштабуванні, оскільки для керування кожним елементом потрібно багато шаблонного коду. Окрім того, через сильну зв’язаність та ієрархічну структуру, з’являється багато перепон при тестуванні окремих компонент.

Jetpack Compose вирішує багато проблем, пов’язаних з імперативним підходом до побудови візуального інтерфейсу.

По-перше, завдяки реактивній природі Jetpack Compose, замість оновлення всього екрану, оновлюються лише ті елементи інтерфейсу, дані яких дійсно змінились. Це призводить до більшої продуктивності та зниження використання ресурсів.

По-друге, фреймворк пропонує компонентно-орієнтований підхід, що дозволяє розділити великі та складні інтерфейси на менші, самодостатні компоненти, які можна перевикористовувати та змінювати незалежно один від одного. Це усуває проблему дублювання коду, забезпечує більшу модульність і розширюваність коду, а також значно спрощує тестування.

По-третє, завдяки декларативному підходу, елементи інтерфейсу оновлюються автоматично, відповідно до зміни моделі, що дозволяє писати менше коду, зменшує ризик помилок та значно пришвидшує розробку.

Узагалі, Jetpack Compose – це перспективна технологія для розробки інтерфейсів для платформи Android, перехід на яку всебічно підтримується Google.

РОЗДІЛ 3. АНАЛІЗ ВИМОГ

Вимоги до програмного забезпечення можна розділити на три основних категорії: системні, функціональні та нефункціональні вимоги.

Функціональні вимоги визначають, які операції повинні бути доступні користувачеві та як система повинна обробляти дані під час виконання цих функцій.

Нефункціональні вимоги описують характеристики, властивості та обмеження системи, які не стосуються безпосередньо функціональності, але впливають на ефективність, безпеку, надійність та інші аспекти програмного забезпечення.

Системні вимоги описують зовнішні умови та обмеження, що стосуються програмного продукту.

3.1 Функціональні вимоги

Доступні операції в системі організовані відповідно ролі користувача. Всього було визначено 3 ролі: Адміністратор, Організація, Відвічувач.

Роль “Організація” надається підтвердженим представникам певної культурної інституції. Відповідно, необхідний механізм подачі заявки на долучення до системи. Кожна заявка повинна бути розглянута вручну та підтверджена перед тим, як установа матиме доступ до авторизації та використання сервісу. Такий підхід необхідний, аби стороння людина не могла видати себе за когось іншого і таким чином спаплюжити репутацію організації.

Після підтвердження заявки, користувачі з роллю “Організація” повинні мати можливість створювати, редагувати, видаляти та переглядати записи про об’єкти під їх юрисдикцією. Таким чином виконуватиметься завдання архівації даних про культурні об’єкти. Також необхідна можливість створення публічних колекцій записів, які можуть переглядати

неавторизовані користувачі системи. Таким чином виконуватиметься завдання популяризації культурних об’єктів.

Роль “Адміністратор” також передбачає обов’язкову авторизацію. Користувачі-адміністратори можуть розглядати подані заявки на приєднання та підтверджувати або відхиляти їх. За виникнення потреби з боку організації, Адміністратор може надати їй статус “Заблоковано”, це заборонить авторизацію для цієї установи та приховає будь-яку публічну про неї.

Роль “Відвідувач” передбачає використання мобільного додатку для перегляду публічних записів та колекцій різноманітних організацій. Задля кращої навігації користувачі-відвідувачі повинні мати можливість підписатись на конкретну організацію та відслідковувати вже переглянуті записи.

3.2 Нефункціональні вимоги

Надійність: Система повинна працювати безперебійно впродовж тривалого часу.

Безпека: Інформаційна система повинна мати механізми захисту даних від несанкціонованого доступу.

Швидкодія: Система повинна забезпечувати швидку відповідь на запити користувачів.

Масштабованість: Система повинна бути готовою до зростання обсягу даних та навантаження.

3.3 Системні вимоги

Для користування системою в цілому необхідний доступ в інтернет.

Вебсервіс повинен гарантувати підтримку для найбільш популярних браузерів: Chrome, Firefox, Opera, Safari.

Мобільний додаток повинен підтримувати версію Android 7.0 (API 24)

РОЗДІЛ 4. РОЗРОБКА ТА РОЗГОРТАННЯ ВЕБСЕРВІСУ

Основна задача вебсервісу – надавати можливість архівації та публікації інформації про пам’ятки культурним установам.

4.1 Огляд схеми бази даних

Android-додатки можуть працювати з віддаленими базами даних, але це не рекомендований підхід з двох причин.

1. Android-застосунки можуть бути декомпільованими, тобто клієнт зможе прочитати паролі та інші дані для доступу до бази даних.
2. Створення і підтримка з’єднання з віддаленою базою даних вимагає багато часу і ресурсів, особливо, якщо клієнт знаходиться далеко від серверу.

Однією з вимог до системи є її безпека та швидкодія, саме тому задачі по обробці чутливих даних було покладено саме на вебсервіс.

З погляду на ці особливості, було спроєктовано схему бази даних.

Організації повинні мати можливість подати заявку на долучення до сервісу. При подачі заявки в базі даних створюється запис в таблиці “organizations” (табл. 1).

Таблиця 1 – Опис структури таблиці “organizations” в БД

Назва поля	Тип	Призначення
id	LONG	Ідентифікатор
organizationName	CHAR(256)	Назва організації
physicalAddress	CHAR(256)	Фізична адреса організації
email	CHAR(128)	Емейл-адреса організації, на яку прийде пароль

fullName	CHAR(256)	ПІП контактної особи для підтвердження
phone	CHAR(20)	Телефон контактної особи для підтвердження
type	CHAR(56)	Тип організації
directions	CHAR(512)	Напрямки роботи організації
status	CHAR(56)	Статус організації (NEW, APPROVED, BANNED)

Після підтвердження заявки відбувається наступне:

1. Статус організації в таблиці organizations змінюється на “APPROVED” (Підтверджено).
2. Випадковим чином генерується пароль для організації та створюється новий запис в таблиці “users”.
3. Організація отримує на email-адресу привітальний лист зі згенерованим паролем для входу.
4. Публічна інформація про цю організацію стає доступною через додаток.

Таблиця “users” відповідає за зберегання інформації для авторизації користувачів (табл. 2).

Таблиця 2 – Опис структури таблиці “users” в БД

Назва поля	Тип	Призначення
id	LONG	Ідентифікатор
username	CHAR(128)	Емейл-адреса користувача/Логін

password	CHAR(128)	Зашифрований пароль користувача
role	SMALLINT	Ідентифікатор ролі (0 або 1)

Доступно 2 ролі: ADMIN, PUBLISHER. Для розподілу ролей не було створено окремої таблиці, бо в системі дозволено лише одного користувача з роллю ADMIN, стовпчик “role” якого містить значення “1”. Усі нові користувачі створюються з роллю PUBLISHER, тобто в стовпчик “role” записується “0”.

Після отримання даних для входу, Організації стає доступним створення експонатів та створення колекцій.

Таблиця “exhibits” відповідає за зберегання інформації про створені експонати (табл. 3).

Таблиця 3 – Опис структури таблиці “exhibits” в БД

Назва поля	Тип	Призначення
id	LONG	Ідентифікатор публікації
user_id	LONG	Ідентифікатор організації, якій належить публікація
title	CHAR(256)	Назва публікації
description	BLOB	Опис публікації
thumbnail	CHAR(1024)	Посилання на мініатюру

subject	CHAR(128)	Тема експонату
type	CHAR(128)	Тип експонату
epoch	CHAR(128)	Епоха, в яку було створено експонат
file	CHAR(1024)	Посилання на основний медіа файл

Таблиця “collections” відповідає за зберегання інформації про створені колекції (табл. 4).

Таблиця 4 – Опис структури таблиці “collections” в БД

Назва поля	Тип	Призначення
id	LONG	Ідентифікатор колекції
title	CHAR(256)	Назва колекції
description	BLOB	Опис колекції
thumbnail	CHAR(1024)	Посилання на мініатюру
user_id	LONG	Ідентифікатор організації, якій належить колекції

Колекції узагальнюють певну групу експонатів, з якими організація вирішила поділитись публічно. Експонат може бути частиною одразу кількох колекцій і колекція може містити одразу декілька експонатів, отже вони мають відношення Many-To-Many. Для репрезентації цього відношення створено таблицю “exhibit_collection” (табл. 5).

Таблиця 5 – Опис структури таблиці “exhibit_collection” в БД

Назва поля	Тип	Призначення
id	LONG	Ідентифікатор
exhibit_id	LONG	Ідентифікатор публікації
category_id	LONG	Ідентифікатор колекції

4.2 Огляд доступних запитів

Відповідно безпекових обмежень, запити поділені на 3 категорії доступу: публічне API, запити для користувача з роллю “Адміністратор” та запити для користувача з роллю “Організація”.

Запити визначені як методи з анотаціями @GetMapping або @PostMapping всередині класів-контролерів, які позначені анотацією @Controller.

Публічне API призначене для отримання вебсторінок, які не потребують авторизації та передачі інформації, яка не розголошує конфіденційні дані. Отже, у запити в цій категорії не повертають такі дані, як: особиста контактна інформація особи, яка подала заяву від імені установи, прямі посилання на файли у сховищі, паролі, записи, які не належать до публічних колекцій.

В цій категорії знаходяться наступні запити:

- /get_publications?subject=&type=&text=&epoch=&media_type= – GET запит для отримання списку усіх публічних експонатів. Доступні фільтри по темі, типу об’єкту, тексту, епосі, типу медіа.
- /get_publication/{id} – GET запит для отримання інформації про експонат за його ідентифікатором “id”.
- /get_organization/{id} – GET запит для отримання інформації про організацію за її ідентифікатором “id”.

- /get_organizations – GET запит для отримання списку усіх організацій
- /get_organization_categories/{id} – GET запит для отримання списку категорій від певної організації за ідентифікатором організації “id”.
- /get_category/{id} – GET запит для отримання інформації про категорію за її ідентифікатором “id”.
- /exhibit_media/{id} – GET запит для отримання вебсторінки, що відображає медіа-файл експонату за його ідентифікатором “id”. Цей запит було поміщено саме на серверну частину, адже обробка великих файлів, а особливо відображення 3D моделей можуть перевантажувати мобільний пристрій та негативно впливати на продуктивність додатку. В перспективі такий підхід також полегшує розширення системи.
- / – GET запит для отримання головної сторінки вебсервісу.
- /register – GET запит для отримання вебсторінки реєстрації.
- /login – GET запит для отримання вебсторінки авторизації.
- /register – POST запит для створення заявки на долучення до сервісу.
- /login – POST запит для автентифікації.
- /logout – POST запит для інвалідації поточної сесії, якщо така є.

Запити для користувача з роллю “Адміністратор” потребують попередньої авторизації.

- /organizations – GET запит для отримання списку з інформацією про організації, включаючи контактні дані та статус.
- /approve/{id} – POST запит для підтвердження заявки на приєднання від організації по ідентифікатору “id”.
- /decline/{id} – POST запит для відхилення заявки на приєднання від організації по ідентифікатору “id”.
- /ban/{id} – POST запит для надання статусу “Заблоковано” організації по ідентифікатору “id”.

Запити для користувача з роллю “Організація” потребують попередньої авторизації.

- /exhibits?subject=&type=&text=&epoch=&media_type= – GET запит на отримання сторінки з усіма створеними експонатами для організації. Доступні фільтри по темі, типу об’єкту, тексту, епосі, типу медіа.
- /exhibit/{id} – GET запит на отримання сторінки з деталями про експонат за ідентифікатором “id”. Перед виконанням відбувається перевірка, чи належить він організації.
- /add-exhibit – GET запит на отримання сторінки для створення експонату для організації.
- /add-exhibit – POST запит на збереження інформації про новий експонат для організації.
- /edit-exhibit/{id} – GET запит на отримання сторінки для редагування експонату за ідентифікатором “id”. Перед виконанням відбувається перевірка, чи належить він організації.
- /edit-exhibit/{id} – POST запит на збереження відредагованої інформації про експонат з ідентифікатором “id”. Перед виконанням відбувається перевірка, чи належить він організації.
- /delete-exhibit/{id} – POST запит на видалення експонату за ідентифікатором “id”. Перед виконанням відбувається перевірка, чи належить він організації.
- /categories – GET запит на отримання сторінки з усіма створеними категоріями для організації.
- /add-category – GET запит на отримання сторінки для створення категорії для організації.
- /add-category – POST запит на збереження інформації про нову категорію для організації.
- /edit-category/{id} – GET запит на отримання сторінки для редагування категорії за ідентифікатором “id”. Перед виконанням відбувається перевірка, чи належить вона організації.

- /edit-category/{id} – POST запит на збереження відредагованої інформації про експонат з ідентифікатором “id” для організації. Перед виконанням відбувається перевірка, чи належить вона організації.
- /delete-category/{id} – POST запит на видалення експонату за ідентифікатором “id” для організації. Перед виконанням відбувається перевірка, чи належить вона організації.

4.3 Авторизація та захист від несанкціонованого доступу

Для забезпечення авторизації та захисту від несанкціонованого доступу було використано Spring Security та стандарт JWT.

Після додавання необхідних залежностей, було створено клас конфігурації помічений анотаціями @Configuration @EnableWebSecurity. Цей клас розширює WebSecurityConfigurerAdapter та перевизначає метод “configure” для налаштування прав доступу. В цьому ж методі зазначено сервлет фільтр для валідації JWT-токена, він виконується перед надсиланням запиту. JWT-токен генерується під час авторизації (рис. 4).

```
Jwts.builder()
        .setSubject(authentication.getName())
        .claim(AUTHORITIES_KEY, authorities)
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() + TOKEN_VALIDITY))
        .signWith(SignatureAlgorithm.HS256, SIGNING_KEY)
        .compact();
```

Рисунок 4 – Генерація JWT-токену під час авторизації

Поле "username" користувача (отримане з authentication.getName()) встановлюється як одне з зарезервованих тверджень "sub" (subject). Значення ролі користувача додається в якості додаткового твердження (claim). Далі встановлюється час життя та алгоритм підпису для токену.

В подальшому токен зберігається та зчитується з Cookie під час виконання фільтру. Токен інвалідується, якщо минув час життя або було виконано запит “/logout”.

Після вищеописаних налаштувань, методи контролерів, виконання яких дозволено лише користувачам конкретної ролі необхідно помітити анотацією @PreAuthorize з параметром "hasAuthority('ADMIN')", якщо це метод для користувача-Адміністратора, або з параметром "hasAuthority('PUBLISHER')", якщо це метод для користувача-Організації.

4.4 Робота з базою даних та впровадження залежностей

Після додавання необхідних залежностей для роботи з базою даних в файлі application.properties необхідно вказати налаштування бази даних (посилання, дані для авторизації, платформа, драйвер-клас). Класи-моделі, що відповідають таблицям, необхідно помітити анотаціями @Entity та визначити поле, яке слугуватиме в якості основного ключа для таблиці за допомогою анотацій @Id та @GeneratedValue. Далі необхідно створити класи-репозиторії з відповідним типом даних, вони повинні розширювати клас JpaRepository та бути поміченими анотацією @Repository, аби Spring міг їх розпізнати. JpaRepository надає базові методи для роботи з базою даних, наприклад, метод для пошуку по ідентифікатору або метод для додавання запису.

Для впровадження залежності, яка має конструктор без параметрів, достатньо додати її як поле класу, де вона необхідна, та помітити її анотацією @Autowired.

4.5 Розгортання проєкту на віддаленому сервері

Для розгортання додатку на віддаленому сервері було використано наступні сервіси від AWS: EC2, RDS, Elastic Beanstalk, S3.

Перш ніж почати використовувати ці сервіси, необхідно пройти реєстрацію на Amazon Web Services та обрати регіон для Elastic Beanstalk.

Було обрано європейський регіон, оскільки він ближче та надає доступ до необхідних сервісів.

Наступним кроком була реєстрація нового додатку в Elastic Beanstalk з середовищем типу "web server environment" [11]. Як тільки середовище закінчить формування та запуститься, на панелі керування сервісом буде доступний ендпойнт, по якому можна буде звернутись до додатку. Під час його розгортання також запускається екземпляр EC2, який виступає хостом для нашого сервера. Оскільки основний фокус дипломної роботи не був спрямований на процес розгортання, jar-файл додатку буде завантажено безпосередньо за допомогою кнопки "Upload and Deploy". Проте, варто відзначити, що для проєктів, над якими працює декілька людей, рекомендується використовувати більш надійні методи розгортання через Docker-контейнери.

Далі на панелі керування RDS необхідно створити базу даних типу MySQL. Після активації БД було отримано посилання на неї, яке необхідно додати в файл application.properties. Для забезпечення безпеки та уникнення помилок при спробі підключення до бази даних було також налаштовано правила груп безпеки відповідно до вимог Amazon.

Для роботи з Amazon S3 необхідно створити клієнт (рис. 5).

```
@Bean
public AmazonS3 amazonS3() {
   return AmazonS3ClientBuilder
           .standard()
           .withCredentials(new AWSStaticCredentialsProvider(credentials()))
           .withRegion(Regions.US_WEST_2)
           .build();
}
```

Рисунок 5 – Конфігурація клієнту AmazonS3

Аннотація @Bean на рисунку вказує, що метод amazonS3() повинен бути визначений як bean і буде керуватися контейнером Spring. Виклик AmazonS3ClientBuilder.standard() повертає новий екземпляр

AmazonS3ClientBuilder, який дозволяє налаштувати параметри підключення. Далі викликаються методи для встановлення облікових даних (withCredentials()) і регіону (withRegion()). Функція credentials() повертає об’єкт BasicAWSCredentials з визначеними параметрами для автентифікації accessKey та secretKey. Значення параметрів для автентифікації можна отримати, зареєструвавши нового "IAM" користувача з необхідними правами доступу до Amazon S3 та створивши для нього ключі на панелі Security credentials.

Подальша робота з сервісом здійснюється через методи об’єкту AmazonS3.

4.6 Рендеринг 3D моделей на вебклієнті

Щоб відобразити 3D модель за допомогою Three.js, потрібно імпортувати залежності, налаштувати сцену, додати світло, камеру, віджет рендерингу та завантажити модель.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, w / h, 0.1, 1000);

var renderer = new THREE.WebGLRenderer();
renderer.setSize(w, h);
div.appendChild(renderer.domElement);
```

Рисунок 6 – Створення основних компонент для відображення 3D моделі

На рисунку 6 продемонстровано код, що створює сцену, камеру, віджет рендерингу та додає це до HTML-сторінки. Далі налаштовується світло та додається до сцени, без цього етапу замість 3D моделі відображатиметься чорне полотно.

Найважливіший етап – завантаження моделі. Для цього необхідно створити об’єкт GLTFLoader та викликати його метод load (рис. 7). В його

параметри необхідно передати посилання на 3D модель та 3 функції: функція на випадок успішного завантаження моделі, функція для слідкуванням за прогресом, функція на випадок помилки. Деякі моделі можуть мати великий розмір, тому важливо повідомляти користувачу, що їх завантаження в процесі. Саме для цього на HTML-сторінці поверх віджету рендерингу знаходиться рядок прогресу, який оновлюється відповідно відсотку завантаження моделі.

```
const loader = new THREE.GLTFLoader();
var progressBar = document.getElementById('progressBar');

loader.load(url,
  function ( gltf ) {
    // Модель успішно завантажена
    // Додавання моделі до сцени
    scene.add( gltf.scene );

    // Приховування рядку прогресу
    progressBar.style.display = 'none'
  },

  function ( xhr ) {
    // Оновлення рядку прогресу
    const percentComplete = (xhr.loaded / xhr.total) * 100
    progressBar.value = percentComplete === Infinity ? 100 : percentComplete
  },

  function ( error ) {
    console.log( 'An error happened' );
  }
);
```

Рисунок 7 – Код для завантаження моделі та оновлення рядку прогресу

Після цих кроків було додатково налаштовано камеру для керування за допомогою миші або дотику на пристроях з сенсорним екраном за допомогою класу OrbitControls бібліотеки Three.js.

І, нарешті, віджет рендерингу повинен відобразити все те, що було налаштовано. Для цього необхідно викликати метод render і передати в нього камеру та сцену.

РОЗДІЛ 5. РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

5.1 Структура проєкту

В рамках даного проєкту код розділено на три основні пакети: “data”, “di” та “ui”.

У пакеті “ui” розміщені файли, що відповідають за візуальне представлення. Ці файли містять Composable функції та класи ViewModel.

Composable функції – це функції, які позначені анотацією @Composable, наданою Jetpack Compose. Вони використовуються для опису зовнішнього вигляду та властивостей окремих візуальних елементів. Функції можуть бути вкладені одна в одну, створюючи дерево компонентів. Composable функції підтримують оновлення користувацького інтерфейсу на основі змін у стані або даних.

ViewModel є компонентом архітектури, який відповідає за збереження та керування даними, пов'язаними з візуальним представленням. Класи ViewModel містять бізнес-логіку та дані, необхідні для відображення та взаємодії з користувачем. Вони надають методи та змінні, до яких можна отримати доступ з Composable функцій, щоб обробляти події, зберігати стан та надавати дані для відображення. Класи ViewModel допомагають відокремити логіку та стан від візуального представлення.

У пакеті “data” знаходяться файли, що відповідають за роботу з даними. Сюди входять моделі, що представляють дані, репозиторії, які надають інтерфейс для маніпуляцій з даними, а також допоміжні класи, що допомагають встановити зв'язок з джерелами даних (наприклад, віддаленим сервером чи локальною базою даних). Інтерфейс, наданий репозиторіями, використовується класами ViewModel з пакету “ui”.

В пакеті “di” визначено класс AppModule. Він створює усі залежності, які можна впровадити в додатку за допомогою анотації @Inject (рис. 8).

```
class RemoteRepository @Inject constructor(private val apiHelper: ApiHelper)
```

Рисунок 8 – Приклад використання анотації @Inject

Крім цього, важливо відзначити роль таких файлів, як AndroidManifest.xml, build.gradle та app/build.gradle.

AndroidManifest.xml визначає необхідні налаштування для правильного функціонування додатку на платформі Android. У цьому файлі можна вказати необхідність дозволів до захищених функцій телефону, таких як камера, мікрофон, геолокація, а також описати додаткові компоненти, такі як сервіси, отримувачі мовлення та активності. Також, в маніфесті вказується основний екран (Launcher Activity), який буде відображатися при запуску додатку.

Файли build.gradle та app/build.gradle використовуються для налаштування процесу збирання (build) проєкту. В них визначаються залежності, компіляційні налаштування, версії SDK та інші параметри, необхідні для правильної збірки додатку. build.gradle є загальним файлом для всього проєкту, тоді як app/build.gradle є файлом конфігурації конкретного додатку (модулю). Ці файли використовуються для керування залежностями, встановлення налаштувань компіляції та виконання інших дій, пов'язаних зі збиранням проєкту.

5.2 Комунікація з віддаленим сервером

Для роботи з віддаленим сервером перш за все необхідно вказати в маніфесті, що додатку для роботи необхідний дозвіл на доступ в інтернет (рис 9).

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Рисунок 9 – Рядок, що оголошує необхідність дозволу на доступ в інтернет

Далі необхідно додати необхідні залежності для Retrofit2 та налаштувати клієнт і інтерфейс сервісу, що займатимуться обробкою запитів. Отож, для роботи Retrofit2 необхідно налаштувати клієнт. Так як в проєкті використовується Hilt, то достатньо визначити цей клієнт в основному класі додатку, що надає залежності (рис. 10).

```
@Provides
@Singleton
fun provideRetrofit(client: OkHttpClient, @Named("BASE_URL") BASE_URL: String
): Retrofit = Retrofit.Builder()
        .client(client)
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
```

Рисунок 10 – Приклад налаштування Retrofit2

Метод повертає об’єкт Retrofit, який необхідний для роботи бібліотеки. Серед його параметрів є BASE_URL, помічений анотацією @Named. Цей параметр визначено в основному класі додатку, він містить в собі доменне ім’я віддаленого серверу.

Окрім того, серед параметрів є okHttpClient - це екземпляр OkHttpClient, необхідний Retrofit для виконання мережевих запитів. OkHttpClient використовується в Retrofit для налаштування параметрів мережевого взаємодії, таких як таймаути, перехоплення запитів і відповідей, інтерсептори та інше. Цей екземпляр був аналогічним чином створений всередині основного класу додатку і позначений анотаціями @Provides та @Singleton, тому він підтягується автоматично і не перестворюється для кожного запиту. Для OkHttpClient було налаштовано інтерсептор, який додає до запиту параметр “limit” з значенням “16”. Цей параметр використовується віддаленим сервером для реалізації пагінації – один запит не повертає більш, як 16 об’єктів.

Для визначення запитів було створено інтерфейс ApiService. В цьому інтерфейси оголошені методи з анотацією @GET, вони визначають URL-шаблони для GET-запитів. Імплементувати інтерфейс немає необхідності, адже Retrofit2 використовує механізм Dynamic Proxy, завдяки якому бібліотека створює динамічні проксі-класи, що перехоплюють виклик методів і реалізують необхідну мережеву взаємодію для відправки запитів та отримання відповідей.

5.3 Робота з локальною базою даних

Room — це бібліотека, що реалізує рівень абстракції над базою даних SQLite. В контексті цього додатку, необхідно створити 2 таблиці: “subscriptions”, “viewed”.

Таблиця “subscriptions” відповідає за збереження ідентифікаторів організацій, на які підписався користувач.

Таблиця “viewed” відповідає за збереження ідентифікаторів публікацій, які вже були переглянуті користувачем – це необхідно для простішої навігації колекціями.

Після додавання необхідних залежностей, необхідно було створити класи з анотацією @Entity, що відображають структуру таблиц та позначити один із параметрів анотацією @PrimaryKey.

Наступним кроком було створення інтерфейсу DAO (Data Access Object) з анотацією @Dao, він відповідає за декларацію методів, що здійснюють запити до бази даних. Методи в інтерфейсі, що здійснюватимуть запити необхідно позначати анотацією @Query та зазначати в її параметрі рядок запиту до бази даних. Для методів також існують готові анотації для частовживаних запитів, наприклад @Inject чи @Delete, для яких не потрібно визначати параметри.

Далі було створено клас MainDatabase, що розширює RoomDatabase (частина фреймворку Android Architecture Components) та помічений анотацією @Database. В параметрах анотації вказано класи, що містять

репрезентацію структури таблиць. Екземпляр MainDatabase служить об’єктом-тримачем бази даних, до нього було додано абстрактні методи, що повертають тип раніше створених DAO-інтерфейсів. Як і клієнт для Retrofit2, екземпляр MainDatabase необхідно створити в основному класі додатку та помітити анотаціями @Provides та @Singleton, щоб в подальшому впроваджувати його в репозиторії.

Після цих налаштувань, класи, що впроваджують MainDatabase, можуть отримати екземпляри DAO-інтерфейсів і викликати визначені в них методи.

5.4 Навігація між елементами представлення

Для налаштування навігації між компонентами було застосовано Navigation component [12], що використовується для визначення зв’язку між візуальними компонентами в додатку.

Navigation Component використовує поняття NavGraph (граф навігації) для опису навігаційної структури додатка. NavGraph визначає всі екрани (destination) та зв'язки між ними. Один з ключових атрибутів NavGraph - це route (маршрут). Він визначає унікальну ідентифікацію кожного екрану та його адресу в навігаційній структурі. Для задання графу в Jetpack Compose можна використати функцію NahHost (рис. 11), яка виступає в якості контейнера, що визначає область, де будуть відображатись різні екрани. Цей контейнер розташовано в основній активності, тобто тій, яка відкривається при старті додатку.

```
@Composable
public fun NavHost(
    navController: NavHostController,
    startDestination: String,
    modifier: Modifier = Modifier,
    route: String? = null,
    builder: NavGraphBuilder.() -> Unit
)
```

Рисунок 11 – Сигнатура функції NavHost

Серед параметрів NavHost, використано navController, builder, startDestination.

Параметр navController – об'єкт, який керує навігацією між екранами, він містить методи, що допомагають пересуватись по елементам графа навігації.

Параметр builder був використаний для визначення усіх можливих шляхів для графа навігації. Шлях задається назвою, списком аргументів в назві, та контентом, тобто Composable функцією, яка буде відображена, якщо navController перейде по цьому шляху.

Параметр startDestination був використаний для задання початкового екрану.

5.5 Розробка елементів представлення

Основний екран додатку містить нижнє меню навігації з трьома опціями: “Експонати”, “Організації”, “Підписки”. За замовчуванням обрана опція “Експонати”. При виборі будь-якої з опцій, над меню навігації відображається відповідний контент.

Якщо обрана опція “Експонати”, то відображається галерея усіх публічних записів, отриманих від сервера по запиту “/get_publications” (рис. 12). На цій сторінці також присутній пошук з параметрами “текст”, “тип медіа”, “предмет”, “тип”, який додає до запиту необхідні параметри для фільтрації та оновлює список. При натисканні на будь-який елемент списку, відкривається екран з деталями про обрану публікацію, необхідна інформація отримується по запиту “/get_publication/{id}”.

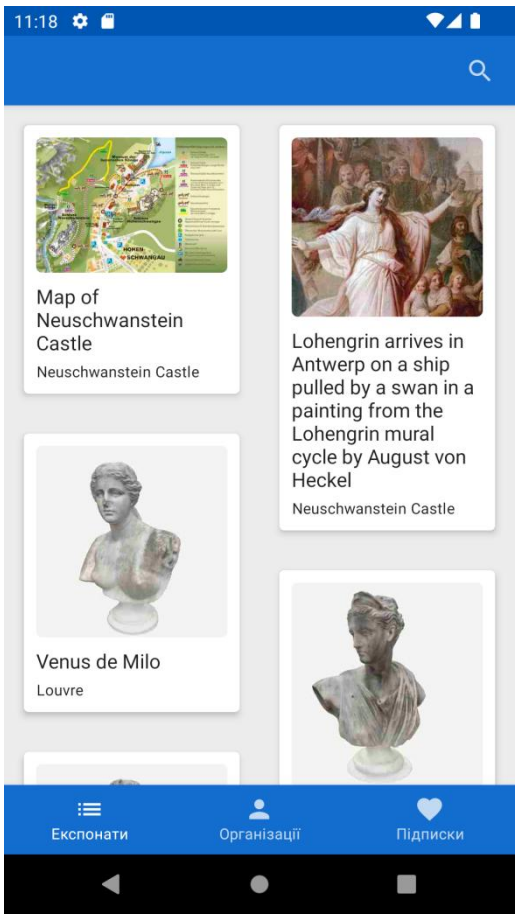

Рисунок 12 – Екран “Експонати”

Якщо обрана опція “Підписки”, то відображається галерея публікацій тільки від тих організацій, на які підписаний користувач. Інформація про підписки отримується з локальної бази даних.

Якщо обрана опція “Організації”, то відображається список усіх організацій, отриманий по запиту “/get_organizations” (рис. 13). При натисканні на будь-який елемент списку відображається детальна інформація про організацію, а також список публічних колекцій. Ця інформація отримується по запиту “/get_organization/{id}” (рис. 14). Якщо натиснути на будь-який елемент зі списку колекцій, відкриється екран з деталями про обрану колекцію. Список публікацій в колекції отримується по запиту “/get_category{id}”. На цьому екрані доступно 2 способи перегляду, які можна змінити за допомогою нижнього меню навігації:

- Опція “Екскурсія” (відображається за замовчуванням) демонструє елементи колекції у вигляді туру: вони впорядковані відповідно

способу, який задала організація, що їх опублікувала, та містять візуальні підказки, що допомагають зорієнтуватись, які публікації користувач вже переглянув (рис. 15).

- Опція “Галерея” не гарантує порядку та не містить підказок, користувач може вільно обирати, яку публікацію хоче переглянути наступною.

При натисканні на будь-який елемент зі списку публікацій в колекції, відкривається вже згаданий екран з деталями про відповідний елемент.

На екрані з деталями про організацію також міститься кнопка “Підписатись”. Після натисканні на неї, користувач підписується на організацію. Якщо користувач вже був підписаний на цю організацію, то замість неї відображатиметься кнопка “Відписатись”, натискання на яку скасує підписку.

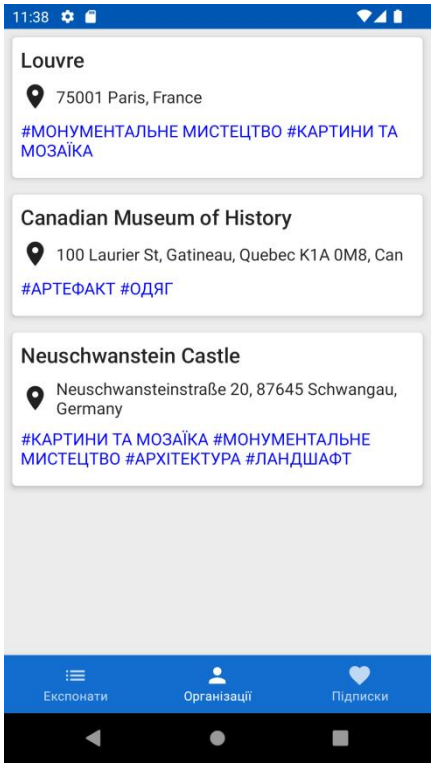

Рисунок 13 – Екран “Організації”

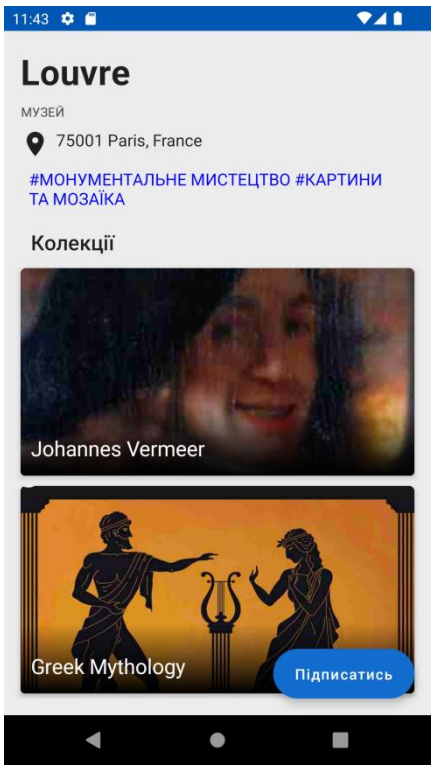

Рисунок 14 – Екран з деталями про організацію

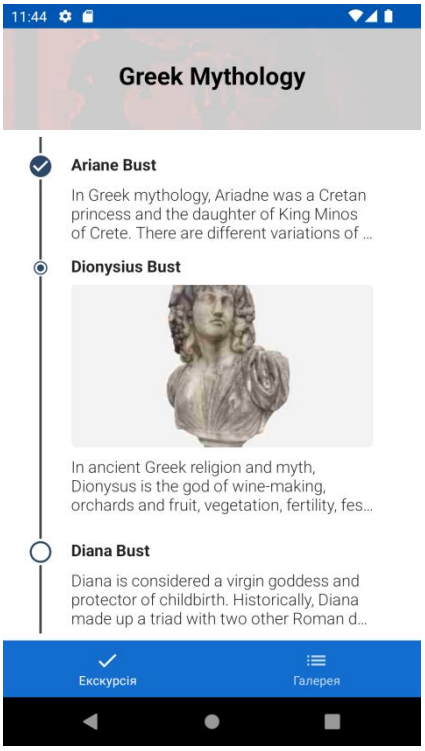

Рисунок 15 – Екран з деталями про колекцію, обрано опцію “Екскурсія”

ВИСНОВКИ

В ході виконання роботи було проаналізовано існуючі рішення в предметній області, сформано завдання, розглянуто необхідний набір технологій для його виконання та розроблено інформаційну систему.

Інформаційна система сформована з двох окремих проєктів: вебсервісу та мобільного додатку. Це дозволило забезпечити кращий рівень безпеки та розширюваності системи.

Для серверної частини вебсервісу використано мову програмування Java, фреймворк Spring, СУБД MySQL та різні допоміжні бібліотеки та інструменти. Клієнтська частина побудована з використанням механізму шаблонів Thymeleaf, фреймворка Bootstrap та JavaScript бібліотеки Three.js. Для розгортання вебсервісу було використано Amazon Web Services.

Для розробки Android-додатку було використано мову програмування Kotlin, набір бібліотек Android Architecture Components, фреймворк Hilt та Jetpack Compose.

Вебсервіс може бути використаний культурними інституціями для архівації та організації об’єктів в їх юрисдикції. Мобільний додаток може бути використаний усіма, хто бажає більше дізнатись про важливі пам’ятки, але не має такої можливості через такі фактори, як відстань, вартість чи фізичні обмеження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Поливач К. А. Культурна спадщина та її вплив на розвиток регіонів України / Катерина Анатоліївна Поливач., 2012.
2. Вебсайт Міністерства культури та інформаційної політики України [Електронний ресурс] – Режим доступу до ресурсу: https://culturecrimes.mkip.gov.ua/
3. Mobile Operating System Market Share Ukraine [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/ukraine
4. Android-додаток «British Museum Audio Buddy» [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=air.com.bm.london.vusiem&hl=en&gl=US
5. Android-додаток «Asian Art Museum SF» [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.acoustiguidemobile.asian_art_museum&hl=en&gl=US
6. Web-портал «Europeana» [Електронний ресурс] – Режим доступу до ресурсу: https://www.europeana.eu/en
7. Heckler M. Spring Boot Up & Running / Mark Heckler., 2021.
8. Dumbravan A. Clean Android Architecture Take a Layered Approach to Writing Clean, Testable, and Decoupled Android Applications / A. Dumbravan, E. Price., 2022.
9. Google I/O 2017 [Електронний ресурс] – Режим доступу до ресурсу: https://blog.google/technology/developers/all-io17-announcements/
10. Google I/O 2019: Empowering developers to build the best experiences on Android + Play [Електронний ресурс] – Режим доступу до ресурсу: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html

11. AWS Elastic Beanstalk - Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/elasticbeanstalk/latest/dg/awseb-dg.pdf#create_deploy_Java
12. Офіційна документація щодо налаштування навігації для Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/jetpack/compose/navigation