МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МЕЛЬНИК ОЛЕГ ОЛЕГОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ КВИТКІВ ДЛЯ ПОДОРОЖЕЙ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
І. В. Богач, доцент кафедри
інформаційних технологій,
к. т. н., доцент

_____

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

Вінниця - 2024

АНОТАЦІЯ

Мельник О. О. Інформаційна система пошуку квитків для подорожей. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній (бакалаврській) роботі досліджено системи пошуку квитків для подорожей. За допомогою таких технологій як Java, Kotlin та Rust розроблено інформаційну систему для пошуку квитків на подорожі, що дозволяє користувачам знаходити та бронювати потрібні квитки.

Ключові слова: інформаційна система, пошук квитків для подорожей, бронювання квитків, Java, Kotlin, Rust.

81 ст., 27 рис., 5 табл., 2 дод., 34 джерел.

ABSTRACT

Melnyk O. O. Information System for Searching Tickets for Travel. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2024.

In the qualification (bachelor's) thesis, travel ticket search systems were investigated. Using technologies such as Java, Kotlin, and Rust, an information system for travel ticket search was developed, allowing users to find and book the necessary tickets.

Keywords: information system, travel ticket search, ticket booking, Java, Kotlin, Rust.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. У сучасному світі використання різноманітних інформаційних систем у різних сферах не є лише додатковою можливістю, але стає необхідністю для прискорення роботи, залучення нових клієнтів, розширення сфери впливу та інших аспектів [6]. Сфера туризму завжди була однією з найбільш популярних, прибуткових та динамічних [1]. Сучасній людині значно комфортніше та швидше вирішувати свої потреби за допомогою Інтернету [15]. Часто виникає потреба знайти та забронювати квиток на поїздку. Проте іноді виникають проблеми з організацією роботи, коли людям доводиться показувати паперові квитки або купувати їх на місці, що може бути незручним. Відповідно до цього було прийнято рішення щодо розробки інформаційної системи для пошуку квитків на подорожі. Для її реалізації одним з найкращих варіантів є мобільний додаток, оскільки наразі більшість інтернет-трафіку припадає на смартфони, і їх кількість постійно зростає.

Існують програмні рішення для вирішення зазначених проблем, такі як сайти з продажу квитків, але вони можуть бути складними у використанні через ряд додаткових функцій, таких як бронювання готелів, оренда автомобілів, інтернет-страхування тощо, що робить деякі сервіси досить перенасиченими. У результаті створення спеціалізованої інформаційної системи, що фокусується на пошуку та бронюванні квитків, стає актуальною.

Мета роботи - розробка інформаційної системи пошуку та бронювання квитків для подорожей, що на відміну від існуючих буде простою в користуванні та забезпечить більш швидку обробку даних в порівнянні з існуючими системами.

Для її реалізації потрібно виконати наступні **завдання дослідження**:

1. Провести аналіз предметної області, особливу увагу приділити типам пошукових систем квитків та їх особливостям.

2. Здійснити аналіз відомих рішень цієї проблеми, визначити переваги та недоліки.

3. Поставити цілі до додатку, визначити тип системи та обрати програмне забезпечення та інструменти для розробки.

4. Розробити інформаційну систему пошуку квитків для подорожей, протестувати функціонал розробленої системи та описати результат.

Об’єкт дослідження – процес пошуку та бронювання квитків для подорожей.

Предмет дослідження – методи та засоби здійснення пошуку та бронювання квитків для подорожей.

Теоретичне та практичне значення одержаних результатів. В результаті роботи розроблено інформаційну систему для пошуку квитків на подорожі, що дозволяє користувачам знаходити та бронювати потрібні квитки із застосуванням Java, Kotlin та Rust, що на відміну від існуючих систем суттєво спрощує процеси пошуку та бронювання квитків.

Розроблена інформаційна система запрограмована для операційної системи Android.

Апробація результатів дослідження

Основна частина роботи подана на отримання свідоцтва на авторське право.

Структура кваліфікаційної роботи – Робота складається зі вступу, чотирьох розділів, логічного висновку, списку використаних джерел з 34 джерел включаючи інтернет-ресурси, документацію обраних технологій, довідників та посібників, 5 таблиць та 27 рисунків. Загальний обсяг основної частини роботи - 61 сторінка.

РОЗДІЛ 1
ОГЛЯД ТА ПОРІВНЯННЯ ІСНУЮЧИХ СИСТЕМ ПОШУКУ КВИТКІВ ДЛЯ ПОДОРОЖЕЙ

1.1 Методологічні засади дослідження

На сьогоднішній день ринок надання різноманітних послуг постійно розширюється. Однією з таких послуг є подорожі, що залишається дуже актуальною темою, оскільки кількість подорожуючих людей зростає з кожним роком [2]. Мільйони людей щодня вирушають на відпочинок або відправляються у ділові поїздки [4]. Усіх їх об’єднує те, що вони потребують квитки на різні види транспорту (такі як літаки, поїзди або автобуси), які вони знаходять та придбають. Зростає популярність онлайн-покупок квитків, коли люди часто роблять попереднє замовлення, не купуючи або замовляючи продукт відразу. З урахуванням тенденції до цифровізації, актуальність розробки програмного забезпечення для таких послуг буде тільки зростати [3].

Для того щоб залишатись конкурентоспроможними, компаніям необхідно постійно вдосконалювати їхні продукти та впроваджувати нові технології [5]. Використання розробленої системи має спростити управління різними процесами, пов’язаними з отриманням та обробкою замовлень, дозволяючи менеджерам отримувати необхідну інформацію для побудови ефективної економічної політики.

1.2 Пошукова система, її типи та принципи

Метою пошукової системи є отримання інформації з бази даних в Інтернеті. Пошукові системи стають важливим щоденним інструментом для пошуку необхідної інформації, не знаючи, де саме вона зберігається.

За останні десятиліття використання Інтернету значно зросло завдяки простоті та доступності пошукових систем, таких як Google, Bing та Yahoo! Ці

системи надають користувачам широкий доступ до різноманітної інформації в мережі. Існують різні типи пошукових систем, які допомагають знаходити конкретну інформацію, яку ви шукаєте.

Пошукові системи є частиною повсякденного життя двох типів людей:

- Користувачі, які шукають та отримують інформацію;
- Власники сайтів, які намагаються оптимізувати свої веб-сайти для отримання найвищого рейтингу в результатах пошуку.

Користувачі здійснюють понад мільярди пошукових запитів лише у Google, щоб знайти необхідну інформацію. Це відкриває величезні можливості для компаній і видавців онлайн-контенту для привертання відвідувачів на свій веб-сайт безкоштовно. Пошукові системи керуються своїми власними алгоритмами та дотримуються певних правил для рейтингування веб-сайтів. Оптимізація веб-сайтів пошукових систем маштабу для Google та інших стає важливою частиною стратегії будь-якого власника веб-сайту, оскільки це дозволяє привернути широку аудиторію. Відвідувачі можуть отримувати дохід для власників сайтів за допомогою реклами, що відображається на сайті, або купуючи продукти.

Пошукові системи поділяються на категорії залежно від того, як вони працюють:

- пошукові системи на основі сканера;
- каталоги, що працюють від людини;
- гібридні пошукові системи;
- інші спеціальні пошукові системи.

Усі пошукові системи, базуючись на сканерах, використовують сканер, бота або павука для проходження по веб-сайтах, сканування їх вмісту та індексації нової інформації у своїй базі даних пошуку. Більшість популярних пошукових систем працюють на основі сканерів і використовують цю технологію для відображення результатів пошуку.

Гібридні пошукові системи комбінують автоматичну індексацію за допомогою сканерів і ручну для включення сайтів у результати пошуку.

Більшість пошукових систем, таких як Google, переважно використовують сканери як основний засіб індексації, доповнюючи їх людськими каталогами як допоміжний метод. Наприклад, Google може взяти опис веб-сторінки з ручних каталогів і відобразити його в результатах пошуку. Оскільки каталоги, що працюють від людини, зникають, гібридні типи стають все більше і більше пошуковими системами на основі сканерів. Оскільки каталоги, що працюють від людини, зникають, гібридні типи стають все більше і більше пошуковими системами на основі сканерів.

Крім перерахованих основних типів, пошукові системи класифікують на інші категорії залежно від використання. Наприклад, пошукові системи використовують різні типи ботів для окремого відображення відео, новин, зображень і різноманітних списків. Наприклад, сторінка Google News дозволяє шукати тільки новини з різних видань. Деякі пошукові системи, як-от Dogpile, збирають метаінформацію зі сторінок інших пошукових систем і каталогів для відображення у своїх результатах. Такі пошукові системи відносять до матепошуку і називаються метапошуковими. Інші пошукові системи, як-от Swoogle, надають точні результати в певній сфері, враховуючи контекстне значення пошукових запитів.

1.3 Аналіз відомих програмних систем

На сьогоднішній день існує низка подібних додатків. Кожен з них має свої переваги та недоліки: деякі відрізняються зручним інтерфейсом, інші пропонують більший вибір критеріїв для пошуку, або приваблюють більше клієнтів.

Такі сайти та додатки отримують прибуток від розміщення реклами інших проектів на своїх сторінках. Proizd.ua (рис.1.1) – це інтернет-сервіс, який дозволяє користувачам знаходити та придбавати квитки на поїзди, автобуси або літаки за будь-якими параметрами. Він розпочав роботу з 21 грудня 2014 року.

Даний сервіс працює як на ПК так і на мобільних пристроях досить швидко, інтерфейс інтуїтивний та зручний. Функціонал сервісу передбачає автоматичне створення кабінету після оплати квитка. За даними аналітики на цей сервіс приходиться 5% ринку проданих квитків в Україні [7].

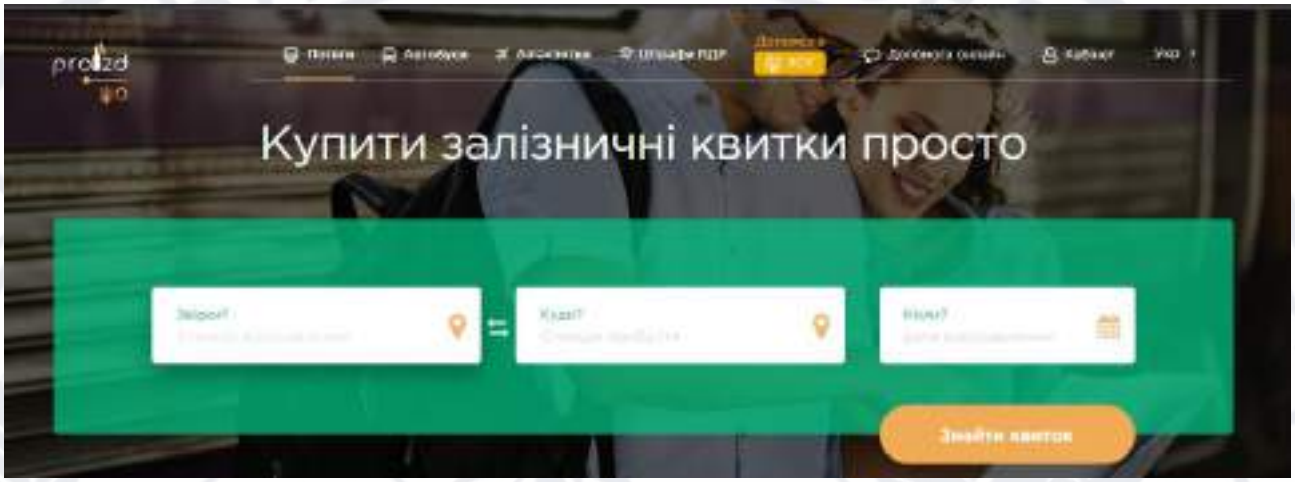

Рисунок 1.1 - Головна сторінка Proizd.ua

До переваг Proizd.ua можна віднести великий вибір типів транспорту, адже вони надають послуги щодо придбання квитків на потяги, автобуси та літаки. Також інтерфейс є досить зручним, і під час роботи не виникає ніяких труднощів.

Проте недоліки також присутні. Наприклад, при переключенні між фільтрами є ймовірність того, що попередні налаштування скинуться, і вам доведеться починати пошук спочатку.

Busfor.ua (рис. 1.2) — міжнародний проект з продажу автобусних квитків на міжміські та міжнародні рейси, який розпочав свою роботу ще в 2010 році. Busfor не перевізник, але надає послуги для придбання квитків на автобус від інших перевізників. Цей сервіс має мобільний додаток та сайт, що працюють на технології GDS (база власної розробки), що представляє собою інноваційну систему дистрибуції та продажу автобусних квитків.

Перевізники вже давно навчились успішно використовувати дану технологію для оперативного розміщення вільних місць на своїх автобусних рейсах щоб продавати їх клієнтам онлайн. Також до системи GDS вже

підключені автовокзали та автостанції, що дозволяє підвищити якість надаваємих послуг, оскільки вони на сервісі Busfor дають пасажирам можливість купувати онлайн автобусні квитки. [10].

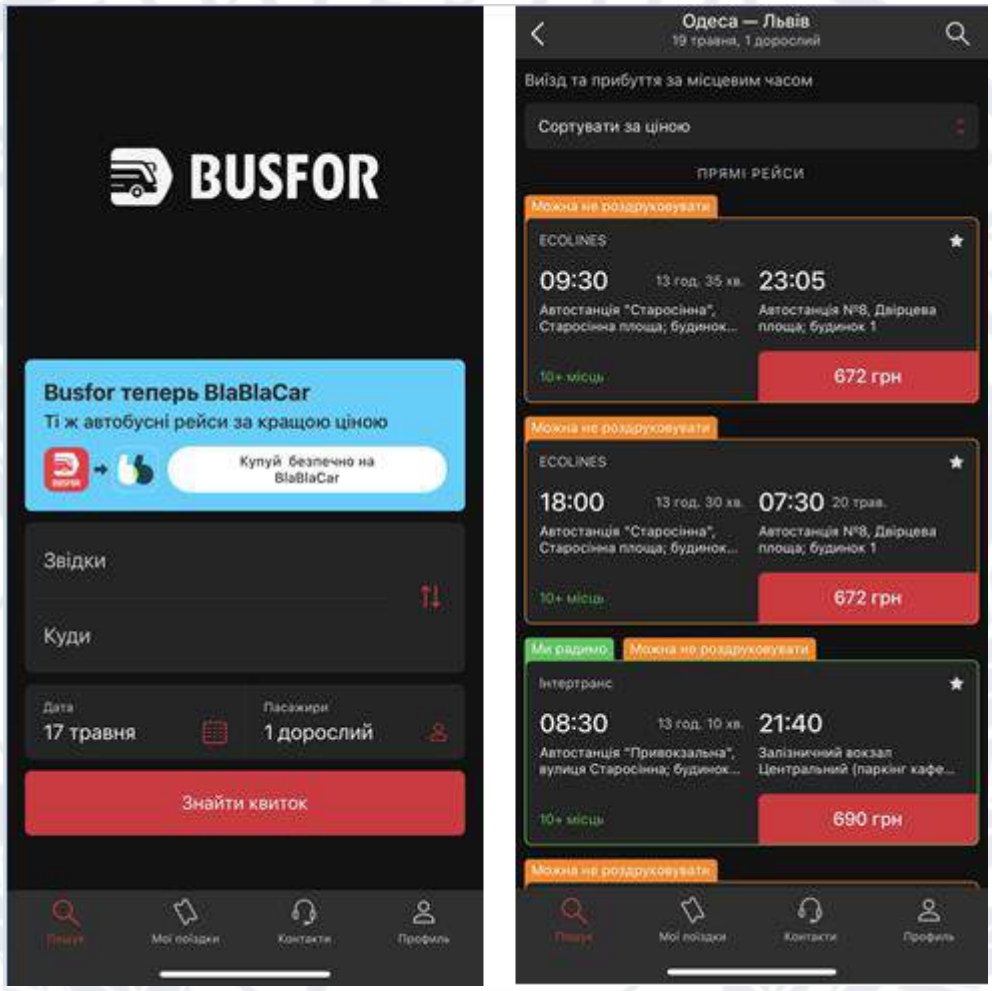

Рисунок 1.2 - Головна сторінка та результат пошуку Busfor

До переваг сервісу Busfor можна віднести великий вибір параметрів для сортування, зокрема: за ціною, за часом виїзду, за часом у дорозі, за часом прибуття та за рейтингом. Також клієнт може вибрати кількість пасажирів для яких потрібні квитки, що спрощує процес пошуку. Крім того, система гарантує надійність пропозицій, оскільки перевізникам необхідно пройти перевірки перед виставленням пропозицій у системі.

Проте серед недоліків можна відзначити відсутність різних видів транспорту, доступні лише квитки на автобуси. Також деякі квитки можуть бути

непридатними, і компанія часто не компенсує їх. Більшість квитків перепродаються іншими операторами, які не володіють автобусами.

Tickets.ua (рис.1.3) – український онлайн-оператор, що пропонує придбати електронні квитки на літак, потяг, автобус, трансфер, різноманітні події та концерти. Також сервіс надає можливість забронювати готель та страхові поліси в Україні та країнах Європи, а також взяти в оренду автомобіль. Компанія почала працювати ще в 2009 році, має таких світових партнерів, як Amadeus, Booking.com та TripAdvisor. Кожен день Tickets.ua продає понад 2000 квитків і має щоденну відвідуваність у 60 000 користувачів.

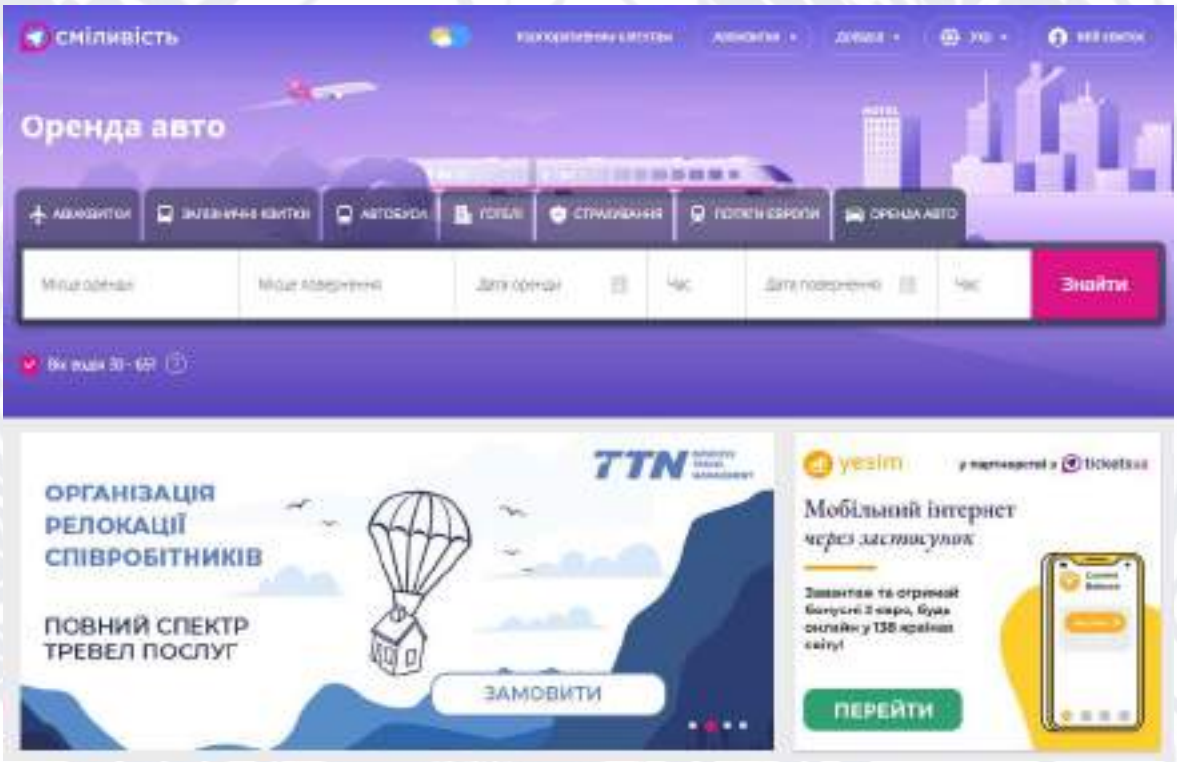

Рисунок 1.3 - Головна сторінка Tickets.ua

Безумовною перевагою сервісу Tickets.ua є великий вибір типів транспорту, крім того, доступна можливість забронювати готель та страхування. Проте серед недоліків можна відзначити перевантаженість головної сторінки та наявність великих рекламних банерів. Також сервіс доступний лише у веб-версії і не має власного мобільного додатку. Відсутній розширений пошук за додатковими параметрами та доступна досить мала кількість фільтрів.

Skyscanner.com.ua (рис. 1.4) - Інтернет-портал, що запущений у 2003 році, присвячений пошуку авіаквитків за низькою ціною. Через цей веб-сайт можна придбати авіаквитки в понад 60 країн. У 2011 році розробник запустив мобільний додаток. Крім того, за допомогою даного сайту клієнт може забронювати готель. Однак основним недоліком є те, що він шукає авіаквитки лише в літаку.

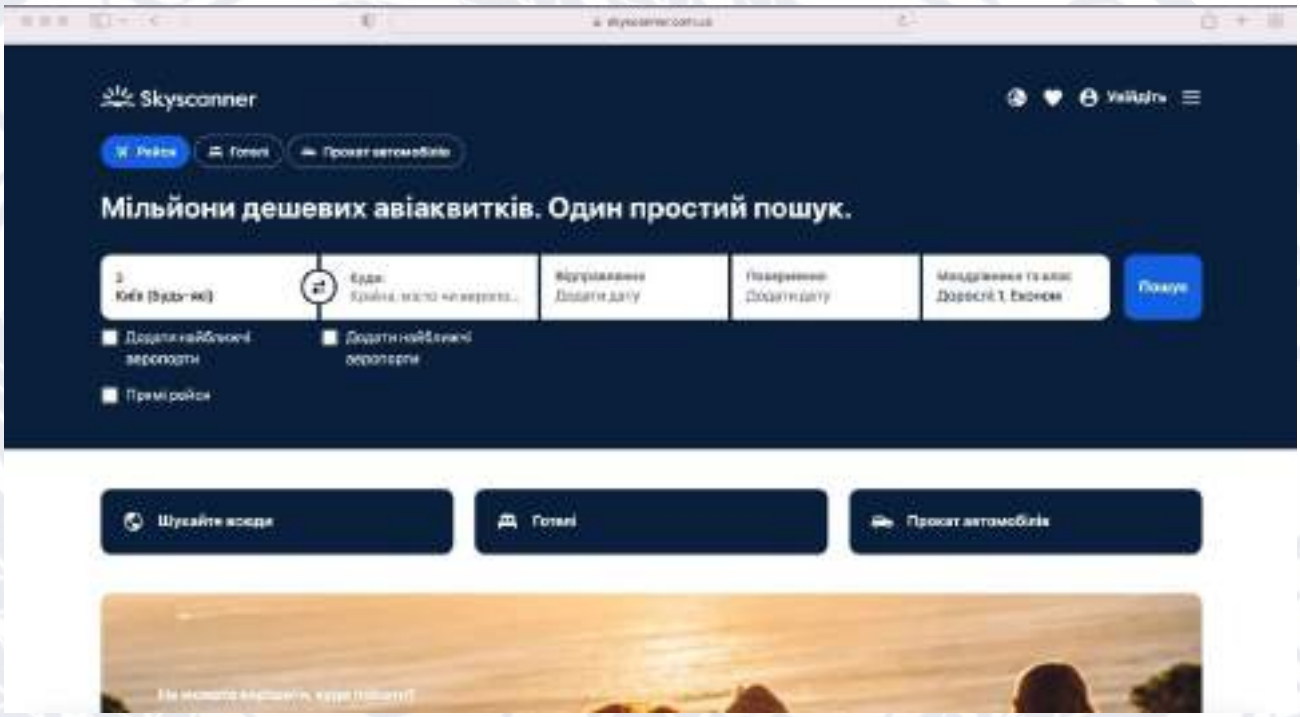

Рисунок 1.4 - Головна сторінка skyscanner.com.ua

Серед переваг сайту Skyscanner.com.ua можна відзначити велику кількість країн, в яких можна придбати квитки. Також бронювання готелю є досить корисною та зручною опцією, а наявність мобільного додатка розширює можливості користувачів. Проте, тут можна придбати квитки виключно на один тип транспорту - літак.

Omio, раніше відомий як GoEuro, є німецьким веб-сайтом для порівняння та бронювання подорожей онлайн. Він був заснований у 2012 році як GoEuro. Omio пропонує мандрівникам можливість визначити потрібні їм вимоги до транспорту, такі як потяги, літаки, автобуси тощо, використовуючи їхню просту у використанні платформу. Покриваючи 207 європейських аеропортів, понад десять тисяч центральних автовокзалів і понад двадцять тисяч залізничних

вокзалів, Omio надає можливість придбати квиток на такі види транспорту, як літак, автобус та потяг (рис.1.5, рис. 1.6).

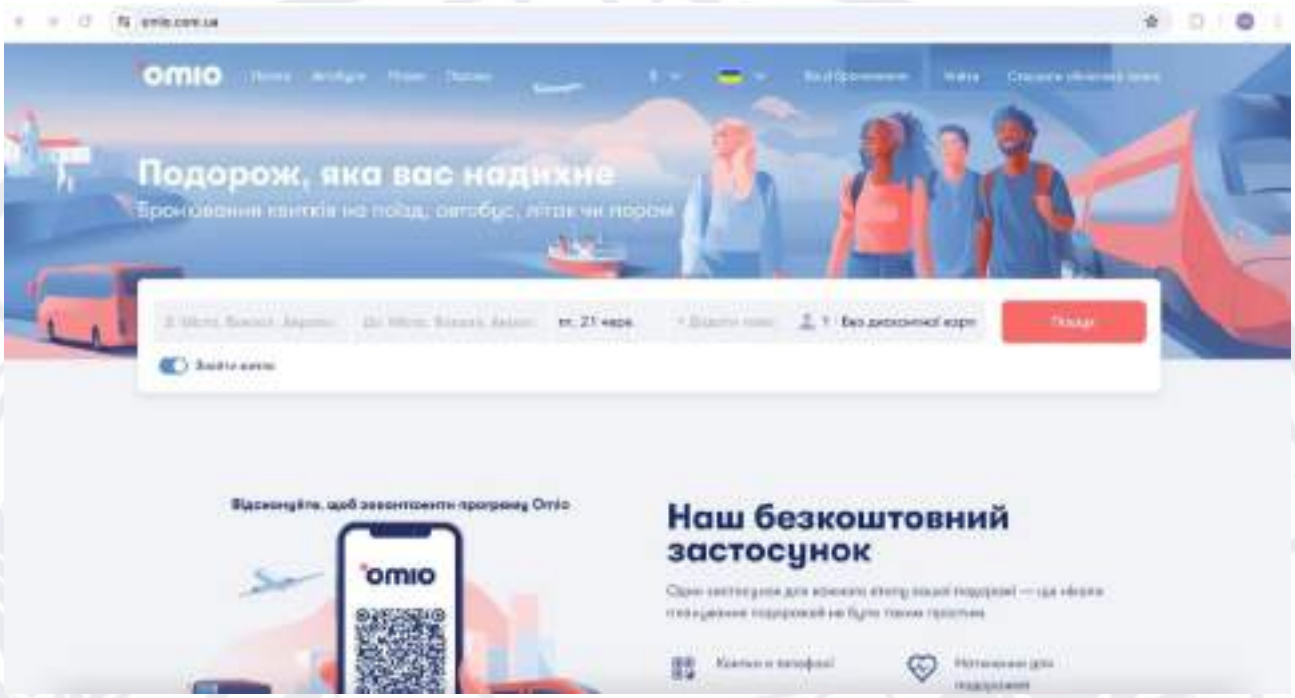

Рисунок 1.5 - Головна сторінка Omio

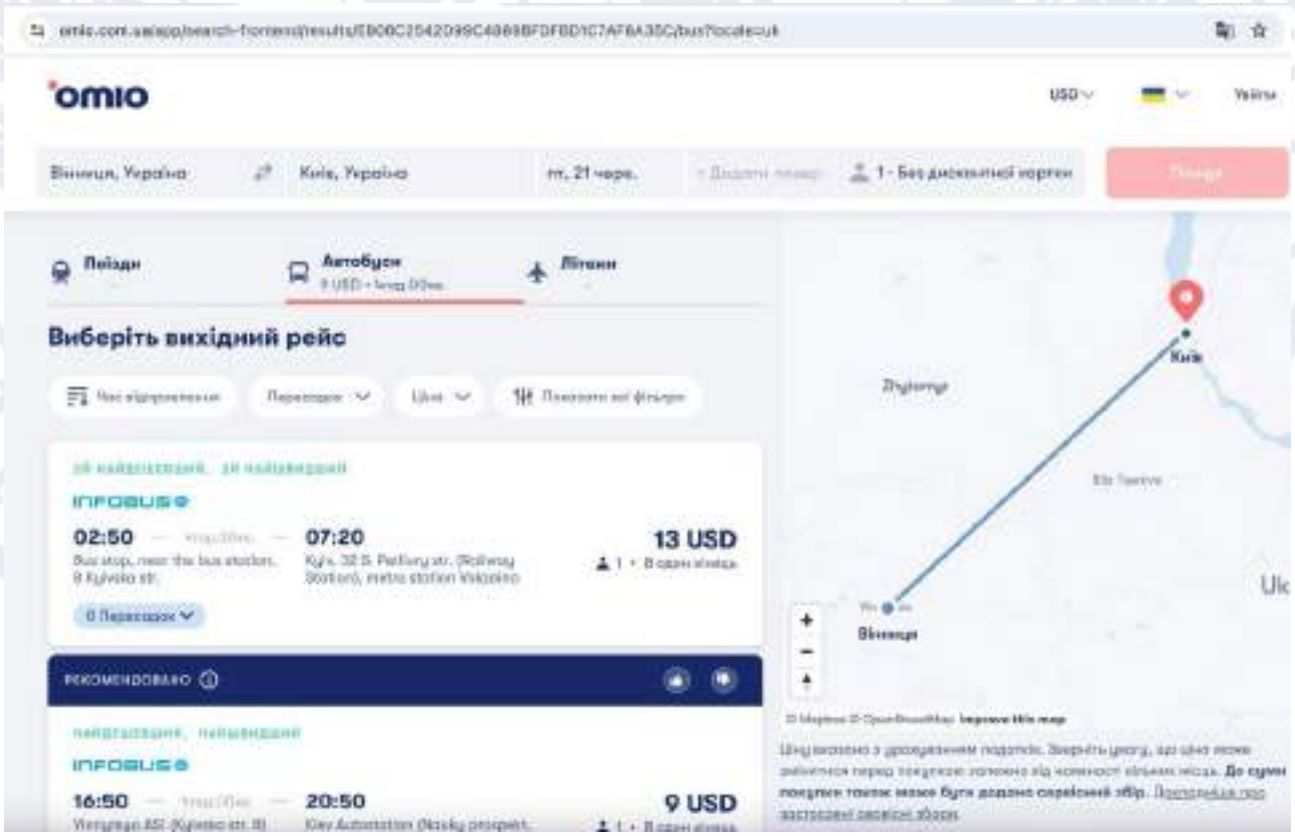

Рисунок 1.6 - Результат пошуку на Omio

До переваг платформи Omio можна віднести можливість вибору валюти оплати, що дозволяє користувачам придбати квитки в різні країни. Також наявність як веб-сайту, так і мобільного додатка розширює можливості користувачів. Проте серед недоліків можна виділити обмежену кількість фільтрів для пошуку та достатню кількість реклами на веб-сайті та в мобільному додатку.

1.4 Висновок до першого розділу

На сьогоднішній день сфера туризму є однією з провідних, отже досить актуальною. Адже більшість людей мають бажання подорожувати, завдяки цьому росте потреба у використанні спеціалізованих інформаційних систем для пошуку та аудиту квитків на транспорт. У даному розділі було проаналізовано найбільш відомі системи з пошуку квитків для подорожей та виявлено їх переваги і недоліки.

РОЗДІЛ 2
СИСТЕМНИЙ АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ

Системний аналіз - це процес вивчення системи з метою аналізу, моделювання та вибору логічної альтернативи. Проекти системного аналізу розпочинаються з проблем, можливостей та директив. Учасники включають системних аналітиків, спонсорів та користувачів. Процес розробки системи можна описати як життєвий цикл розробки системи, який включає завдання, прийоми та інструменти, що використовуються в методології. Існують три класифікації методологій: традиційні, інформаційні інженерні та об'єктно-орієнтовані. Інструменти CASE - це автоматизовані інструменти, що підтримують певні методології.

Проектування систем - це процес планування та проектування нової бізнес системи або внесення змін до існуючої системи за допомогою визначення та розробки її модулів або компонентів для задоволення конкретних вимог кінцевого користувача чи установи. Перед плануванням важливо ретельно вивчити стару систему та визначити оптимальне використання комп'ютерів для ефективної роботи. Проектування системи фокусується на досягненні мети системи. [14].

Системний аналіз і проектування (SAD) в основному зосереджені на:

- системи;
- процеси;
- технологія.

Система — це «упорядковане групування взаємозалежних компонентів, пов'язаних разом відповідно до плану для досягнення певної мети» [14].

Система повинна мати три основні обмеження:

- Система повинна мати деяку структуру і поведінку, які призначені для досягнення заздалегідь сформульованої мети;
- Між компонентами системи повинен, обов'язково, бути взаємозв'язок і взаємозалежність;

- Очевидно, що цілі організації повинні мати вищий пріоритет на відмінність від цілей її підсистем.

Система має наступні властивості:

- Організація визначає структуру і порядок. Завдяки розташуванню компонентів можна і швидше досягти заздалегідь визначених цілей;
- Взаємодія – визначає як компоненти працюють між собою;
- Взаємозалежність означає, як елементи системи залежать один від одного. Для належного функціонування елементи системи координуються та з'єднуються згідно з визначеним планом, де, відповідно, вихід однієї підсистеми є вхідним для іншої;
- Інтеграція показує як елементи системи поєднані між собою. Це означає, що частини системи працюють разом, навіть якщо кожна з них виконує окрему функцію;
- Мета системи повинна бути основною. Вона може бути реальною або заявленою. Нерідкі випадки, коли організація ставить мету і діє для досягнення іншої. Користувачам потрібно показувати основну мету комп'ютерної програми на початку аналізу для успішного проектування та реалізації.

2.1 Дерево цілей

Дерево цілей – це інструмент, який використовується для окреслення та візуалізації етапів, необхідних для досягнення складної довгострокової мети [12]. Крім того, дерево цілей включає 3 основні компоненти:

- Складна довгострокова мета;
- Критичні фактори успіху (CSF), які необхідні для досягнення мети;
- Необхідні умови, необхідні для виконання CSF.

Дерево цілей спонукає приймати правильні рішення про те, над чим працювати зараз. І що важливо сьогодні, наступного тижня і наступного місяця. У переслідуванні цілей, які не можуть бути повністю досягнуті протягом кількох років.

Дерево цілей для розробленої системи на рис. 2.1 включає головну ціль *Створення інформаційної системи пошуку квитків*. Для досягнення цієї основної мети необхідно виконати цілі другого рівня, такі як *Зберігання даних*, *Відображення даних* та *Пошук даних*.

Ціль *Зберігання даних* передбачає розробку функціоналу для зберігання інформації про квитки та інформації про користувача.

Відображення даних включає в себе відображення збережених даних, таких як інформація про квитки та інформація про користувача в їх профілі.

Ціль *Пошук даних* передбачає виконання пошуку квитків за вхідними параметрами, щоб користувач міг знайти потрібний йому квиток.

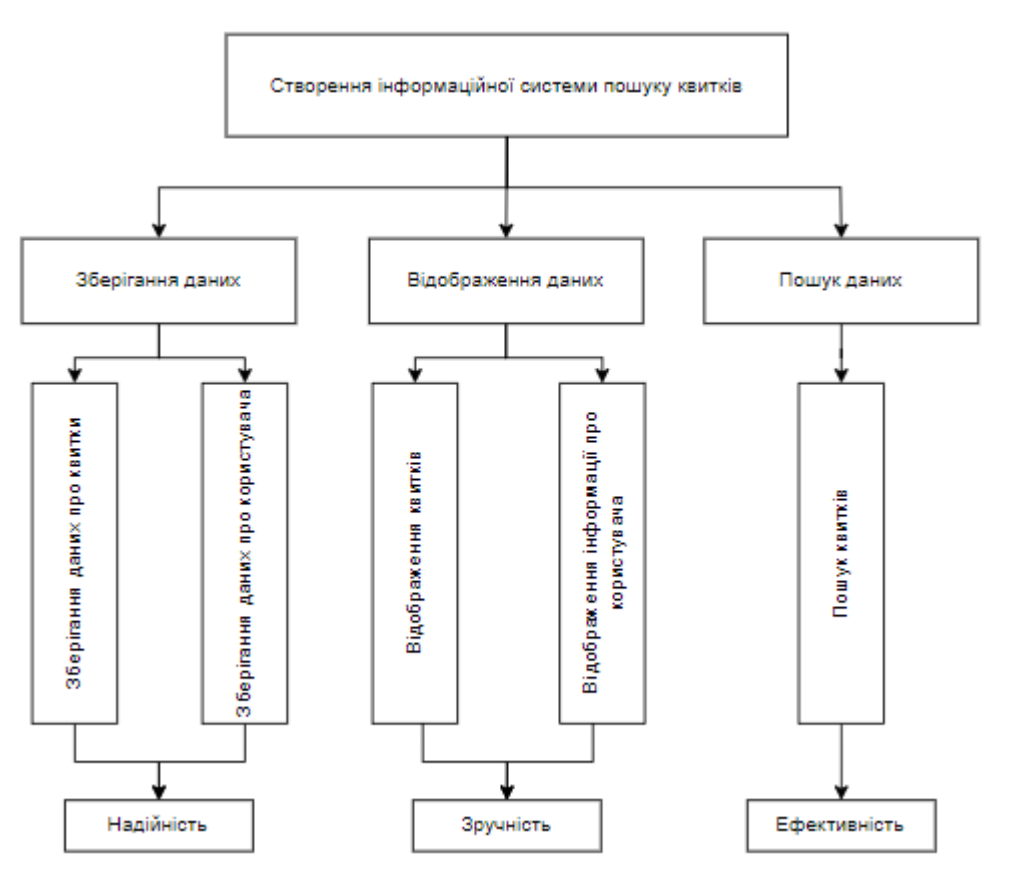

Рисунок 2.1 - Дерево цілей

Далі потрібно визначити тип інформаційної системи. На розгляд було взяти такі типи систем як інформаційно-довідкова система, система підтримки прийняття рішень, інформаційно-пошукова система та інтелектуальна інформаційна система. Для цього потрібно скласти матрицю попарних порівнянь

визначених критеріїв – табл. 2.1 та визначити оцінки переваг одних критеріїв над іншими.

Таблиця 2.1 Матриця попарних порівнянь критеріїв

№	Критерій	1	2	3	Вектор пріоритетів
1	Надійність	1	2	1/3	0,249
2	Зручність	1/2	1	1/3	0,157
3	Ефективність	3	3	1	0,593

Після створення матриці попарних порівнянь критеріїв потрібно створити матриці попарних порівнянь критеріїв до типів систем, результати у таблицях 2.2 – 2.4.

Таблиця 2.2 - Матриця попарних порівнянь для критерію *Надійність*

№	Критерій	1	2	3	4	Вектор пріоритетів
1	Інформаційно-довідкова	1	2	3	2	0,423
2	Інформаційно-пошукова	1/2	1	2	2	0,270
3	Система підтримки прийняття рішень	1/3	1/2	1	1	0,145
4	Інтелектуальна інформаційна система	1/2	1/2	1	1	0,160

Таблиця 2.3 – Матриця попарних порівнянь для критерію *Зручність*

№	Критерій	1	2	3	4	Вектор пріоритетів
1	Інформаційно-довідкова	1	1/2	2	2	0,277
2	Інформаційно-пошукова	2	1	2	2	0,392
3	Система підтримки прийняття рішень	1/2	1/2	1	1	0,165
4	Інтелектуальна інформаційна система	1/2	1/2	1	1	0,165

Таблиця 2.4 – Матриця попарних порівнянь для критерію *Ефективність*

№	Критерій	1	2	3	4	Вектор пріоритетів
1	Інформаційно-довідкова	1	1/2	1/2	1/3	0,119
2	Інформаційно-пошукова	2	1	3	3	0,457
3	Система підтримки прийняття рішень	2	1/3	1	1	0,2
4	Інтелектуальна інформаційна система	3	1/3	1	1	0,222

Після цього було проведено ієрархічний синтез, шляхом перемноження векторів пріоритетів критеріїв та систем. Результати показані у таблиці 2.5.

Таблиця 2.5 - Ієрархічний синтез критеріїв та типів систем

Критерій / Система	Надійність	Зручність	Ефективність	Векторний пріоритет
Інформаційно-довідкова	0,105416029	0,019605466	0,150463441	0,275484937
Інформаційно-пошукова	0,067354845	0,067932317	0,219201109	0,354488271
Система підтримки прийняття рішень	0,036179695	0,032972333	0,140057037	0,209209065
Інтелектуальна інформаційна система	0,04004943	0,036489884	0,083278412	0,159817727

Після того як було використано метод ієрархічного аналізу було виявлено, що для того щоб реалізувати інформаційну систему пошуку квитків потрібно обрати інформаційно-пошукову систему, яка буде шукати квитки за вхідними даними користувача.

2.2 Конкретизація роботи системи

IDEF0 є методологією функціонального моделювання для аналізу, розробки, реінжинірингу та інтеграції інформаційних систем.

IDEF0 дозволяє:

- описувати любі системи, а не лише інформаційні;
- створювати опис системи та її оточення (зовнішнього) до фінального визначення вимог до системи.

Іншими словами, за допомогою методології IDEF0 можна систематично розробляти та аналізувати систему, включаючи випадки, коли важко уявити її

реалізацію. Цей підхід може бути корисним на початкових етапах розробки різноманітних систем і дозволяє аналізувати функції існуючих систем для подальшого їх вдосконалення.

Основу методології IDEF0 становить графічна мова опису процесів. Модель у форматі IDEF0 складається з ієрархічно впорядкованих та взаємопов'язаних діаграм. Кожна діаграма представляє собою окремий опис системи і знаходиться на окремому аркуші.

Початково була розроблена контекстна діаграма згідно з методологією IDEF0, яка ілюструє систему узагальнено. Основним процесом на цій діаграмі є *Пошук квитків*, з вхідними даними, такими як "ПІБ користувача", *Логін, Пароль* та *Інформація про квиток*. Елементами контролю є "Документація Room Database" та "Документація SQL Search", а механізми включають "SharedPreferences", "Room Database" та "SQL Search". Результатом роботи системи, керуючись вказаними механізмами та елементами контролю, є "Шуканий квиток" та "Інформація про користувача в профілі".

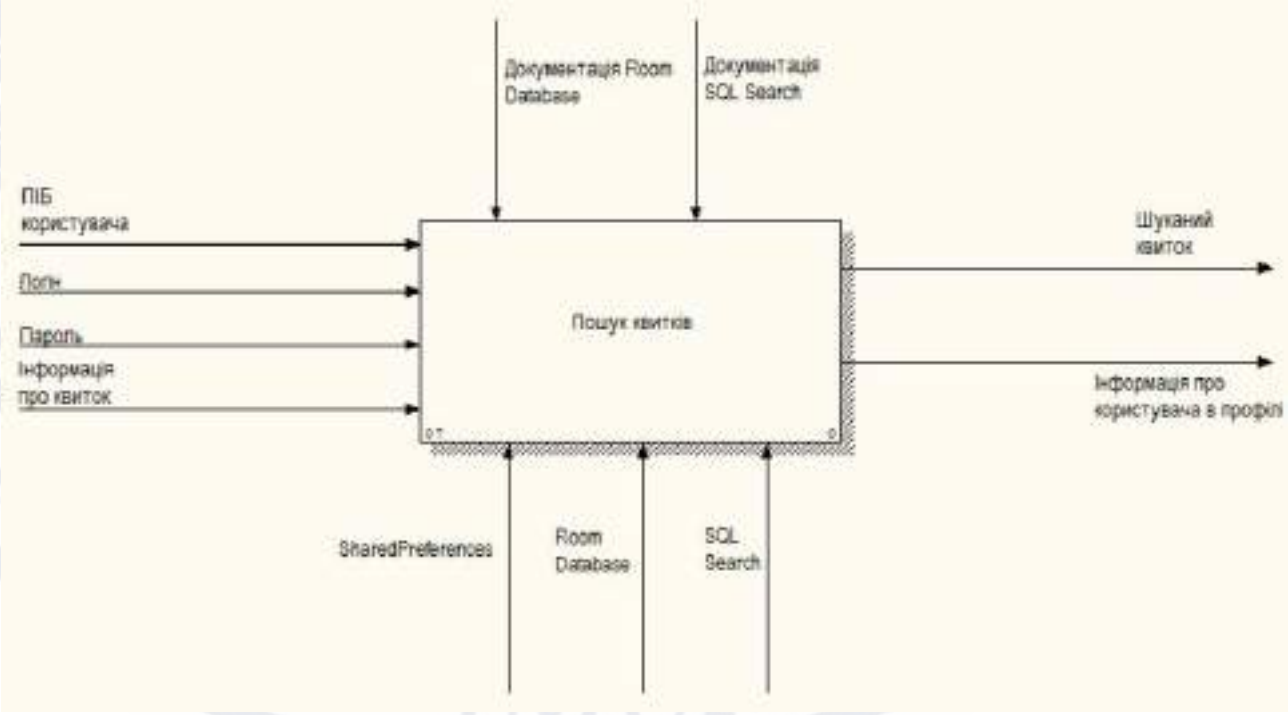

Рисунок 2.2 - IDEF діаграма

Далі, контекстну діаграму потрібно розкласти, щоб забезпечити більш детальне зображення функціональних можливостей системи, в результаті чого утворюється розбита діаграма першого рівня – рис. 2.3 на якому описано наступні процеси: *Реєстрація*, *Створити профіль* та *Пошук квитка*. Щоб зареєструвати користувача потрібні лише дані про нього такі як: *«ПІБ користувача»*, *Логін* та *Пароль*.

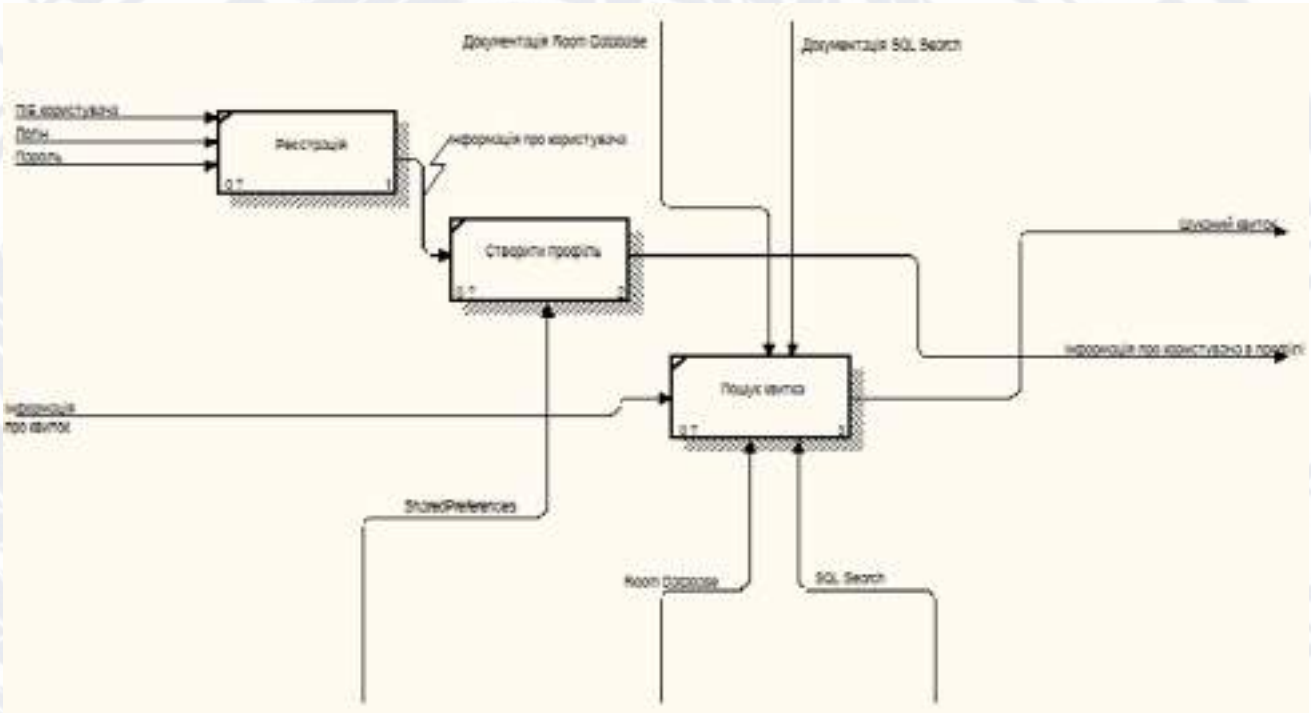

Рисунок 2.3 - Діаграма декомпозиції першого рівня

Наступним кроком є виконання декомпозиції другого рівня для кожного процесу, щоб отримати діаграми декомпозиції другого рівня. Перший процес, який потрібно детально розглянути, це процес *Реєстрація* - рис. 2.4. Для того щоб користувач пройшов реєстрацію необхідні такі дані як *ПІБ користувача*, *«Логін»* та *«Пароль»* та провести перевірку логін та пароль.

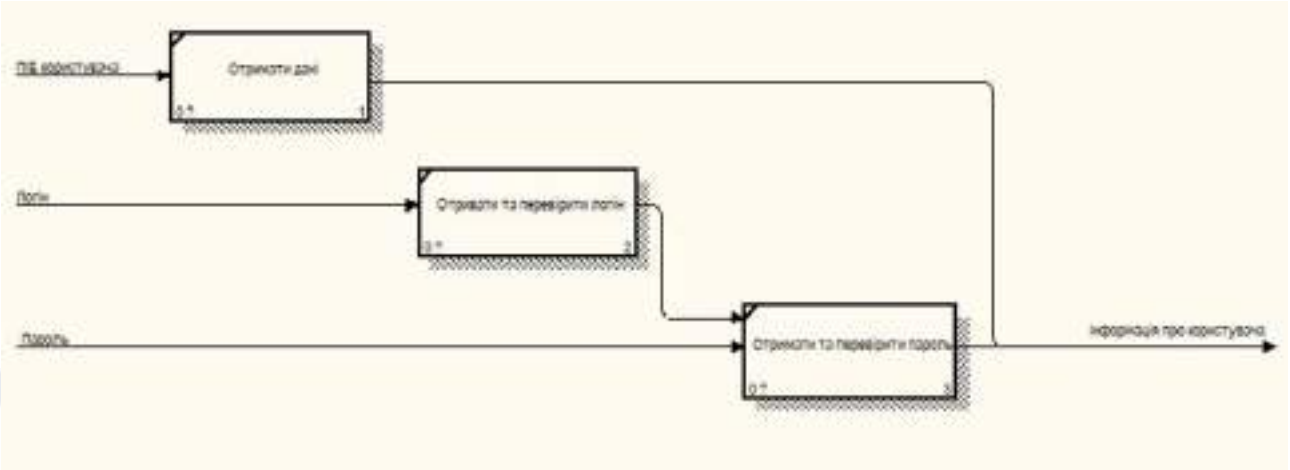

Рисунок 2.4 - Діаграма декомпозиції другого рівня процесу *Реєстрація*

У процесі *Створення профілю* (рис. 2.5) перший крок полягає в Отриманні перевірених даних, що стосуються інформації, введеної користувачем на попередньому етапі, такі як *ПІБ користувача*, *Логін* та *Пароль*. Після цього, інформація повинна бути "Збережена", щоб користувач міг переглянути або відредагувати її у своєму профілі. Для збереження цих даних застосовується відповідний механізм SharedPreferences.

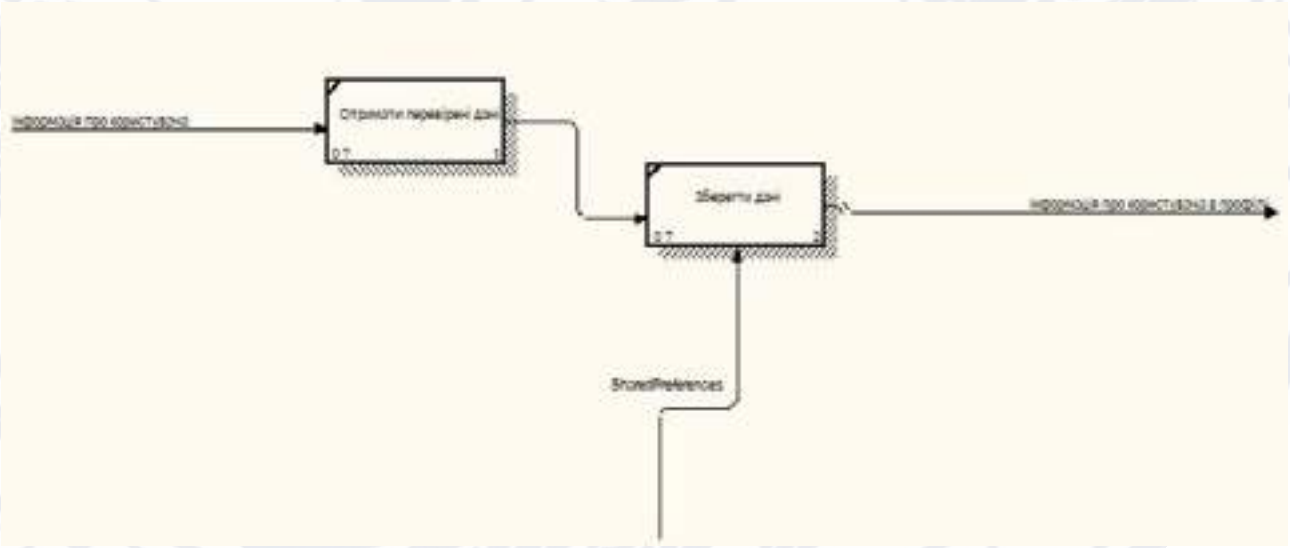

Рисунок 2.5 - Декомпозиція другого рівня процесу *Створити профіль*

Наступним процесом є Пошук квитка - рис. 2.6. Для реалізації цього процесу необхідно спочатку отримати вхідну інформацію про квиток, який шукає користувач. Після чого, за допомогою відповідних механізмів *Room Database* та *SQL Search* буде здійснено «*Пошук певних квитків*», при цьому

застосовуються критерії *Документація Room Database* та «*Документація SQL Search*». На виході процесу буде отримано *Шуканий квиток*.

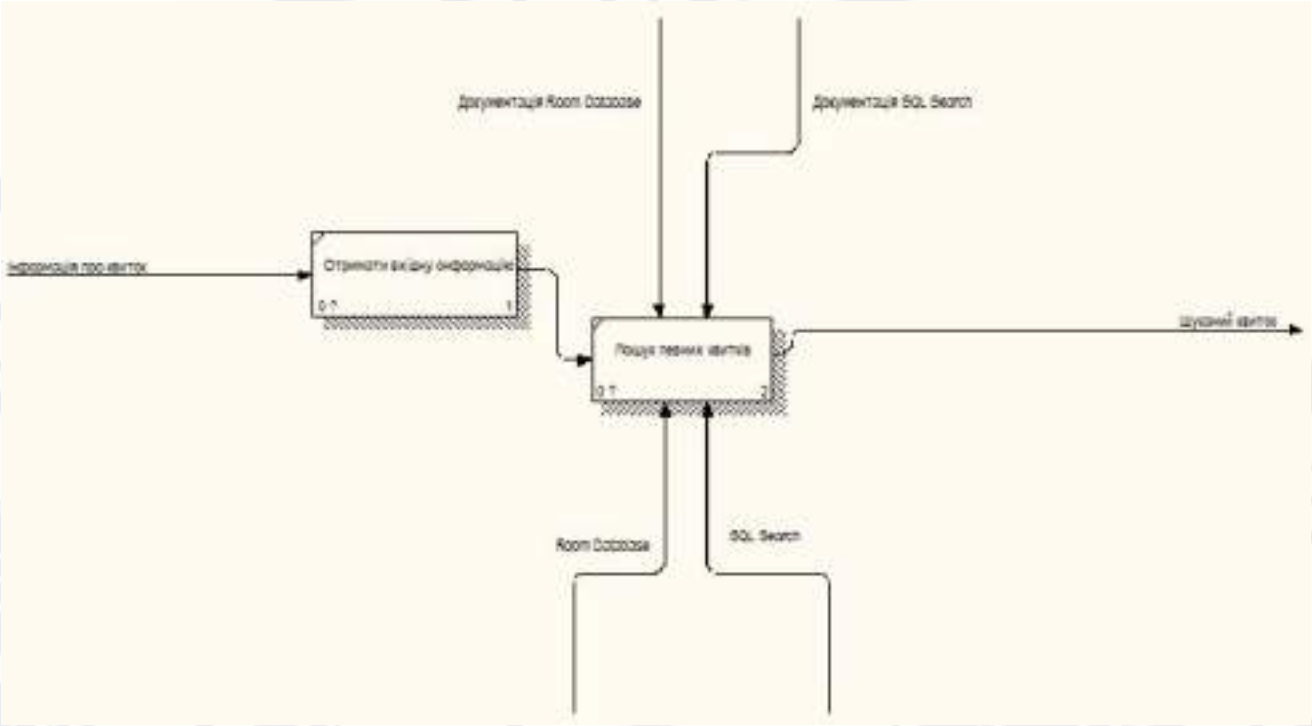

Рисунок 2.6 - Декомпозиція другого рівня процесу *Пошук квитка*

У процесі *"Пошуку конкретних квитків"* (рис. 2.7), на третьому рівні деталізації, спочатку відбувається Парсинг квитків на основі введеної інформації про квиток. Цей крок здійснюється з використанням механізму *Room Database* та з врахуванням елементів контролю, таких як *Документація Room Database*.

Після цього проводиться Пошук в базі даних квитків за допомогою механізмів *"Room Database"* та *SQL Search*, враховуючи відповідні критерії з документації обох механізмів.

Останнім етапом цього процесу є "Представлення результатів пошуку" користувачу, який проводив пошук квитків.

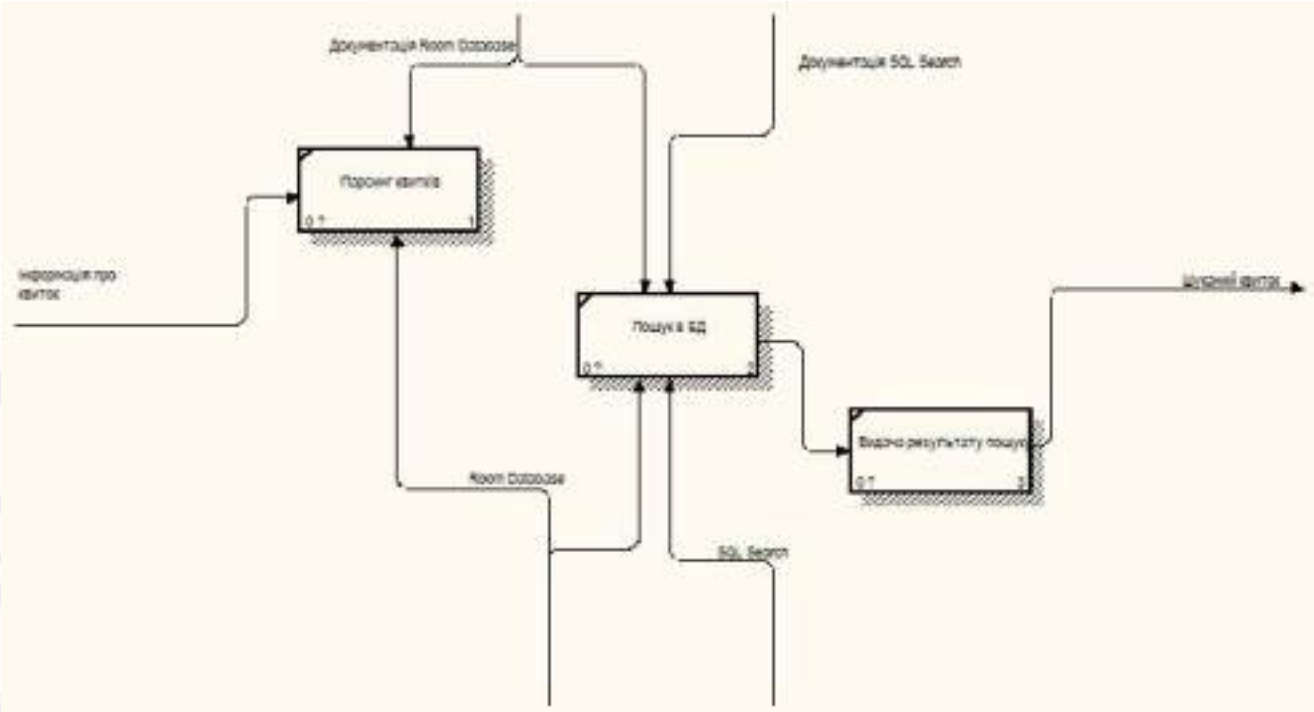

Рисунок 2.7 - Декомпозиція третього рівня процесу *Пошук певних квитків*

2.3 Побудова ієрархії процесів (функцій, задач)

Діаграма ієрархії процесів (PHD), також відома як діаграма функціональної декомпозиції (FDD), є інструментом, що допомагає визначити структуру системи шляхом розкриття функцій високого рівня незалежно від їх послідовності виконання або умов. У PHD процеси декомпозуються до належного рівня деталізації, щоб з'ясувати необхідні функції для вашої системи. Ця декомпозиція процесів відома як ієрархія процесів, де всі процеси пов'язані між собою ланками декомпозиції. PHD зазвичай використовується на етапі аналізу проекту.

Бізнес-аналітики та менеджери використовують його для:

- визначення всіх процесів, що виконуються в рамках бізнес-функції;
- зосередження на розпізнаванні та перерахуванні процесів; на цьому кроці визначаються лише імена процесів;
- розкладання вже ідентифікованих процесів на підпроцеси, поки не буде досягнутий відповідний рівень;

- відображення в одному поданні всієї ієрархії вже описаного процесу або будь-якого декомпонованого підпроцесу.

Ієрархічна діаграма процесів показує всі завдання, необхідні для роботи системи. На вершині ієрархії розташовані основні завдання, які поділяються на більш детальні завдання, що утворюють рівні ієрархії [13].

Головною задачею і вершиною ієрархії є «Пошук квитків». Для її досягнення необхідно щоб були виконані всі задачі, які знаходяться нижче неї такі як: «Реєстрація», «Створити профіль» та «Пошук квитка».

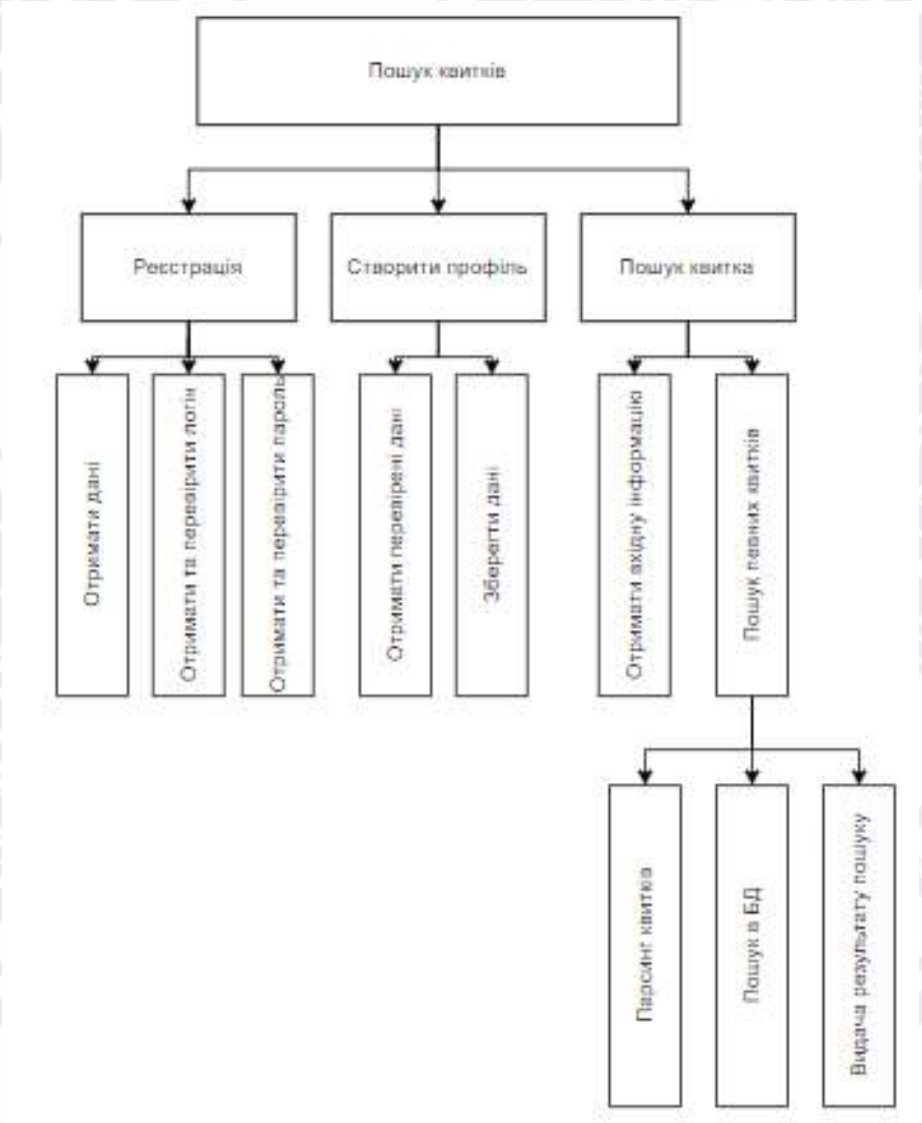

Рисунок 2.8 - Ієрархія процесів

2.4 Висновок до другого розділу

В даному розділі було проведено комплексний системний аналіз досліджуваної системи - *Інформаційної системи пошуку квитків*. Основні концепції системного аналізу були визначені, а також пояснено, для чого використовується цей підхід.

Було створено дерево цілей для вивченої інформаційної системи, де у вершині розташувалася головна ціль - Створення інформаційної системи пошуку квитків. Ця ціль була поділена на менші цілі, які послужили критеріями: Надійність, Зручність та Ефективність. Застосувавши метод аналізу ієрархій, було визначено, що система відповідає типу інформаційно-пошукової системи.

Крім того, у даному розділі була розроблена контекстна діаграма, використовуючи нотацію IDEF0, яка деталізувала основні процеси та функції системи. Декомпозиція цієї діаграми до третього рівня дозволила докладніше проаналізувати структуру системи. Надалі була побудована діаграма ієрархії процесів, яка ілюструвала основні процеси та їх розташування у структурі системи.

РОЗДІЛ 3
ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ДОДАТКУ

3.1 Вибір та обґрунтування засобів розв'язання задачі

Наразі доступна велика кількість різних мов програмування. Однак, одним з ключових етапів розробки є вибір мови програмування, платформи та технологій, оскільки це має великий вплив на складність інформаційної системи та витрати на її впровадження. Вам потрібно обрати програмні інструменти, які дозволять працювати з базами даних та надати достатній функціонал для створення інтерфейсу користувача.

3.1.1 Мова програмування Swift

Swift - це мова програмування загального призначення з відкритим вихідним кодом, створена компанією Apple, переважно для розробників iOS і macOS. Вона взаємодіє з фреймворками Cocoa і Cocoa Touch і має сумісність з основною кодовою базою Apple Objective-C. Основною метою розробки Swift було створення мови, яка була б легше читати і має більшу стійкість до помилок програміста, ніж попередник Objective-C. Програми на Swift компілюються за допомогою LLVM, який входить до складу інтегрованого середовища розробки Xcode 6 і вище. Мова підтримує середовище виконання Objective-C, що дозволяє використовувати обидві мови (і C) в одній програмі.

У багатьох відношеннях Swift перевершує свого попередника. Вона поєднує продуктивність і ефективність компільованих мов з простотою та інтерактивністю популярних мов сценаріїв [26].

Плюси використання Swift:

- Швидкий процес розвитку: Чиста й виразна мова зі спрощеним синтаксисом та граматикою Swift простіше читається й пишиться. Він дуже

стислий, і для виконання того ж самого завдання необхідно менше коду, ніж Objective-C.

- Автоматичний підрахунок посилань виконує всю роботу, відстежуючи та керуючи використанням пам’яті програми, тому розробникам не потрібно витрачати час і зусилля на це вручну;
- Swift було створено з фокусом на покращенні продуктивності, швидкості розробки і безпеці. Його розробка спрямована на перевершення попередньої мови програмування. Однією з сильних сторін Swift є його система безпеки. Вона включає потужний механізм введення та обробки помилок, що дозволяє уникнути збоїв коду та помилок у виробництві. Це допомагає знизити час і зусилля для виправлення помилок, і суттєво зменшити ризики розгортання низькоякісного коду, оскільки Swift має значно коротший цикл зворотнього зв'язку, що в роботі дозволяє миттєво виявляти та виправляти знайдені помилки;
- Зменшення обсягу пам'яті: стандартні бібліотеки Swift інтегровані в кожен випуск macOS, iOS, tvOS і watchOS. Це означає, що будь-який додаток, створений на цих платформах, буде меншим, оскільки ці бібліотеки включені. Завдяки цьому прогресу також був випущений стабільний бінарний інтерфейс програми (ABI).

Мінуси мови програмування Swift:

- Погана сумісність із сторонніми інструментами - Багато в чому через часті оновлення, про що писалось вище, часто складно знайти потрібні правильні інструменти, щоб допомогти з певними завданнями. Більше того, офіційна Apple IDE, XCode, відстають в питанні інструментів і підтримки Swift;

Swift підтримує всі платформи Apple та Linux і Windows, проте його кросплатформна підтримка є неповною. Хоча мова була спочатку призначена для розробки на iOS, вона також може бути використана на інших платформах. Проте, вона найкраще працює для розробки нативних додатків для iOS. Swift єдине ще не готове програмне забезпечення; Apple продовжує його розвивати, і, швидше за все, буде додавати нові функції в найближчий час.

Інтерфейс налагодження Xcode включає інтерактивну версію мови програмування Swift під назвою Interactive Playground.

Це означає, що розробники можуть використовувати синтаксис Swift для оцінки та взаємодії з існуючим додатком, написання нового коду, щоб побачити, як він працює в середовищі, і навіть створювати нові алгоритми.

3.1.2 Мова програмування Java

Java - це мова програмування, яка вирішує різноманітні завдання та відзначається своєю потужністю та універсальністю. Однією з особливостей цієї мови є її незалежність від платформ: для всіх операційних систем є віртуальні Java-машини, які виконують будь-який Java-додаток однаково. Хоча машини для різних систем можуть мати різні можливості, наприклад, для desktop, серверних, мобільних або вбудованих систем, основні відмінності стосуються інтерфейсу користувача та принципів роботи з пам'яттю та процесами. Синтаксис Java базується на синтаксисі мови C++, але має значно більший функціонал [22].

Переваги Java:

- Універсальність коду для всіх платформ;
- Ефективна реалізація Java-машин для пристроїв, де критично важлива швидкість обробки, таких як мобільні та вбудовані пристрої;
- Набір бібліотек, призначених для роботи на конкретних платформах;
- Комплект вбудованих інструментів, що значно полегшує процес розробки програмного забезпечення.

Недоліки Java:

- Java працює повільно і має низьку продуктивність, споживає пам'ять і значно повільніше, ніж рідні мови, такі як C або C++;
- Java не забезпечує резервного копіювання, в основному працює на сховищі, а не зосереджується на резервному копіюванні даних. Це серйозний недолік, через який він втрачає інтерес і рейтинги серед користувачів;

- Код Java є багатослівним, що ускладнює його читання та розуміння. Це може знизити читабельність коду. Java зосереджується на керованості, але в той же час вона повинна мати складний код та довгі пояснення для кожної речі [21].

Компілятор Java генерує архітектурно-нейтральний формат об’єктного файлу, що дозволяє виконувати скомпільований код на різних процесорах, якщо доступна система середовища виконання Java.

Багатопотокова функція Java дозволяє писати програми, які здатні виконувати кілька завдань одночасно. Це дає розробникам змогу створювати інтерактивні програми, що працюють плавно, без перерв.

Java розглядається як більш динамічна мова програмування порівняно з C або C++, оскільки вона спроектована для адаптації до постійно змінюючого середовища. Програми на Java можуть носити значну кількість інформації під час виконання, що можна використовувати для перевірки та контролю доступу до об’єктів.

Незважаючи на всі перелічені обмеження, Java є однією з найбільш використовуваних мов у сфері програмного забезпечення завдяки своїй платформенній незалежності, безпеці та можливостям обслуговування.

3.1.3 Мова програмування Kotlin

Kotlin - це мова програмування статичного типу з відкритим вихідним кодом, яка підтримує як об'єктно-орієнтоване, так і функціональне програмування. Вона надає схожий синтаксис та концепції з інших мов, таких як C#, Java та Scala, серед інших.

Kotlin стала популярною мовою програмування для розробки додатків для Android. В порівнянні з Java, Kotlin відрізняється більшою чіткістю, оскільки вона використовує агресивний висновок для визначення виразів і значень. У Java часто потрібно вказувати кілька декларацій типів.

Переваги Kotlin:

- Kotlin може допомогти скоротити час, необхідний для розробки програми для Android. Програми Kotlin менші та швидше компілюються, ніж код Java, а це означає, що ви витрачатимете менше часу на очікування внесення змін у код [19];
- Коли у вашій програмі виникають помилки, вони зазвичай відображаються в трасі стека. Повідомлення про помилки Kotlin є більш корисними, ніж у Java, що полегшує пошук і виправлення помилок;
- Kotlin — дуже лаконічна мова. Це може допомогти вам писати менше коду, полегшуючи читання та обслуговування коду. Крім того, Kotlin має багато функцій, яких немає у Java, наприклад, нульовий захист і класи даних;
- Kotlin є більш надійною мовою програмування, ніж Java. Він має нульовий захист, який запобігає збою вашої програми під час спроби отримати доступ до нульового об’єкта. Kotlin також має класи даних, які допомагають уникнути шаблонного коду [19].

Недоліки Kotlin:

- Хоча Kotlin і Java мають схожість, вони є двома різними мовами. Розробникам знадобиться деякий час на вивчення Kotlin, щоб швидко перейти від однієї мови до іншої.
- У деяких випадках Kotlin швидший за Java - в основному при виконанні інкрементальних збірок. Але все ж, що Java є переможцем, коли справа доходить до чистих збірок [18].

3.1.4 Мова програмування Rust

Rust - це мова програмування зі статичним типом, спроектована для забезпечення продуктивності та безпеки, зокрема безпечного паралельного керування та управління пам'яттю. Її синтаксис схожий на синтаксис C++. Початково розроблена Mozilla Research, Rust розв’язує питання та проблеми, з якими раніше розробники C/C++ стикалися протягом тривалого часу: в більшості

це паралельне програмування та помилки пам'яті, що робить його головною перевагою.

У Rust існують два режими написання коду: Safe Rust і Unsafe Rust. Safe Rust накладає додаткові обмеження на програміста, забезпечуючи належну роботу коду, тоді як Unsafe Rust надає більше автономії, але може призвести до виникнення проблем у коді. Небезпечний режим Rust відкриває більше можливостей, проте вимагає від програміста особливої обережності, щоб переконатися, що код дійсно безпечний. Для цього можна використовувати абстракції вищого рівня, які гарантують безпечне використання коду. Написання небезпечного коду потребує особливої обережності у будь-якій мові програмування, включно з Rust. Це дозволяє уникнути невизначеної поведінки та зменшити ризик вразливостей, які можуть виникнути через проблему небезпеки пам'яті.

Дворежимна модель Rust є однією з найбільших переваг. В C++ ми ніколи не зможемо бути впевненими, що маємо безпечний код, адже проблеми можуть виявитися тільки у майбутньому під час виконання програми або виникнення порушення безпеки.

Rust також суттєво спрощує написання паралельних програм, що запобігає перегонам даних при компіляції. Перегони даних відбуваються, коли різні потоки намагаються отримати доступ до одного фрагмента пам'яті одночасно, причому принаймні один з них намагається внести зміни, і відсутність синхронізації призводить до непередбачуваного результату. Доступ до пам'яті без синхронізації може призвести до невизначеності результатів.

Як мова системного програмування, Rust дозволяє контролювати низькорівневі деталі, що відкриває можливість вибору між зберіганням даних в стеку (для статичного розподілу пам'яті) або в купі (для динамічного виділення пам'яті). Це дозволяє більш ефективне використання пам'яті.

Rust є низькорівневою мовою програмування, що має прямий доступ до апаратного забезпечення та пам'яті, завдяки чому ми отримуємо чудове рішення для розробки та вповадження вбудованих систем та розробки без використання.

Можна використовувати Rust для написання програм мікроконтролера та операційних систем. Низькорівнева природа Rust робить його відмінним вибором для таких завдань.

Деякі операційні системи, такі як Redox, intermezzOS, QuiltOS, Rux і Tock, написані на Rust. Крім того, Mozilla, де мова була спочатку розроблена, застосовує її у своїх браузерних механізмах.

Rust демонструє вражаючу швидкість, що робить його незамінним для обчислювальної біології та машинного навчання, в яких необхідна швидка обробка великих обсягів даних. Ця властивість робить його привабливим для таких областей.

Rust є відносно новою технологією, тому деякі потрібні бібліотеки можуть бути ще недоступними. Крім того, для розробників, що не працюють з мовами, в яких помилки в коді виявляються під час компіляції, отримання багато повідомлень про помилки може бути дратівливим. Це може призвести до повільнішого темпу розробки коду порівняно з більш популярними мовами, такими як Python [24].

Основні переваги Rust:

- Висока продуктивність при умові забезпечення безпеки пам'яті;
- Він підтримує паралельне програмування;
- Забезпечена зворотна сумісність і стабільність;
- У репозиторії crates.io зростає кількість пакетів Rust.

3.2 Технічні характеристики обраних програмних засобів розроблення

В попередньому аналізі мов програмування для реалізації системи пошуку квитків були виявлені переваги і недоліки кожної мови. Розробка мобільного додатку була обрана як найкращий варіант через його зручність для користувачів і перспективність.

Згідно з проведеними дослідженнями, у 2020 році понад половина трафіку веб-сайтів створювалась з мобільних пристроїв, що свідчить про велику

популярність мобільних пристроїв серед користувачів. Крім того, кількість мобільних користувачів у світі постійно зростає.

Враховуючи результати проведеного дослідження, мова програмування Kotlin була обрана як оптимальний вибір для реалізації системи. Однією з особливостей Kotlin є можливість використання функцій розширення, які допомагають розробникам у різних ситуаціях. На відміну від Java, де функції можуть бути використані тільки у звичайних класах, Kotlin не має цього обмеження, що полегшує розробку додатку.

У Kotlin, розробники можуть використовувати функції як розширення, що робить процес додавання нових функцій плавним. У Java, збірка об'єктів проходить кілька етапів, що може впливати на продуктивність програми. Особливо коли йдеться про обробку колекційних об'єктів, які є важливою частиною розробки програм на Java. У Kotlin, який є сучасною мовою програмування, передбачено ефективну обробку незмінних колекцій. Багаті функціональні API Kotlin автоматично конвертуються у вигляді колекцій і мають ідентичну функціональність.

Для створення інтерфейсу користувача використовується XML. Це текстова мова розмітки, в якій теги ідентифікують дані та використовуються для їх зберігання та організації, а не для визначення того, як вони повинні бути відображені, як у випадку з тегами HTML, що відповідають за відображення даних. У програмуванні для Android XML часто використовується для реалізації інтерфейсу користувача. Тому розуміння основ інтерфейсу користувача з використанням XML є важливим. Інтерфейс користувача для Android-додатків побудований як ієрархія основних макетів та віджетів. Макети представляють собою об'єкти або контейнери ViewGroup, які керують розташуванням дочірніх елементів на екрані.

Віджети в Android - це об'єкти перегляду, такі як кнопки та текстові поля, які використовуються для відображення різних елементів інтерфейсу користувача. XML теги визначають дані та організують їх, але не визначають, як саме вони мають бути відображені. Використання XML для реалізації

інтерфейсу користувача в Android дійсно спрощує розробку, оскільки це проста та легко масштабована мова розмітки.

Щодо розробки баз даних, вирішено використовувати Room, який є новим способом збереження даних додатків у програмах Android. Room є частиною нової архітектури Android, що підтримує доречну архітектуру програм. Цей інструмент пропонується як альтернатива іншим бібліотекам, таким як Realm, ORMLite, GreenDao та іншим.

Room надає високорівневий інтерфейс для роботи з низькорівневими прив'язками до SQLite, які вбудовані в Android. Він автоматизує багато рутинних завдань і створює інтерфейс API поверх вбудованого SQLite API, що спрощує роботу з базами даних без необхідності прямої роботи з Cursor або ContentResolver.

Room відрізняється від більшості ORM тому що використовує обробники інструкцій для здіййснення всієї своєї магії збереження даних. Це означає, що класи додатків та класи моделей не потребують розширення в Room на відмінність від інших ORM, таких як Realm і SugarORM.

Також Room дозволяє вам відслідковувати зміни даних, інтегруючись з API LiveData архітектурних компонентів і з RxJava 2. Якщо схема складна і змінам даних в базі даних потрібно з'являтися в кількох місцях програми, Room зробить сповіщення про зміни. Це потужне доповнення може бути включено одним рядком коду.

3.2 Висновок до третього розділу

У даному розділі було взято на розгляд декілька мов програмування (Swift, Java, Kotlin, Rust). Було розглянуто переваги та недоліки кожного варіанту та їх особливості при роботі. Крім цього було описано можливості кожної з обраних мов програмування для розробки інформаційної системи пошуку квитків.

За допомогою обраної мови програмування повинна бути відповідною до наступних вимог:

- Реалізація графічного інтерфейсу користувача;
- Робота з базою даних;
- Швидка компіляція.

Основуючись на цих вимогах, було вирішено вибрати мову програмування Kotlin. Цей варіант виявився найкращим для реалізації інформаційної системи пошуку квитків, оскільки має чіткий синтаксис та відмінно підходить для роботи.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ДОДАТКУ ДЛЯ ПОШУКУ КВИТКІВ ДЛЯ ПОДОРОЖЕЙ

4.1 Опис створеного програмного засобу

4.1.1 Структура бази даних

Для того щоб працювати з даними системи було прийнято рішення використовувати Room Database. Розроблена база даних призначена для зберігання даних про квитки.

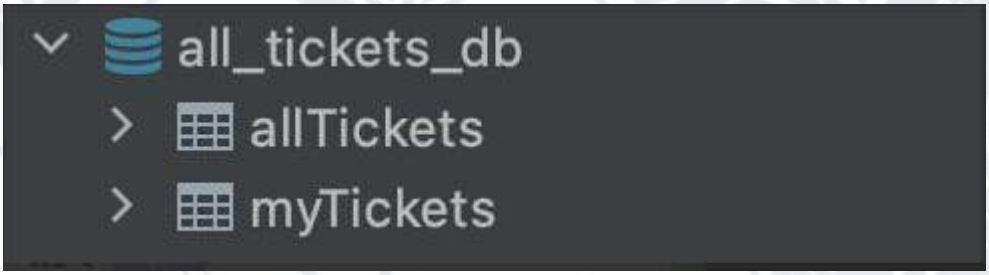

Рисунок 4.1 - Структура розробленої бази даних

База даних складається з двох таблиць:

- allTickets;
- myTickets;

Таблиця allTickets містить в собі такі поля як:

- id – первинний ключ;
- route – маршрут, міста відправлення та прибуття;
- DepartureTime – час відправлення;
- isSelected - вказує на додані користувачем квитки, якщо значення 1, то користувач додав квиток до обраних.

Рисунок 4.2 - Структура таблиці allTickets

Таблиця myTickets містить в собі наступні поля:

- id – первинний ключ;
- route – маршрут, міста відправлення та прибуття;
- DepartureTime – час відправлення.

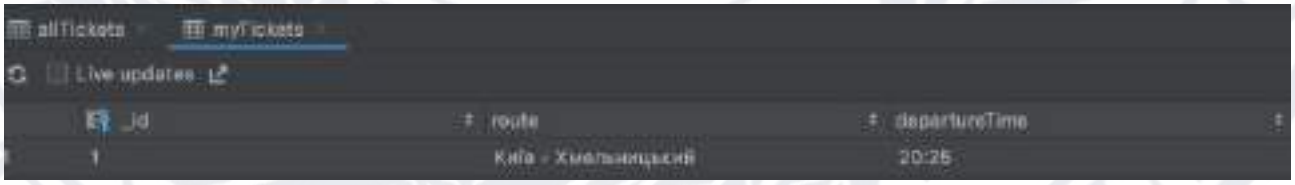

Рисунок 4.3 - Структура таблиці myTickets

4.1.2 Опис механізмів

Головні функції інформаційної системи було поділено на певні модулі. В результаті було отримано:

- модуль реєстрації;

- модуль відображення доступних квитків;
- модуль роботи з внутрішньою памя’ттю;
- модулі інтерфейсу користувача;
- модуль роботи з БД.

Перелік основних функцій в коді:

```
    <application
        android:name=".presentation.app.App"
        <activity
            android:name=".presentation.launcher.MainActivity"
            android:exported="true"
            android:screenOrientation="fullSensor"
            android:theme="@style/Theme.Mel_d">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"
/>

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".presentation.auth.Login"
            android:exported="false"
            android:screenOrientation="fullSensor"
            android:theme="@style/Theme.Mel_d" />
        <activity
            android:name=".presentation.bottomNavig.BottomNavig"
            android:exported="true"
            android:screenOrientation="fullSensor"
            android:theme="@style/Theme.Mel_d" />
    </application>

</manifest>
```

AndroidManifest.xml – файл, завдяки якому система Android розуміє та показує наш додаток, у ньому прописані усі головні компоненти Android системи [23]. В AndroidManifest.xml можна налаштувати такі можливості:

- вказати ім'я Java-пакета програми, який є унікальним ідентифікатором;
- описати компоненти програми, служби;
- вказати список необхідних дозволів для звернення до захищених частин API та взаємодії з іншими програмами;

- дозволити доступ до ресурсів, які сторонні програми повинні мати для взаємодії з компонентами цієї програми;
- вказати мінімальний та максимальний рівень API Android, необхідний для роботи програми.

Тег <manifest> - це найголовніший тег, в якому вкладена вся конфігурація проекту. За замовчуванням він створюється разом із файлом AndroidManifest і спочатку має початковий набір параметрів.

Тег <application> - один з основних елементів маніфесту, що містить опис компонентів програми, що доступні у пакеті. Також містить дочірні елементи, що представляють кожен компонент, що є у складі програми. У маніфесті може бути лише один елемент *application.*

Тег <activity> - вказує саме активність. Якщо у програмі є кілька активностей, вони повинні бути оголошені в маніфесті, при цьому створиться для кожної свій елемент <activity>. Якщо активність не оголошена в маніфесті, вона не буде видною для системи і не буде запущена при виконанні програми або виводитиметься повідомлення про помилку.

Елемент <intent-filter> визначає типи намірів, на які можуть відповісти діяльність, сервіс чи приймач намірів. Фільтр намірів надає для компонентів-клієнтів можливість отримання намірів типу, що оголошується, відфільтровуючи ті, що не є значимими для компоненту, і містить дочірні елементи *action, category, data*. Елемент *action* додає дії до фільтру намірів. Елемент *intent-filter* повинен містити один або більше елементів *action*. При умові, що в елементі *intent-fiiter* не буде саме цих елементів, об'єкти намірів не зможуть пройти через фільтр. *category* - визначає категорію компонента, що буде опрацьовувати намір. Це саме строкові константи, що визначені у класі intent:

```
package com.example.Mel_d.presentation.auth

import…
@AndroidEntryPoint
class LoginActivity : AppCompatActivity() {

    private val binding by lazy {
```

```
            ActivityLoginBinding.inflate(layoutInflater)
        }

        private val dateAndTime: Calendar by lazy {
    Calendar.getInstance() }

        private val loginViewModel: LoginViewModel by viewModels()

        @Inject
        lateinit var sharedPreferencesManager:
    SharedPreferencesManager
            binding.viewModel = loginViewModel

            observeViewModel()

            dateClick()
            continueClick()
        }

        private fun observeViewModel() {
            loginViewModel.isLoginButtonEnable.observe(this) {
                if (it) {
                    binding.continueButton.setCardBackgroundColor(
                        ContextCompat.getColor(
                            this@LoginActivity,
                            R.color.teal_200
                        )
                    )
                    binding.continueButton.isClickable = true
                } else {
                    binding.continueButton.setCardBackgroundColor(
                        ContextCompat.getColor(
                            this@LoginActivity,
                            R.color.purple_200_20
                        )
                    )
                    binding.continueButton.isClickable = false
                }
            }
            loginViewModel.progressLiveData.observe(this) {
                if (it) {
                    binding.allMask.visibility = View.VISIBLE
                    binding.progressBar.visibility = View.VISIBLE
                } else {
                    binding.allMask.visibility = View.GONE
                    binding.progressBar.visibility = View.GONE
                }
            }
            loginViewModel.successLiveData.observe(this) {
                if(it) {
                    startActivity(Intent(this@LoginActivity,
    BottomNavigActivity::class.java))
                }
```

```
        }
    }

    private fun setUserDateOfBirth() {
        val format = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault())

binding.dateOfBirth.setText(format.format(dateAndTime.time))
    }

    private fun dateClick() {
        binding.dateOfBirth.setOnClickListener {
            DatePickerDialog(
                this@LoginActivity, datePickerCallback,
                dateAndTime[Calendar.YEAR],
                dateAndTime[Calendar.MONTH],
                dateAndTime[Calendar.DAY_OF_MONTH]
            ).show()
        }
    }

    private var datePickerCallback =
        OnDateSetListener { view, year, monthOfYear, dayOfMonth ->
            dateAndTime.set(Calendar.YEAR, year)
            dateAndTime.set(Calendar.MONTH, monthOfYear)
            dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth)
            setUserDateOfBirth()
        }

    private fun continueClick() {
        binding.continueButton.setOnClickListener {
            loginViewModel.saveUserData()
        }
    }
}
```

Клас відповідає за реєстрацію користувача.

Функція onCreate() відповідає за створення Activity - візуального компонента додатку, має в собі виклик setContentView(binding.root) - призначенний для скріплення файлу kotlin та xml файлу.

observeViewModel() відповідає за отримання колбеків від полів класу LoginViewModel, так як ViewModel добре працює з реактиввним програмуванням, ми 1 раз підпишеммось за оновлення її полів і будемо отримувати дані як тільки вони зміняться.

Змінна datePickerCallback відповідає за отримання данних при закритті вікна з календарем.

Функція setUserDateOfBirth відповідає за встановленя дати у відповідне поле.

Функція continueClick відповідає за обробку кліку на кнопку продовження.

У фрагменті коду нижче: private val binding by lazy {ActivityLoginBinding.inflate(layoutInflater)} - встановлюється змінна для керування xml файлом.

Змінна private val dateAndTime: Calendar by lazy {Calendar.getInstance()} - відповідає за керуваввння датою.

private val loginViewModel: LoginViewModel by viewModels() створюється змінна типу LoginViewModel.

by viewModels() - делегат котліну для створення вьюмоделей.

```
<layout xmlns:android=" "
    xmlns:app=" "
    xmlns:tools=" ">

    <data>

        <variable
            name="viewModel"

type="com.example.Mel_d.presentation.auth.LoginViewModel" />
    </data>

    <androidx.constlayout.widget.Constlayout
        <TextView
            android:id="@+id/text1"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="36dp"
            android:gravity="center"
            android:text="Вітаємо!"
            app:layout_Left_toLeftOf="parent"
            app:layout_Right_toRightOf="parent"
            app:layout_Top_toTopOf="parent" />

        <TextView
            android:id="@+id/text2"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:gravity="center"
            android:text="Пройдіть реєстрацію!"
            android:textColor="@color/black"
            android:textSize="20sp"
            android:textStyle="bold"
```

```
            app:layout_Left_toLeftOf="parent"
            app:layout_Right_toRightOf="parent"
            app:layout_Top_toBottomOf="@id/text1" />

        <EditText
            android:id="@+id/nameEditText"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:gravity="start"
            android:hint="Введіть Ваше ім'я"
            android:text="@={viewModel.nameLiveData}"
            android:textColor="@color/black"
            app:layout_End_toEndOf="parent"

        <EditText
            android:id="@+id/surnameEditText"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginHorizontal="24dp"
            android:gravity="start"
            android:hint="Введіть Ваше прізвище"
            android:text="@={viewModel.surnameLiveData}"
            android:textColor="@color/black"
            app:layout_End_toEndOf="parent"
            app:layout_Start_toStartOf="parent"
            app:layout_Top_toBottomOf="@+id/nameEditText" />

        <EditText
            android:id="@+id/emailEditText"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginHorizontal="24dp"
            android:gravity="start"
            android:hint="Введіть Вашу електрону адресу"
            android:inputType="textEmailAddress"
            android:text="@={viewModel.emailLiveData}"
            android:textColor="@color/black"
            app:layout_End_toEndOf="parent"
            app:layout_Start_toStartOf="parent"
            app:layout_Top_toBottomOf="@+id/surnameEditText" />
```

Фрагмент з файлу - activity_login.xml. Даний файл відповідає за дизайн екрану реєстрації.

Основні коммпоненти :

- Constlayout - це компонент який дозволяє в собі розмістити інші компоненти;
- Textiew - це тег який дозволяє відобразити текстове поле;
- EditText - тег який дозволяє відобразити поле для вводу данних;

- MaterialCardView - елемент картки для більш гарного відображення елементів;
- ProgressBar - елемент для візуального відображення завантаження;
- LinearLayout - це компонент який дозволяє в собі розмістити інші компоненти(старіший за Constlayout) [16].

```
package com.example.Mel_d.presentation.bottomNavigActivity

import androidx.appcompat.app.AppCompatDelegate
import androidx.navig.NavController
import androidx.navig.fragment.NavHostFragment
import androidx.navig.ui.setupWithNavController
import com.example.Mel_d.R
import com.example.Mel_d.databinding.ActivityBottomNavigBinding
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class BottomNavigActivity : AppCompatActivity() {

    private val binding by lazy {
        ActivityBottomNavigBinding.inflate(layoutInflater)
    }

    private lateinit var navController: NavController

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT
_NO)

        setContentView(binding.root)
        setupBottomNavigBar()

    }

    private fun setupBottomNavigBar() {
        val navHostFragment =

supportFragmentManager.findFragmentById(R.d.mainNavContainer) as
NavHostFragment
        navController = navHostFragment.navController


binding.mainBottomNav.setupWithNavController(navController)
        binding.mainBottomNav.setOnItemSelectedListener {
            when (it.itemId) {
                R.d.myTickets -> {
                    navController.navigate(R.d.myTicketsFragment)
                    true
                }
```

```
                R.d.allTickets -> {
                    navController.navigate(R.d.allTicketsFragment)
                    true
                }
                else -> {
                    navController.navigate(R.d.userInfoFragment)
                    true
                }
            }
        }
    }
}
```

Клас BottomNavigActivity - призначений для того, щоб відображати екрани на яких можна побачити головну інформацію додатку, як і кожен клас який унаслідує activity має метод onCreate().

Функція setupBottomNavigBar(), яка відповідає за налаштування керуванням фрагментами та переходу між ними.

```
<navig 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"
    android:id="@+id/main_bottom_nav"
    app:startDestination="@id/allTicketsFragment">

    <fragment
        android:id="@+id/allTicketsFragment"

android:name="com.example.Mel_d.presentation.allTickets.AllTickets
Fragment"
        android:label="AllTicketsFragment"
        tools:layout="@layout/fragment_all_tickets" />
    <fragment
        android:id="@+id/myTicketsFragment"

android:name="com.example.Mel_d.presentation.myTickets.MyTicketsFr
agment"
        android:label="MyTicketsFragment"
        tools:layout="@layout/fragment_my_tickets" />
    <fragment
        android:id="@+id/userInfoFragment"

android:name="com.example.Mel_d.presentation.userInfo.UserInfoFrag
ment"
        android:label="UserInfoFragment"
        tools:layout="@layout/fragment_user_info" />
```

```
</navig>
```

Файл – main_botton_nav.xml. Даний файл призначений для створення конфігурації фрагментів, їх анімації та переходів між ними.

```
package com.example.Mel_d.data.sharedPreferencesManager
import com.example.Mel_d.data.user.UserModel
import dagger.hilt.android.qualifiers.ApplicationContext
import javax.inject.Inject
import javax.inject.Singleton

@SuppressLint("CommitPrefEdits")
@Singleton
class SharedPreferencesManager @Inject
constructor(@ApplicationContext context: Context) {

    private val PREFERENCES_FILE_NAME = "app.spref"
    private val USER = "USER"
    private val IS_DB_FILLED = "IS_DB_FILLED"

    private val sharedPreferences: SharedPreferences =
        context.getSharedPreferences(PREFERENCES_FILE_NAME,
Context.MODE_PRIVATE)
    private val editor: SharedPreferences.Editor =
sharedPreferences.edit()

    var userModel: UserModel?
        get() {
            val gson = Gson()
            val json: Strg? = sharedPreferences.getString(USER,
null)
            return gson.fromJson(json, UserModel::class.java)
        }
        set(value) {
            val gson = Gson()
```

```
            val json: String = gson.toJson(value)
            editor.putString(USER, json)
            editor.apply()
        }

    var isDBFilled: Boolean
        get() = sharedPreferences.getBoolean(IS_DB_FILLED, false)
        set(isDBFilled) {
            editor.putBoolean(IS_DB_FILLED, isDBFilled)
            editor.apply()
        }
}
```

Клас SharedPreferencesManager - це клас який описує роботу з внутрішною памятю (кеш).

private val sharedPreferences: SharedPreferences = context.getSharedPreferences(PREFERENCES_FILE_NAME, Context.MODE_PRIVATE) – змінна, яка дає доступ для створення кешу

private val editor: SharedPreferences.Editor = sharedPreferences.edit() – змінна, яка дозволяє редагувати кеш.

Ключі доступу до змінних:

- private val PREFERENCES_FILE_NAME = "app.spref"
- private val USER = "USER"
- private val IS_DB_FILLED = "IS_DB_FILLED"

```
package com.example.Mel_d.data.roomDB

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.Query
import androidx.room.Update
import com.example.Mel_d.data.roomDB.data.MyTicketEntity
import com.example.Mel_d.data.roomDB.data.TicketEntity

@Dao
interface TicketsDao {
    @Query("SELECT * from allTickets")
```

```
    suspend fun getAllTickets(): List<TicketEntity>

    @Query("SELECT * FROM allTickets WHERE route LIKE '%' ||
:search || '%'")
    fun findTicket(search: Strg?): List<TicketEntity>

    @Update
    suspend fun updateTicket(ticket: TicketEntity)

    @Insert
    suspend fun addNewItem(ticket: MyTicketEntity)

    @Query("DELETE FROM myTickets WHERE _id LIKE '%' || :id ||
'%'")
    suspend fun deleteItem(id: String)

    @Query("SELECT * FROM myTickets")
    suspend fun findMyTickets(): List<MyTicketEntity>

    @Insert
    suspend fun addItem(vararg ticket: TicketEntity)
}
```

Інтерфейс – TicketsDao. Даний інтерфейс описує роботу з локальною базою даних, прописані усі методи які використовуються при роботі з нашою БД.

```
 package com.example.Mel_d.domain.usecases.addTicketToMyList


import com.example.Mel_d.data.roomDB.data.MyTicketEntity


interface AddTicketToMyListUseCase {
    suspend fun addTicketToMyList(ticket: MyTicketEntity):
Result<Any>
}
@Singleton
class DefaultAddTicketToMyListUseCase @Inject constructor(
    private val addTicketToMyListRepository:
DefaultAddTicketToMyListRepository
) : AddTicketToMyListUseCase {
    override suspend fun addTicketToMyList(ticket:
MyTicketEntity): Result<Any> {
        return kotlin.runCatching {
addTicketToMyListRepository.addTicketToMyList(ticket) }
```

```
    }
}
```

Інтерфейс AddTicketToMyListUseCase та його реалізація DefaultAddTicketToMyListUseCase. Даний інтерфейс описує як потрібно звертатись до репозиторія, який працює з базою данних, у DefaultAddTicketToMyListUseCase ми реалізуєм це звернення.

```
interface AddTicketToMyListRepository {
    suspend fun addTicketToMyList(ticket: MyTicketEntity)
}
package com.example.Mel_d.data.repository

import com.example.Mel_d.data.roomDB.TicketsDao
import com.example.Mel_d.data.roomDB.data.MyTicketEntity
import
com.example.Mel_d.domain.repositories.AddTicketToMyListRepository
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class DefaultAddTicketToMyListRepository @Inject constructor(
    private val ticketsDao: TicketsDao
):AddTicketToMyListRepository {
    override suspend fun addTicketToMyList(ticket: MyTicketEntity)
{
        ticketsDao.addNewItem(ticket)
    }
}
```

Інтерфейс AddTicketToMyListRepository - описує правило як потрібно додавати білет до списку користувача. Клас наслідник DefaultAddTicketToMyListRepository - реалізує це правило.

```
package com.example.Mel_d.data.user

data class UserModl(
    val name: Strg?
    val surname: Strg?,
    val email: Strg?,
    val dateOfBirth: Strg?
)
```

Модель користувача в якій вказані відповідні змінні. Змінна name – ім’я користувача; surname – прізвище користувача; email – електронна пошта користувача; dateOfBirth – дата народження користувача.

```
<androidx.constlayout.widget.Constlayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    >

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/mainNavContainer"
        android:name="androidx.navig.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:defaultNavHost="false"
        app:layout_Bottom_toTopOf="@id/mainBottomNav"
        app:layout_Top_toTopOf="parent"
        app:navGraph="@navig/main_bottom_nav" />

    <com.google.android.material.bottomnavig.BottomNavigView
        android:id="@+id/mainBottomNav"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:layout_gravity="bottom"
        app:labelVisibilityMode="labeled"
        app:layout_Bottom_toBottomOf="parent"
        app:menu="@menu/bottom_menu" />
</androidx.constlayout.widget.Constlayout>
```

Файл - activity_bottom_navig.xml. FragmentContainerView - компонент, контейнер для відображення фрагментів. BottomNavigView - компонент для відображення нижньої панелі меню, за допомогою якої користувач має змогу переключатись між екранами пошуку квитків, екраном обраних квитків та екраном профілю з персональною інформацією.

4.2 Інструкція користувача

Вступ. Розроблена інформаційна система для пошуку квитків для подорожей надає користувачам можливість здійснювати пошук та бронювання квитків. Інтуїтивний та простий у використанні інтерфейс дозволяє користувачам швидко та зручно редагувати свої персональні дані у вікні профілю, а також додавати та видаляти квитки, які вони обрали раніше.

Загальні відомості про систему

Розроблена інформаційна система реалізована на мові програмування Kotlin для мобільного додатка на операційній системі Android. Для створення інтерфейсу користувача використовувалась XML версії 1.0. Для зберігання даних використано базу даних Room, а для роботи з внутрішньою пам'яттю — sharedpreferences.

Основна мета продукту полягає в тому, щоб користувач мав можливість знайти потрібний квиток серед доступних або вибрати з них і забронювати його.

Класи вирішуваних завдань

Головною метою даної системи є пошук квитків. Перш ніж користувач зможе розпочати пошук, йому необхідно пройти реєстрацію. На сторінці реєстрації користувач вводить своє прізвище, ім'я, дату народження, логін і пароль, після чого натискає кнопку "Продовжити". Система перевіряє коректність введених даних та зберігає їх у внутрішній пам'яті.

Після реєстрації користувач потрапляє на головне вікно "Всі квитки", де він може обрати квиток з наявних або здійснити пошук, введши назву міста відправлення у відповідному полі.

Для забронювання квитка користувач просто натискає кнопку "Забронювати" біля обраного квитка. Після цього квиток з'являється у вікні "Мої квитки", де користувач може видалити обраний ним раніше квиток за допомогою кнопки "Видалити".

Крім того, користувач може переглянути введену при реєстрації інформацію про себе у вікні "Профіль". У цьому вікні є кнопки "Вийти", за допомогою якої користувач може вийти зі свого профілю, та "Редагувати", щоб змінити інформацію про себе.

Опис основних характеристик і особливостей системи

Розроблена інформаційна система пошуку квитків для подорожей значно спрощує процес пошуку та бронювання, оскільки вона дозволяє користувачам здійснювати ці операції прямо через мобільний додаток, без необхідності відвідувати офлайн каси на вокзалах. Ця інформаційна система є автономною і

не потребує інтеграції з іншими системами, що робить її використання ще більш зручним для користувачів.

Відомості про функціональні обмеження на застосування

Наразі дані розробленої інформаційної системи знаходяться локально, але для майбутнього розміщення додатку на таких платформах, як "Google Play" або "AppStore", потрібне буде інтернет-з'єднання. Однак для роботи з системою не потрібне додаткове обладнання, лише смартфон з встановленим додатком буде достатнім.

4.3 Аналіз контрольного прикладу

Фінальним кроком розробки інформаційної системи є тестування, для того щоб перевірити працездатність усіх функцій системи. Після того як користувач інсталював додаток він потрапляє на вікно реєстрації (рис 4.4).

Рисунок 4.4 - Екран реєстрації

Після надання усієї необхідної особистої інформаціїї про себе, користувачу необхідно натиснути кнопку «Продовжити». Після чого система перевірить коректність усіх даних та збереже їх (рис 4.5).

Рисунок 4.5 - Заповнена форма реєстрації

Після реєстрації користувач потрапляє у вкладку «Всі квитки», де може обрати з наявних (рис 4.6).

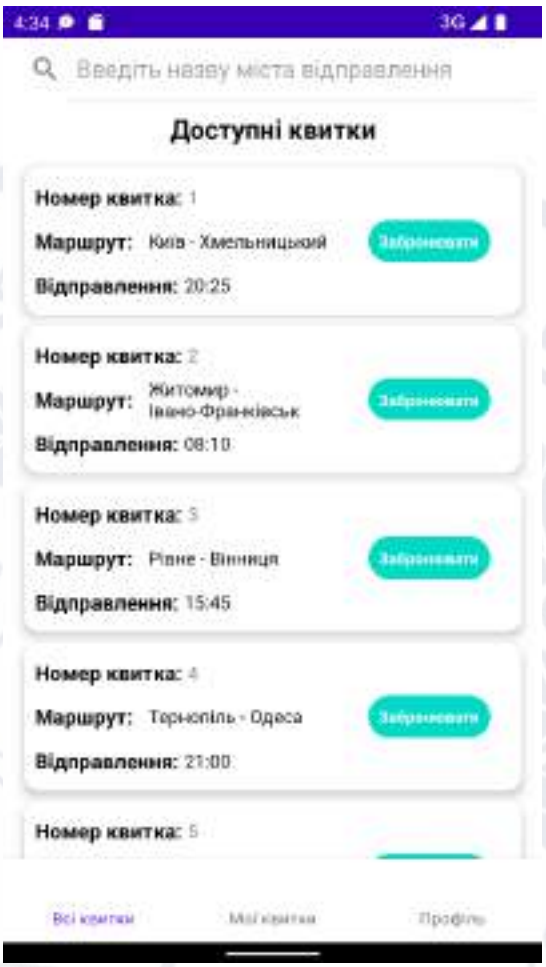

Рисунок 4.6 - Вікно «Всі квитки»

Для того щоб обрати квиток серед наявних, потрібно натиснути на кнопку «Забронювати» напроти обраного квитка (рис 4.7).

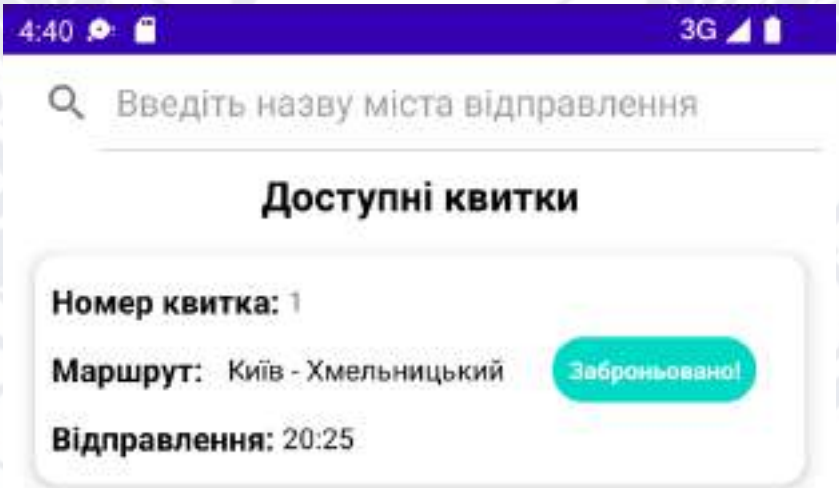

Рисунок 4.7 - Заброньований квиток

Після того як користувач забронював квиток він автоматично з’являється у вікні «Мої квитки» (рис 4.8). Якщо ж користувачу необхідно видалити обраний квиток, потрібно натиснути на кнопку «Видалити» напроти квитка і він одразу ж зникне (рис 4.9).

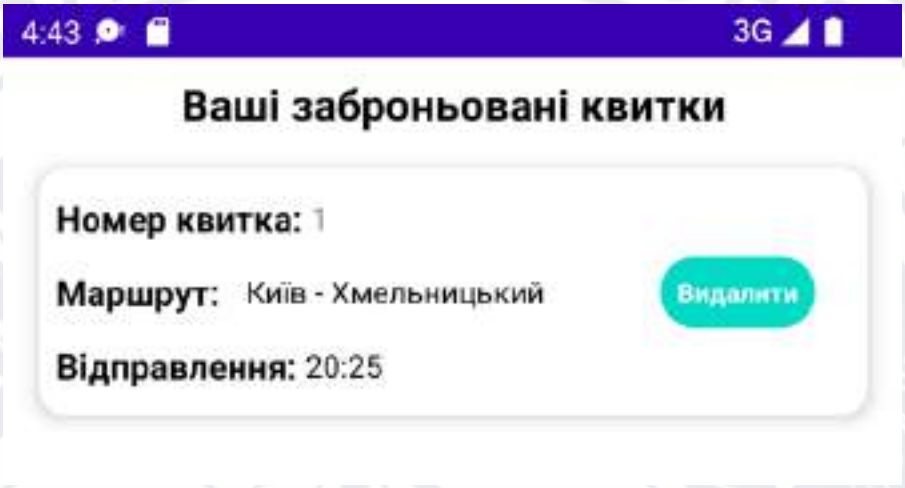

Рисунок 4.8 - Вікно «Мої квитки»

Рисунок 4.9 - Вікно «Мої квитки» після видалення квитка

Для того щоб здійснити пошук серед наявних квитків, користувач повинен ввести назву міста відправлення у відповідному полі (рис 4.10). Далі система знайде доступні на даний момент квитки (рис 4.11).

Рисунок 4.10 - Поле для пошуку

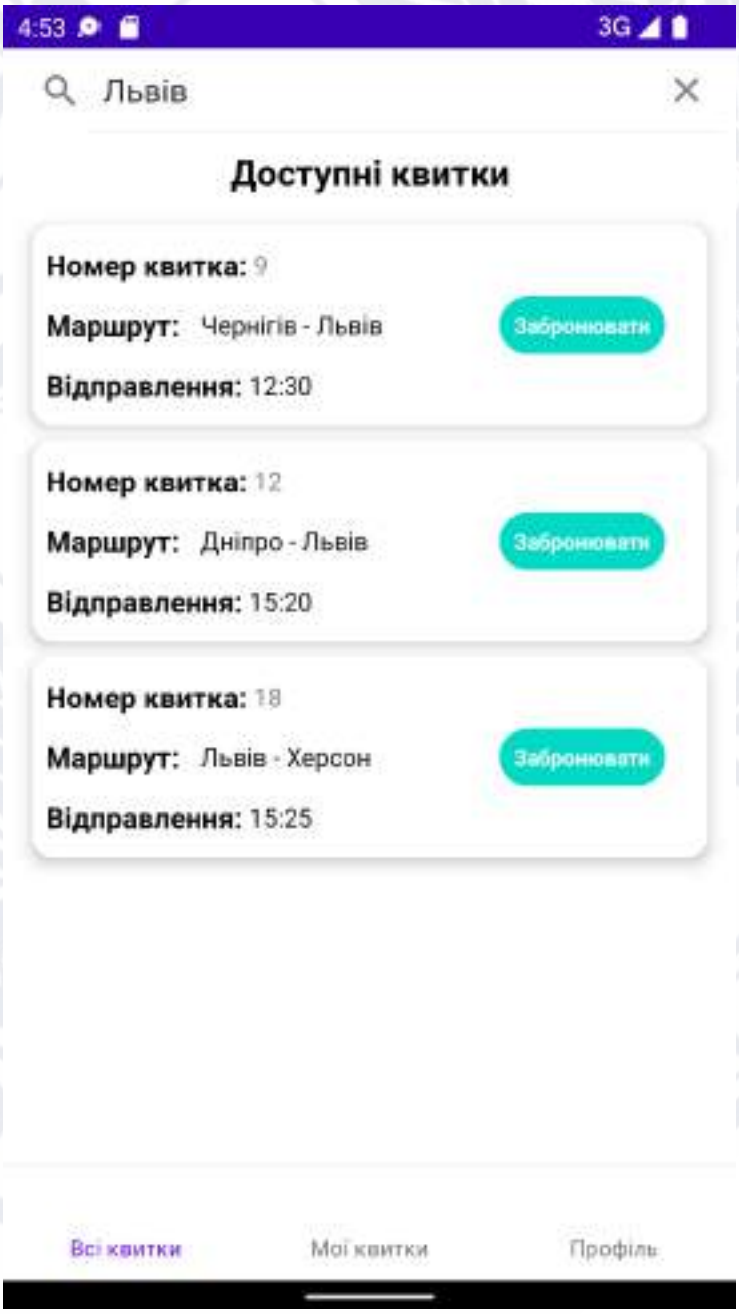

Рисунок 4.11 - Результат пошуку

Для навігації по додатку в нижній частині екрану знаходиться меню, за допомогою якого користувач може переключатись між вікнами (рис 4.12).

Рисунок 4.12 - Меню для навігації

Якщо користувач хоче переглянути персональну інформацію, потрібно перейти у вкладку «Профіль» (рис 4.13).

Рисунок 4.13 - Вікно «Профіль»

Для того щоб відредагувати особисту інформацію користувачу потрібно натиснути кнопку «Редагувати», ввести нові дані у відповідні поля та натиснути «Зберегти» (рис 4.14). Якщо ж потрібно вийти з профілю то натиснути кнопку «Вийти».

Рисунок 4.14 - Відредагована інформація в профілі

4.4 Висновок до четвертого розділу

У цьому розділі було представлено та розроблено програмну реалізацію інформаційної системи пошуку квитків для подорожей. Описано структуру бази даних, яка була розроблена за допомогою Room Database. База даних містить дві

таблиці: одна для зберігання даних про наявні квитки, а інша - для квитків, які було заброньовано користувачем. Програмна реалізація була розділена на окремі модулі, такі як:

- модуль реєстрації;
- модуль відображення доступних квитків;
- модуль роботи з внутрішньою памя’ттю;
- модулі інтерфейсу користувача;
- модуль роботи з БД.

Усі вищезазначені модулі містять функції та об'єкти, які взаємодіють між собою. Крім того, була розроблена інструкція користувача, в якій описано ключові пункти та особливості системи. Зазначено, що система може функціонувати самостійно, і визначено обрану мову програмування та необхідні технології для її реалізації. Для використання розробленої інформаційної системи користувачу потрібно мати лише смартфон з встановленим додатком.

Пізніше було проведено аналіз контрольного прикладу, під час якого була перевірена основна функціональність системи. У розробленій системі доступні кілька вікон, за допомогою яких можна зареєструватися, забронювати квитки, здійснювати пошук та переглядати або редагувати персональну інформацію в профілі.

ВИСНОВКИ

В результаті роботи було досягнуто поставлену мету, а саме розроблено інформаційну системи пошуку квитків для подорожей. Зважаючи на постійну популярність сфери туризму, такий проект є надзвичайно актуальним. Ось чому все більше людей віддають перевагу інтеграції інформаційних систем у різноманітні сфери, щоб полегшити процеси та збільшити ефективність. Сьогодні використання інформаційних систем стало не просто додатковою можливістю, але й необхідністю. Це робить багато процесів більш зручними та швидшими. Наприклад, зараз не обов'язково стояти в чергах на вокзалах для бронювання або придбання квитків, оскільки це можна зробити онлайн.

Під час дослідження предметної області було детально вивчено питання пошуку квитків. Визначено різні типи пошукових систем та проаналізовано їх особливості. Серед виділених типів можна відзначити:

- пошукові системи на основі сканера;
- каталоги, що працюють від людини;
- гібридні пошукові системи;
- інші спеціальні пошукові системи.

Також було проведено аналіз аналогічних систем, визначено та описано їхні недоліки та переваги. У наступному розділі було проведено системний аналіз. В результаті було сформовано дерево цілей, де основною ціллю визначено *Створення інформаційної системи пошуку квитків для подорожей*. Після створення дерева цілей було визначено критерії, якими повинна відповідати система. За допомогою методу аналізу ієрархій було визначено тип системи – інформаційно-пошукова система. Завдяки використанню методології IDEF0 було розроблено контекстну діаграму та проведено її декомпозицію до третього рівня включно, для того щоб визначити функціональні можливості системи. В наступному розділі було здійснено аналіз програмних засобів, які можна використати для розробки інформаційної системи.

Було досліджено такі мови програмування, як Java, Kotlin, Swift та Rust. Серед усіх варіантів було обрано Kotlin, оскільки ця мова програмування має достатньо можливостей для реалізації інформаційної системи пошуку квитків. Для роботи з даними було обрано Room Database. Наступним кроком була програмна реалізація. Було розроблено структуру бази даних, яка складається з двох таблиць. Перша таблиця містить дані про усі наявні квитки, а друга - про квитки, обрані користувачем. Програмний код було розподілено на окремі модулі, такі як:

- модуль реєстрації;
- модуль відображення доступних квитків;
- модуль роботи з внутрішньою памя’ттю;
- модулі інтерфейсу користувача;
- модуль роботи з БД.

Для розробленої програми було складено інструкцію користувача, що описує основний функціонал та особливості інформаційної системи. Пізніше було проведено аналіз контрольного прикладу, де перевірено працездатність системи та коректність її роботи.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Тягунова Н., Спориш О.А., Іржавська Л.В. Основи організації туристичного бізнесу. Центр учбової літератури. 2024. 130 с.

2. Офіційний сайт Всесвітньої туристичної організації URL: http://www.world-tourism.org. (останнє звернення: 25.05.2024)

3. Tourism and the world economy URL: http://www.unwto.org. (останнє звернення: 25.05.2024)

4. Мальська М.П., Паньків Н.М., Ховалко А.Б. історія розвитку туризму. URL: https://tourlib.net/books_history/malska-irt.htm (останнє звернення: 24.05.2024)

5. Шаховалов М. Інтернет-технології в туризмі. URL: https://infotour.in.ua/shahovalov.htm (останнє звернення: 24.05.2024)

6. Спориш О.А., Тягунова Н. Використання сучасних інформаційних технологій для просування туристичного продукту. URL: https://tourlib.net/statti_ukr/sporysh.htm (останнє звернення: 25.05.2024)

7. Proizd.ua. URL: https://proizd.ua/ (останнє звернення: 25.05.2024)

8. Skyscanner.com. URL: www.skyscanner.com.ua (останнє звернення: 25.05.2024)

9. Tickets.ua. URL: https://tickets.ua (останнє звернення: 25.05.2024)

10. Busfor.ua. URL: https://busfor.ua/ (останнє звернення: 25.05.2024)

11. Omio.ua. URL: https://www.omio.com.ua (останнє звернення: 25.05.2024)

12. What is a Goal Tree URL:https://dividendsdiversify.com/goal-tree/ (останнє звернення: 25.05.2024)

13. Process Hierarchy Diagram Basics URL:https://infocenter-archive.sybase.com/ (останнє звернення: 25.05.2024)

14. Авраменко В.С., Авраменко А.С. Проектування інформаційних систем: навчальний посібник – Черкаси: Черкаський національний університет ім. Б. Хмельницького, 2022. – 434 с.: іл.

15. Орлик О., Демёянчук К.Ф. Можливості Інтернету у формуванні, просуванні й реалізації продуктів та послуг // Інформатика та інформаційні технології : студ. наук. конф., 20 квітня 2015 р. : матер. конф. - Одеса: ОНЕУ, 2015. - С. 88-91.

16. XML For Android App Development URL:https://www.geeksforgeeks.org/a-complete-guide-to-learn-xml-for-android-app-development/#:~:text=XML%20tags%20define%20the%20data,need%20to%20be%20just%20invoked (останнє звернення: 25.05.2024)

17. Rust's growing popularity URL:https://codilime.com/blog/why-is-rust-programming-language-so-popular/#:~:text=What%20is%20so%20special%20about,developed%20originally%20at%20Mozilla%20Research (останнє звернення: 25.05.2024)

18. Advantages and disadvantages of Kotlin URL:https://krify.co/advantages-and-disadvantages-of-kotlin/ (останнє звернення: 25.05.2024)

19. Pros and Cons of the Kotlin URL:https://www.netguru.com/blog/kotlin-pros-and-cons` (останнє звернення: 25.05.2024)

20. Kotlin overview URL: https://developer.android.com/kotlin/overview#:~:text=Kotlin%20is%20an%20open%2Dsource,and%20Scala%2C%20among%20many%20others (останнє звернення: 25.05.2024)

21. Advantages and Disadvantages of Java URL:https://techvidvan.com/tutorials/pros-and-cons-of-java/ (останнє звернення: 25.05.2024)

22. Мова програмування JavaURL:https://java.fandom.com/ru/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_Java (останнє звернення: 25.05.2024)

23. AndroidManifest.xml URL: https://devcolibri.com/androidmanifest (останнє звернення: 25.05.2024)

Rust URL:https://www.rust-lang.org/ (останнє звернення: 25.05.2024)

24. What's new in Rust URL:https://www.infoworld.com/article/3267624/whats-new-in-the-rust-language.html (останнє звернення: 25.05.2024)

25. The Good and the Bad of Swift URL:https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-swift-programming-language/

26. Перелік національних стандартів для Перелік Національних стандартів України для створення, впровадження та супроводження автоматизованих і інформаційних систем http://nbuv.gov.ua/node/1469. (дата звернення 21.05.2024)

27. Уніфікована система організаційно-розпорядчої документації. Вимоги до оформлювання документів. ДСТУ 4163-2003; чинний від 01.09.2003. URL: http://zakon3.rada.gov.ua/rada/show/v0055609-03. (дата звернення 21.05.2024)

28. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання: ДСТУ 3008-2015, 2016. 26 с.

29. Інформація та документація Бібліографічне посилання Загальні положення та правила складання ДСТУ 8302:2015, 2016. 16 с.

30. Що таке стандарт ISO? URL: https://onmedu.edu.ua/shho-take-standart-iso-9001/. (дата звернення 21.05.2024)

31. ICO. URL: https://www.ukrcsm.kiev.ua/index.php/en/services-ua/standard-ua/inter-org-standardua/inter-org-iso-ua. (дата звернення 21.05.2024)

32. МЕЖДУНАРОДНЫЙ СТАНДАРТ ISO 9000. 2015. URL: http://isomanagement.com/wp-content/uploads/2018/09/ISO-9000-2015.pdf. (дата звернення 21.05.2024)

33. ДСТУ ISO/IEC/IEEE 16326:2015 Розроблення систем та програмного забезпечення. Процеси життєвого циклу. Керування проектами (ISO/IEC/IEEE 16326:2009, IDT) URL : http://online.budstandart.com/ua/catalog/docpage.html?id_doc=67052. (дата звернення 21.05.2024)

34. Методичні рекомендації до виконання, оформлення та захисту кваліфікаційної (бакалаврської) роботи здобувачами спеціальності 122 «Комп’ютерні науки» О. В. Зелінська, Т. В. Січко, Н. А. Потапова, К. О. Якубич. Вінниця: ДонНУ імені Василя Стуса, 2024. 27 с.

ДОДАТКИ

ДОДАТОК А

Лістинг програми

AllTicketsFragment.kt:

```
package com.example.Mel_d.presentation.allTickets

import android.annotation.SuppressLint
import android.os.Bundle
import androidx.appcompat.widget.SearchView
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.recyclervw.widget.LinearLayoutManager
import com.example.Mel_d.data.roomDB.data.TicketEntity
import com.example.Mel_d.databinding.FragmentAllTicketsBinding
import dagger.hilt.android.AndroidEntryPoint
import javax.inject.Inject

@AndroidEntryPoint
class AllTicketsFragment : Fragment() {

    private val binding by lazy {
        FragmentAllTicketsBinding.inflate(layoutInflater)
    }

    private val allTicketsViewModel: AllTicketsViewModel by
viewModels()

    @Inject
    lateinit var allTicketsAdapter: AllTicketsAdapter

    lateinit var lastSelectedTicket: TicketEntity
        setUpViewModelCallbacks()
        setUpRecyclervw()
    }

    override fun onResume() {
        super.onResume()
        getAllTickets()
    }

    private fun setUpRecyclervw() {
        binding.allTicketsRecyclervw.apply {
            adapter = allTicketsAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }

        allTicketsAdapter.onItemClickListener = { ticket ->
            allTicketsViewModel.saveAndUpdateTicket(ticket)
            lastSelectedTicket = ticket
        }
    }
```

```
    @SuppressLint("NotifyDataSetChanged")
    private fun setUpViewModelCallbacks() {
        allTicketsViewModel.tickets.observe(viewLifecycleOwner) {
result ->
            if (result.isSuccess) {
                val items = result.getOrNull()
                if (items != null) {
                    allTicketsAdapter.items = items
                }
                allTicketsAdapter.notifyDataSetChanged()
            } else {
                Toast.makeText(
                    requireContext(),
                    result.exceptionOrNull()?.message.toString(),
                    Toast.LENGTH_SHORT
                ).show()
            }
        }

        allTicketsViewModel.addSuccess.observe(viewLifecycleOwner)
{ result ->
            if (result.isSuccess) {
                allTicketsAdapter.items.find {
                    it._id == lastSelectedTicket._id
                }?.isSelected = true

                allTicketsAdapter.notifyDataSetChanged()
            }
        }
    }
}
```

LoginActivity.kt:

```
package com.example.Mel_d.presentation.auth

import android.app.DatePickerDialog
import android.app.DatePickerDialog.OnDateSetListener
import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.ContextCompat
import com.example.Mel_d.R
import
com.example.Mel_d.data.sharedPreferencesManager.SharedPreferencesM
anager
import com.example.Mel_d.databinding.ActivityLoginBinding
import
com.example.Mel_d.presentation.bottomNavigActivity.BottomNavigActi
vity
import dagger.hilt.android.AndroidEntryPoint
```

```
@AndroidEntryPoint
class LoginActivity : AppCompatActivity() {

    private val binding by lazy {
        ActivityLoginBinding.inflate(layoutInflater)
    }

    private val dateAndTime: Calendar by lazy {
Calendar.getInstance() }

    private val loginViewModel: LoginViewModel by viewModels()

    @Inject
    lateinit var sharedPreferencesManager:
SharedPreferencesManager
    override fun onCreate(savedInstState: Bundle?) {
        super.onCreate(savedInstState)
        setContView(binding.root)
        binding.viewModel = loginViewModel

        observeViewModel()

        dateClick()
        continueClick()
    }

    private fun observeViewModel() {
        loginViewModel.isLoginButtonEnable.observe(this) {
            if (it) {
                binding.continueButton.setCardBackgroundColor(
                    ContextCompat.getColor(
                        this@LoginActivity,
                        R.color.teal_200
                    )
                )
                binding.continueButton.isClickable = true
            } else {
                binding.continueButton.setCardBackgroundColor(
                    ContextCompat.getColor(
                        this@LoginActivity,
                        R.color.purple_200_20
                    )
                )
                binding.continueButton.isClickable = false
            }
        }
        loginViewModel.progressLiveData.observe(this) {
            if (it) {
                binding.allMask.visibility = View.VISIBLE
                binding.progressBar.visibility = View.VISIBLE
            } else {
                binding.allMask.visibility = View.GONE
                binding.progressBar.visibility = View.GONE
```

```
            }
        }
        loginViewModel.successLiveData.observe(this) {
            if(it) {
                startActivity(Intent(this@LoginActivity,
BottomNavigActivity::class.java))
            }
        }
    }

    private fun setUserDateOfBirth() {
        val format = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault())

binding.dateOfBirth.setText(format.format(dateAndTime.time))
    }

    private fun dateClick() {
        binding.dateOfBirth.setOnClickListener {
            DatePickerDialog(
                this@LoginActivity, datePickerCallback,
                dateAndTime[Calendar.YEAR],
                dateAndTime[Calendar.MONTH],
                dateAndTime[Calendar.DAY_OF_MONTH]
            ).show()
        }
    }

    private var datePickerCallback =
        OnDateSetListener { view, year, monthOfYear, dayOfMonth ->
            dateAndTime.set(Calendar.YEAR, year)
            dateAndTime.set(Calendar.MONTH, monthOfYear)
            dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth)
            setUserDateOfBirth()
        }

    private fun continueClick() {
        binding.continueButton.setOnClickListener {
            loginViewModel.saveUserData()
        }
    }
}
```

AllTicketsViewModel.kt:

```
package com.example.Mel_d.presentation.allTickets

import android.util.Log
import com.example.Mel_d.data.roomDB.data.MyTicketEntity
import com.example.Mel_d.data.roomDB.data.TicketEntity
import
com.example.Mel_d.domain.usecases.addTicketToMyList.DefaultAddTick
etToMyListUseCase
import
```

```
com.example.Mel_d.domain.usecases.allTickets.DefaultAllTicketsUseC
ase
import
com.example.Mel_d.domain.usecases.findTickets.DefaultFindTicketsUs
eCase
import
com.example.Mel_d.domain.usecases.updateTicket.DefaultUpdateTicket
UseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import java.lang.Exception
import javax.inject.Inject

@HiltViewModel
class AllTicketsViewModel @Inject constructor(
    private val allTicketsUseCase: DefaultAllTicketsUseCase,
    private val findTicketsUseCase: DefaultFindTicketsUseCase,
    private val updateTicketUseCase: DefaultUpdateTicketUseCase,
    private val addTicketToMyListUseCase:
DefaultAddTicketToMyListUseCase
) : ViewModel() {
    private val _tickets =
MutableLiveData<Result<List<TicketEntity>>>()
    val tickets: LiveData<Result<List<TicketEntity>>> = _tickets

    private val _addSuccess =
MutableLiveData(Result.success(false))
    val addSuccess: LiveData<Result<Boolean>> = _addSuccess

    fun getTickets(searchValue: Strg?) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                _tickets.postValue(
                    if (searchValue.isNullOrEmpty()) {
                        allTicketsUseCase.getAllTickets()
                    } else {

findTicketsUseCase.findTickets(searchValue.toString())
                    }
                )
            } catch (e: Exception) {
                _tickets.postValue(Result.failure(e))
                Log.e("AllTicketsViewModel", e.toString())
            }
        }
    }

    fun saveAndUpdateTicket(ticketEntity: TicketEntity) {
        viewModelScope.launch(Dispatchers.IO) {
            try {
                val awaitAll = awaitAll(
                    async {
```

```
                        updateTicketUseCase.updateTicket(
                            TicketEntity(
                                _id = ticketEntity._id ?: 0L,
                                route = ticketEntity.route,
                                departureTime =
ticketEntity.departureTime,
                                isSelected = true
                            )
                        )
                    },
                    async {

addTicketToMyListUseCase.addTicketToMyList(
                            MyTicketEntity(
                                _id = ticketEntity._id ?: 0L,
                                route = ticketEntity.route,
                                departureTime =
ticketEntity.departureTime,
                                isSelected = true
                            )
                        )
                    }
                )

                val result = awaitAll[0].isSuccess &&
awaitAll[1].isSuccess
                if (result) {
                    _addSuccess.postValue(Result.success(true))
                } else {

_addSuccess.postValue(Result.failure(Exception()))
                }
            } catch (e: Exception) {
                Log.e("AllTicketsViewModel", e.toString())
                _addSuccess.postValue(Result.failure(e))
            }
        }
    }
}
```

MyTicketsFragment.kt:

```
package com.example.Mel_d.presentation.myTickets

import android.annotation.SuppressLint
import com.example.Mel_d.data.roomDB.data.MyTicketEntity
import com.example.Mel_d.databinding.FragmentMyTicketsBinding
import dagger.hilt.android.AndroidEntryPoint
import javax.inject.Inject

@AndroidEntryPoint
class MyTicketsFragment : Fragment() {
    private val binding by lazy {
        FragmentMyTicketsBinding.inflate(layoutInflater)
```

```
    }

    private val myTicketsViewModel: MyTicketsViewModel by
viewModels()

    @SuppressLint("NotifyDataSetChanged")
    private fun setupRecyclervw() {
        binding.myTicketsRecyclervw.apply {
            adapter = myTicketsAdapter
            layoutManager = LinearLayoutManager(requireContext())
        }

        myTicketsAdapter.onItemClickListener = {
            myTicketsViewModel.deleteMyTicketAndUpdate(it)
            myTicketsAdapter.items.remove(it)
            myTicketsAdapter.notifyDataSetChanged()
        }
    }

    private fun getUserTickets() {
        myTicketsViewModel.showMyTickets()
    }

    @SuppressLint("NotifyDataSetChanged")
    private fun observeViewModel() {
        myTicketsViewModel.apply {
            myTicketsLiveData.observe(viewLifecycleOwner) { result
->
                if (result.isSuccess) {
                    val items = result.getOrDefault(listOf())
                    if (items.isNotEmpty()) {
                        binding.noBookedTickets.visibility =
View.GONE
                        myTicketsAdapter.items = items as
ArrayList<MyTicketEntity>
                        myTicketsAdapter.notifyDataSetChanged()
                    } else {
                        binding.noBookedTickets.visibility =
View.VISIBLE
                    }

                } else {
                    Toast.makeText(
                        requireContext(),

result.exceptionOrNull()?.message.toString(),
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
        }
    }
}
```

UserInfoFragment.kt:

```
package com.example.Mel_d.presentation.userInfo

import android.app.DatePickerDialog
import androidx.fragment.app.viewModels
import com.example.Mel_d.databinding.FragmentUserInfoBinding
import com.example.Mel_d.presentation.auth.LoginActivity
import
com.example.Mel_d.presentation.bottomNavigActivity.BottomNavigActi
vity
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class UserInfoFragment : Fragment() {

    private val binding by lazy {
        FragmentUserInfoBinding.inflate(layoutInflater)
    }

    private var datePickerCallback =
        DatePickerDialog.OnDateSetListener { view, year,
monthOfYear, dayOfMonth ->
            dateAndTime.set(Calendar.YEAR, year)
            dateAndTime.set(Calendar.MONTH, monthOfYear)
            dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth)
            setUserDateOfBirth()
        }

    private val dateAndTime: Calendar by lazy {
Calendar.getInstance() }

    private val userInfoViewModel: UserInfoViewModel by
viewModels()

    private fun setupClicks() {
        binding.exitButton.setOnClickListener {
            userInfoViewModel.logOut()
            (activity as BottomNavigActivity).finish()
            startActivity(Intent(requireContext(),
LoginActivity::class.java))
        }

        binding.editProfileButton.setOnClickListener {
            if (userInfoViewModel.isEditModeLiveData.value ==
true) {
                userInfoViewModel.changeEditMode(false)
                binding.editProfileText.text = "Редагувати"
                userInfoViewModel.saveChanges()

            } else {
                userInfoViewModel.changeEditMode(true)
                binding.editProfileText.text = "Зберегти зміни"
```

```
            }
        }

        binding.dateOfBirth.setOnClickListener {
            if (userInfoViewModel.isEditModeLiveData.value ==
true) {
                DatePickerDialog(
                    requireContext(), datePickerCallback,
                    dateAndTime[Calendar.YEAR],
                    dateAndTime[Calendar.MONTH],
                    dateAndTime[Calendar.DAY_OF_MONTH]
                ).show()
            }
        }
    }

    private fun setupDataBinding() {
        binding.viewModel = userInfoViewModel
    }

    private fun getUserInfo() {
        userInfoViewModel.getUserData()
    }

    private fun setUserDateOfBirth() {
        val format = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault())

binding.dateOfBirth.setText(format.format(dateAndTime.time))
    }

    private fun observeViewModel() {

userInfoViewModel.progressLiveData.observe(viewLifecycleOwner) {
            if (it) {
                binding.allMask.visibility = View.VISIBLE
                binding.progressBar.visibility = View.VISIBLE
            } else {
                binding.allMask.visibility = View.GONE
                binding.progressBar.visibility = View.GONE
            }
        }


userInfoViewModel.isEditButtonEnabled.observe(viewLifecycleOwner)
{
            if (userInfoViewModel.isEditModeLiveData.value ==
true)
                binding.editProfileButton.isClickable = it
        }

userInfoViewModel.isEditModeLiveData.observe(viewLifecycleOwner) {
            binding.apply {
```

```
                binding.nameEditText.isFocusableInTouchMode = it
                binding.surnameEditText.isFocusableInTouchMode =
it
                binding.emailEditText.isFocusableInTouchMode = it
            }

        }
    }
}
```

UserInfoViewModel.kt:

```
package com.example.Mel_d.presentation.userInfo

import androidx.lifecycle.*
import
com.example.Mel_d.data.sharedPreferencesManager.SharedPreferencesM
anager
import com.example.Mel_d.data.user.UserModel
import com.example.Mel_d.presentation.utils.isValidEmail
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class UserInfoViewModel @Inject constructor(
    private val sharedPreferencesManager: SharedPreferencesManager
) : ViewModel() {

    val emailLiveData = MutableLiveData("")
    val nameLiveData = MutableLiveData("")
    val surnameLiveData = MutableLiveData("")
    val dateOfBirthLiveData = MutableLiveData("")
    val progressLiveData = MutableLiveData(false)

    var isEditModeLiveData = MutableLiveData(false)

    val isEditButtonEnabled = MediatorLiveData<Boolean>().apply {

        addSource(emailLiveData) {
            validateUserData()
        }
        addSource(nameLiveData) {
            validateUserData()
        }
        addSource(surnameLiveData) {
            validateUserData()
        }
        addSource(dateOfBirthLiveData) {
            validateUserData()
        }
    }
```

```
    private fun validateUserData() {
        if (emailLiveData.value?.isValidEmail() == true
            && nameLiveData.value?.isNotEmpty() == true
            && surnameLiveData.value?.isNotEmpty() == true
            && dateOfBirthLiveData.value?.isNotEmpty() == true
        ) {
            isEditButtonEnabled.postValue(true)
        } else {
            isEditButtonEnabled.postValue(false)
        }
    }


    fun getUserData() {
        emailLiveData.value =
sharedPreferencesManager.userModel?.email
        nameLiveData.value =
sharedPreferencesManager.userModel?.name
        surnameLiveData.value =
sharedPreferencesManager.userModel?.surname
        dateOfBirthLiveData.value =
sharedPreferencesManager.userModel?.dateOfBirth
    }

    fun changeEditMode(value: Boolean) {
        isEditModeLiveData.value = value
    }

    fun saveChanges() {
        viewModelScope.launch {
            progressLiveData.postValue(true)
            delay(2000)
            sharedPreferencesManager.userModel = UserModel(
                name = nameLiveData.value,
                surname = surnameLiveData.value,
                dateOfBirth = dateOfBirthLiveData.value,
                email = emailLiveData.value
            )
            progressLiveData.postValue(false)

        }
    }

    fun logOut() {
        sharedPreferencesManager.userModel = null
    }
}
```

Декларація щодо унікальності текстів роботи

та невикористання матеріалів інших авторів без посилань

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)