ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи

на ступінь вищої освіти магістр

на тему: «**РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ НЕЗАЛЕЖНИХ ТРЕНУВАНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ»**

Виконав: студент 7 курсу, групи ППЗМ–71

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Мельник Т.С.

(прізвище та ініціали)

Керівник Щербина І.С.

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Нормоконтроль_____

(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра – Інженерії програмного забезпечення

Ступінь вищої освіти - «Магістр»

Спеціальність - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

Негоденко О.В.

«____» _____2021 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

МЕЛЬНИКУ ТАРАСУ СЕРГІЙОВИЧУ

(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка рекомендаційної системи для організації незалежних тренувань з використанням штучного інтелекту

Керівник роботи Щербина І.С.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «12» березня 2021 року № 65.

2. Строк подання студентом роботи «01» червня 2021 року

3. Вхідні дані до роботи:

3.1 Вимоги до кваліфікаційної роботи магістра з актуальних завдань спеціальності;

3.2 Нормативні матеріали (стандарти, ГОСТи);

3.3 Технічнівимоги;

3.4 Науково-технічна література з питань, пов'язаних з темою роботи.

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

4.1 Порівняльний аналіз результатів, отриманих іншими авторами;

4.2 Методика дослідження;

4.3 Результати дослідження;

4.4 Висновки.

5. Перелік графічного матеріалу:

5.1 Статистика росту ШІ

5.2 Переваги технологій ШІ

5.3 Структура рекомендаційної системи

5.4 Класифікація рекомендаційних систем

5.5 Фільтрація вмісту

5.6 Комбінування методів

5.7 Відповідальності сутностей

5.6 Приклад реалізації

6. Дата видачі завдання «15» жовтня 2020 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
	Підбір науково-технічної літератури	16.10-20.10	Виконано
	Дослідження історії розвитку штучного інтелекту	16.10-27.10	Виконано
	Аналіз процесу роботи рекомендаційних систем	16.10-30.10	Виконано
	Розробка додатку згідно прототипу	21.10-5.03	Виконано
	Вступ, висновки, реферат	11.03-16.03	Виконано
	Розробка обов’язкових демонстраційних матеріалів	10.04-28.04	Виконано
	Попередній захист роботи	03.06-04.06	
	Здача роботи	18.06.2021	

Студент _____ Мельник Т.С.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Щербина І.С.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи 90 с., 20 рис., 34 джерел.

Дана робота містить опис програми для мобільного пристрою на платформі Android. Створений мобільний додаток виконує такі задачі:

- введення даних для авторизації;
- завантаження даних з сервера;
- обробка та вивід даних с сервера;
- відправлення даних на сервер .

Програма взаємодіє з користувачем завдяки графічною інтерфейсу користувача. При розробці мобільного додатку було використано мову Kotlin та Java у середовищі програмування Android Studio з використанням системи автоматичної збірки проекту Gradle

Об'єкт дослідження – рекомендаційна система для організації незалежних тренувань.

Предмет дослідження – є додаток з використанням штучного інтелекту.

Мета роботи – підвищення ефективності побудови формування рекомендацій на основі ітеративної оцінки результатів роботи рекомендаційних систем.

Методи дослідження – Методи оцінки рекомендаційних систем. Розробка інтерфейсу системи вимагає використання теорії алгоритмічних мов. Дослідження характеристик розробленої системи планується виконувалося за допомогою теорії статистичного аналізу. удосконалення оцінювання рекомендаційних систем на основі колаборативної фільтрації на основі пропущених даних з неявним зворотним зв'язком.

Галузь використання – організація групових спортивних тренувань.

ЗМІСТ

ВСТУП

Актуальність дослідження. Враховуючи темпи сучасного життя, безумовно можна стверджувати, що єкономія часу для людини стає досить актуальним моментом. А мобільні програми якраз розроблені, щоб заощадити вільний час користувача. Зазвичай коистувачі, які відвідують спортинвні майданчики мають певні незручності, а саме:

1. Переповненість спортивного поля;
2. Недостатня кількість гравців для спортивних змагань чи інших спільних заходів;
3. Неможливість проведення деяких закритих заходів, наприклад, коли команда хотіла тренуватися.
4. Через відкритість спортивної зони деякі люди можуть надмірно користуватися спортивними спорудами та своїм спортивним спорядженням. З цієї причини деякі люди не зможуть користуватися майданчиком та спортивними спорудами, розташованими на ньому;
5. Якщо групи людей з різними інтересами виходять на поле, це може призвести до неможливості використання або втрати часу однією з груп людей;
6. Велика кількість людей та дезорганізація можуть призвести до того, що на матч прийдуть гравці з різною кваліфікацією. Це може призвести до незграбності у грі як тих, хто грає менш добре, так і тих, хто набагато сильніший у дії.

Використовуючи спортивний додаток, користувач зможе уникнути таких організаційних питань, як бронювання, створення спортивних заходів, вибір місця та часу проведення спортивного заходу. Штучний інтелект можна визначити як наукову дисципліну, що займається моделюванням інтелектуальної поведінки. У цього визначення є один суттєвий недолік - поняття інтелекту важко пояснити. Проблема визначення штучного інтелекту зводиться до проблеми визначення інтелекту в цілому: чи він єдиний, чи цей термін містить набір різнобічних здібностей? Штучний інтелект забезпечує інструмент і тест-модель для теорії інтелекту: ці теорії можна сформулювати мовою комп’ютерних програм, а потім

перевірити. На сьогодні розроблено багато алгоритмів та підходів до створення штучного інтелекту. Системи рекомендацій - це програми, основною метою яких є формування рекомендацій щодо різних продуктів чи послуг для користувачів на основі їхніх уподобань. Веб-сайти збирають інформацію про уподобання користувачів і намагаються запропонувати їм корисні продукти. В даний час існує багато методів створення рекомендацій, але кожен має свої переваги та недоліки. Тому дослідження в цій галузі є важливими.

Ця проблема особливо важлива для нових, нещодавно розроблених систем. Проблема точності оцінок часто виникає у разі холодного старту, оскільки нові об'єкти або користувачі ускладнюють створення точних рекомендацій. Методи машинного навчання відіграють важливу роль у багатьох аспектах сучасного суспільства - від веб-пошуку до фільтрування в соціальних мережах, які визначають актуальність дослідження.

Метою дослідження є підвищення ефективності побудови формування рекомендацій на основі ітеративної оцінки результатів роботи рекомендаційних систем.

Завдання дослідження:

1. Провести огляд літератури, з обраної тематики;
2. Дослідити історію розвитку штучного інтелекту
3. Проаналізувати процес роботи рекомендаційних систем
4. Розробити додаток згідно прототипу;
5. Проаналізувати отримані дані.

Об’єктом дослідження є рекомендаційна система для організації незалежних тренувань.

Предметом дослідження є додаток з використанням штучного інтелекту

Методи дослідження: Методи оцінки рекомендаційних систем.

Розробка інтерфейсу системи вимагає використання теорії алгоритмічних мов. Дослідження характеристик розробленої системи планується виконувалося за допомогою теорії статистичного аналізу, удосконалення оцінювання

рекомендаційних систем на основі колаборативної фільтрації на основі пропущених даних з неявним зворотним зв’язком.

СТРУКТУРА РОБОТИ: Магістерська робота містить 104 сторінок, 20 рисунків, 1 таблицю, 1 додаток.

1 ШТУЧНИЙ ІНТЕЛЕКТ У СУЧАСНОМУ СВІТІ

1.1 Поняття «штучний інтелект»

Штучний інтелект (ШІ) - це моделювання процесів людського інтелекту за допомогою машин, зокрема комп'ютерних систем. Ці процеси включають навчання, міркування та самокорекцію. Деякі програми включають експертні системи, розпізнавання мови та машинне бачення [2].

Штучний інтелект можна класифікувати як слабкий або сильний. Слабкий штучний інтелект, також відомий як вузький штучний інтелект, - це система, яка розроблена і підготовлена до конкретного завдання. Віртуальні особисті помічники, такі як Apple Siri, є формою слабкого ШІ. Сильний штучний інтелект, також відомий як загальний штучний інтелект, - це система, яка має узагальнені когнітивні здібності людини. Коли виникає невідоме завдання, потужна система може знайти рішення без втручання людини. Оскільки вартість обладнання, програмного забезпечення та персоналу для ШІ може бути дорогою, багато постачальників включають компоненти ШІ у свою стандартну пропозицію, а також надають доступ до платформ ШІ як послугу (AIaaS). Штучний інтелект як послуга дозволяє приватним особам та компаніям експериментувати з ШІ для різних бізнес-цілей та вибору з різних платформ. Популярні пропозиції хмарного ШІ включають Amazon AI, IBM Watson Assistant, Microsoft Cognitive Services та Google AI.

Хоча інструменти штучного інтелекту пропонують ряд нових функціональних можливостей для бізнесу, їх використання породжує етичні проблеми [4]. Це тому, що алгоритми глибокого навчання, які лежать в основі багатьох передових інструментів, настільки ж розумні, як і дані, які вони отримують під час навчання. Оскільки людина вибирає, які дані використовувати для підготовки програми, потенціал людського упередження є невід'ємним і повинен ретельно контролюватися. Деякі

експерти в галузі вважають, що термін штучний інтелект занадто тісно пов'язаний із популярною культурою, що викликає нереальні побоювання спільноти щодо ШІ та неймовірні сподівання щодо того, як він змінить робоче місце та життя в цілому. Вчені та маркетологи сподіваються, що доповнений інтелект, який має більш нейтральні відтінки, допоможе людям зрозуміти, що ШІ просто вдосконалює продукти та послуги, а не замінює людей, які ними користуються. Аренд Гінце, доцент кафедри інтегративної біології та інформатики та техніки в Мічиганському державному університеті, класифікує штучний інтелект на чотири типи - від виду систем, які існують сьогодні, до розумних систем, які ще не існують. Його категорії такі [6]:

- Реактивні машини. Прикладом може служити Deep Blue, шахова програма IBM, яка перемогла Гаррі Каспарова в 1990-х. Deep Blue може ідентифікувати фігури в шахи і робити прогнози, але він не має пам’яті і не може використовувати минулий досвід для визначення майбутніх. Він аналізує можливі ходи - свій і суперника - і обирає найбільш стратегічний крок. Google Deep Blue та AlphaGO розроблені для вузьких цілей і не можуть бути легко застосовані інакше. На малюнку 1.1 показано суперкомп’ютер Deep Blue [7].

Обмежена пам’ять. Використання минулий досвіду для майбутніх рішень. Деякі функції прийняття рішень сконструйовані таким чином. Спостереження інформують про заходи, які відбудуться найближчим часом, наприклад. Теорія розуму. Цей психологічний термін означає розуміння того, що інші мають власні переконання, бажання та наміри, що впливають на рішення, які вони приймають. Цей вид штучного інтелекту ще не існує. Самосвідомість. У цій категорії системи штучного інтелекту мають почуття себе, мають свідомість. Машини з самосвідомістю розуміють свій сучасний стан і можуть використовувати інформацію, щоб зробити висновок про те, що відчувають інші.

ШІ має місце зараз в багатьох сферах. Розглянемо їх:

Автоматизація: завдяки якій система або процес запускаються автоматично. Наприклад, автоматизований робототехнічний процес (RPA) може бути запрограмований на виконання багатьох повторюваних завдань, які зазвичай виконується людиною. RPA відрізняється від IT-автоматизації тим, що може адаптуватися до мінливих обставин.

Машинне навчання: вивчення того, як змусити комп'ютер працювати без програмування. Поглиблене навчання - це підмножина машинного навчання, яку, простіше кажучи, можна розглядати як автоматизацію прогнозної аналітики. Існує три типи алгоритмів машинного навчання:

Контрольоване навчання: Набори даних позначаються тегами, щоб шаблони могли бути виявлені та використані для ідентифікації нових наборів даних.

Навчання без нагляду: дані не мають маркеру та діляться за подібністю чи відмінностями. Укріплене навчання: набори даних немарковані, але після виконання дії або декількох дій система ШІ отримує зворотній зв'язок.

Machine Vision: вивчення того, як дозволити комп'ютерам бачити. Ця технологія фіксує та аналізує візуальну інформацію за допомогою камери, аналого-цифрового перетворення та цифрової обробки сигналу. Його часто порівнюють із людським зором, але машинне бачення не пов'язане з біологією, і його можна запрограмувати, щоб бачити крізь стіни, наприклад. Він використовується в широкому спектрі застосувань, від ідентифікації підписів до аналізу медичних зображень. Комп'ютерний зір, який зосереджений на машинній обробці зображень, часто асоціюється з машинним зором.

Обробка природної мови (NLP): обробка людської мови, а не комп'ютерної мови, за допомогою комп'ютерної програми. Одним з найдавніших і найвідоміших прикладів NLP є виявлення спаму, який аналізує тему та текст електронного листа та вирішує, чи є це непотрібним. Сучасні підходи до НЛП базуються на машинному навчанні. Завдання НЛП включають переклад тексту, аналіз настрою та розпізнавання мови [9].

Робототехніка: машинобудівна галузь, присвячена проектуванню та виробництву роботів. Роботи часто використовуються для виконання завдань, які

важко виконувати людям або виконуються безперервно. Вони використовуються на складальних лініях для виробництва автомобілів або в NASA для переміщення великих предметів через космос. Вчені також використовують машинне навчання для створення роботів, які можуть взаємодіяти соціально.

Самостійні машини: використовуйте комбінацію комп'ютерного зору, розпізнавання зображень та поглибленого навчання, щоб розвинути здатність автоматично управляти транспортним засобом на певній смузі та уникати несподіваних перешкод, таких як пішоходи. [10].

До того ж, ШІ часто має місце в інших галузях людської діяльності. Штучний інтелект в охороні здоров'я. Найбільші установи займаються покращенням результатів для пацієнтів та зменшенням витрат. Компанії використовують машинне навчання, щоб ставити кращі та швидші діагнози, ніж люди.

Однією з найвідоміших технологій охорони здоров'я є IBM Watson. Він розуміє природну мову і здатний відповідати на поставлені ним запитання. Система витягує дані пацієнта та інші доступні джерела даних, щоб сформулювати гіпотезу, яка потім представляє схему перевірки. Інші програми ШІ включають чати, комп'ютерну програму, яка використовується в Інтернеті для відповіді на запитання та надання допомоги клієнтам, призначення зустрічей або допомоги пацієнтам у процесі виставлення рахунків, а також віртуальні фельдшери для надання основної медичної інформації.

ШІ і бізнес. Автоматизація роботизованих процесів використовується для дуже повторюваних завдань, які зазвичай виконуються людьми. Алгоритми машинного навчання інтегровані з аналітикою та CRM-платформами, щоб розкрити інформацію про те, як найкраще обслуговувати своїх клієнтів. Чат-боти були включені в веб-сайти для надання негайного обслуговування клієнтів. Автоматизація робочих місць також стала темою розмов між науковцями та IT-аналітиками. [11].

ШІ і освіта. Штучний інтелект може автоматизувати класифікацію, надаючи вчителям більше часу. Штучний інтелект може оцінювати учнів та

пристосовуватися до їхніх потреб, допомагаючи їм працювати у своєму власному темпі. Вчителі можуть надати додаткову підтримку студентам, переконавшись, що вони не відхиляються від курсу. Штучний інтелект може змінити, де і як навчаються учні, можливо, навіть замінивши деяких вчителів.

ШІ і фінанси. ШІ у програмах особистого фінансування, таких як монетний двір та податок на турбо, порушує фінансові установи. Такі програми збирають особисту інформацію та надають фінансові консультації. Інше програмне забезпечення, наприклад IBM Watson, було використано в процесі купівлі житла. Сьогодні це програмне забезпечення виконує більшість аукціонів на Уолл-стріт [12].

ШІ і право. Процес документів у законі важливий для людей. Автоматизація цього процесу значить ефективніше використання часу. Стартапи також створюють комп'ютерних асистентів із питаннями-відповідями, які можуть розпізнавати відповіді на програмні питання, вивчаючи систематику пов'язану з базою даних.

ШІ і виробництво. Промислові роботи використовувались для виконання конкретних завдань і тримались окремо від працівників, але з розвитком технологій, які змінилися.

З використанням штучного інтелекту в галузі самокерованих автомобілів виникають питання безпеки та етики [13]. Автомобілі можуть бути зламані, і коли автономний транспортний засіб потрапляє в аварію, відповідальність незрозуміла. Автономні транспортні засоби також можуть опинитися в ситуації, коли аварія неминуча, що змушує програму прийняти етичне рішення про те, як мінімізувати шкоду.

Ще однією серйозною проблемою є можливість зловживання інструментами штучного інтелекту. Хакери починають використовувати передові інструменти машинного навчання для доступу до вразливих систем, ускладнюючи безпеку, що перевищує сучасний рівень техніки.

Поглиблені аудіо- та відеосистеми на основі глибокого навчання також становлять загрозу для знаменитостей, оскільки ці інструменти

використовуються для створення так званих глибокі фейки, тобто переконливо сфабриковані відео публічних діячів, які говорять або роблять те, чого ніколи не було.

Незважаючи на ці потенційні небезпеки, існує мало законів, що регулюють використання засобів штучного інтелекту, і там, де закони існують, вони, як правило, поширюються лише на опосередкований інтелект. Наприклад, федеральні правила справедливого кредитування вимагають від фінансових установ пояснювати кредитні рішення потенційним клієнтам, що обмежує ступінь використання кредиторами глибоких алгоритмів навчання, які за своєю суттю є непрозорими. Європейський регламент GDPR встановлює жорсткі обмеження щодо того, як компанії можуть використовувати споживчі дані, ускладнюючи вивчення та функціонування багатьох програм, що стосуються споживачів.

У 2016 році Національна рада з питань науки і техніки опублікувала звіт, що вивчає потенційну роль державного регулювання у розвитку штучного інтелекту, але не рекомендувала розглядати конкретні нормативні акти. З того часу питанню не приділялося особливої уваги з боку законодавців [14].

1.2 Історія розвитку штучного інтелекту

ШІ - молода, шістдесятирічна дисципліна, що являє собою сукупність наук, теорій і методів, включаючи математичну логіку, статистику, ймовірність та обчислювальну нейробіологію, метою якої є імітація когнітивних здібностей людини. Він був створений під час Другої світової війни [15], його розвиток тісно пов’язаний з комп’ютерними технологіями та змусив комп’ютери виконувати дедалі складніші завдання, які раніше могли доручати лише людям. Однак ця автоматизація залишається далекою від людського інтелекту в суворому сенсі, що робить цей заголовок

вразливим для критики з боку деяких експертів. Заключний етап їхніх досліджень ("сильний" штучний інтелект, тобто здатність повністю автономно контекстуалізувати дуже різні спеціалізовані проблеми) абсолютно незрівнянний із сучасними досягненнями ("слабкий" або "сильний" ШІ надзвичайно ефективний у своїй галузі науки). "Сильний" штучний інтелект, який лише набуває своєї форми у науковій фантастиці, потребує успіхів у фундаментальних дослідженнях (а не лише ефективності), щоб мати змогу моделювати весь світ.

Однак з 2010 року дисципліна пережила новий бум, значною мірою завдяки значному збільшенню обчислювальної потужності та доступу до величезних обсягів даних. Обіцянки, поновлення та страхи, про які іноді фантазують, ускладнюють об’єктивне розуміння цього явища. Короткі історичні нагадування можуть допомогти дисциплінувати та підвищити обізнаність щодо поточних дебатів. На рисунку 1.2 представлена інфографіка, що зображує історію штучного інтелекту.

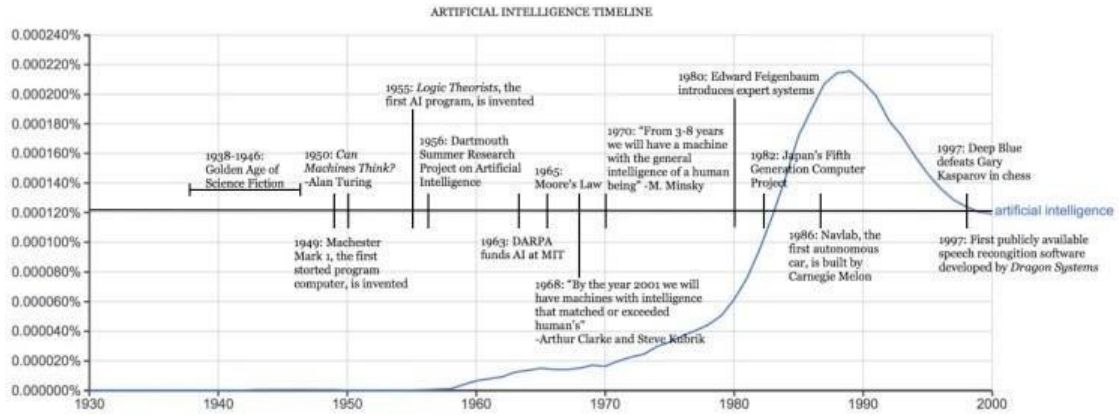

Рисунок 1.2 – Інфографіка розвитку штучного інтелекту

1940-1960: народження штучного інтелекту та пробудження кібернетики. 2520/5000 Період між 1940 та 1960 роками був сильно ознаменований поєднанням технологічних розробок (прискорювачем яких була Друга світова війна) та прагненням зрозуміти, як поєднати

функціонування машин та органічних істот. Для Норберта Вінера, піонера кібернетики, його метою було об’єднати математичну теорію, електроніку та автоматику як «цілу теорію управління та зв’язку як у тварин, так і в машинах». Незадовго перша математична та комп’ютерна модель біологічного нейрона (формальний нейрон) була розроблена Уорреном Маккаллохом та Уолтером Пітсом ще в 1943 році [16].

Джон Фон Нейман і Алан Тьюрінг не засновували термін «ШІ», але були засновниками технології, яка стоїть за ним: вони здійснили перехід від комп'ютерів до десяткової логіки 19 століття і машин до двійкової логіки. Роблячи це, двоє дослідників формалізували архітектуру сучасних комп’ютерів і продемонстрували, що це універсальна машина, здатна виконувати все, що запрограмовано. Тюрінг, навпаки, вперше звернувся до питання можливого машинного інтелекту у своїй відомій статті "Обчислення та інтелект" 1950 року та описав "гру імітації", в якій людина повинна вміти розрізняти телелітичний діалог, незалежно від того, є він чи ні. спілкування з людиною або з машиною. Якою б суперечливою не була ця стаття (цей "тест Тьюрінга" [17], здається, не підходить для багатьох експертів), його часто цитують як джерело для допитів щодо інтерфейсу людина-машина.

Термін "ШІ" визначає "побудову комп'ютерних програм, які вирішують завдання, які зараз люди виконують більш задовільно, оскільки вони вимагають розумових процесів на високому рівні, таких як перцептивне навчання, організація пам'яті та критичне мислення".

Хоча технологія все ще захоплююча та перспективна, її популярність знизилася на початку 1960-х рр. Машини мали дуже мало пам'яті, що ускладнювало використання комп'ютерної мови. Однак сьогодні є деякі основи, такі як дерева рішень для усунення несправностей: IPL, мова обробки інформації.

Herbert Simon, економіст і соціолог, передбачив в 1957 році, що штучний інтелект переможе людину в шахах протягом наступних 10 років.

Бачення виявилося точним лише через 30 років. 1980-1990 та експертні системи. У 1968 році Stanley Kubrick зняв фільм

"Космічна одісея 2001", в якій комп'ютер - HAL 9000 (лише один лист від IBM) узагальнює цілу купу етичних питань, поставлених ШІ: це буде високий рівень витонченості, користь для людства чи загроза? Вплив фільму, звичайно, буде не науковим, але допоможе популяризувати тему, а також автору наукової фантастики Philip K. Dick, який ніколи не перестане замислюватися, чи машини коли-небудь відчуватимуть емоції.

Саме з появою перших мікропроцесорів наприкінці 1970-х років ШІ знову з'явився і увійшов у золотий вік експертних систем. Цей шлях був фактично відкритий в MIT в 1965 році завдяки DENDRAL (експертна система, що спеціалізується на молекулярній хімії), і в Uniwersytecie Stanforda 1972 завдяки MYCIN (що спеціалізується на діагностиці захворювань).

Планування передбачало масштабний розвиток, але манія знову зазнала краху наприкінці 1980-х - на початку 1990-х. Програмування таких знань насправді вимагало багато зусиль, і, від 200 до 300 правил, був ефект "чорного ящика", де незрозуміло, як машина аргументувала. Тому розробка та обслуговування стали надзвичайно проблематичними і, перш за все, швидшими, і можна було використовувати багато інших, менш складних та дешевих методів. Слід пам'ятати, що в 1990-х термін штучний інтелект став майже табу, а його скромніші варіації навіть увійшли до університетської мови, наприклад, «сучасна інформатика».

Успіх Deep Blue (експертна система IBM) у травні 1997 року. У шаховій партії з Harry Kasparov він через 30 років здійснив пророцтво Герберта Саймона 1957 року, але не підтримав фінансування та розвиток цієї форми ШІ. Операція Deep Blue була заснована на систематизованому алгоритмі грубої сили, де всі можливі рухи оцінювались і зважувались. Побиття людини залишалося дуже символічним протягом історії, але Deep

Blue насправді міг розглянути лише обмежений ланцюжок (той, що стосується правил шахів), дуже далекий від моделювання складності світу.

Новий бум розпочався у 2010 р. На основі великої кількості даних, що підлягають обробці, та нових обчислювальних можливостей [20]. Два фактори пояснюють новий бум цієї дисципліни в 2010 році:

- доступ до величезних обсягів даних. Наприклад, щоб мати можливість використовувати алгоритми для класифікації зображень та розпізнавання, довелося зробити вибірку самостійно. Сьогодні ви можете знайти мільйони за допомогою простого пошуку Google.
- Потім виявлення дуже високої продуктивності комп'ютерних процесорів відеокарт для прискорення обчислення алгоритмів навчання. Процес був дуже повторюваним, обробка всієї вибірки могла зайняти десь від тижнів до 2010 року. Обчислювальна потужність цих карток досягла значного прогресу з обмеженими фінансовими витратами.

Це нове технологічне обладнання дозволило досягти значного успіху в суспільстві та збільшити фінансування: у 2011 році IA Watson від IBM переможе в іграх з 2 чемпіонами в Danger! "У 2012 році Google X (Google Search Lab) зможе мати розпізнавання AI-cat на відео. Для цього останнього завдання було використано понад 16 000 процесорів, але потенціал надзвичайний: машина навчиться щось розрізняти. У 2016 році , AlphaGO (ШІ, що спеціалізується на Google Games) перемагає чемпіона Європи (Фан Хуей) і чемпіона світу (Лі Седол), а потім самого себе (AlphaGo Zero). Зверніть увагу, що Го має комбінаторику набагато більше, ніж шахи (більший, ніж кількість частинок). у Всесвіті), і отримання таких значущих результатів неможливо при сильній силі (як це було у випадку з Deep Blue у 1997 р.).

Повна зміна парадигми для експертних систем. Підхід став індуктивним: мова вже не про правила кодування, як у експертних системах,

а про здатність комп'ютерів виявляти їх окремо, співвідносячи та класифікуючи з величезного обсягу даних.

Серед методів машинного навчання глибоке навчання видається найбільш перспективним для багатьох програм (включаючи розпізнавання голосу та зображень). У 2003 році Джеффрі Хінтон (Університет Торонто), Джошуа Бенджо (Університет Монреаля) та Ян Лекун (Університет Нью-Йорка) вирішили розпочати дослідницьку програму, спрямовану на модернізацію нейронних мереж. Одночасні експерименти в Microsoft, Google та IBM за допомогою лабораторії Хінтона в Торонто показали, що такий тип навчання зміг зменшити вдвічі кількість помилок розпізнавання мови.

1.3 Штучний інтелект в сучасному світі

ШІ - галузь інформатики, яка вивчає розвиткок машин з інтелектом, мисленням та роботою як люди. Наприклад, розпізнавання мовлення, вирішення проблем, навчання та планування. 32% керівників заявляють, що розпізнавання голосу є найбільш використовуваною технологією ШІ у їхньому бізнесі. ШІ є дуже популярною темою, яка широко обговорюється у технологічних та ділових колах. Багато галузевих експертів та аналітиків стверджують, що за штучним інтелектом чи машинним навчанням - це майбутнє, але якщо ми оглянемося назад, то побачимо, що це не майбутнє - це сьогодення.

З розвитком технологій ми вже якимось чином пов'язані зі штучним інтелектом - будь то Siri, Watson чи Alexa [22]. Так, технологія перебуває в зародковому стані, і все більше компаній інвестує ресурси в машинне навчання, що свідчить про стабільне зростання продуктів і програм ШІ найближчим часом. У 2014 році в стартапи AI було інвестовано понад 300 млн доларів, що на 300% більше порівняно з попереднім роком (Bloomberg)

До 2018 року 6 мільярдів підключених пристроїв будуть активно просити про підтримку. (Гартнер). До кінця 2018 року цифрові помічники клієнтів розпізнаватимуть клієнтів за обличчям та голосом через канали та партнерів (Gartner). Штучний інтелект замінить 16% робочих місць у США до кінця десятиліття (Форрестер) - 15% власників телефонів Apple використовують розпізнавання голосу Siri. (BGR). На відміну від загальноприйнятого уявлення, штучний інтелект не обмежується лише IT-галузями або технологічними галузями, але широко використовується в інших сферах, таких як медицина, бізнес, освіта, право та виробництво.

ВИСНОВКИ ДО 1 РОЗДІЛУ

Штучний інтелект (ШІ), машинне навчання та нейронні мережі - це терміни, що використовуються для опису потужних технологій, заснованих на машинному навчанні, які можуть вирішити багато проблем у реальному світі. Порівняно з можливостями людського мозку в машинах, далеких від ідеалу (звичайно, не ідеальних для людей), останнім часом було зроблено кілька важливих відкриттів у галузі технологій ШІ та пов'язаних з ними алгоритмів. Важливу роль відіграє збільшення кількості великих зразків різноманітних даних, доступних для навчання ШІ. Галузь ШІ пронизує багато інших галузей, включаючи математику, статистику, теорію ймовірностей, фізику, обробку сигналів, машинне навчання, комп'ютерний зір, психологію, лінгвістику та науку про мозок.

Питання, пов’язані із соціальною відповідальністю та етикою створення ШІ, залучають людей, які цікавляться філософією.

2 АНАЛІЗ МЕТОДІВ ОЦІНКИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ І ПОСТАНОВКА ЗАДАЧІ

2.1 Дослідження характеристик рекомендаційних систем

На початку слід визначити систему рекомендацій. Системи рекомендацій - це програми, основним завданням яких є формування рекомендацій щодо різних продуктів чи послуг для користувачів на основі їхніх уподобань.

З появою Інтернету кількість інформації, з якою люди стикаються щодня, значно зросла. Це означає, що людям доводиться орієнтуватися через величезну кількість альтернативних варіантів, коли вони хочуть щось знайти. Але з іншого боку, власники інтернет-магазинів та послуг: їх цікавить особиста реклама та рекомендації для кожного користувача, адже такий підхід може значно збільшити прибуток компанії. Відповідно, за останні роки інтерес до розробки та вдосконалення існуючих систем направлення значно зріс.

У наш час системи рекомендацій є досить поширеними і мають багато застосувань. Перш за все, системи рекомендацій використовуються в електронній комерції, щоб допомогти користувачам вибрати правильні товари. Такі веб-сайти збирають інформацію про уподобання користувачів і намагаються запропонувати їм корисні продукти. В даний час існує багато методів створення рекомендацій, але кожен має свої переваги та недоліки. Тому дослідження в цій галузі є важливими.

Якість системи рекомендацій залежить від багатьох властивостей її рекомендацій, які зазвичай оцінюються за допомогою показників ефективності, що відображають вузьке уявлення про поведінку системи.

Найчастіше розробники додатків оцінюють релевантність рекомендації за допомогою метрик, що вимірюють здатність системи рекомендацій точно складати набір відомих переваг. Точність вимірюється резервуванням частини набору даних оцінки як набору відомих уподобань та використанням інших оцінок для навчання системи рекомендацій та надання рекомендацій.

Відомі переваги можуть бути використані певними показниками, такими як точність або відповідь, для обчислення оцінки, що представляє точність алгоритму рекомендацій для цього набору даних. Структура системи рекомендацій представлена нижче.

Рис.2.1 – Структура рекомендаційної системи

Однак показники точності дають лише приблизне одновимірне уявлення про ефективність системи рекомендацій. Зокрема, точність і реакція обмежуються кількома способами. По-перше, вони дають мало інформації про поведінку об'єктів з невідомими уподобаннями. По-друге, коли набір відомих фаворитів малий, значення, як правило, низькі, оскільки обрані фаворити навряд чи будуть включені до відомого набору переваг.

Оскільки точність і реакція не дають жодної інтуїції щодо поведінки об’єктів, які не входять до набору відомих уподобань, висновки про відносну перевагу однієї системи над іншою можуть ввести в оману. Рекомендації складаються індивідуально для кожної людини на основі їх попередньої діяльності на певному Інтернет-ресурсі або на основі попередньої діяльності. Крім того, важливо також зберегти попередніх учасників процесу.

Це важлива функція для інтернет-магазинів, а для великих каталогів, таких як Amazon, - один із небагатьох способів робити добро. Метод рекомендацій у цьому випадку не є звичайною додатковою опцією, він забезпечує легкість навігації користувачем через Інтернет-ресурси. Якщо електронний каталог містить понад 20 000 одиниць, орієнтація вже здається непропорційно складною, що робити, якщо товарів є мільйони?

Наскільки виснажливо для потенційного покупця взаємодіяти з такою сторінкою? Відповідь очевидна. І на допомогу приходить віджет для пошуку товарів, які візуально схожі на шукані, або належать до однієї товарної групи, або додаткового виробництва (коли пропонують вибрати сумку для пари взуття, наприклад). Це рішення не тільки збільшує кількість переглядів сторінки, але й позитивно впливає на конверсію.

На практиці, не тільки інтернет-магазини використовують цю техніку. Ці методи також можна легко побачити на різних платформах соціальних медіа, сайтах літератури, туристичних сайтах, новинних ресурсах, інтернет-магазинах, коротше кажучи - майже скрізь. Ця техніка дійсно дуже популярна. Рекомендаційні веб-сайти збирають різну інформацію про дану людину, використовуючи кілька методів, загальних для всіх систем.

І так, перший тип - це явний збір даних. Як можна здогадатися з назви, користувач сам надає матеріали, необхідні для роботи. Наприклад, коли системи переходу Google або інші пошукові системи просять людину оцінити різні елементи, створити список обраних для певного поля або відповісти на кілька запитань. Якщо фізична особа відмовляється самостійно надавати інформацію, може бути доцільним наступний спосіб.

Другий тип - прихований збір даних. Це шпигунська місія, за допомогою якої дії учасника процесу фіксуються програмою для подальшої обробки та використання. Програма розпізнає покупки, рейтинги на сторінках, збирає інформацію про перегляди, коментарі. Звичайно, вибір такої техніки веде до

певні етичні питання, оскільки захист персональних даних - одна з головних вимог користувачів до пошуку сайтів. Але поки факт залишається фактом - можливе якесь відстеження, і відвідувачі сайту не можуть підтвердити, що такі події насправді відбуваються.

Існують також типи систем рекомендацій, які визначаються підходами, які вони використовують.

2.2 Аналіз процесу роботи рекомендаційних систем

Відгуки збираються з багатьох причин, а не лише для машинного навчання. Кожна навчальна система має певні вимоги перед поданням даних на аналіз. Вибір конкретних даних для тестування може сильно вплинути на модель навчання, з цих причин підготовка даних є важливою частиною будь-якої мети машинного навчання. Підготовка даних часто є найбільш трудомісткою частиною будь-якого проекту машинного навчання. Вибір конкретного типу фільтрації або поєднання декількох методів безпосередньо залежить від двох факторів - складності проекту та обсягу його фінансування. Наприклад, створення алгоритму для системи тематичних блогів, що перетинаються, є відносно простим і помірно дорогим завданням. Більші та різнорідніші проекти, такі як Інтернет-магазини, є дорогими, особливо якщо проблема полягає у збільшенні конверсій на дійсно значні суми. Як правило, в таких проектах неможливо обмежитися одним типом рекомендованого алгоритму, і необхідно використовувати гібридну фільтрацію, що збільшує вартість і складність розробки на порядки.

У будь-якому випадку, при створенні дизайну, який пропонує користувачеві можливість вибору конкретних предметів із загального набору, важливо враховувати швидкий прогрес у зручності використання у всіх сферах людського життя - від оптимізації сну за допомогою пристроїв, які аналізують всі процеси сну та давати рекомендації щодо його вдосконалення з метою автоматичного підбору повсякденних товарів на основі поточних потреб користувача.

Розглянемо процес роботи рекомендаційних систем на рисунку 2.2.

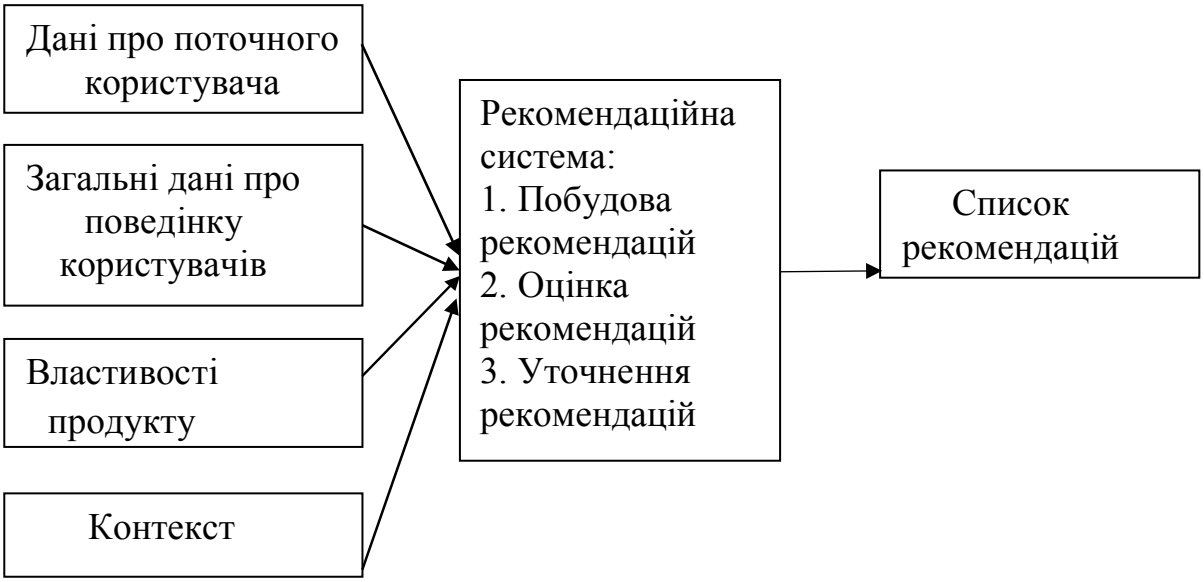

Рис. 2.2 – Процес роботи рекомендаційних систем

Дотепер для створення реферальних систем використовувались дві основні стратегії: фільтрація вмісту та спільна фільтрація. Варто зазначити, що на практиці найчастіше застосовуються гібридні методи, що поєднують переваги наступних підходів.

На малюнку нижче ми бачимо, як класифікуються системи рекомендацій.

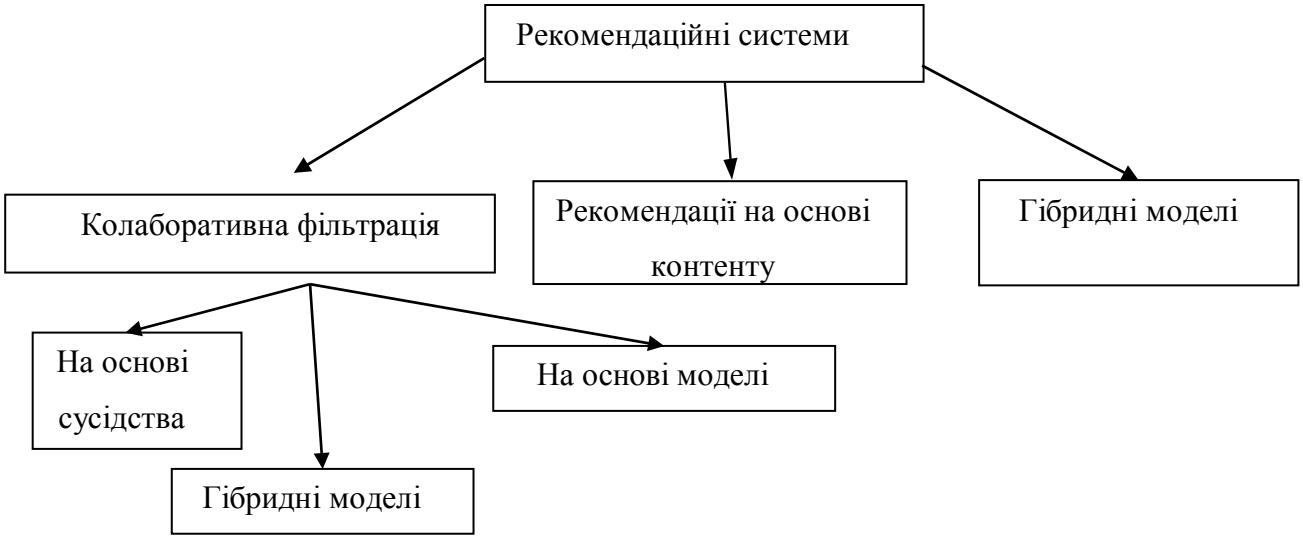

Рис.2.3 – Класифікація рекомендаційних систем

Фільтрування вмісту (рис. 2.4) базується на створенні профілю користувача та профілю об’єкта. Слід враховувати параметри об’єкта та їх відповідність уподобанням користувачів. З цією метою системи рекомендацій використовують теги (ключові слова) для опису об’єктів, а профіль користувача відображає оцінку конкретних тегів або їх комбінацій.

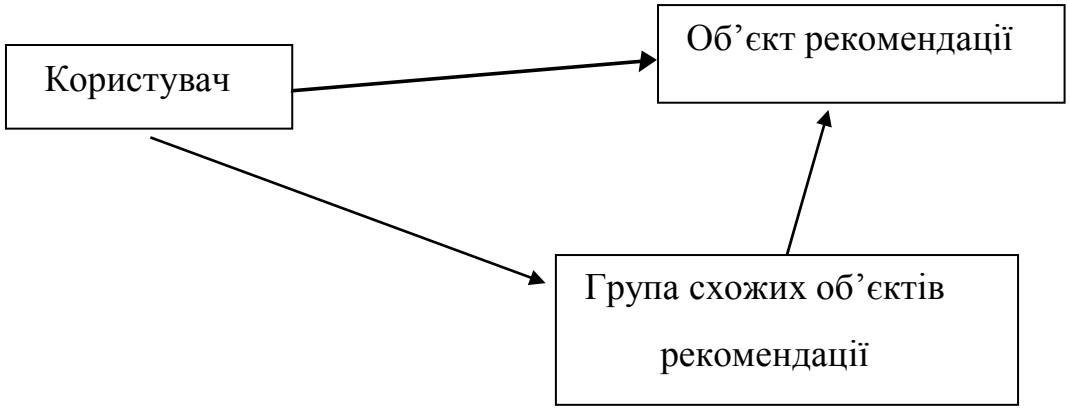

Рис. 2.4 – Фільтрація вмісту

У системах рекомендацій із фільтрацією вмісту функція задоволення користувачів h (u, s) u для певного об'єкта s визначається на основі інформації про задоволеність користувачів об'єктами si ∈ S, подібними до s. Тобто, якщо система рекомендацій є використовується на веб-сайті музичного потокового додатка, щоб рекомендувати музику користувачеві, система повинна визначити, що поєднує різні музичні композиції, які користувач раніше оцінював високо.

Потім система рекомендуватиме користувачеві музичні твори з максимальною точністю по відношенню до попередньо високо оцінених користувачем. Профіль користувача в системі створюється на основі вмісту, який споживає користувач, у вигляді набору параметрів, що визначають об'єкт s. Ці параметри, як правило, є ключовими словами та відповідними вагами для кожного об'єкта. Тому необхідно визначити ваги цих параметрів. Під час пошуку інформації показник TF-IDF є одним із найвідоміших способів визначення ваги ключових слів. Припустимо, що N - загальна кількість об’єктів, які, ймовірно, будуть рекомендовані користувачеві, і що ключове слово kj зустрічається в n об’єктах, а f, j - кількість випадків входження цього слова в об’єкт dj. Тоді TFi, j (частота терміна) - це відношення кількості повторень даного тегу до загальної кількості тегів об'єкта, тобто (1.1):

$$TF_{i.j} = \frac{f_{i,j}}{max_z f_{z,j}}. \tag{1.1}$$

Однак, якщо врахувати лише частоту даного тегу, найпоширеніші теги матимуть максимальну вагу в більшості об’єктів, що може призвести до неправильної оцінки уподобань користувачів. Щоб уникнути цього, використовується IDFi (зворотна частота документа) - обернене значення до частоти тегу в об'єкті колекції. Ми визначаємо це як (1.2):

$$IDF_i = log \frac{N}{n_i}. \tag{1.2}$$

Таким чином, вага $w_{i,j}$ у ключового слова k_i в об’єкті d_j позначається як добуток частоти входження тега на зворотну частоту об’єкта (1.3):

$$w_{i.j} = TF_{i,j} \times IDF_i\,, \tag{1.3}$$

В такому випадку контент об’єкту d_j можна визначити так (1.4):

$$Content(d_{j,i}) = (w_{1,j}, \dots, w_{k,j}). \tag{1.4}$$

Як зазначалося вище, рекомендовані системи фільтрації вмісту пропонують функції, засновані на тих, які користувач оцінив раніше. Різні об'єкти порівнюються з тими, що сподобалися користувачеві, і рекомендуються об'єкти з максимальною схожістю. Набір об'єктів, які користувач попередньо оцінив, створює визначений користувачем ContentBasedProfile (u) або, іншими словами, вектор ваги (wu, 1, ..., wu, k), де кожна вага wu, i визначає значення тегу ki користувачеві u. Таким чином, ContentBasedProfile (u) та Content (s) можуть бути представлені як вектори ID ve і $\vec{w}$ T TF-IDF, а функція задоволення користувача h (u, s) може бути представлена
як коефіцієнт косинусів векторів $\vec{w}\,u$ і $\vec{w}\,s$, де K - загальна кількість міток у системі (1.5):

$$h(u,s) = \cos(\vec{w}_u, \vec{w}_s) = \frac{\vec{w}_u \vec{w}_s}{\|\vec{w}_u\| \|\vec{w}_s\|} = \frac{\sum_{i=1}^{K} w_{i,u} w_{i,s}}{\sqrt{\sum_{i=1}^{K} w^2_{i,u}} \sqrt{\sum_{i=1}^{K} w^2_{i,s}}}, \tag{1.5}$$

На додаток до евристики, що базується головним чином на методах пошуку інформації, часто використовуються інші методи рекомендацій щодо фільтрування вмісту, такі як простий байєсівський класифікатор, різні техніки машинного навчання, включаючи нейронні мережі, дерева рішень та кластеризацію.

Однак, на відміну від методів пошуку інформації, ці методи намагаються передбачити задоволеність користувачів, використовуючи евристичний коефіцієнт косинуса як основу. Наприклад, у потоковому передаванні музики ви можете використовувати інформацію про музичні доріжки, які користувачеві подобаються і не подобаються, через наївний байєсівський категоризатор, щоб вибрати пісні, які користувач не чув, та оцінити ймовірність того, що пісня pj належить до певного класу Ci (сподобався і ні).

Спільна фільтрація (рис. 2.5) використовує відомі переваги групи користувачів, яка включає користувача, для якого ми хочемо передбачити рекомендації. Основною причиною такого підходу є те, що користувачі оцінювали деякі функції однаково в минулому і частіше оцінювали ті самі функції в майбутньому.

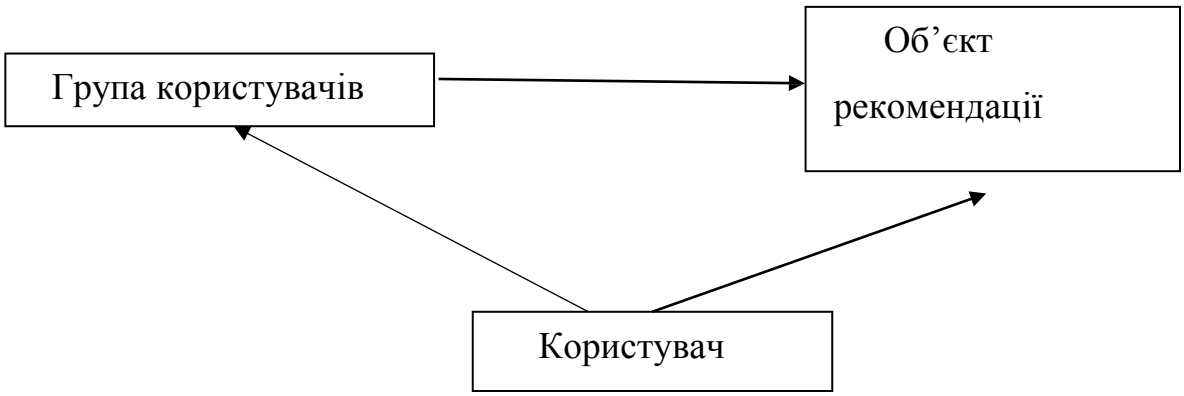

Рис. 2.5 – Колаборативна фільтрація

На відміну від фільтрації вмісту, методи спільної фільтрації намагаються передбачити, чи буде заданий об'єкт відповідати уподобанням користувача на основі оцінок інших користувачів тієї самої групи, тобто ми отримуємо оцінку задоволеності h (u, s) деякими об'єктами для користувач u, який обчислюється залежно від задоволеності h (uw, s) одним і тим же об'єктом s uu є u користувачів, які мають подібні характеристики до користувача u

Зазначимо важливу особливість методу: ознаки того, що один користувач схожий на інших, не завжди є частиною цієї системи. Це означає, що в системі потокового передавання музики групи користувачів можуть визначатися не тільки за музичними уподобаннями користувачів, а й за більш загальними характеристиками, такими як географічне розташування, профілі соціальних мереж та інші параметри.

Цей вид інформації може бути використаний для початкової оцінки схожості нового користувача з користувачами, вже зареєстрованими в системі, і може бути використаний для вирішення загальної проблеми спільної фільтрації «холодного старту».

Існує багато систем рефералів на основі користувачів, розроблених за участю науковців та комерційних структур з різних дисциплін. Така система називається "системою Грунді" - бібліотекар задає користувачам запитання на різні теми, створюючи таким чином профіль користувача та рекомендуючи книги, які можуть їм сподобатися. Якщо користувач відповів, що його не цікавить запропонована книга, система почала досліджувати причини, поглиблюючи тим самим інформацію про профілі. Таким чином, "система Грюнді" створила особисту модель для кожного користувача, завдяки якій вона могла дуже точно порадити цікаву літературу. У той же час інша система під назвою «Гобелен», замість того, щоб вказати схожість через питання, запропонувала користувачеві створити список користувачів із відповідним вибором уподобань.

Також вважається, що спільні методи фільтрації вмісту діляться на дві основні категорії: на основі пам'яті та на основі моделі.

Перша категорія алгоритмів намагається передбачити рейтинг об’єкта, використовуючи знання всіх об’єктів, які користувач вже оцінив.

Величина рейтингу $r_{u,s}$ для користувача u і об’єкта s обчислюється, як агреговане значення оцінок інших найбільш схожих N користувачів для цього самого об’єкта s (1.6):

$$r_{u,s} = \text{aggr}_{u \epsilon \hat{U}} r_{u',s}, \tag{1.6}$$

де U позначає множина N користувачів, які оцінювали об’єкт s і найбільш схожі на користувача u. В якості найпростішого прикладу функції агрегування можна привести обчислення середнього (1.7):

$$r_{u,s} = \frac{1}{N} \sum_{u \epsilon \hat{U}} r_{u',s}. \tag{1.7}$$

Найпоширенішим випадком агрегування є (1.8; 1.9):

$$r_{u,s} = k \sum_{u \epsilon \hat{U}} \text{sim}(u, u') \times r_{u',s}, \tag{1.8}$$

$$r_{u,s} = \bar{r}_u + k \sum_{u \epsilon \hat{U}} \text{sim}(u, u') \times (r_{u',s} - \bar{r}_{u'}), \tag{1.9}$$

де k використовується для нормалізації і, як правило, sim (u, u ') є мірою "подібності", яка є зворотною відстанню і в більшості випадків приймається за вагу. Таким чином, чим більше схожі користувачі u та u ', тим більший вага буде врахований при обчисленні ru, s. Ми бачимо, що різні показники подібності можуть використовуватися в різних системах рекомендацій. Давайте розглянемо два з них, які є найбільш поширеними. Є кілька труднощів із використанням зважених сум, у тому числі неможливість врахувати той факт, що різні користувачі не завжди оцінюють об'єкти з однаковим рівнем суворості.

Існує величезна кількість модифікацій, що підвищують ефективність цих методів. Наприклад, голосування за замовчуванням - одна із таких модифікацій методу, заснованого на пам’яті, згаданого раніше.

У той же час, коли вищезазначені підходи використовувались для обчислення подібності користувачів, Б. Сарвар використовував ті самі підходи для

визначення співвідношень між об’єктами для обчислення оцінок для них. Пізніше ця ідея була вдосконалена в алгоритм Top-N. Існують також емпіричні докази того, що алгоритми фільтрації вмісту здатні ефективніше працювати на витрачених обчислювальних ресурсах, надаючи не менш точні рекомендації, ніж більшість алгоритмів спільної фільтрації.

Деякі рекомендаційні системи використовують поєднання підходів до фільтрації вмісту та спільної фільтрації, які називаються гібридними методами. Вони дозволяють до певної міри уникнути недоліків обох підходів. Є кілька основних можливостей поєднання різних методів у системах рекомендацій гібридного типу, це такі на рисунку 2.6.

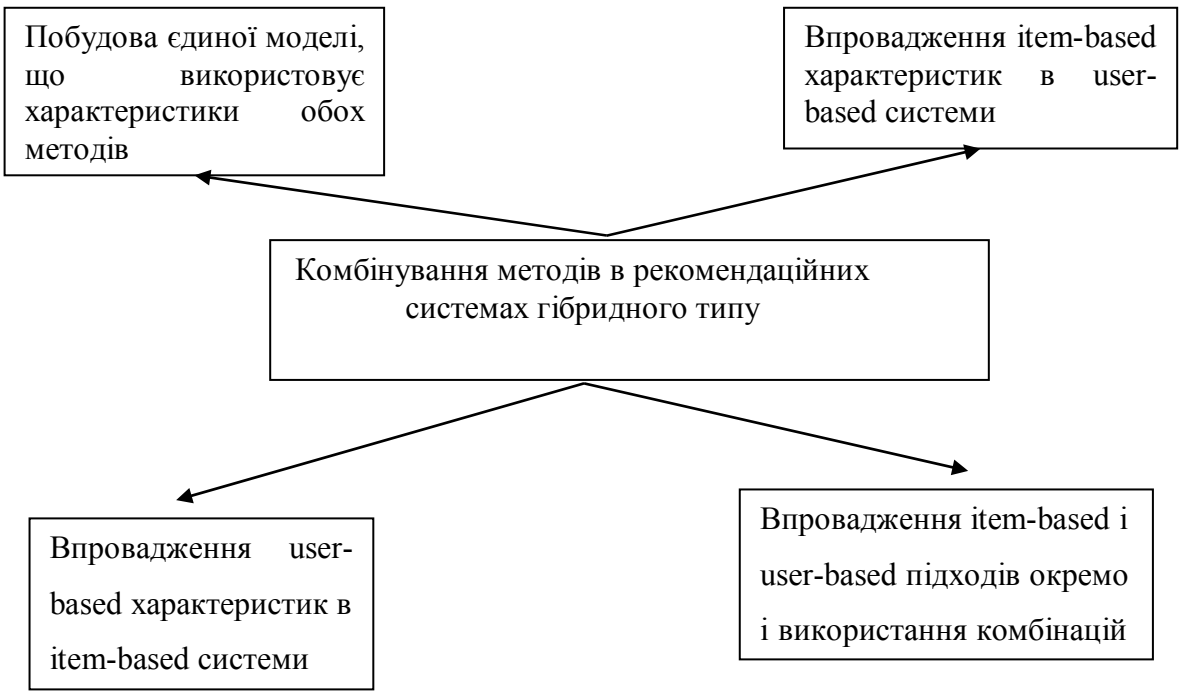

Рис. 2.6 – Комбінування методів в рекомендаційних системах гібридного типу

Останнім часом цей варіант дуже популярний у великої кількості дослідників. Основним принципом цього методу є використання функцій фільтрації вмісту та спільної фільтрації (наприклад, жанр пісні та заняття користувача) в єдиній системі рекомендацій.

Такі підходи включають уніфікований імовірнісний метод із використанням прихованого семантичного аналізу.

Такі гібридні системи рекомендацій побудовані на основі спільного компонента, але також містять деякі дані фільтрації вмісту в профілі користувача. Потім ці дані служать основою для розрахунку схожості користувачів замість загальної оцінки функцій. Цей підхід дозволяє уникнути проблеми дефіциту даних, що виникає через невелику кількість пар користувачів з великою кількістю цілочисельних об'єктів. Крім того, цей метод дозволяє рекомендувати користувачеві не лише предмети з позитивними думками інших користувачів, але й об’єкти, які можуть сподобатися користувачеві на основі його особистих уподобань. які використовують фільтрацію вмісту. Один з таких методів використовує прихований семантичний аналіз (LSA) для створення спільного вмісту профілів користувачів, які представлені набором векторів. Цей підхід значно покращує продуктивність порівняно з системами, які використовують лише фільтрацію вмісту.

Однією з комбінацій гібридної реферальної системи є окрема реалізація підходів на основі предмета та користувачів. Тоді є два шляхи розвитку. У першому випадку бали з двох систем можуть поєднуватися за допомогою лінійної комбінації балів. Другий спосіб пропонує використовувати систему, яка повинна бути найбільш підходящою в даному випадку. Існує система "DailyLearner", яка використовує систему рекомендацій, яка дає рекомендації з мінімальним рівнем помилок.

Гібридні системи рекомендацій іноді доповнюються методами, які використовують базу знань для поліпшення якості рекомендацій та вирішення основних проблем більшості систем рекомендацій (холодний старт та інші). Є статті різних авторів, в яких порівняння результатів роботи демонструє перевагу гібридних підходів перед чистими системами рекомендацій на основі позицій та користувачів.

2.3 Аналіз критеріїв та методів оцінювання рекомендаційних систем

Існує багато різних критеріїв, за якими можна оцінювати систему рекомендацій - наприклад, точність, новизна, здатність до несподіваності, стійкість до атаки, залежність від холодного завантаження, переконливість тощо, але одним з найважливіших все-таки є точність. Це показує, наскільки наші прогнози наближені до контрольного результату. Існує багато методів вимірювання точності, я рекомендую уважно підходити до вибору, оскільки від цього залежить багато чого.

Одним з найпопулярніших методів є обчислення середньої квадратичної помилки (RMSE). Після того, як алгоритм зробив прогноз на основі даних тесту, помилку можна обчислити за наступною формулою:

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (p_{ui} - r_{ui})^2} \qquad (1.10)$$

Де:

U – користувач;

i – предмет;

r – оцінка;

p–прогнозована

оцінка;

T – загальна кількість тестових оцінок, чим менше тим краще.

Дослідники досліджували властивості систем рекомендацій, які можуть бути корисними розробникам, окрім релевантності рекомендацій, а також метрики їх захоплення.

У літературі щодо систем рекомендацій було виявлено кілька таких властивостей, часто з відповідними показниками, такими як: охоплення, різноманітність, новизна, пристосованість. Наприклад, різноманітні рекомендації, коли елементи відрізняються один від одного, можуть мати більшу цінність для користувача. Якщо ми знаємо, що користувач захоплюється науково-фантастичними фільмами, рекомендація всіх фільмів "Зоряних воєн" може бути доречною, але не різноманітною і може бути незадовільною для користувача.

Друге завдання, на якому ми зосереджуємося, - це оцінка стабільності. Стабільність системи рекомендацій вимірює узгодженість рекомендації після внесення змін до набору даних. Розглянемо систему рекомендацій, яка

пропонує відео «Міжзоряне» як найвищу оцінку для 1% користувачів. Якщо користувач додає новий рейтинг до Interstellar, і в результаті він більше не буде рекомендований нікому, то

рекомендаційна система може вважатися нестабільною і негативно впливати на довіру користувачів.

Ми визначаємо загальну модель систем рекомендацій, яку потім використовуємо для формального визначення та систематичного вивчення властивостей простору систем рекомендацій.

Кешування можна використовувати для підвищення продуктивності сторінки. Це дозволяє генерувати збережені версії замість того, щоб створювати сторінки, що часто запитуються кожного разу. Завдяки цьому можна значно скоротити час завантаження сторінки та зменшити навантаження сервера. Наприклад, система управління базами даних може зберігати результати часто використовуваних запитів, окремі фрагменти інформації можна завантажувати в оперативну пам’ять сервера, HTML-код сторінок або їх невеликі блоки можна витягувати з файлової системи веб-сервера, сторінки можна кешувати в Інтернеті.

Інформація повинна використовуватися знову і знову, щоб кешування було ефективним. З іншого боку, персоналізовані сторінки можуть містити дані, що стосуються конкретного користувача. Веб-сайт ідентифікує відвідувача за допомогою файлів cookie та генерує сторінки спеціально для нього.

Такі сторінки неможливо повністю кешувати. Навіть якщо сторінка кешована, шанси перезавантажити її з кешу, як правило, досить низькі, тому обсяг пам'яті на жорсткому диску сервера значно зросте, тоді як завантаження центрального процесора буде трохи меншим.

Рішенням цієї проблеми може бути призначення всіх користувацьких блоків окремим URL-адресам та їх динамічне завантаження за допомогою технології AJAX.

На додаток до проблеми неможливості кешувати сторінки, існують труднощі, обумовлені специфічністю вихідних даних.

Кожен користувач має відповідну пару "документ-оцінка", де документом може бути сторінка веб-сайту або конкретна тема (тег), а рейтинг - деякий показник зацікавленості користувача цим документом, наприклад, кількість переглядів сторінки.

Ви можете сприймати ці дані як матрицю з кожним рядком, що відповідає користувачеві, і стовпцем документу. У цьому випадку проблема зводиться до обчислення невідомих елементів матриці. Матриця рейтингів, як правило, дуже розпорошена - користувачів і сторінок багато, і на практиці рейтинги значно менші, ніж їх робота, оскільки середній користувач відвідує мало сторінок. Решта елементів матриці невідомі, і їх значення слід передбачити, тобто кількість прогнозованих оцінок значно перевищує кількість відомих оцінок. Важливо, щоб система могла ефективно прогнозувати оцінки на основі невеликої кількості прикладів. Також необхідна критична кількість користувачів. Проблему рідкісних рейтингів можна подолати, використовуючи інформацію про користувача у своєму профілі для пошуку схожих користувачів.

Спільна фільтрація користувачів - один із найефективніших алгоритмів рекомендацій, але він має ряд недоліків. Вони включають, серед інших неможливість створення рекомендацій для нових користувачів та погана масштабованість.

Першу проблему можна вирішити, застосувавши спільну фільтрацію на основі подібності сторінок. Щоб використовувати спільну фільтрацію на веб-сайтах із високою кількістю сторінок, варто розрахувати схожість користувачів у кількох потоках одночасно. Ви можете збільшити швидкість створення рекомендації, зменшивши розмірність вихідних даних. Попередній вибір груп користувачів зі схожою поведінкою значно покращить продуктивність. У цьому випадку відстань між двома користувачами, які перебувають у різних групах, можна вважати нескінченною. Це можна зробити, попередньо

вибравши групи подібних користувачів та застосувавши спільний алгоритм фільтрації в групах. Це вирішить проблему розріджених вихідних даних та покращить масштабованість.

Щоб зменшити кількість рекомендацій, ви можете навчити алгоритм не на всіх даних користувача, а на якомусь відповідному зразку. Ви також можете використовувати вибір підмножин із вихідних даних для створення навчальних та тестових зразків. У цьому випадку параметри алгоритму коригуються на основі даних з першої вибірки, а точність рекомендацій оцінюється з решти даних. Основним завданням вибірки є пошук відповідної підмножини вихідних даних. Найпростіший спосіб - використовувати випадкову вибірку, де всі елементи вихідних даних мають однакову ймовірність потрапити до вибірки. У деяких випадках виправданим є використання стратометричного відбору, коли загальна сукупність поділяється на групи зі специфічними характеристиками (стать, вік, політичні уподобання, освіта, рівень доходів тощо) та вибираються елементи з відповідними характеристиками. Як правило, використовуються випробування без повторення - один і той же елемент не можна вибрати двічі. Поширений підхід, який використовує випадкову вибірку без повторення з вибором навчальних та тестових наборів даних у пропорції 80/20.

Процес навчання алгоритму можна повторити багато разів, щоб уникнути помилок використання невеликої підмножини замість усіх вихідних даних. Після створення навчальних та тестових зразків створюється модель рекомендаційної системи та оцінюється її точність. Потім створюється нова пара зразків, і процес навчання та тестування повторюється k разів. В результаті використовується середнє значення k-моделей. Цей підхід називається перехресною валідацією. Це може бути здійснено кількома способами. У разі множинної випадкової вибірки підмножини, що підлягають навчанню, обираються k разів випадковим чином. Іншим способом може бути вибір n підмножин

вихідних даних. Один з них використовується як тест, а решта - як тренування. Процес навчання та тестування повторюється n разів, так що кожна підмножина виявляється тестом одноразово.

Учасників сторінки веб-сайту можна класифікувати таким чином, що вихідними даними є інформація про перегляд, а результатом є клас відвідувачів, до якого належить користувач. Існує багато способів класифікації, два типи яких можна виділити - навчальний та ненавчальний. У класифікаційних алгоритмах з навчанням класи відомі заздалегідь, і є дані про належність певних елементів до класів. На основі цих даних можна засвоїти алгоритм і класифікувати наступні елементи. Якщо класи не відомі заздалегідь, розділіть усі елементи на групи так, щоб елементи, що належать до однієї групи, були схожі між собою, але не схожі на елементи з інших груп.

Алгоритм k найближчих сусідів дозволяє розділити вихідні дані на різні класи, використовуючи інформацію про подібність окремих елементів. Користувача, клас якого ще ніхто не знає, буде призначено до класу, до якого належать k найближчих користувачів. Однією з головних переваг цього методу є те, що при додаванні нових користувачів не потрібно перекваліфікувати алгоритм для всіх даних - потрібно лише вказати найбільш підходящі класи лише для нових відвідувачів. Однак, класифікуючи доданих користувачів, їх вектори слід порівнювати в парах з даними всіх існуючих відвідувачів, що може вимагати великого обсягу пам’яті для обчислень і тривати довго. Це може бути неприпустимо повільним для сайтів із великим трафіком.

Байєсівський класифікатор можна використовувати для вибору груп подібних користувачів. Цей метод заснований на принципі задньої максимізації ймовірності. Для користувача обчислюються функції вірогідності кожного класу і на їх основі розраховуються задні ймовірності класів. Як результат, користувач належить до класу, для якого задня ймовірність максимальна.

Спрощене значення класу, якщо вважати, що всі параметри, що характеризують поведінку користувача і утворюють його вектор, незалежні один від одного.

Завдання функціонування рекомендаційної системи можна сформулювати як пошук N найцікавіших користувачеві посилань, а значить оцінити ефективність рекомендаційної системи, заснованої на класифікації, можна перевіривши, наскільки точно вона класифікує сторінки як цікаві для конкретного користувача. Таким чином, для подібних систем можна застосовувати такі поняття, як точність і повнота. Для рекомендаційної системи точність – це частка цікавих користувачеві сторінок серед запропонованих в рекомендаціях. Повнота – частка відомих системою як цікаві користувачеві сторінок серед всіх існуючих.

2.4 Постановка задачі дослідження.

У наш час ера мобільних телефонів розвивається досить динамічно. Сучасний смартфон може робити фотографії та відео, працювати з документами, користуватися Інтернетом, виконувати функції роутера, оплачувати рахунки та багато іншого. Аналітики вважають, що сучасні смартфони незабаром стануть повноцінними комп'ютерами, до яких можна підключити всю периферію. З появою стандарту 5G (швидкість передачі даних до 7 Гб / с) люди відмовляться від старих джерел Інтернету, таких як Wi-Fi. Крім того, смартфон може бути досить дешевим, тому ним можуть користуватися всі, від дітей до літніх представників.

Разом з розвитком мобільних технологій розвивається і напрямок мобільних додатків. Велика кількість додатків створює велику конкуренцію на ринку, а це означає, що потреби користувачів будуть зростати і надалі. Люди потребують швидкого та простого доступу до

необхідної інформації та функціональних можливостей. Багато з них в даний час зосереджені на розробці мобільних додатків, починаючи від базових калькуляторів і закінчуючи програмами, що використовують штучний інтелект. Мобільні додатки сьогодні є потужним інструментом не тільки для користувача, але і для самого бізнесу. Вони дозволяють компаніям створювати імідж, піклуватися про бренд і допомагають впроваджувати додаткові інструменти для роботи або спілкування між відділами компанії..

Сьогодні існує багато способів створити рекомендацію, але всі вони мають свої переваги та недоліки. Тому дослідження в цій галузі є важливими. Ця проблема особливо важлива у випадку нових, нещодавно розроблених систем. Проблема достовірності оцінок часто виникає у разі холодного старту, оскільки нові об'єкти або користувачі ускладнюють створення точних рекомендацій.

ВИСНОВКИ ДО 2 РОЗДІЛУ

Щоб система мала якісні рекомендації, необхідно постійно оновлювати інформацію про інтереси користувача. Сучасні методи підготовки рекомендацій враховують як властивості предметів, так і користувачів, які їх вибирають. Однак ефективність цих методів оцінюється лише під час їх впровадження. На практиці існує потреба в адаптації цих методів на регулярній основі, на основі оцінки їх ефективності.

3 ОСОБЛИВОСТІ РОЗРОБКИ

3.1 Особливості розробки Android

Android – відома платформа, попит на додатки для неї високий, як і конкуренція між ними. Лише одна компанія виробляє пристрої для iOS і багато для Android. Розробникам доводиться мати справу не тільки з версією операційної системи, але і з пристроями, які сильно відрізняються один від одного. Однією з головних перешкод є розмір та роздільна здатність екрану. Додатки Android потребують більше часу, щоб перевірити, чи працюють вони належним чином на більшості смартфонів та планшетів. Ця функція призводить до додаткових витрат часу та грошей на розробку програми. Крім того, виникають труднощі при роботі з Інтернетом, оскільки можуть виникнути проблеми з термінами тривалих операцій, таких як Інтернет-запити та життєвий цикл [36].

Протягом сьогодення розробники надають багато уваги навантаженню акумулятора і пам'яті телефону. Творці кожної версії Android намагаються максимізувати продуктивність та швидкість роботи програми, оновлюючи SDK. Всі сфери дизайну зосереджені на простоті для максимальної функціональності та простоти використання. Якщо раніше розробники та дизайнери намагалися звернути увагу на шрифти та орнаменти, які використовувались у зовнішньому вигляді, то зараз все зроблено максимально мінімалістично та просто для зручності користувача.

3.2 Види мобільних додатків та мобільні сайти

Завдяки смартфонів нинішніх поколінь ви можете побачити звичайний веб-сайт. Такі пристрої - це майже все, що ми бачимо у настільних додатках, оскільки можливість підтримки HTML5 робить свою справу. Такі веб-додатки - чудове рішення для початку, адже лише завдяки їм ми можемо досягти дуже значного ефекту за невеликі гроші та за короткий час. Крім того, перевагою мобільних веб-сайтів у порівнянні з іншими мобільними додатками є т.зв. крос-платформерність.

Веб-програми використовують стандартні веб-технології, такі як HTML5, JavaScript та CSS. Завдяки такому підходу можна створювати міжплатформні мобільні додатки, які працюють на декількох пристроях [9]. Додатки HTML5 можуть працювати на різних операційних системах та типах пристроїв. Програма масштабується залежно від розміру пристрою [5]. Якщо вам потрібно оновлення, слід оновити та протестувати одну програму, яка буде доступна на всіх пристроях одночасно. Великою проблемою безпеки в HTML5 є можливість попереднього перегляду вихідного коду. Це означає, що ми можемо не тільки зрозуміти, як це працює, але і використовувати його для своїх цілей.

3.3 Гібридні додатки

Маючи такий підход отримуємо доступ до всіх переваг програмного інтерфейсу програмного забезпечення (API) операційної системи, а саме - серед приємних аспектів є те, що додаток кишить push-повідомленнями. Крім того, ваші товари можуть бути на складі. Однак основним вмістом як і раніше залишається незалежний від платформи макет сторінки, розміщений на сервері. Це дозволяє вносити косметичні

зміни до продукту без необхідності випускати нову версію, оскільки завантаження змін на сервер досить проста. Тож гібридні програми можуть бути чудовим рішенням для тих, хто хоче розпочати бізнес або реалізувати свої ідеї. Гібридні додатки поєднують в собі деякі функції нативних та веб-додатків: крос-платформенність та можливість використання програмного забезпечення на телефоні. Їх можна завантажити з магазинів додатків і одночасно мати можливість самостійно оновлювати інформацію (їм потрібне з'єднання з Інтернетом, оскільки веб-частина оновлюється через Інтернет). Гібридні програми є найпопулярнішими в наші дні, оскільки розробка відбувається швидше і дешевше, ніж власні програми, хоча оболонка написана "рідною" мовою програмування, "начинка" може бути записана в тій чи іншій формі в html5.

Навряд чи користувач помітить різницю між власною програмою та гібридною програмою. Програми HTML5, як правило, дешевші у розробці та обслуговуванні, ніж власні програми, оскільки це лише одна програма, необхідна для запуску декількох операційних систем.

Даний додаток може створити один веб-розробник. Гібридні програми є портативними; може бути побудований майже з такою ж швидкістю, як додаток HTML5, оскільки основна технологія однакова; можна створити майже за ту саму ціну, що і додаток HTML5, але для більшості фреймворків потрібна ліцензія, що додає додаткові витрати на розробку; доступ та розповсюдження через відповідний магазин додатків, а також власні програми; мають доступ до апаратних ресурсів, як правило, за допомогою власного API. Однак не всі апаратні ресурси доступні для гібридних програм [7]. Програми відображатимуться для кінцевого користувача так само, як і ваші власні програми, але працюватимуть набагато повільніше. Те саме обмеження стосується HTML5. Візуалізація складних макетів CSS займе більше часу, ніж візуалізація подібного макета.

3.4 Нативні додатки

Програми цього типу вважаються найбільш ресурсоємними, але вони також дозволяють максимально використовувати функції, пропоновані окремими операційними системами. Як результат, власні програми перевершують інші типи програм не тільки за функціональністю, але і за швидкістю завантаження. Компанії, які використовують пакетні програми, покладаються на цей підхід.

Наприклад, всесвітньо відомий Facebook розпочав свій бізнес із комбінованого типу додатків: вмістом був веб-сайт, а рідними елементами управління були вкладки, перемикачі тощо. Є проблеми з продуктивністю, тому багато розробників уникають підключення до Інтернету. Оригінальна програма - це програма, побудована повністю з використанням технологій, пов’язаних з певною операційною системою. Це може бути Android, iOS, Windows, Blackberry тощо. У випадку з Android, користувацькі програми зазвичай будуються за допомогою Java, тоді як у випадку з iOS програма може бути побудована за допомогою Objective C або Swift. Запатентовані програми унікальні для кожної операційної системи, тому для підтримки декількох мобільних операційних систем потрібно створити окремі програми для кожної ОС.

Якщо потрібне оновлення, кожна програма повинна оновлюватися незалежно одна від одної. Під час розробки створюються різні макети, щоб програма могла адаптуватися до різних розмірів пристрою / екрану та орієнтації. Спеціальні програми для всіх основних мобільних операційних систем зазвичай вимагають спеціалізованого розробника для кожної операційної системи (Java для Android, Objective C / Swift для IOS, C # для Windows), вартість набагато дорожча, ніж для одного веб-розробника. Створення програми для певної платформи вимагає більше знань, ніж створення веб-програми в HTML5. Фірмові

програми можуть взаємодіяти з різними пристроями, доступними на пристрої, включаючи Глобальну систему позиціонування (GPS), камеру, акселерометр тощо.

ВИСНОВКИ ДО 3 РОЗДІЛУ

Ця частина присвячена аналізу того, що таке мобільний додаток і для яких операційних систем він існує. Переваги, недоліки та складність створення додатків для Android та iOS. У цьому розділі описано три типи мобільних додатків: власні, гібридні та мобільні сторінки. Розглянуто їх переваги та недоліки, а також коротко описані методи створення кожного з цих типів додатків. Було вивчено, чому власні програми є найважливішими у розробці таких систем.

РОЗДІЛ 4 ВИМОГИ ДО РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ «ОГРАНІЗАЦІЇ НЕЗАЛЕЖНИХ ТРЕНУВАНЬ»

4.1 Характеристика проекту

Метою цього програмного забезпечення є впровадження або розповсюдження такого типу мобільних додатків, як система бронювання спортивних майданчиків. Мобільний додаток повинен контролювати бронювання та інші послуги програми, щоб користувачі могли завантажувати та використовувати програму незалежно від обладнання свого смартфона.

Користувачі - звичайні люди, яким потрібно зареєструватися. Мобільний додаток повинен мати певну систему бронювання, а потім підтримувати профіль користувача. Система повинна підтримувати події - тип системних даних, що описують спортивні події, які повинні містити певну інформацію:

- Час початку події;
- Час закінчення події;
- Опис події;
- Назва події;
- Тип події (футбол);
- Назва спортивного поля;
- Координати спортивного поля.
- Електронна пошта користувача;
- Правильне ім’я користувача;
- Можливість входу в систему;
- Можливість реєстрації;
- Можливість перегляду власного профілю;

- Можливість виходу з програми;
- Можливість перегляду списку поточних спортивних подій;
- Можливість перегляду створених користувачем подій;
- Можливість перегляду подій, до яких користувач підключився;
- Можливість створити власний захід;
- Можливість приєднатися до конкретної, створеної події;
- Можливість відключення від події.

4.2 Системні вимоги до додатку

Було вибрано операційну систему - Android, необхідно якісно проаналізувати ринкові версії цієї операційної системи, кількість користувачів, кількість пристроїв, що підтримують цю операційну систему, і важливість розробки на ній. Гігант пошукової системи Google, який є творцем цієї операційної системи, надає доступ до статистики версій. Крім того, повинен бути проведений статистичний аналіз пристроїв з різними розмірами екрану, цю інформацію також надає Google.

Проаналізувавши цю інформацію, можна припустити, що більшість пристроїв мають попередньо встановлені версії, такі як: Lollipop, Marshmallow, Nougat, Oreo, Pie. Отже, починаючи з версії Lollipop.

Проаналізувавши екрани та конфігурацію, ми бачимо, що більшість пристроїв мають у своїй конфігурації такі розміри та щільність екрану, як: xhdpi, hdpi, xxhdpi.

Виходить зі з’ясованої раніше інформації, можна прописати вимоги для мобільного додатку:

- Операційна система: Android Lollipop 5.1 або більш нова;
- Екрани: xhdpi, xxhdpi, hdpi;
- Сумісність з усіма архітектурами Android;

- Об’єм пам’яті: 150 Мб. (мінімум)

4.3 Основні інструменти для реалізації додатку

Програми для Android, як правило, розробляються з використанням мови програмування, подібної до Java, та комплекту розробки програмного забезпечення (SDK), але є й інші можливості. SDK включає емулятор платформи Android, він заснований на qemu і дуже повільний (принаймні). Емулятор дозволяє створювати віртуальні пристрої на різних версіях Android, на різних екранах. (Віртуальний пристрій Android або AVD за термінологією SDK), на якому ви можете запускати та тестувати свої програми. Емулятор дозволяє користувачеві не тільки використовувати функціонал операційної системи, але і імітувати всю функціональність реального пристрою:

- Змінити орієнтацію екрана;
- Зміна рівня заряду акумулятора;
- Емуляція роботи камери;
- Емуляція SMS-повідомлень та телефонних дзвінків;
- Емуляція роботи датчика відбитків пальців;
- Здійснення записів екрану;
- емуляція роботи модуля GPS.

Android SDK можна розділити на кілька груп: одна з них містить дані модулів для створення мобільних додатків, тобто для мобільної платформи Android ці модулі також містять деякі інструменти та компоненти для деяких пристроїв, таких як планшет Samsung Galaxy. Інша група модулів включає всі інші модулі, такі як код, документація, API, сервіси Google.

Зазвичай SDK розпаковується в каталозі ~ / android / adt-bundle-<OS-PLATFORM> platform / platform-NNN, де NNN - номер версії (номер) платформи API. Для кожного основного випуску платформа

випускала нову версію API, наприклад, для Android 2.2, версія API номер 8, для Android 2.3.1 - 9, для Android 2.3.3-10, для Android 4.2.2 - 17 тощо.

Модуль містить файли, необхідні для запуску цієї платформи в емуляторі платформи Android.

Але в цьому модулі Google не встановлений, наприклад, для Google Maps. Обробники API Google перераховані окремо і зазвичай їх називають Google Inc. Загалом, усі модулі, знайдені в ~ / android / adt-bundle- <OS-PLATFORM> /, мають приблизно однакову структуру і містять файли, з яких створюється образ віртуального пристрою AVD.

Android SDK має засоби для підключення до пристрою Android, який працює точно так само як із реальним, так і з віртуальним. Одним з них є Android Debug Bridge - утиліта командного рядка під назвою adb, яка знаходиться в ~ / android / adt-bundle- <OS-PLATFORM> / platform-tools і дозволяє виконувати роботу з налагодження на підключеному пристрої. Ви можете побачити всі доступні вам параметри adb за допомогою команди adb help, вона відобразить довгий список різних параметрів з дуже докладним описом кожного.

Додатки Android упаковані у форматі .apk і зберігаються в папці / data / app на Android (ця папка доступна лише кореневому користувачеві з міркувань безпеки). Файли .Apk включають файли .dex (компільовані файли байт-коду, що називаються виконуваними файлами Dalvik), файли ресурсів тощо.

Код, написаний на C / C ++, можна скомпілювати в ARM або власний код x86 (або 64-розрядний) за допомогою Android Native Development Kit (NDK). NDK використовує компілятор Clang для компіляції C / C ++. GCC був включений до NDK r17, але вилучений у r18 у 2018 році.

4.4 Структура та компоненти

Структура (рис. 4.1) створеного проекту Android досить велика, вона містить як самі програмні модулі, такі як файли з класами, функціями та програмами, так і системні файли, такі як маніфест програми, автоматичні файли збірки системи проекту. У самому проекті також є папка ресурсів для мобільних додатків:

- Файли розмітки екрана;
- Файли тегів для деяких компонентів та віджетів;
- Терміновий ресурсний набір;
- Набір стилів;
- Набір векторних зображень;
- Набір растрових зображень;
- Набір кольорів у шістнадцятковому форматі;
- Набір примітивів;

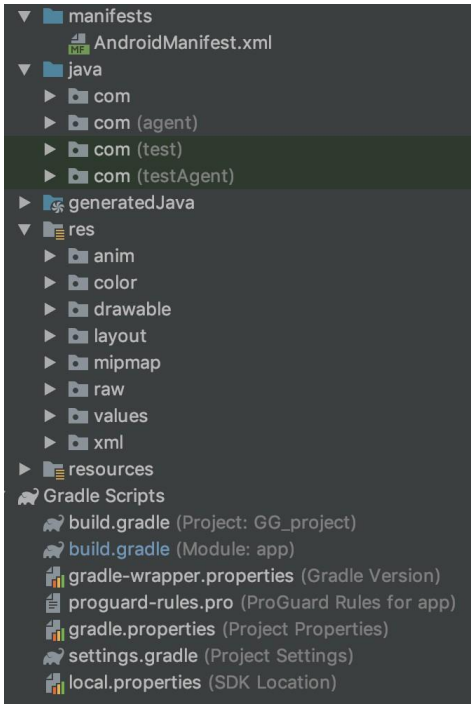

Рисунок 4.1 — Структура проекту.

Файл маніфесту містить основну системну інформацію про програму. Маніфест є дуже важливим елементом проекту, це XML-файл, який описує властивості програми (ім'я, опис) залежно від вимог до платформи на пристрої, функціональних можливостей, що надаються пакетом, та інших подібних речей.

Ви можете використовувати вбудований редактор ADT для редагування маніфесту. Цей файл також повинен чітко вказувати на контекстні компоненти Android, на які додаток має дозволи, такі як доступ до Інтернету або доступ до галереї.

У полі <application> цей файл повинен містити конкретні налаштування проекту:

- Поле коду версії - номер версії програми у вигляді цілого числа, кожна наступна версія програми повинна бути більшою за попередню. Це значення використовується системою для порівняння версій;
- Поле імені версії - версія програми у вигляді будь-якого рядка, тут дозволено майже все, система використовує це поле лише як опис;
- Поля «Спільний ідентифікатор користувача» та «Спільний ярлик користувача» використовуються для групування різних програм одного виробника, програми, які відповідають полям Спільного ідентифікатора користувача та підписані тим самим сертифікатом, можуть мати доступ до ресурсів інших програм.

Завдання є основою побудови Gradle. Завдання - це набір інструкцій щодо побудови, який Gradle запускає під час створення програми. Порівняно з іншими системами побудови, завдання можуть бути добре відомими абстракціями. Gradle працює у віртуальній машині Java, використовує системні бібліотеки Ant build, використовує

елементи керування Apache Ivy та інші інструменти (TestNG, JUnit, SureFire тощо).

Git - система контролю версій, яка передбачає встановлення версій програмного коду. Він був створений Лінусом Торвальдсом для управління розвитком ядра Linux. Система контролю версій - це система, яка повинна керувати або реєструвати зміни у файлі чи декількох файлах протягом певного періоду. Це робиться для того, щоб ви могли пізніше повернутися до певної версії. Головною перевагою цієї системи є те, що існує багато віддалених сховищ, які можуть завантажити код проекту на свій сервер, зменшуючи ймовірність втрати коду.

Такими послугами, наприклад, є: BitBacket, GitLab, GitHab. Крім того, ця система використовує децентралізовану систему контролю версій (рис. 4.2). Це означає, що у випадку смерті сервера кожне зі сховищ може бути відновлено.

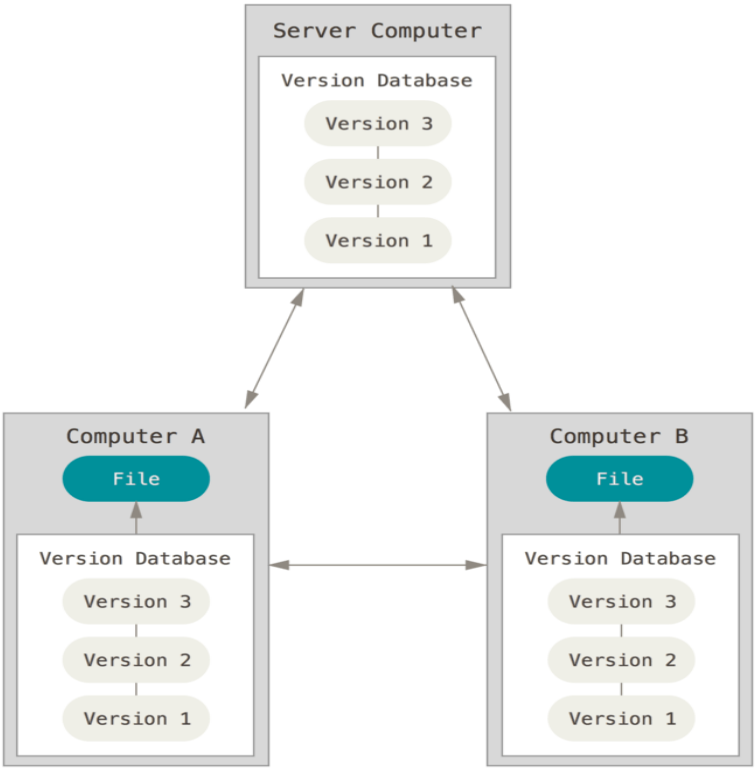

Рисунок 4.2 — Децентралізована система контролю версій

Git має три основні стани:

- Зберігається у коміті - це означає, що файл зберігається в базі даних;
- Змінено - це означає, що файл зберігається в базі даних, але містить деякі зміни, які ще не доступні;
- Проіндексовано - це стан, який виникає, коли помічений змінений файл помічається, щоб його зміни можна було застосувати до бази даних при наступному коміті.

Git зберігає як метадані, так і базу даних об’єктів проекту. Це те, що копіюється при клонуванні сховища Git. Індекс - це файл, який зазвичай знаходиться в каталозі Git і містить інформацію про те, що буде збережено в наступній команді компіляції. Цей файл також відомий як «інтерактивна область». Кожен файл у цій системі може мати два стани, відстежуваний або відстежений.

Після створення клону проекту всі файли перебувають у контрольованому стані. Під час редагування файлів проекту звернення помічає їх зміну стану. Якщо помічені змінені файли, вони позначаються статусом - незмінені. Однією з популярних запитів є «git status» (рисунок 4.3)

```
MBP-Vladisla:maydanchik vladyslavboiko$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   app/src/main/java/com/kpi/maydanchik/data/events/EventType.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/ApiPlaygroundRepository.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/Playground.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundRepository.kt
        new file:   app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundService.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/CreateActivity.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/CreatePresenter.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/CreateView.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/di/CreateComponent.kt
        new file:   app/src/main/java/com/kpi/maydanchik/screens/create/di/CreateModule.kt
```

Рисунок 4.3 — Консольний вивід команди «git status»

Ця команда дозволяє пересчитувати всі змінені та неконтрольовані файли у вашій системі Git. Наступною базовою командою є "git add.". Ця команда дозволяє індексувати всі змінені та неконтрольовані файли проекту. Найважливіша команда - "git commit -m <повідомлення>".

Ця команда створює комітет (рис. 4.4) і зберігає всі відслідковувані зміни у файлах проекту.

```
MBP-Vladisla:maydanchik vladyslavboiko$ git add .
MBP-Vladisla:maydanchik vladyslavboiko$ git commit -m "Your commit message"
[master dc31102] Your commit message
 63 files changed, 1577 insertions(+), 363 deletions(-)
 rewrite .idea/codeStyles/Project.xml (81%)
 create mode 100644 .idea/encodings.xml
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/events/EventType.kt
 rewrite app/src/main/java/com/kpi/maydanchik/data/events/Events.kt (78%)
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/ApiPlaygroundRepository.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/Playground.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundRepository.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/data/playgrounds/PlaygroundService.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/screens/create/CreateActivity.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/screens/create/CreatePresenter.kt
 create mode 100644 app/src/main/java/com/kpi/maydanchik/screens/create/CreateView.kt
```

Рисунок 4.4 — Консольний вивід команди «git commit -m <message>»

4.5 Середовище розробки Android Studio

Це середовище базується на програмному забезпеченні Intellij Idea від розробників JetBrains. З початку з 2017 року, це офіційне середовище для розробки додатків для Android і рекомендується Google. Android Studio підтримує наступні мови програмування, як Java, C ++, Kotlin. Вбудована підтримка XML. Середовище розробки пристосоване для виконання загальних завдань, які вирішуються під час розробки додатків для платформи Android. Середовище має інструменти, що спрощують тестування програмного забезпечення на сумісність з різними версіями платформи та засоби для розробки програм, що працюють на пристроях з різною роздільною здатністю (планшети, смартфони, ноутбуки, годинники, окуляри). На додаток до функцій, доступних в IntelliJ IDEA, Android має багато додаткових функцій, таких як нова уніфікована підсистема для встановлення, тестування та розгортання програм на основі інструментів колектора Gradle та підтримка використання засобів безперервної інтеграції. Основна функціональність IDE:

- Можливість роботи з компонентами інтерфейсу за допомогою функції перетягування, що дозволяє відображати макет у декількох конфігураціях екрана;
- Створення програми на основі Gradle;
- Генерація різних версій APK та App Bundle;
- Рефакторинг коду;
- Статистичний аналізатор коду, що дозволяє виявляти проблеми, пов’язані з несумісністю версій та продуктивністю мобільного додатку;
- Можливість затушувати код за допомогою ProGuard;

- Можливість підписувати файл APK власними ключами;
- Можливість використання основних шаблонів Android;
- Підтримка Android NDK;
- Вбудована підтримка деяких функцій Google Cloud Platform;
- Вбудована підтримка Google Cloud Messaging та Firebase;
- Вбудована підтримка Kotlin та KotlinX;
- Можливість створення додатків для Android Wear, Android TV.

Однією з важних особливостей представленого середовища є інтелектуальний редактор коду. Тому користувач може швидше використовувати ті функції та методи, переглядати перелік пропозицій функцій, методів, полів із середовища (рис. 4.5), що значно значно спрощує процес створення коду.

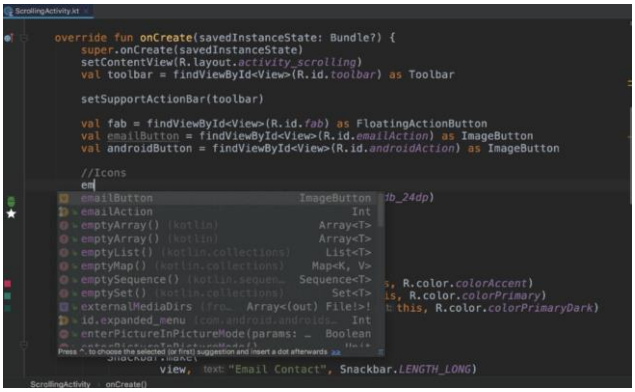

Рисунок 4.5 — Розумний редактор коду

4.6 Мови програмування

Мова програмування Java - це високотекстова об'єктно-орієнтована мова програмування, розроблена Sun Microsystems, а потім придбана Oracle. В даний час багато великих компаній, таких як Oracle, RedHead, IBM, Google, JetBrains, спільно створюють OpenJDK - пакет, що містить основні засоби програмування, такі як бібліотеки. Програми Java (не лише

мобільні) перекладаються у спеціальний байт-код, який був створений для віртуальної машини Java (JVM). Завдяки цій машині та байт-коду програми Java можуть працювати на будь-якій платформі, операційній системі, на якій встановлено JVM. На 2019 рік ця мова програмування є найпопулярнішою у світі. Однією з істотних переваг цієї мови є гнучка система безпеки, в якій виконання програми повністю контролюється віртуальною машиною. Будь-яка операція, яка теоретично може перевищити права цієї програми, наприклад, несанкціонований доступ до Інтернету або доступ до певних даних, негайно припиняє роботу програми. Недоліком є менша швидкість порівняно з C або C ++. За деякими даними, швидкість приблизно в два рази менша. Однак із кожним оновленням вихідного коду швидкість програми Java зростає. Найбільшою перевагою цієї мови є безпека пам'яті, оскільки немає вказівників або явних посилань на об'єкти. Крім того, швидкість створення програм на цій мові набагато вища, ніж на мовах програмування низького рівня, таких як C або C ++. Згідно з таблицею 4.1, переваги використання цієї мови перевищують недоліки.

Мова програмування Kotlin - це статично набрана мова OSS, яка підтримує Java, JavaScript та Native. Ця мова була розроблена в 2010 році JetBrains, версія 1.0 випущена в 2016 році. З 2019 року це рекомендована мова для розробки платформи Android. Ця мова програмування є загальнодоступною, що означає "відкритий код", весь вихідний код є загальнодоступним на GitHub. Котлін є одночасно об’єктно-орієнтованою та функціональною мовою.

Це означає, що мова приймає властивості Java у контексті ООП, але має можливість писати у функціональному стилі, наприклад, писати функцію без обгортки в класі.

Kotlin можна використовувати для побудови таких платформ, як будь-який сервер, клієнт, веб або Android. З випуском Kotlin / Native (який зараз розробляється) з’являється підтримка таких платформ, як macOS та iOS.

Значна частина синтаксису взята з наступних мов програмування: Pascal, TypeScript, Haxe, PL / SQL, F #, Go і Scala, C ++, Java, C #, Rust і D. Ця мова, через те, що він прийняв найкраще з багатьох мов набагато стисліше. Код Kotlin в середньому на 40% коротший за код Java. На сьогоднішній день майже всі сучасні середовища розробки ID підтримують Kotlin, серед основних - Intellij Idea, Android Studio, Eclipse, Visual Studio Code. Котлін також є стандартним. Через нульовий захист програми менш сприйнятливі до NullPointerException. Kotlin також має багато інших переваг перед Java: функції вищого порядку, функції розширення та лямбда-вирази для споживачів. Це дозволяє програмісту писати виразний код і підтримує створення DSL.

Переваги Kotlin перед Java. Залишається питання, використовувати Kotlin замість Java чи ні. Основна відмінність між цими мовами програмування полягає в тому, що Kotlin абсолютно безпечний, що дозволяє не використовувати блоки для спроб захоплення. Це означає, що у Котліна немає т. Зв позначені винятки, тому немає необхідності постійно перевіряти та ловити винятки. Kotlin також дозволяє створювати додаткові потоки, як і Java, але вводить новий механізм управління цими потоками, який називається спільними програмами. Kotlins реалізовані без використання стека, що означає, що вони використовують менше пам'яті, ніж звичайні потоки Java.

У реальних проектах, які стають все більшими і більшими, т. Зв клас POJO - звичайний старий об'єкт Java. Зазвичай такі класи описують якісну модель буття, їх завдання полягає лише у збереженні даних та їх описі. Існує простий спосіб створення таких об’єктів на мові програмування Kotlin. Досить додати слово клас до специфікатора даних. Такий клас автоматично створює геттери, сетери, створює кілька конструкторів класів.

Крім того, він замінює стандартні функції, такі як toString (), hashCode (), equals (). Це значно спрощує розробку програмного забезпечення.

Важлива перевага Kotlin полягає в тому, що він підтримує функції вищого порядку та лямбда-функції. Оскільки мінімальною версією Android є 5.1, це означає, що проект буде використовувати лише Java 1.6, яка не має цієї функціональності. Як статично набрана мова програмування, Kotlin використовує багато типів функцій для представлення функцій. Регулярні функції в Kotlin - це функції першого порядку, що означає, що вони можуть зберігатися в структурах даних та у вигляді полів, а також можуть передаватися як аргументи.

Ще однією перевагою є те, що Kotlin дозволяє розробникам використовувати т. Зв функції розширення. Вони дозволяють продовжувати певні класи, не змінюючи їх зсередини..

4.7 Архітектура та бібліотеки

Успішна архітектура дає розробникам шанс, що вони ніколи не зможуть відновити цілу програму чи програмний продукт [3]. Розробка правильного проекту та вибір відповідної архітектури може значно полегшити ваше життя. Найважливішими змінами є редагування певних модулів коду без зміни клієнта. Через неправильну архітектуру кожен рядок коду збільшує складність обслуговування, розширення та тестування проекту.

Архітектура MVP - розшифровується як Model View Presenter. Це одна з найпопулярніших і фундаментальних архітектур для розробки Android. Цей шаблон дизайну дозволяє програмісту розподілити відповідальність за розробку інтерфейсу користувача.

Основним компонентом системи Android є активність, вона має всю логіку, відповідальну за дії користувача, бізнес-логіку, презентацію, роботу з базами даних та Інтернетом.

Якщо ви не розділите ці обов'язки на класи, у вас можуть виникнути труднощі з розширенням та підтримкою програми. У цій архітектурі є 3 суттєві сутності (Таблиця 4.6):

- View View - це клас, який відповідає за всі взаємодії користувачів.
- Сутність Presenter - це клас, який відповідає за основну бізнес-логіку програми.
- Сутність моделі - це класи, які відповідають за всі дані мобільних додатків, маршрутизацію, зберігання тощо.

Таблиця 4.6 - Відповідальності сутностей в MVP

Model	**View**	**Presenter**
Відповідає за маршрутизація даних	Відповідає за створення сутності Presenter та механізм його приєднання або від’єднання;	Відповідає за завантаження моделей
Відповідає за взаємодію з базою даних	Відповідає за оповіщення Presenter про важливі життєві події додатку	Відповідає за збереження посилання моделей
Відповідає за взаємодію з різними джерелами даних такими як інтернет	Відповідає за оповіщення Presenter про події від користувача	Відповідає за форматування та маппінг, того що повинно бути відображено на View
Відповідає за моделі	Відображає дані, що були оброблені та повернуті Presenter	Відповідає за команди до View, що вона повинна показати, зробити і т.п.
	Відповідає за розміщення View - елементів та інших віджетів	Взаємодія з репозиторіями
	Відповідає на переходи до інших екранів мобільного додатка	Відповідає за визначення певних дій після того як отримані певні події від View
	Відповідає за анімації	

Dagger 2 - це якісно статична система реалізації залежністей. Ця система використовується для реалізації шаблону IoC (інверсія управління), який зменшує узгодженість проекту. Це значно полегшує розробку та тестування системи.

Однією з реалізацій цієї моделі є реалізація залежностей. Існують різні типи реалізації залежностей:

- Реалізація через конструктор;
- Реалізація методом класу;
- Реалізація через інтерфейс реалізації;

Фреймворк дає змогу користати всі ці типи реалізації. Це працює шляхом генерації коду. Dagger також екземпляри класів програм і задовольняє їх залежності. Анотація javax.inject.Inject визначає, які конструктори та поля представляють інтерес. Основні анотації фреймворку:

- Анотація ін’єкцій
- Модуль анотацій
- Абстрактна складова
- Забезпечити абстракцію
- Анотація Сінглтон

Усі залежності створюються в спеціальному модулі - класі, який містить методи, позначені анотацією "Надати". Ці методи зазвичай створюють або використовують готові залежності, щоб задовольнити інші залежності, позначені анотацією Inject.

Оскільки модулів можливо буває багато, а для створення однієї залежності може знадобитися інший, реалізовано спеціальну сутність Component..

Рисунок 4.7 - Компонент Dagger 2

Кожен компонент створює графік залежностей, що означає, що він з'єднує модулі і тим самим дозволяє їм взаємодіяти. Наприклад, якщо одному модулю потрібна залежність від іншого модуля, щоб створити його власний, компонент забезпечує доступ до цієї залежності.

Крім того, цей фреймворк дозволяє вам створити Sigleton, тобто формуючий шаблон програмування, який гарантує, що для даного класу створюється лише один об'єкт і має глобальний доступ. Цей шаблон реалізований в рамках Dagger, він може бути реалізований, використовуючи лише одне оголошення @Singleton. Це також дозволяє створювати власні анотації, наприклад, для локальних одиночних файлів, що може бути корисним, коли потрібно встановити деякі обмеження на термін служби об’єкта.

Однією з основних бібліотек для роботи з Інтернетом в Android є бібліотека оновлень.

З 2021 року він вважається стандартним стандартом у розробці програмного забезпечення для Android. Однією з причин його популярності є те, що він має чудову підтримку REST API, а також досить швидко з HTTP. Крім того, ця бібліотека добре працює з RxJava, про що буде сказано нижче.

Бібліотека дозволяє робити всі типи HTTP-запитів, включаючи запити Mutlipart, які можуть знадобитися для передачі певних файлів, таких як зображення. Ця бібліотека використовує анотації для позначення типів Інтернет-запитів:

- Реферат GET;
- Реферат POST;
- Анотація PUT.

Крім того, для передачі деяких параметрів використовуються анотації про типи цих параметрів:

- абстракція ТІЛА;
- абстракція PATH;
- ГОЛОВНА абстракція;

Щоб отримати данну можливість користуватися цією бібліотекою, треба зробити інтерфейс, що буде описувати всі запити, які отримують дані, і які дані вони надсилають. Щоб користуватися сайтом, потрібно зробити посилання на клас модернізації.

Для цього використовуйте Retrofit.Builder (), цей об’єкт має можливість налаштувати HTTP-клієнт за запитом користувача. Потім вам потрібно скористатися цим посиланням, щоб скористатися методом retrofit.create (serviceClass). Його параметром є serviceClass, який є інтерфейсом, описаним вище. Потім модернізація поверне об’єкт цього інтерфейсу, який ви створили, що дозволить надсилати посилання в Інтернет.

RxJava Library - це реалізація потокових розширень Java. Це дає можливість реалізувати потокове програмування в проекті.

Потокове програмування - це парадигма програмування, яка фокусується на потоках даних та їх розповсюдженні та розповсюдженні змін. Головна його особливість - робота з асинхронними потоками даних.

Насправді ця бібліотека реалізує шаблон програмування Observer. Він забезпечує методи обробки послідовностей даних та конкретних подій, додає велику кількість операторів, які можуть декларативно компілювати та описувати послідовності дій, одночасно усуваючи занепокоєння з таких питань, як безпека потоку, безпека виходу та входу, потоки низького рівня.

Бібліотека Rx базується на двох основних типах, а решта розширюють свою функціональність. Основними типами є Observable та Observer. Тип Observable - це об'єкт, який має можливість модифікувати та повідомляти про певні зміни своїм передплатникам - Observer. Однією з особливостей цієї бібліотеки є різноманітність абонентів з різними кінцевими станами:

- Спостережуваний тип;
- Одинарний тип;
- Плавний тип;
- Комплектується типу;
- Можливо тип.

Кожен тип має різне призначення. Якщо розробнику потрібно отримати лише один результат запиту, використовуйте тип Single. Він має два стани: OnSuccess, OnError.

Перший стан виконується при отриманні результату, другий - при появі помилки.

Якщо результат не є актуальним, але важливо знати, що дія була виконана, скористайтеся програмою, яка заповнюється. Він має ті самі два стани, що й Single, але нічого не повертає.

Тип Maybe на основі його назви може або повернути результат, або ні, і обидва будуть успішними.

Найбільш складними типами є Observable та Flowable - це типи з 3 кінцевими станами: OnNext, OnSuccess, OnError. Останні два мають ті ж властивості, що й інші типи. Стан OnNext викликається, коли дані потрібно надсилати абоненту, але він не останній, тому він не працює повністю. Основною відмінністю двох класів є стратегія зворотного тиску.

Термін зворотний тиск позначає подію, коли Observable отримує набагато більше інформації, ніж його абоненти можуть обробити, що спричиняє проблеми з пам'яттю. Тип Flowable має можливість вирішувати такі проблеми, використовуючи стратегію зворотного тиску.

За допомогою Google Map Api ви можете використовувати карти на основі карт Google у своєму мобільному додатку. Цей API автоматично обробляє доступ до інформації, пов’язаної з картою, отримує таку інформацію, як назви вулиць, назви міст, кілька громадських місць, та обробляє жести на карті. Завдяки цьому API ви можете додавати на карту маркери, багатокутники, які перекриваються та позначають вибрану область. Ви також можете реалізувати групування, якщо на карті занадто багато маркерів. До фрагмента карти можна додати різні графічні елементи:

- Значки, прикріплені до певних позицій на карті (Маркери).
- Набори відрізків ліній (поліліній).
- Вкладені відрізки (багатокутники).
- Графічні растрові зображення прив’язані до певних позицій на карті (наземні накладання).
- Набори зображень, що відображаються у верхній частині основних накладених плиток.

Для використання цього API необхідно отримати спеціальний ключ через сервіс Google Cloud Platform, який використовується майже з усіма сервісами, пов’язаними з геолокацією та картами.

Використовуючи Google Static Map, ми можемо заощадити багато ресурсів, якщо нам потрібно показати карту, з якою неможливо взаємодіяти.

Це можуть бути невеликі фрагменти із розташуванням конкретного об’єкта, що додається до опису. Щоб використовувати цей API, ви також повинні отримати спеціальний ключ, який буде використовуватися в параметрі запиту.

Головна перевага полягає в тому, що для отримання картки потрібно зробити запит в Інтернеті з певними параметрами. Цей запит (рис. 4.8) з часом поверне зображення, яке використовується для відображення карти.

```
https://maps.googleapis.com/maps/api/staticmap?center=Brooklyn+Bridge,New+York,NY&zoom=13&size=600x30
&markers=color:blue%7Clabel:S%7C40.702147,-74.015794&markers=color:green%7Clabel:G%7C40.711614,-74.01
&markers=color:red%7Clabel:C%7C40.718217,-73.998284
&key=YOUR_API_KEY
```

Рисунок 4.8 — Приклад запиту до Google Static Map Api

Однією з переваг є те, що картка надзвичайно гнучка. Велика кількість параметрів дозволяє не тільки відключити або активувати певні маркери, але і змінити колір, наприклад, будівель та доріг. Завдяки цій гнучкості ви можете додавати власні маркери і навіть використовувати власні фотографії.

Одним з необхідних параметрів є ключ. Цей параметр повинен мати власний ключ API. Наступними основними параметрами є координати центру карти, вони описуються словами lat і lnt. Де lat - широта, а lng - довгота. Існує три основні варіанти налаштування стилів об’єктів на карті:

- Параметр функції;
- Параметр елемента;
- Набір правил стилю;
- Параметр Feature відповідає за визначення функцій, які слід вибрати для модифікації стилю, це можуть бути дороги, будівлі, орієнтири.
- Якщо цей параметр не вказаний, карта матиме всі елементи, за які вона відповідає.
- Параметр Element відповідає характеристикам функції, це може бути геометрія або мітки. Не є обов’язковим елементом, якщо його немає у запиті, стиль буде застосовано до всіх елементів зазначеної функції.

ВИСНОВКИ ДО 4 РОЗДІЛУ

У цьому розділі представлений аналіз ринку мобільних пристроїв, їх специфікацій та конфігурацій. Були проаналізовані таблиці версій та конфігурації екрана. Були враховані та визначені межі проекту, визначена функціональність мобільного додатку. Системні вимоги мобільного додатку визначаються відповідно до таких характеристик: сумісність, обсяг пам'яті, специфікація екрану, операційна система, архітектура процесору. В цьому розділі було проведено детальний перегляд Android SDK, особливостей, компонентів. Було обрано нативний додаток в якості розробки, через ефективність та швидкодію. Було розглянуто систему контролю версій Git, показано необхідність застосування у реальному проекті. Показано детальну роботу с цією системою, наведено приклади роботи в проекті. Розглянуто систему автоматичної збірки проекту, її роботу в системі розробки на платформы Android. Розглянуто принцип роботи з програмним середовищем Android Studio, її переваги та мінуси. Розглянуті основні особливості мов Kotlin та Java, порівняно їх переваги та мінуси. Обгрунтовано вибір Kotlin в якості мови програмування та розробки мобільного додатка.

Також у цьому розділі було обрано архітектуру мобільного додатка - MVP. Детально описані основні переваги цієї архітектури, її ідея та взаємодія компонентів системи з основною сутністю. Dagger 2 був обраний в якості основи для реалізації інжекції залежностей, що значно полегшує реалізацію залежностей. Модернізація була обрана як бібліотека http через її простоту в реалізації та описі. Описано основні типи запитів та їх реалізацію. Пояснено цілеспрямованість використання Google Static Map Api та Google Map Api.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ НЕЗАЛЕЖНИХ ТРЕНУВАНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

5.1 Загальний опис роботи додатку

Однією з основних цілей магістерської роботи було впровадження рекомендаційної системи для організації незалежних тренувань з використанням штучного інтелекту. Програмний продукт був реалізований на платформі Android у нативному середовищі.

Після першого запуску програми відкривається головний екран MainActivity (рис. 5.1), який був вказаний у маніфесті проекту як екран запуску. Цей екран перед тим, як намалювати основні елементи екрана - віджети та подання, запитує «Спільні налаштування», щоб перевірити, чи авторизований користувач чи ні.

Спільні налаштування - це, по суті, файл із певними записами, які зберігаються на мобільному пристрої не тільки під час запуску програми, але й коли програма вимкнена. У цьому файлі прийнято зберігати деякі системні налаштування, вибрані користувачем, та маркер авторизації. Саме цей маркер авторизації перевіряє наявність MainActivity.

5.2 Авторизація

Якщо маркер доступний, головний екран залишається, а віджети головного екрану малюються; якщо маркер відсутній, відображається екран авторизації LoginActivity (рис. 5.1), що дозволяє користувачеві зареєструватися або увійти.

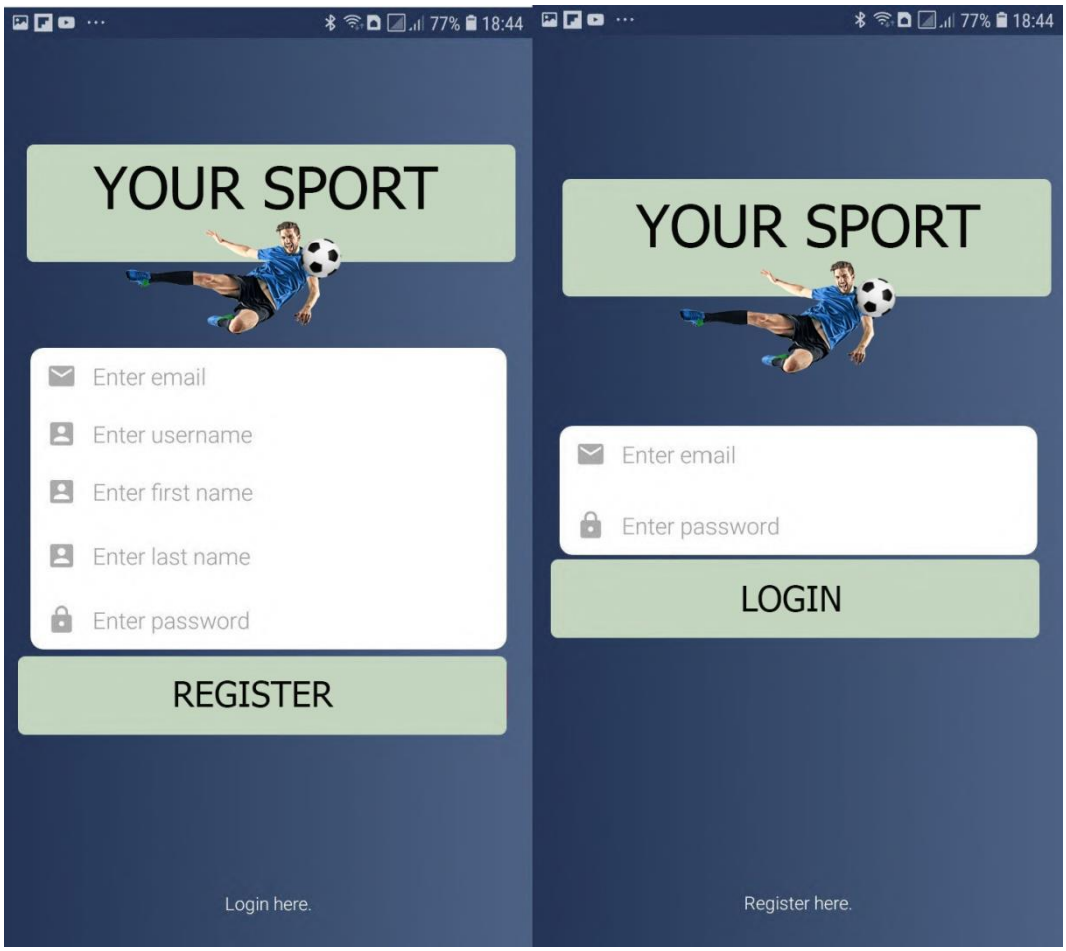

Рисунок 5.1 — Екран авторизації

Після натискання кнопки «Зареєструватися тут» внизу екрана починають з’являтися або зникати додаткові анімовані поля для реєстрації програми.

Існує також перевірка правильності введених даних, але якщо якесь із полів заповнено неправильно, на ньому з’явиться повідомлення. Для перевірки введених даних були прийняті наступні правила:

- Поле електронної пошти повинно відповідати всім стандартам електронної пошти: воно повинно мати знак «@», принаймні один знак собаки, а за собакою повинно бути дійсне доменне ім’я.
- Поле пароля має містити щонайменше шість символів.
- Ім’я користувача, прізвище, ім’я має складатися щонайменше з двох символів.

Після натискання на кнопку «Вхід» або «Реєстрація» сервер запитує, чи всі поля правильні.

Під час усіх тривалих операцій, таких як запит на підключення до Інтернету, відображається віджет «ProgressBar», який вказує на завантаження.

Головний екран мобільного додатку має список усіх актуальних спортивних подій на всіх спортивних майданчика. Кожен елемент списку може натискатися, через користувачеві буде показаний екран події. На головному екрані також є кнопка оновити, що робить запит на сервер та оновлює весь список подій.

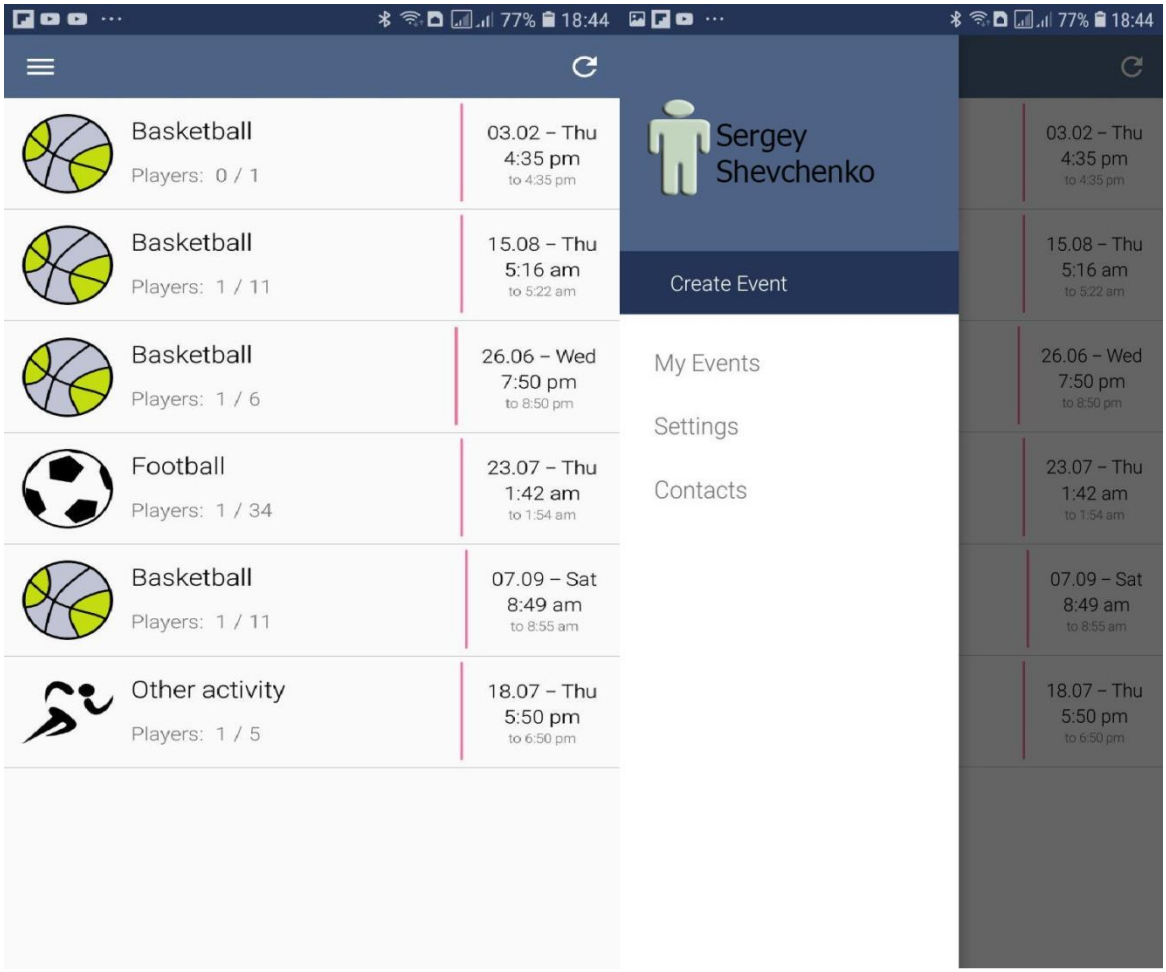

Рисунок 5.2 — Головний екран

5.3 Налаштування

Екран налаштувань (рис. 5.3) дозволяє користувачеві змінювати певну інформацію у своєму профілі. Він містить 3 змінних поля, які перевіряються перед тим, як попросити сервер змінити інформацію. Правила перевірки схожі на екран авторизації.

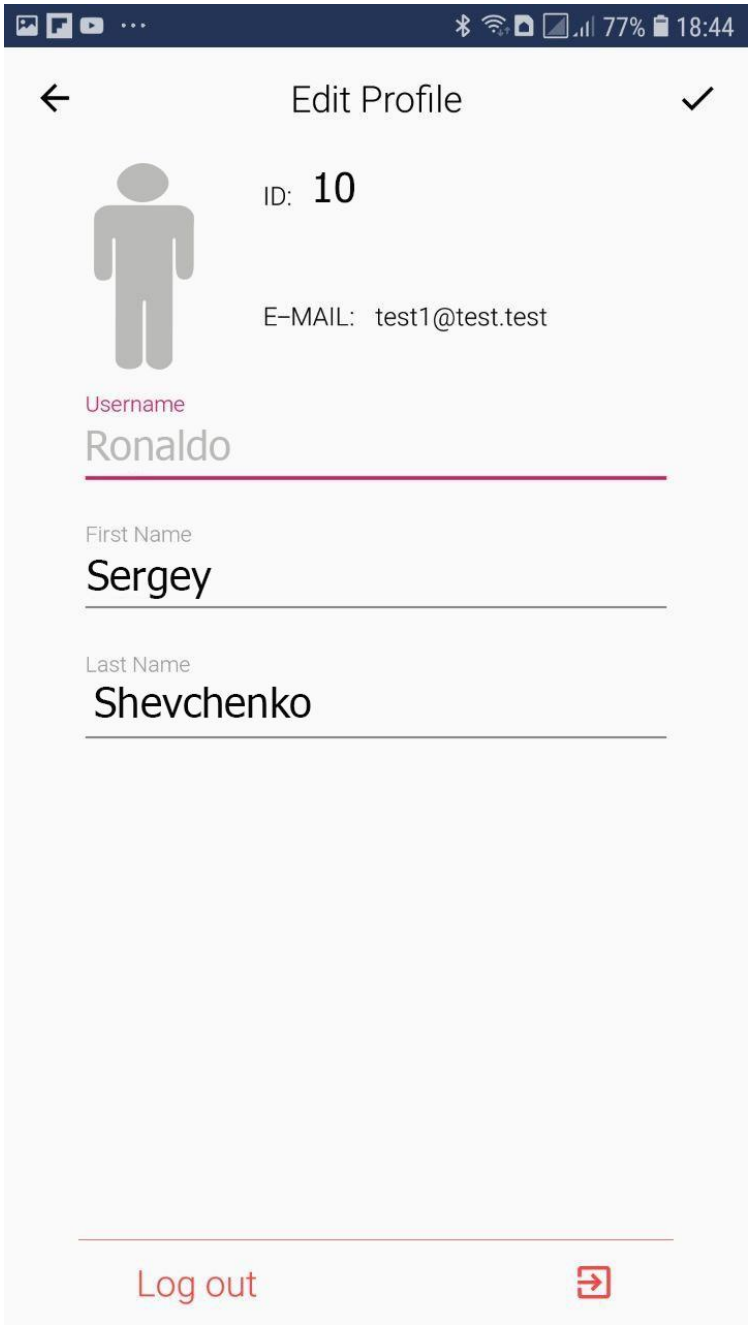

Рисунок 5.3 — Екран налаштувань

Внизу екрану бачимо кнопка, вона дає змогу вийти з мобільного додатка. Після того, як вийшли, всі налаштування налаштуваннях скидаються, включаючи видалення маркера реєстрації.

Екран спортивної події надає повну інформацію про спортивну подію: час початку та закінчення, назву, опис, назву та місце розташування спортивного поля, кількість зареєстрованих гравців та максимальну кількість гравців. Усі поля незмінні.

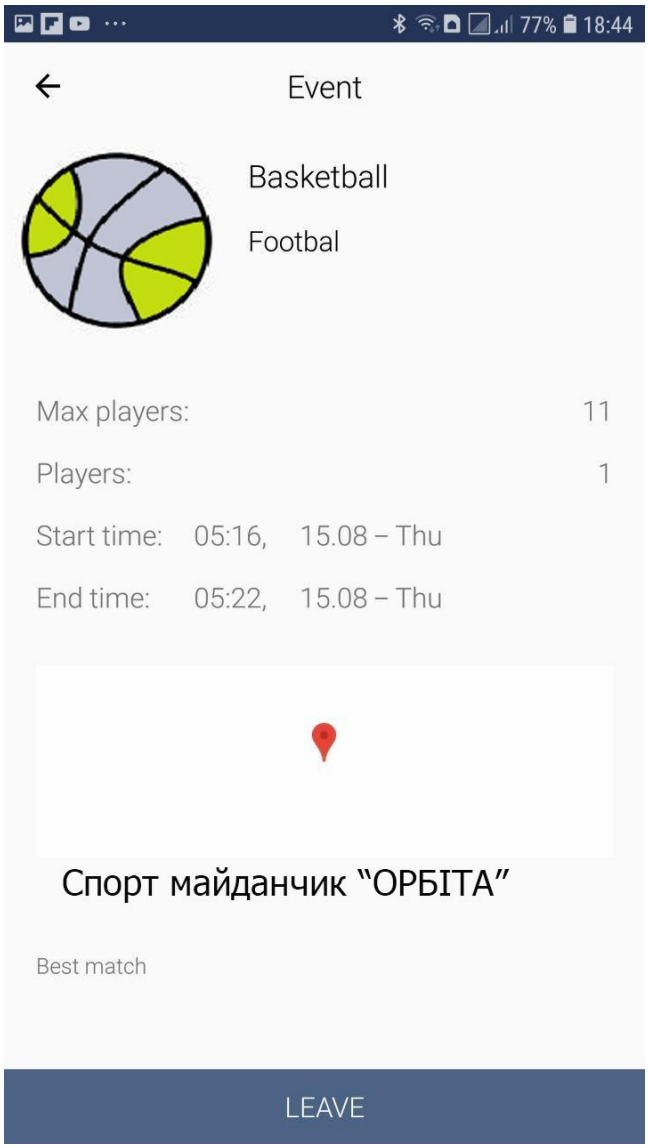

Рисунок 5.4 — Екран спортивної події

На екрані є віджет «ImageView», який дозволяє відображати конкретні зображення. Доступ до цього екрану здійснюється через Google Static Map і ініціалізує віджет.

Крім того, внизу екрана є кнопка, яка дозволяє користувачеві або підключитися до спортивної події, якщо вона ще не підключена, або відключити. Щоразу, коли ви натискаєте, на сервер діє запит на активність.

5.4 Екран спортивної події

Існує два екрани для створення події (рис. 5.5), які дозволяють детально та якісно налаштувати власну подію.

Перший екран дозволяє вибрати спортивний заклад, побачити, де він розташований завдяки інтерактивній карті. Він містить випадаючий список з усіма майданчиками, зареєстрованими в системі.

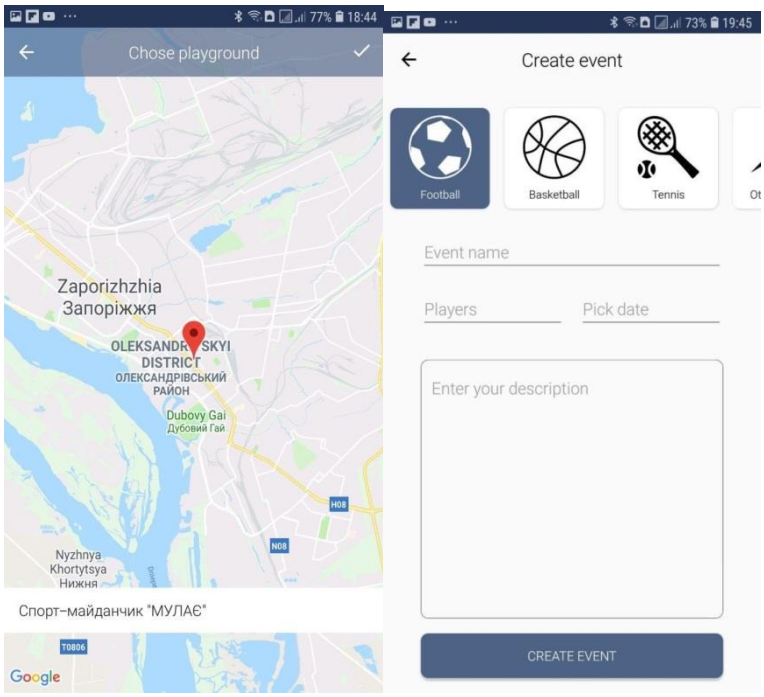

Рисунок 5.5 — Вікно створення події

Другий екран дозволяє вводити деталі про спортивну подію. Користувач може вибрати тип спортивної події, наприклад, футбол, завдяки спеціальному віджету у вигляді списку кнопок з картинками. Крім того, користувач може ввести назву спортивної події, детальний опис події, кількість гравців та час завдяки спеціальним полям введення. Після натискання поля «Вибрати дату» з’являється спеціальне діалогове вікно для вибору часу та дати (рис. 5.6).

Рисунок 5.6 — Вікно вибору часу та дати тренування

5.5 Створення події

Екран «власні події» (рис. 5.7) надає користувачеві доступ до власних подій, до яких він приєднався або створив. На екрані є список, подібний до списку на головному екрані, він також має елементи, які натискаються та ведуть на сторінку події. Крім того, у верхній частині екрана є дві вкладки, які дозволяють користувачеві змінювати результати власних подій, як створених, так і вкладених.

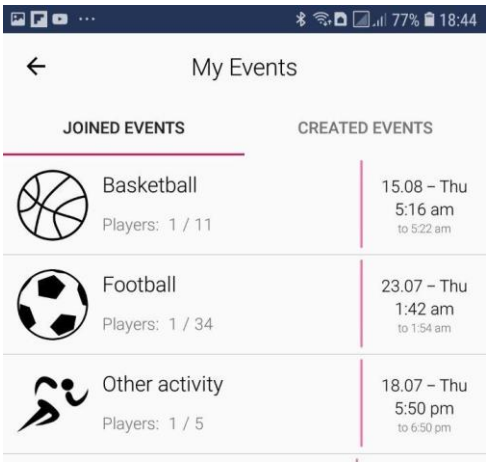

Рисунок 5.7 — Екран власних подій

ВИСНОВКИ ДО 5 РОЗДІЛУ

У п’ятому розділі наведено детальний опис програми. Попередньо побудована функціональна технологія має перевагу, заощаджуючи час, необхідний для запуску програми. Тому цю технологію слід застосовувати у всебічному моделюванні гідроакустичних процесів, що забезпечить більш швидкі результати.

ЗАГАЛЬНІ ВИСНОВКИ

Проаналізовано ринок користувачів, які використовують даний тип додатків. технології створення мобільних додатків. Поставлене конкретне технічне завдання та вимоги щодо роботи мобільного додатка та його якості. Досліджені типи мобільних додатків, їх властивості та переваги. Досліджені програмні мови для розробки, визначено їх плюси, мінуси, та зроблено висновки щодо використання. Також досліджено конкурентів та аналогів даного додатка. Було досліджено інструменти на технології для розробки мобільних додатків. Було спроектовано архітектуру мобільного додатку, механізм спілкування програми з сервером. Реалізовано та спроектовано зручний інтерфейс системи бронювання для мобільної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Методичні вказівки щодо розробки та оформлення магістерської атестаційної роботи за спеціальністю 8.05010101 – Інформаційні управляючі системи та технології. Освітньо-кваліфікаційний рівень – магістр / Упоряд.: Левикін В.М., Міхнов Д.К., Саєнко В.І., Євланов М.В., Міхнова А.В., Керносов М.А. – Харків: ХНУРЕ, 2012. – 28 С.

2. Chalyi S. Доповнення вхідних даних рекомендаційної системи в ситуації циклічного холодного старту з використанням темпоральних обмежень типу «next» / S. Chalyi, V. Leshchynskyi, I. Leshchynska // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2019. – Т. 4 (56). – С. 105-109. – doi:https://doi.org/10.26906/SUNZ.2019.4.105.

3. Чалий С.Ф., Лещинський В.О., Лещинська І.О. Моделювання контексту в рекомендаційних системах. Науковий журнал «Проблеми інформаційних технологій», 2018, №. 1(023). С. 21-26.

4. Chalyi S. Доповнення вхідних даних рекомендаційної системи в ситуації циклічного холодного старту з використанням темпоральних обмежень типу «next» / S. Chalyi, V. Leshchynskyi, I. Leshchynska // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2019. – Т. 4 (56). – С. 105-109. – doi:https://doi.org/10.26906/SUNZ.2019.4.105.

5. Савчук Т.О., Застосування кластерного аналізу для колаборативної фільтрації / Т.О. Савчук, А.В.Сакалюк // Вісник Хмельницького національного університету. –2011 – №1– С. 186-192

6. Sarwar B. M. Item-based collaborative filtering recommendation algorithms

/ B. M. Sarwar, G. Karypis, J. A. Konstan // Proceedings of ACM WWW ‘01, pp. 285–295, ACM, 2001.

7. Hu Y., Koren Y. and Volinsky C. (2008), “Collaborative filtering for implicit feedback datasets”, Data Mining, ICDM’08. Eighth IEEE International Conference on. IEEE, pp. 263–272.

8. Yehuda Koren, Robert Bell, and Chris Volinsky. (2009), “Matrix factorization techniques for recommender systems”, Computer No.8, pp. 30–37.

9. Linden G. Amazon.com recommendations: Item-to-item collaborative filltering / G. Linden, B. Smith, J. York // IEEE Internet Computing, vol. 7, no. 1, pp. 76–80, 2003.

10. Xiaoyuan Su and Taghi M. Khoshgoftaar «A Survey of Collaborative Filtering Techniques A Survey of Collaborative Filtering Techniques» // Hindawi Publishing Corporation, Advances in Artificial Intelligence archive, USA : 2009. — p. 1-19.

11. Jannach D., Zanker M., Felfernig A. Friedrich G. Recommender Systems. An Introduction. New York: Cambridge University Press 32 Avenue of the Americas, 2011. 352 P.

12. Melville P., Sindhwani V. Recommender systems. Encyclopedia of Machine Learning. 2010. p. 30

13. Guo G., Zhang J. and Yorke-Smith N. (2015), “TrustSVD: Collaborative Filtering with Both the Explicit and Implicit Influence of User Trust and of Item Ratings” AAAI, pp. 123–129

14. Christian Desrosiers and George Karypis. A comprehensive survey of neighborhoodbased recommendation methods. In Recommender systems handbook, pages 107–144. Springer, 2011.

15. Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on, pages 263–272. IEEE, 2008.

16. Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In SDM, volume 5, pages 1–5. SIAM, 2005.

17. Istv´an Pil´aszy and Domonkos Tikk. Recommending new movies: even a few ratings are more valuable than metadata. In Proceedings of the third ACM conference on Recommender systems, pages 93–100. ACM, 2009.

18. Steffen Rendle. Factorization machines with libfm. ACM Transactions on Intelligent Systems and Technology (TIST), 3(3):57, 2012.

19. Nathan Srebro, Tommi Jaakkola, et al. Weighted low-rank approximations. In ICML, volume 3, pages 720–727, 2003.

20. Cantador I., Konstas I., Jose J. Categorising social tags to improve folksonomy-based recommendations // Web Semantics: Science, Services and Agents on the World Wide Web. — 2011. — Vol. 9, no. 1. — P. 1–15.

21. Groh G., Ehmig C. Recommendations in taste related domains: collaborative filtering vs. social filtering // Proceedings of the 2007 international ACM conference on Supporting group work / Citeseer. — 2007. — P. 127–136.

22. Matrix factorization and neighbor based algorithms for the netflix prize problem / G. Tak´acs, I. Pil´aszy, B. N´emeth, D. Tikk // Proceedings of the 2008 ACM conference on Recommender systems / ACM. — 2008. — P. 267–274.

23. Linden G. Amazon.com recommendations: Item-to-item collaborative filltering / G. Linden, B. Smith, J. York // IEEE Internet Computing, vol. 7, no. 1, pp. 76–80, 2003.

24. Xiaoyuan Su and Taghi M. Khoshgoftaar «A Survey of Collaborative Filtering Techniques A Survey of Collaborative Filtering Techniques» // Hindawi Publishing Corporation, Advances in Artificial Intelligence archive, USA : 2009. — p. 1-19.

25. Jannach D., Zanker M., Felfernig A. Friedrich G. Recommender Systems. An Introduction. New York: Cambridge University Press 32 Avenue of the Americas, 2011. 352 P.

26. Melville P., Sindhwani V. Recommender systems. Encyclopedia of Machine Learning. 2010. p. 30

27. Guo G., Zhang J. and Yorke-Smith N. (2015), “TrustSVD: Collaborative Filtering with Both the Explicit and Implicit Influence of User Trust and of Item Ratings” AAAI, pp. 123–129

28. Офіційний сайт бібліотеки brain.js. – [Електронний ресурс] - Режим доступу https://brain.js.org/.

29. Офіційний сайт бібліотеки machinelearn.js. – [Електронний ресурс] - Режим доступу https://www.machinelearnjs.com/.

30. Офіційний сайт бібліотеки math.js. – [Електронний ресурс] - Режим доступу https://mathjs.org/.

31. Репозиторій бібліотеки face-api.js. – [Електронний ресурс] - Режим доступу https://github.com/justadudewhohacks/face-api.js/.

32. Офіційний сайт бібліотеки r.js. – [Електронний ресурс] - Режим доступу https://requirejs.org/.

33. Офіційний сайт бібліотеки stdlib-js. – [Електронний ресурс] - Режим доступу https://stdlib.io/.

34. Офіційний сайт бібліотеки tensorflow.js. – [Електронний ресурс] - Режим доступу https://www.tensorflow.org/js.

ДОДАТОК

```
package com.kpi.maydanchik.screens.main
import
com.kpi.maydanchik.core.base.Ba
sePresenter                    import
com.kpi.maydanchik.data.events
.Event
                import    com.kpi.maydanchik.screens.main.domain.MainInteractor
    class     MainPresenter(private     val     interactor:     MainInteractor)     :
      BasePresenter<MainView>(interactor)    {    override    fun    onAttach(view:
      MainView) {
                      s
         uper.on
         Attach(
         view)
         getEve
         nts()
                }

      private  fun  getEvents()  {
         mvpView?.showLoading
         ()
         interactor.getEvents(::onG
         etEvent, ::onError)
                }

      fun
         refresh(
         )       {
         mvpVie
         w?.sho
         wLoadi
         ng()
                      interactor.refresh({ mvpView?.hideLoading() }, ::onError)
                }

      private                  fun
         onGetEvent(events:
         List<Event>)           {
         mvpView?.hideLoadi
         ng()
         mvpView?.onGetEven
         ts(events)
                }
              }
              package com.kpi.maydanchik.screens.main

              i
    mport
    android.cont
    ent.Intent
    import
```

```
android.os.
Bundle
        import androidx.core.view.GravityCompat
        import
androidx.recyclerview.widget.LinearLa
youtManager                 import
com.kpi.maydanchik.R
        import
com.kpi.maydanchik.core.base
.BaseActivity          import
com.kpi.maydanchik.data.ev
ents.Event
        import com.kpi.maydanchik.screens.event.EventActivity
        import
com.kpi.maydanchik.screens.main.di.DaggerMainActivit
yComponent                                  import
com.kpi.maydanchik.screens.main.di.MainActivityM
odule
        import
com.kpi.maydanchik.screens.main.ui.E
ventsAdapter                import
kotlinx.android.synthetic.main.activit
y_main.* import javax.inject.Inject
class   MainActivity   :
  BaseActivity(),
  MainView     {     val
  component by lazy {
          DaggerMainActivityComponent.builder()
              .applicationComponent(getComponent())
              .mainActivityModule(MainActivityModule())
              .build()
        }

        @Inject
  lateinit          var
  presenter:
  MainPresenter
  private  lateinit  var
  adapter:
  EventsAdapter
  override                  fun
    onCreate(savedInstanceState:
    Bundle?)                  {
    super.onCreate(savedInstanc
    eState)
    setContentView(R.layout.acti
    vity_main)
    component.inject(this)
          p
    resenter.
    onAttac
    h(this)
```

```
    initVie
    ws()
            }

  private   fun
    initView
    s()       {
    filterTv.s
    etOnClic
    kListener
    { }
    refreshTv.setOnClickListener   {   presenter.refresh()   }
    menuIv.setOnClickListener                                 {
    drawer.openDrawer(GravityCompat.START)                    }
    initEventList()
            }

            private fun initEventList() {
              adapter = EventsAdapter(::showEvent)


          eventsList.layoutManager          =
    LinearLayoutManager(this)
    eventsList.adapter = adapter
            }

            private fun showEvent(id: Int) {
              val        intent        =
    Intent(this,
    EventActivity::class.java)
    intent.putExtra(EventActivity.INT
    ENT_EVENT_ID,                 id)
    startActivity(intent)
            }

  override                    fun
    onGetEvents(events:
    List<Event>)              {
    adapter.events = events
            }
          }
          package com.kpi.maydanchik.screens.main.di

          import com.kpi.maydanchik.core.network.ServiceProvider
          import
com.kpi.maydanchik.data.preferences.Preferenc
esRepository                              import
com.kpi.maydanchik.data.user.UserRepositor
y
          import com.kpi.maydanchik.di.ActivityScope
          import
com.kpi.maydanchik.di.app.Application
Component                         import
```

```
com.kpi.maydanchik.screens.main.M
ainActivity                    import
com.kpi.maydanchik.utils.Workers
        import dagger.Component

        @ActivityScope
        @Component(dependencies = [ApplicationComponent::class], modules =
[MainActivityModule::class])interface MainActivityComponent {
          fun inject(activity: MainActivity)

          fun
  serviceProvider(
  ):
  ServiceProvider
  fun    workers():
  Workers
          fun preferencesRepository(): PreferencesRepository

          fun userRepository(): UserRepository
        }
        package com.kpi.maydanchik.screens.main.di

        import
com.kpi.maydanchik.core.network.Se
rviceProvider                  import
com.kpi.maydanchik.data.events.ApiEv
entsRepository                 import
com.kpi.maydanchik.data.events.Eve
ntsRepository                  import
com.kpi.maydanchik.data.events.Eve
ntsService
        import
com.kpi.maydanchik.data.preferences.Preferenc
esRepository                           import
com.kpi.maydanchik.di.ActivityScope
        import com.kpi.maydanchik.di.FragmentScope
        import com.kpi.maydanchik.screens.main.MainPresenter
        import
com.kpi.maydanchik.screens.main.domain.Main
Interactor                             import
com.kpi.maydanchik.utils.Workersimport
dagger.Module                          import
dagger.Provides@Module
        class MainActivityModule {@ActivityScope @Provides
          fun provideMainPresenter(interactor: MainInteractor) = MainPresenter(interactor)
          @ActivityScope @Provides
  fun                provideMainInteractor(preferencesRepository:
                PreferencesRepository,          eventsRepository:
                EventsRepository,
                workers:          Workers)          =
      MainInteractor(preferencesRepository,
      eventsRepository, workers)
```

```
}
package com.kpi.maydanchik.di.app

import android.content.Context
import
android.content.Shared
Preferences import
com.kpi.maydanchik.M
aydanchikApp
import
com.kpi.maydanchik.core.network.RetrofitServi
ceProvider import
com.kpi.maydanchik.core.network.ServicePro
vider
import com.kpi.maydanchik.core.network.TokenInterceptor

import
com.kpi.maydanchik.data.events.ApiEv
entsRepository import
com.kpi.maydanchik.data.events.Eve
ntsRepository import
com.kpi.maydanchik.data.events.Eve
ntsService
import
com.kpi.maydanchik.data.preferences.Preferenc
esRepository import
com.kpi.maydanchik.data.preferences.PrefsRe
pository import
com.kpi.maydanchik.data.user.ApiUserRepos
itory
import
com.kpi.maydanchik.data.user.User
Repository import
com.kpi.maydanchik.data.user.Us
erService import
com.kpi.maydanchik.di.ActivityS
cope
import
com.kpi.maydanchi
k.utils.Workers
import
dagger.Module
import dagger.Provides
import
io.reactivex.android.schedulers.Android
Schedulers import
io.reactivex.schedulers.Schedulers
import javax.inject.Singleton

@Module
class ApplicationModule(val
  application: MaydanchikApp) {
  companion object {
```

```
        const val PREFERENCES = «PREFERENCES»
        const val BASE_URL = «http://46.101.218.209/»
    }@Provides @Singleton
    fun provideApplicationContext(): Context = application@Provides @Singleton
    fun provideWorkers(): Workers = Workers(Schedulers.io(), AndroidSchedulers.mainThread())@Provides @Singleton
fun provideServiceProvider(tokenInterceptor: TokenInterceptor):
    ServiceProvider = RetrofitServiceProvider(BASE_URL,
    tokenInterceptor)@Provides @Singleton
fun provideTokenInterceptor(prefRepository:
    PreferencesRepository): TokenInterceptor =
    TokenInterceptor(prefRepository)@Provides @Singleton
fun providePreferencesRepository(preferences: SharedPreferences):
    PreferencesRepository = PrefsRepository(preferences)@Provides
    @Singleton
fun provideSharedPreferences(): SharedPreferences =
    application.getSharedPreferences(PREFERENCES,
    Context.MODE_PRIVATE)@Provides @Singleton
    fun provideUserService(serviceProvider: ServiceProvider) =
serviceProvider.createService(UserService::class.java)@Provides @Singleton
    fun provideUserRepository(userService: UserService): UserRepository = ApiUserRepository(userService)@Singleton
@Provides
    fun provideEventRepository(service: EventsService): EventsRepository = ApiEventsRepository(service)@Singleton
@Provides
    fun provideEventService(serviceProver: ServiceProvider) = serviceProver.createService(EventsService::class.java)
}
package com.kpi.maydanchik.di.app

import com.kpi.maydanchik.MaydanchikApp
import
com.kpi.maydanchik.core.network.Serv
iceProvider import
com.kpi.maydanchik.data.events.Eve
ntsRepository
import
com.kpi.maydanchik.data.preferences.Preferenc
esRepository import
com.kpi.maydanchik.data.user.UserRepositor
y
import com.kpi.maydanchik.utils.Workers

i
mport
dagger.
Compo
nent
import
javax.i
nject.S
ingleto
n@Sin
gleton
```

```
@Component(modules =
[ApplicationModule::class]) interface
ApplicationComponent {

    fun inject(app: MaydanchikApp)

    fun serviceProvider(): ServiceProvider

    fun
  preferencesRepository():
  PreferencesRepository fun
  eventRepository():
  EventsRepository
    fun workers(): Workers

    fun userRepository(): UserRepository
  }
  package com.kpi.maydanchik.core.base

  i
mport
android.cont
ent.Intent
import
android.wid
get.Toast
  import
androidx.appcompat.app.App
CompatActivity import
com.kpi.maydanchik.Mayda
nchikApp
  import
com.kpi.maydanchik.screens.login.Login
Activity abstract class BaseActivity :
AppCompatActivity(), BaseView {
    private var loadingDialog: LoadingDialog? = null
    protected fun getComponent() = (application as
  MaydanchikApp).applicationComponentoverride fun hideLoading()
  {
      loadingDialog?.dismiss()
    }

    override fun showError(error: Throwable) {
      Toast.makeText(this, error.message, Toast.LENGTH_LONG).show()
    }

    override fun showError(error: String?) {
      error?.let { Toast.makeText(this, error, Toast.LENGTH_LONG).show() }
    }

  override fun
    showLoadi
    ng() {
```

```
    loadingDial
    og          =
    LoadingDia
    log()
                loadingDialog?.show(supportFragmentManager,     LoadingDialog::class.java.simpleName)
            }

            override fun showNetworkErrorDialog() {
              //todo
            }

            override fun showInitScreen() {
              val       intent       =       Intent(this,       LoginActivity::class.java)
    intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK                                  or
    Intent.FLAG_ACTIVITY_CLEAR_TASK)startActivity(intent)
            }

        }
        package com.kpi.maydanchik.core.base

        import
io.reactivex.disposables.Composite
Disposable      abstract      class
BaseInteractor {
            protected val disposables = CompositeDisposable()

            open fun unsubscribe() = disposables.clear()
        }
        package com.kpi.maydanchik.core.base

        import android.os.Bundle
        import androidx.fragment.app.Fragment


abstract  class  BaseFragment  :
  Fragment(),  BaseView  {  val
  safeActivity: BaseActivity
              get() {
                return activity as BaseActivity
              }val safeArguments: Bundleget() {
                return arguments ?: throw NullPointerException(«Arguments are null.»)
              }override fun hideLoading() {safeActivity.hideLoading()
            }override fun showError(error: Throwable) {safeActivity.showError(error)
            }override fun showInitScreen() {safeActivity.showInitScreen()
  }package
com.kpi.maydanchik
.core.base   interface
BaseView       {fun
hideLoading()    fun
showLoading()fun
showError(error:
Throwable)       fun
showError(error:
```

```
String?)              fun
showNetworkErrorD
ialog()fun
showInitScreen()}
  override                    fun
    showLoading()             {
    safeActivity.showLoading
    ()
              }

  override                    fun
    showNetworkErrorDialo
    g()                       {
    safeActivity.showNetwor
    kErrorDialog()
              }

  override          fun
    showError(erro
    r:    String?)    {
    safeActivity.sh
    owError(error)
              }
            }
            <?xml version=«1.0» encoding=«utf-8»?>
<androidx.constraintlayout.widget.ConstraintLayout
  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=«matc
  h_parent»>

  <ImageView
    android:id=«@
    +id/backIv»
    android:layout_
    width=«wrap_c
    ontent»
                android:layout
    _height=«wrap_content»
    android:padding=«16dp»
    android:src=«@drawable/
    ic_arrow_black»
    app:layout_constraintStart
    _toStartOf=«parent»
    app:layout_constraintTop_t
    oTopOf=«parent» />

  <TextView
    android:layout_width=«
    wrap_content»
```

```
    android:layout_height=
    «wrap_content»
    android:text=«Contacts
    »
    android:textColor=«@an
    droid:color/black»
    android:textSize=«20sp
    »
                app:layout_constraintBottom_toBottomOf=«@id/backIv»
    app:layout_constraintLeft_toLeftOf=«parent»
app:layout_constraintRight_toRightOf=«parent»
    app:layout_constraintTop_toTopOf=«@id/backIv» />

 <TextView
    android:id=«@+i
    d/addressTv»
    android:layout_
    width=«wrap_co
    ntent»
    android:layout_he
    ight=«wrap_conte
    nt»
    android:layout_
    marginTop=«32
    dp»
    android:layout_
    marginBottom=«
    32dp»
                android:text=«22 Yangelya st, Kyiv, Ukraine»
    app:layout_constraintBottom_toBottomOf=«parent»
    app:layout_constraintLeft_toLeftOf=«parent»
    app:layout_constraintRight_toRightOf=«parent» />

 <ImageView
    android:layout_width=«m
    atch_parent»
    android:layout_height=«wr
    ap_content»
    android:src=«@drawable/
    map»
                app:layout_constraintBottom_toBottomOf=«parent»
    app:layout_constraintLeft_toLeftOf=«parent»
    app:layout_constraintRight_toRightOf=«parent»
    app:layout_constraintTop_toTopOf=«parent» />
            </androidx.constraintlayout.widget.ConstraintLayout>


            package
com.kpi.maydanchik.sc
reens.login        import
android.content.Intent
            import android.os.Bundle
```

```
import
com.kpi.maydanchik.M
aydanchikApp import
com.kpi.maydanchik.R
import com.kpi.maydanchik.core.base.BaseActivity
import
com.kpi.maydanchik.screens.login.di.DaggerLo
ginComponent import
com.kpi.maydanchik.screens.login.di.LoginM
odule
import
com.kpi.maydanchik.screens.main.
MainActivity import
com.kpi.maydanchik.utils.setClic
kability
import
kotlinx.android.synthetic.main.acti
vity_login.* import
javax.inject.Inject
class LoginActivity :
  BaseActivity(),
  LoginView { private val
  component by lazy {
                DaggerLoginComponent.builder()
                    .applicationComponent((application as MaydanchikApp).applicationComponent)
                    .loginModule(LoginModule(this))
                    .build()}private var forLogin = true@Inject
              lateinit var presenter: LoginPresenter

  override fun
    onCreate(savedInstanceState:
    Bundle?) {
    super.onCreate(savedInstanc
    eState)
    setContentView(R.layout.acti
    vity_login)
    component.inject(this)
                initViews()
              }override fun onDestroy() {super.onDestroy() presenter.onDetach()
              }

  private fun initViews() {
    loginButton.setOnClickLi
    stener {
                  if(forLogin)
        presenter.login(emailEt.text.toString(),
      passwordEt.text.toString()) else
        presenter.register(em
            ailEt.text.toStri
            ng(),
            usernameEt.te
            xt.toString(),
```

```
                firstnameEt.tex
                t.toString(),
                lastnameEt.tex
                t.toString(),


                    passwordEt.text.toString())
                }

    registerEt.setOnClickListener   {
    changeScreenState()             }
    loginEt.setOnClickListener      {
    changeScreenState() }
            }

            private fun changeScreenState() {
                if    (forLogin)
    motionLayout.transitionT
    oEnd()                  else
    motionLayout.transition
    ToStart()

    forLogin   =   !forLogin
    registerEt.setClickability(f
    orLogin)
    loginEt.setClickability(!f
    orLogin)
                loginButton.text = if (forLogin)  getString(R.string.hint_login)  else  getString(R.string.hint_register)
            }

private             fun             setRegisterFieldsClickable()            {
    firstnameEt.setClickability(!firstnameEt.isClickable)
    lastnameEt.setClickability(!lastnameEt.isClickable)
    usernameEt.setClickability(!usernameEt.isClickable)
            }override fun showAuthError() {
                showError(«You have entered wrong email or password.»)}

overridefun
showEmail
Error()     {
emailEt.err
or          =
«Invalid
email»}

override            fun
showPasswordErro
r()                     {
passwordEt.error  =
«Invalid password»
            }
```

```
override fun
showUsernameE
rror() {
usernameEt.error
= «Invalid
username»
}

override fun
showFirstna
meError() {
firstnameEt.
error =
«Invalid
name»
}

override fun
showSecondna
meError() {
lastnameEt.err
or = «Invalid
name»
}

override fun startMainActivity() {
val intent = Intent(this, MainActivity::class.java)
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK)startActivity(intent)
}
}
package com.kpi.maydanchik.screens.login

import com.kpi.maydanchik.core.base.BasePresenter
import
com.kpi.maydanchik.screens.login.domain.Logi
nInteractor import retrofit2.HttpException
import java.net.HttpURLConnection
class LoginPresenter(private val interactor: LoginInteractor) :
BasePresenter<LoginView>(interactor) { fun login(email: String, password:
String) {
mvpView?.showLoading()
interactor.lo
gin(email,
password,
{
mvpView
?.hideLoa
ding()
mvpView
?.startMai
nActivity(
```

```
            )},
            ::onLogin
            Error)}
fun register(email: String, username: String, firstname: String, lastname: String,
password: String) {mvpView?.showLoading()
        interactor.register(email, username, firstname,
            lastname,              password,              {
            mvpView?.hideLoading()
                    mvpView?.startMainActivity()

                }, ::onInvalidValueError)
            }

        private                fun
            onLoginError(error:
            Throwable) {if (error
            is HttpException) {
              when        (error.code())        {
                HttpURLConnection.HTTP_U
                NAUTHORIZED -> {
                            mvpV
                  iew?.hideLoadin
                  g()
                  mvpView?.showA
                  uthError()
                        }
                    }
                } else onInvalidValueError(error)
             }

        private                        fun
            onInvalidValueError(error:
            Throwable)                 {
            mvpView?.hideLoading()
                    when (error) {
                      is  LoginInteractor.InvalidPasswordException  ->
              mvpView?.showPasswordError()                          is
              LoginInteractor.InvalidEmailException                 ->
              mvpView?.showEmailError()
                      is LoginInteractor.InvalidFirstNameException -> mvpView?.showFirstnameError()
                      is    LoginInteractor.InvalidSecondNameException    ->
              mvpView?.showSecondnameError()                         is
              LoginInteractor.InvalidUsernameException               ->
              mvpView?.showUsernameError()
                      else -> onError(error)
                  }
                }
```