

Міністерство освіти і науки України
Державний ВНЗ «Національний гірничий університет»

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
дипломної роботи

магістра
(назва освітньо-кваліфікаційного рівня)

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

спеціалізація *Програмне забезпечення систем*
(код і назва спеціалізації)

освітній рівень *магістр*
(назва освітнього рівня)

кваліфікація *2132.2 Інженер-програміст*
(назва кваліфікації)

на тему: *Дослідження ефективності обробки великих
об'ємів даних на прикладі використання хмарного та виділеного
серверів у мобільному додатку*

Виконавець:

студент 2 курсу, групи 121М-16-1

(підпис)

Козаков А. О.

(прізвище та ініціали)

Керівники	Посада, прізвище, ініціали	Оцінка	Підпис
проекту	<i>проф. Куваєв В.М.</i>		
розділів:			
Економічний	<i>доц. Касьяненко Л.В.</i>		

Рецензент	<i>проф. Головка В. І.</i>		
-----------	----------------------------	--	--

Нормоконтроль	<i>доц. Коротенко Л.М.</i>		
---------------	----------------------------	--	--

Дніпро
2018

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів, що очікуються, полягає в реалізації та використанні власної системи аналізу даних серверних технологій в мобільних додатках, та розробки мобільного додатка для тестування описаної методики.

Практична цінність результатів полягає в розробці програмного продукту, що дозволить оцінити переваги та недоліки новітніх технологій розробки серверних систем для мобільних додатків.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати магістерської роботи повинні відповідати вимогам паспорту наукової спеціальності 121 – «Інженерія програмного забезпечення». Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання системи аналізу передачі великих об'ємів даних в серверних технологіях.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Аналіз стану питання «Серверні технології»	1.05.2017 15.06.2017
Використання хмарних та виділених серверних технологій при проектуванні мобільних додатків	16.06.2017 30.07.2017
Використання методики статичного аналізу вхідних даних в мобільних додатках	10.08.2017 25.11.2017
Реалізація мобільного додатка для дослідження ефективності серверних технологій	26.11.2017 04.01.2018

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним та може бути отриманий за рахунок скорочення часу та затрат на розробку програмного забезпечення, інформаційних систем.

Соціальний ефект від реалізації результатів роботи очікується позитивним завдяки створенню програмного забезпечення, що значно зменшує затрати на розробку, а також легкість у використанні для користувачів.

7 ДОДАТКОВІ ВИМОГИ

Відповідність оформлення ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення, ЕДИНОЙ СИСТЕМЕ ПРОГРАММНОЙ ДОКУМЕНТАЦИИ (ЕСПД – ГОСТ 19.101-77, ГОСТ 19.102-77, ГОСТ 19.103-77, ГОСТ 19.104-78, ГОСТ 19.105-78, ГОСТ 19.106-78, ГОСТ 19.201-78, ГОСТ 19.202-78, ГОСТ 19.401-78, ГОСТ 19.402-78, ГОСТ 19.404-79)

Завдання видав	_____	<i>Куваєв В.М.</i>
	(підпис)	(прізвище, ініціали)
Завдання прийняв до виконання	_____	<i>Козаков А.О.</i>
	(підпис)	(прізвище, ініціали)

Дата видачі завдання: 12.03.2017р.

Термін подання дипломного проекту до ДЕК _____

Реферат

Пояснительная записка: 119 с., 30 рис., 18 табл., 4 приложения, 50 источников.

Объект исследования: технология облачной реализации серверной технологии, разработка системы для анализа данных из полученных результатов выполнения операций запросов на сервер.

Цель магистерской работы: разработка программного средства Android приложение для статистического анализа и исследования эффективности обработки больших объемов данных на примере использования выделенного и облачного серверов, который предназначен для оценки использования облачных технологий и выделенных серверов, их сопоставление, выделение преимуществ и недостатков, описания различий с работой над большими объемами данных.

Методы исследования. При решении поставленной задачи использовались научные достижения в областях разработки мобильных приложений и программного обеспечения.

Научная новизна заключается в реализации и использовании собственной системы анализа данных серверных технологии в мобильных приложениях, и разработки мобильного приложения для тестирования описаной методики.

Практическое значение работы заключается в разработке системы анализа полученных данных на примере различных серверных технологий, и построении выводов с использованием диаграмм на мобильном устройстве.

Область применения. Разработанная информационная система может применяться для решения широкого спектра задач, в частности, для создания программного обеспечения, проектирования информационных систем.

Значение работы и выводы. Усовершенствованная методика позволяет проектировать информационные системы со значительным сокращением как материальных затрат, так и временных, что подтверждается разработанным программным продуктом в данной магистерской работе.

Прогнозы по развитию исследований. Разработать универсальные программные модули, которые могут быть использованы для поддержки проектирования мобильных приложений и систем. Разработать комплекс программных средств и пользовательский интерфейс для графического представления результатов.

В разделе «Экономика» проведены расчеты трудоемкости разработки программного обеспечения, расходов на создание ПО и длительности его разработки, а также проведены маркетинговые исследования рынка сбыта созданного программного продукта.

Список ключевых слов: ANDROID ПРИЛОЖЕНИЕ, BAAS, JAVA, ANDROID STUDIO, BACKENDLESS, FIREBASE, SQL, ОБЛАЧНЫЙ СЕРВИС, MATERIAL DESIGN.

Реферат

Пояснювальна записка: 119 с., 30 рис., 18 табл., 4 додатки, 50 джерел.

Об'єкт дослідження: технологія хмарної реалізації бекенду, розробка системи для аналізу даних з отриманих результатів виконання операцій запитів на сервер.

Мета магістерської роботи: розробка програмного засобу Android додатку для статистичного аналізу та дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів, який призначений для оцінки використання хмарних технологій та виділених серверів, їх зіставлення, виділення переваг та недоліків, опису відмінностей з роботою над великими об'ємами даних.

Методи дослідження. При рішенні поставленої задачі використовувалися наукові досягнення в областях розробки мобільних додатків та програмного забезпечення.

Наукова новизна полягає в реалізації та використанні власної системи аналізу даних серверних технологій в мобільних додатках, та розробки мобільного додатка для тестування описаної методики.

Практична цінність полягає в розробці системи аналізу отриманих даних на прикладі різних серверних технологій, та побудові висновків з використанням діаграм на мобільному пристрої.

Область застосування. Розроблений програмний додаток може застосовуватися для вирішення широкого спектру завдань, зокрема, для створення програмного забезпечення, проектування інформаційних систем.

Значення роботи та висновки. Удосконалена методика дозволяє проектувати інформаційні системи зі значним скороченням як матеріальних витрат, так і тимчасових, що підтверджується розробленим програмним продуктом в даній магістерській роботі.

Прогнози щодо розвитку досліджень. Розробити універсальні програмні модулі, які можуть бути використані для підтримки проектування мобільних додатків та систем. Розробити комплекс програмних засобів і призначений для користувача інтерфейс для графічного представлення результатів.

У розділі «Економіка» проведені розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки, а також провести маркетингові дослідження ринку збуту створеного програмного продукту.

Список ключових слів: ANDROID ДОДАТОК, BAAS, JAVA, ANDROID STUDIO, BACKENDLESS, FIREBASE, SQL, ХМАРНИЙ СЕРВІС, MATERIAL DESIGN.

THE ABSTRACT

Explanatory slip: 119 p., 30 pic., 6 tab., 18 applications, 50 of sources.

Object of research: the technology of cloud implementation of server technology, the development of a system for analyzing data from the obtained results of the operations of requests to the server.

The purpose of the degree project: software development Android application for statistical analysis and research on the efficiency of processing large amounts of data using the example of using dedicated and cloud servers that is designed to assess the use of cloud technologies and dedicated servers, their comparison, highlighting the advantages and disadvantages, describing the differences with working on large amounts of data.

Methods of research. At the decision of a task in view, scientific achievements in the field of development of mobile applications and software were used.

The scientific novelty. The application has no analogues with the described functions. Therefore, it is necessary to develop a product that can use most of these functions and capabilities, provide auxiliary functions to developers.

The practical value of work is to develop a system for analyzing the data obtained through the example of various server technologies, and drawing conclusions using diagrams on a mobile device.

The scope. The developed information system can be used to solve a wide range of tasks, in particular, for creating software, designing information systems.

The value of the work and conclusions. An advanced methodology allows you to design information systems with a significant reduction in both material costs and time, which is confirmed by the developed software product in this master's work.

Projections on development research. Develop universal software modules that can be used to support the design of mobile applications and systems. Develop a set of software tools and a user interface for graphical presentation of results.

In section "Economics" calculated the complexity of software development, the cost of creating the software and the duration of its development, and marketing studies market created by the software.

List of keywords: ANDROID APPLICATION, BAAS, JAVA, ANDROID STUDIO, BACKENDLESS, FIREBASE, SQL, CLOUD SERVICE, MATERIAL DESIGN.

ЗМІСТ

Перелік скорочень	6
ВСТУП	7
РОЗДІЛ 1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	14
1.1 Призначення та область застосування	14
1.2 Постановка задачі.....	14
1.3 Аналіз предметної області.....	16
1.3.1 Опис проблеми	16
1.3.2 Актуальність дослідження	16
1.3.3 Аналіз останніх досліджень і публікацій	17
1.3.4 Мета	17
1.3.5 Виклад основного матеріалу	17
1.3.6 Висновки	18
1.4 Огляд літературних та інтернет джерел.....	19
1.4.1 Технологія MBaaS.....	19
1.4.2 Хмарний сервіс Firebase	20
1.4.3 Інтерфейс Google Maps Android API.....	22
1.4.4 Інтерфейс Google Places Android API.....	23
1.4.5 Використання бази даних на мобільному пристрої.....	23
1.4.6 Використання Material Design.....	24
РОЗДІЛ 2. ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ	27
2.1 Зовнішнє проектування	27
2.1.1 Опис функціональних характеристик	27
2.1.2 Вхідні дані.....	28
2.1.3 Вихідні дані.....	29
2.1.4 Формалізація задачі.....	31
2.2 Проектування бази даних	32
2.2.1 Специфікація вимог до бази даних	32
2.2.2 Визначення типів сутностей та зв'язків між ними	33
2.2.3 Логічне проектування бази даних	40
2.2.4 Схема бази даних	46
РОЗДІЛ 3. РОЗРОБКА МОБІЛЬНОГО ДОДАТКА ДЛЯ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ХМАРНИХ ТА ВИДІЛЕНИХ СЕРВЕРІВ	48
3.1 Вибір мови програмування	48
3.1.1 Мова програмування Java.....	48
3.1.2 Опис середовища розробки Android Studio	51
3.1.3 Опис СКБД MS SQL Server.....	52

3.2 Об'єктно-орієнтоване проектування	53
3.3 Розробка інтерфейсу	60
3.4 Вибір стратегії тестування	67
3.4.1 Тестування функціональності.....	69
3.4.2 Тест користувацького інтерфейсу	70
3.4.3 Тестування методами чорної шухляди	71
3.4.4 Метод еквівалентних розбиттів	74
3.4.5 Тестування методом граничних умов	78
3.4.6 Тестування методом припущення помилки	80
РОЗДІЛ 4. ЕКОНОМІЧНИЙ РОЗДІЛ	82
4.1. Розрахунок трудомісткості і вартості розробки програмного продукту....	82
4.2. Витрати на створення програмного забезпечення.....	85
4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту	86
4.4. Оцінка економічної ефективності впровадження програмногозабезпечення	89
4.5 Висновки	92
ВИСНОВКИ.....	93
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	95
Додаток А.....	99
Додаток Б	125
Додаток В	126

Перелік скорочень

- ПК – персональний комп'ютер
ОС – операційна система
ПЗ – програмне забезпечення
БД – база даних
BAAS – Backend as a service

ВСТУП

Актуальність роботи. Кожен день людина користується електронними пристроями, завдяки їм спілкується з друзями, заводить нових знайомих, замовляє необхідні товари. Таким чином виникає необхідність для створення нових програм та додатків для роботи з цими пристроями, як розважального характеру, так і для роботи, освіти та ні. Отже мобільна розробка це стрімко розвиваюча галузь, яка охоплює всі сфери життя людини і надає широкий вибір послуг, розрахованих на потреби різного типу. Мобільні додатки допомагають користувачам економити власний час, виконують багато важливих функцій та роблять життя простішим. Дуже мало програм працюють без певного підключення до Інтернету, тобто вони взаємодіють з бекендом, веб-службами або API. Ці API можуть бути надані різноманітними сервісами.

Мета та задачі дослідження. Метою даної магістерської роботи є розробка програмного засобу Android додатку для статистичного аналізу та дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів, який призначений для оцінки використання хмарних технологій та виділених серверів, їх зіставлення, виділення переваг та недоліків, опису відмінностей з роботою над великими об'ємами даних.

Для отримання необхідних результатів були сформульовані наступні задачі:

1. Аналіз існуючих серверних технологій для розробки мобільних додатків, виявлення їх переваг та недоліків.
2. Реалізація додатка для аналізу описаних систем, складання графіків та діаграм з отриманих висновків.

Об'єкт досліджень – технологія хмарної реалізації бекенду, розробка системи для аналізу даних з отриманих результатів виконання операцій запитів на сервер.

Предмет дослідження – методики проектування серверу для мобільних додатків.

Ідея роботи полягає в описі технологій для роботи з бекендом, удосконалення методів аналізу даних на мобільних пристроях, надання детальних висновків у виді графіків та діаграм, опис проблем їх використання та ін.

Методи дослідження. При вирішенні поставленого завдання використовувалися наукові досягнення в областях розробки інформаційних систем і програмного забезпечення.

Наукові положення та очікувані наукові результати.

1. Аналіз існуючих серверних технологій для розробки мобільних додатків, виявлення їх переваг та недоліків.
2. Реалізація додатка для аналізу описаних систем, складання графіків та діаграм з отриманих висновків.

Обґрунтованість і достовірність наукових положень.

Обґрунтованість і достовірність наукових положень, висновків і рекомендацій магістерської роботи обґрунтована коректністю поставлених проблем і прийнятих припущень при математичному описі процесів, обґрунтованістю вихідних посилок, достатнім обсягом вибірки даних і петрифікованими на модельних об'єктах результатами обчислень.

Наукова новизна отриманих результатів полягає в реалізації та використанні власної системи аналізу даних серверних технології в мобільних додатках, та розробки мобільного додатка для тестування описаної методики. Додаток не має аналогів з описаними функціями. Тому необхідно розробити продукт який може використовувати більшість зазначених функцій та можливостей, надавати допоміжні функції розробникам.

Практична цінність отриманих результатів полягає в розробці системи аналізу отриманих даних на прикладі різних серверних технологій, та побудові висновків з використанням діаграм на мобільному пристрої.

Зв'язок роботи з державними програмами, планами науково-дослідних робіт.

Результати дипломної роботи можуть бути використані розробниками програмних продуктів для проектування мобільних додатків, створення програмного продукту, що використовує серверні та хмарні технології.

Особливий внесок магістра складається з:

- вибору методів досліджень і технологій реалізації;
- створення програмної системи, що реалізує механізми роботи з різними серверними технологіями;
- розробці теоретичної частини роботи, в якій досліджені і систематизовані знання про існуючі підходи розробки програмних систем і технології спілкування клієнта з сервером;
- оцінки отриманих результатів.

Апробація результатів магістерської роботи.

Основні положення і результати були докладені та обговорені на студентській науковій конференції.

Структура і обсяг роботи.

Робота складається з вступу, чотирьох розділів і висновків. Містить 119 сторінок друкованого тексту, в тому числі 62 сторінки тексту основної частини з 30 рисунками, списку використаних джерел з 50 найменуваннями на 3 сторінках, 3 додатка на 20 сторінках.

РОЗДІЛ 1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Призначення та область застосування

Програмний додаток використовуватиметься користувачами у яких є мобільний пристрій або планшетний комп'ютер на операційній системі Android та доступ до мережі інтернет. Призначення додатку надання розробнику мобільних додатків отримувати повну інформацію про серверні технології, а також дослідження їх характеристик та відмінностей на власному мобільному пристрої.

Функціональне призначення додатка є отримання детальної інформації про використання різних видів розробки серверної частини мобільних додатків. Користувач може виконати запрограмовані запити на один з вибраних серверів, які розроблені за різними технологіями (хмарні або виділений), отримати інформацію про реакцію на запити, переглянути статистичні дані у вигляді графіків, діаграм, вивести інформацію у лог-файл. Переглядати всю необхідну інформацію завдяки зручному інтерфейсу.

Експлуатаційним призначенням додатка є економія часу розробника мобільних додатків на вибір технології збереження та отримання даних з мережі інтернет. Дослідження принципів розробки мобільних додатків, тестування систем надання послуг по обробці даних; передбачений швидкий, зручний, простий та інтуїтивно зрозумілий інтерфейс користувача, який не потребує наявності спеціальних вміння та навичок для використання додатку; підтримка переважної більшості мобільних пристроїв на операційній системі Android.

Потреба в розробці даного програмного продукту полягає в тому, щоб надавати розробнику мобільних додатків правильний вибір технологій в залежності від цілей використання даних.

1.2. Постановка задачі

Спроекувати та розробити Android додаток для дослідження та статистичного аналізу ефективності обробки великих об'ємів даних на

прикладі використання виділеного та хмарного серверів, який дозволить економити час розробникам програмних продуктів, пропонувати нові та оптимальні технології розробки мобільних додатків та серверів, а також стане корисних для людей, які прагнуть навчитися розробки мобільних додатків. Програма повинна бути складена з двох частин – серверної, яку представляє “backend” за допомогою хмарних сервісів “firebase”, “BackendLess”, “Amazon AWS”, “Microsoft Azure”, “Kinvey”, власного серверу, та клієнтської, на базі мобільної операційної системи Android, яка знаходиться на мобільному пристрої користувача. Серверна частина повинна містити перелік користувачів з даними для входу, запрограмований об’єм даних з запитами на їх отримання, та додання даних з клієнта. Всі запити повинні мати історію та необхідну інформацію для статистичного аналізу. Клієнтська частина повинна мати інтерактивний інтерфейс для перегляду даних з серверу, завантаження даних, формування графіків та діаграм, відображення інформації з запитів та ін.

Додаток повинен пропонувати користувачу наступні функції:

- відображення великої кількості даних, отриманої від запиту на обраній сервер, з часом відгуку, та об’ємом інформації;
- реєстрація та авторизація користувача через соціальні мережі: "facebook", "twitter", "google", або електронну пошту;
- побудова графіку залежності об’єму даних від швидкості виконання запиту на різних серверних технологіях;
- пошук за допомогою запиту на сервер, який повинен повертати результат, додаток буде зберігати час затрачений на відгук серверу та об’єм переданої інформації;
- побудова діаграми залежності з зібраними статистичними даними всіх протестованих серверних технологій;
- відображення результатів аналізу протестованих серверних технологій, використаних в додатку з детальними висновками;
- отримання оповіщень з обраного серверу;

- збереження результатів в лог-файл та мобільну базу даних.

Користувач зможе зберігати отримані результати, додавати до них власні помітки, обмінюватись результатами з друзями через популярні соціальні мережі. Особливістю даного додатку є визначення та аналіз недоліків та переваг описаних технологій, завдяки розробці власного аналізатора даних.

1.3. Аналіз предметної області

1.3.1. Опис проблеми

Найчастіше навіть чітко прописана специфікація протоколу взаємодії сервер/клієнт не може забезпечити коректну інтеграцію. Тому перед розробником виникне проста, очевидна проблема: протоколи клієнта і сервера не збігаються. З точки зору розробки клієнта це призводить до того, що мобільний додаток отримує некоректні дані, або не отримує їх зовсім. При плануванні розробки проекту недооцінюється необхідний обсяг ресурсів і час створення серверу. Ще одна проблема - обмеженість доступних розробнику ресурсів. Найчастіше розробляти сервер доводиться за допомогою тих інструментів і технологій, якими володіють члени команди. Процес виходить тривалим, а сам додаток – досить складним і дорогим з точки зору супроводу. А тривала розробка додатку навіть в простих проектах веде до збільшення витрат і інших ризиків. [1].

1.3.2. Актуальність дослідження

Характерною рисою сучасного суспільства є активне застосування в економіці та перспективне домінування інформаційно-комунікаційних технологій. Завдяки цим технологіям, зокрема їх Internet-компонентам, значна кількість компаній, у тому числі в сфері розробки програмного забезпечення набувають суттєвих конкурентних переваг і отримують можливість ефективного виходу на національні та глобальні ринки [2].

1.3.3. Аналіз останніх досліджень і публікацій

У сфері розробки мобільного бекенду є ряд вагомих розробок, серед яких можна виділити праці Макконнелл С., Хант Е., Томас Д., Мостової Н.О., Нечаюк Л.І., Новикова О.В., П'ятницького Г.Т., Сірого В.М. та інших. Проте особливості та переваги застосування Інтернет-технологій у діяльності цього сегменту залишаються не до кінця дослідженими, через те, що ця тема перебуває на етапі стрімкого розвитку і кожен рік ми отримуємо новіші технології для мобільної розробки. У час науково-технічного прогресу інновації в програмному бізнесі відіграють чи не головну роль у висококонкурентній боротьбі компаній за кожного клієнта. Застосування новітніх технологій розробки мобільних додатків дозволяють програмістам підвищити ефективність програмних продуктів, знаходити нові резерви підвищення якості серверної частини, надання нових послуг.

1.3.4. Мета

Метою цієї роботи є аналіз основних форм застосування інтернет та мобільних технологій на підприємствах сфери розробки мобільних додатків та серверів, а також дослідження досвіду їх запровадження у вітчизняній та закордонній практиці.

1.3.5. Виклад основного матеріалу

Хмарні сервіси дозволяють керувати централізованою базою даних, яка пропонує користувачам спільно використовувати вміст через хмару. Розробнику не потрібно розробляти свій власний сервер з використанням серверної технології, такої як Ruby або PHP. Коли сторонній постачальник послуг відповідає за базову IT-інфраструктуру, розробник додатка більше не відповідає за придбання виділеного сервера або віртуальної машини. У той же час, постачальник послуг може вирішити, як ефективно використовувати свою інфраструктуру, щоб обслуговувати запити від усіх своїх клієнтів. Як наслідок, постачальник послуг зазвичай не зобов'язаний виконувати постійне

навантаження для конкретного клієнта. Замість цього програма обробляє запити від усіх клієнтів одночасно, витрачаючи лише обмежений час на обробку кожного запиту конкретного клієнта. Тому такі постачальники, як правило, платять своїм клієнтам на основі загальної кількості запитів, частоти запитів протягом певного періоду або загального часу, витраченого на обслуговування всіх запитів від клієнта.

Такий самий підхід добре працює для різних типів навантажень, оскільки сторонні постачальники послуг, як правило, мають еластичні, масштабовані послуги. Коли клієнти починають користуватись сторонньою службою, очікувана кількість запитів може бути низькою, а клієнти платять значно менше, ніж придбання, налаштування та управління власною інфраструктурою на місці або в хмарі. Для більшої кількості запитів (або очікуваних, або несподіваних піків навантаження) клієнтам не потрібно додавати сервери, щоб масштабувати їх застосування. Натомість постачальник послуг піклується про збільшення навантаження. Безумовно, обробка більшої кількості запитів обійдеться дорожче, але в більшості випадків таке рішення ще ефективніше, ніж використання спеціалізованої ІТ-інфраструктури.

1.3.6. Висновки

Найпродуктивнішим та найшвидшим каналом передачі інформації у світі є глобальна мережа Інтернет, що дає змогу ефективно використовувати її для впровадження інновацій у розробці мобільних додатків. Специфіка роботи Інтернету дозволяє кардинально змінювати способи і методи взаємодії між суб'єктами бізнесу, підвищувати рівень якості розробки програмних продуктів, зміцнювати конкурентні позиції бізнес суб'єктів, одержувати максимально повну і об'єктивну інформацію для ведення бізнесу, створювати нові перспективні форми соціальної та економічної діяльності.

1.4. Огляд літературних та інтернет джерел

Для реалізації поставленого завдання були знайдені необхідні літературні та веб джерела, в яких були представлені наступні технології розробки:

- використання особливостей мови програмування Java;
- робота з зовнішніми хмарним сервісом Firebase та функціями Firebase API;
- робота з базою даних, яка буде зберігатись на самому пристрої;
- робота з API функціями популярних соціальних мереж;
- використання існуючих бібліотек від Google;
- розробка алгоритмів пошуку місцезнаходження на основі геоданих;
- використання нового дизайну Material.

При розробці додатків Android використовується Java – одна з найбільш поширених мов програмування. Використання Java стало логічної вибором для платформи Android, тому що це потужна, вільна і відкрита мова, відома мільйонам розробників. Досвідчені програмісти Java можуть швидко освоїти Android-програмування, використовуючи інтерфейси Google Android API (Application Programming Interface) та інші розробки незалежних фірм. Мова Java є об'єктно-орієнтована, надає розробникам доступ до потужних бібліотекам класів, що прискорюють розробку програм. Програмування графічного інтерфейсу користувача є керованою подією.

Далі детально наведений опис використаних засобів розробки.

1.4.1. Технологія MBaaS

BaaS – це модель, що дозволяє розробникам веб-додатків і мобільних програм зв'язати їх застосування з серверним хмарним сховищем і API, що виставляються серверними додатками, а також надає такі функції, як управління користувачами, повідомлення оповіщень, інтеграція зі службами соціальних мереж. Веб-додатки вимагають схожий з мобільними додатками набір функцій бекенду, повідомлень та оповіщень, інтеграцію з соціальними

мережами і хмарні сховища. Кожна з цих послуг має власний API, що індивідуально вбудовується в додаток, це – затяжний і трудомісткий для розробників процес. Постачальники BaaS вибудовують міст між інтерфейсом користувача додатка і різними хмарними технологіями за допомогою єдиного API і SDK.

Забезпечення послідовного управління внутрішніми даними значить, що розробникам не потрібно перебудовувати їх бекенд для кожного з сервісів, до яких додатки повинні мати доступ, тим самим заощаджуючи час і гроші.

Ідея BaaS в тому, що замість того, щоб самим розробляти і надалі підтримувати серверні сервіси, можна скористатися готовими наборами, які разом формують необхідний універсальний крос-платформний бекенд для будь-якого проекту. Відповідно, не потрібно взаємодіяти з сервером додатка, базою даних, клієнт-серверної бібліотекою, хостингом, необов'язково писати адміністративну панель, продумувати дизайн свого API. [2]

BaaS може стати стандартом індустрії для прототипів і мобільних, крос-платформних і виробничих програм.

Незважаючи на схожість з іншими хмарно-обчислювальними засобами розробки, таких як “ПЗ як послуга” (SaaS), “інфраструктура як послуга” (IaaS), “платформа як послуга” (PaaS), BaaS відрізняється від них тим, що безпосередньо стосується хмарно-обчислювальних вимог розробників як веб, так і мобільних додатків, надаючи єдині засоби підключення додатків до хмарних сервісів.

1.4.2. Хмарний сервіс Firebase [5]

Firebase – це потужний сервіс, що надає API для зберігання і синхронізації даних в реальному часі, та сервер, на якому ці дані зберігаються. Головними перевагами цього сервісу є: використання бази даних в режимі реального часу (зберігати і синхронізувати дані з базою даних NoSQL хмари. Дані зберігаються в форматі JSON, синхронізуються

для всіх підключених клієнтів в режимі реального часу, і доступні навіть коли додаток переходить в автономний режим); автентифікація (автентифікація користувачів за допомогою електронної пошти та паролю, соціальними мережами Facebook, Twitter, GitHub, Google, анонімність автентифікації, легкість інтеграції з існуючою системою автентифікації); хостинг (сервіс дозволяє здійснювати розгортання веб-додатків в лічені секунди, статичний хостинг активностей, використання захищеного з'єднання за допомогою SSL, та мережевої інфраструктури CDN).

Firebase SDK підтримує розробку крос-платформних мобільних та веб-додатків, також існує можливість підключення до існуючого backend'у використовуючи серверні бібліотеки або REST API функції.

Також перевагою цього сервісу є швидка розробка додатків, що досить помітно зменшує кількість рядків коду порівняно з іншими рішеннями цього типу. На рис. 1.3 зображено панель управління додатком з структурою бази даних, як можна побачити база даних нагадує структуру класу. Крім управління базою даних можна перейти до управління приватністю, аналітикою, автентифікацію.

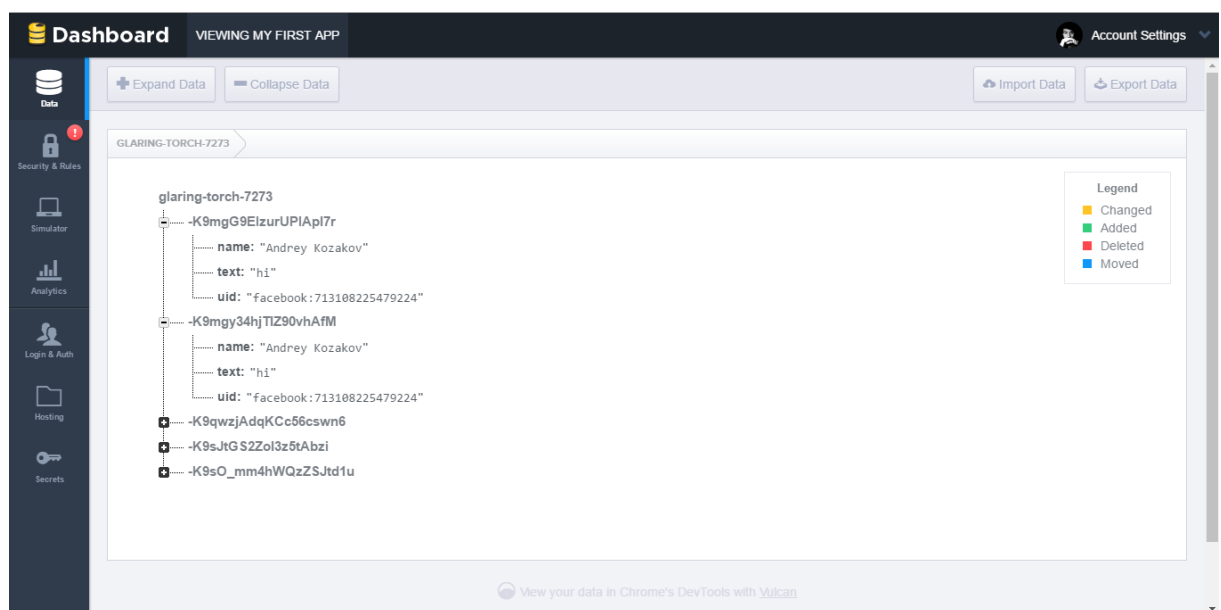


Рис. 1.1. Панель управління додатком

Далі наведена частина коду, яка дозволяє зв'язати сервіс з мобільним додатком, який розробляється:

```
// встановлення з'єднання з власною базою даних Firebase
Firebase ref = new Firebase("https://<ServerAnalizer>.firebaseio.com");
// збереження даних
ref.setValue("Kozakov Andrii");
// чекаємо на зміни у реальному часі
ref.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot snap) {
        System.out.println(snap.getName() + " -> " + snap.getValue());
    }
    @Override public void onCancelled(FirebaseError error) { }});
```

1.4.3. Інтерфейс Google Maps Android API [6]

За допомогою Google Maps Android API можна додавати в свій додаток карти на основі даних Google Карт. Цей API-інтерфейс автоматично управляє доступом до серверів Google Карт, завантаженням даних, відображенням карт і реакцією на жести, виконувані на картах. Крім того, можна використовувати виклики API, щоб додавати маркери, багатокутники і накладення до основної картки, а також змінювати спосіб відображення певної області на карті. Ці об'єкти надають додаткову інформацію про місця на карті і забезпечують можливості взаємодії користувачів з картою. API дозволяє додавати на карту такі графічні елементи:

- позначки, пов'язані з певними місцями на карті (маркери);
- позначки користувача (зображення, фото);
- набори сегментів ліній (ламані лінії);
- замкнені сегменти ліній (багатокутники);
- растрові графічні елементи, пов'язані з певними місцями на карті (наземні накладення);
- набори зображень, які відображаються поверх листів основної карти (мозаїчні накладення).

Для використання Google Maps Android API необхідно підключити останню версію бібліотеки Google Play Services, для цього потрібно додати у розділ «dependencies», файлу build.gradle проекту наступний рядок:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.google.android.gms:play-services:7.5.0'
}
```

1.4.4. Інтерфейс Google Places Android API [7]

За допомогою Google Places можна отримувати географічні дані, дані про організації та картографічні дані для будь-якої точки світу. Можна використовувати базу даних, яка застосовується для служб Google Карти і Google+ Адреси. Ця служба охоплює більше 100 мільйонів організацій і пам'яток, інформація про яких регулярно підтверджується власниками і доповнюється користувачами.

Головні переваги сервісу наведено далі:

- вибір місць – за допомогою вбудованого віджету користувач може вибрати місце зі списку місць поблизу, що відображаються на карті;
- додавання місць – додавання інформації з вашого додатка в базу даних Google Адреса;
- підказки місць – автоматичне надання підказок назв і адрес місць по мірі введення тексту.

Щоб почати роботу з Google Places API for Android, спочатку потрібно додати в свій додаток ключ API-інтерфейсу Google. Всі додатки Android підписуються з використанням цифрових сертифікатів, для яких є приватні ключі. Ключі Android API пов'язані з певними парами "сертифікат-пакет". Для кожного сертифіката буде потрібно лише один ключ, кількість користувачів програми при цьому не має значення. Для того щоб отримати ключ для свого додатку потрібно завітати за адресою <https://developer.android.com/tools/publishing/app-signing.html#debugmode>, та виконати ряд зазначених дій.

1.4.5. Використання бази даних на мобільному пристрої [8]

В ОС Android існує вбудована підтримка бази даних SQLite. Підтримуються всі функції SQLite, надається API оболонки з сумісним

інтерфейсом. API Android SQLite є типовим, розробнику слід реалізувати всю обробку бази даних, включаючи створення, управління версіями, оновлення бази даних та інші налаштування. Якщо потрібно використовувати заздалегідь заповнену базу даних SQLite, потрібна додаткове налаштування.

Залежно від вимог до додатка може знадобитися заздалегідь заповнити базу даних програми початковим набором даних. У випадку з ресторанним додатком заздалегідь заповнюється базовий набір стандартних категорій меню і відомостей про кожен елемент меню.

Android API (наприклад, SQLiteOpenHelper) можна використовувати для заповнення первинного набору даних при створенні та ініціалізації бази даних. Проте такий підхід не завжди є оптимальним, особливо при великому обсязі набору даних. Крім того, не рекомендується використовувати деякі виклики SQLiteOpenHelper в головному потоці. Залежно від потужності пристрою користувачі можуть зіткнутися з тривалою ініціалізацією і значними затримками в роботі призначеного для користувача інтерфейсу при запуску. Ще один можливий підхід – заздалегідь заповнити базу даних і упакувати її в складі ресурсів програми. Залежно від вимог до програми і сценаріїв використання можна додати в ресторанний додаток і інші можливості: підтримку поновлення бази даних, управління версіями і навіть підтримку серверів. При серверної реалізації внутрішньої бази даних можна використовувати локальну базу даних SQLite в якості тимчасового кеша і для можливості автономної роботи.

1.4.6. Використання Material Design [9]

Material design – принцип, система, парадигма дизайну сайтів, програмного забезпечення і додатків, а також правила дизайну інтерфейсів для операційної системи Android від компанії Google. Концепція побудови логіки роботи та зовнішнього вигляду сервісів та додатків, що уніфікує всі продукти від Google. Ідея дизайну полягає в інтерфейсі, поведінка і вигляд

якого наслідують правила поведінки і вигляду паперових карток в реальному житті.

Material Design ґрунтується на чотирьох основних принципах:

- тактильні поверхні. У Material Design інтерфейс складається з відчутних шарів так званого «цифрового паперу». Ці шари розташовані на різній висоті і відкидають тіні один на одного, що допомагає користувачам краще розуміти анатомію інтерфейсу і принцип взаємодії з ним;
- поліграфічний дизайн. Якщо вважати шари шматками «цифрового паперу», то в тому, що стосується «цифрового чорнила» (всього того, що зображується на «цифровому папері»), використовується підхід з традиційного графічного дизайну: наприклад, журнального та плакатного;
- осмислена анімація. У реальному світі предмети не виникають нізвідки і не зникають в нікуди. Тому в Material Design використовуються принципи, що за допомогою анімації в шарах і в «цифрових чорнилах» користувачі отримують підказки про роботу інтерфейсу;
- адаптивний дизайн. Йдеться про те, як в дизайні застосовується попередні три концепції на різних пристроях з різними дозволами і розмірами екранів.

Приклад використання Material design у проекті наведено далі у вигляді xml файлу:

```
<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:contentPadding="8dp"
    card_view:cardBackgroundColor="@color/primary_bgr"
    card_view:cardUseCompatPadding="true"
    card_view:cardCornerRadius="4dp">
<RelativeLayout
```

```
android:id="@+id/articleLayout"  
android:background="@color/primary_bgr"  
android:layout_width="match_parent"  
android:layout_height="wrap_content">
```

//---

Цим рядком виконується створення розмітки елемента CardView, який є віджетом, та імплементує такий елемент дизайну Material, як картка. По суті це контейнер, у якого можна задавати радіус заокруглення кутів, колір картки і висоту по осі z. Вигляд карток зображено на рис. 1.4.

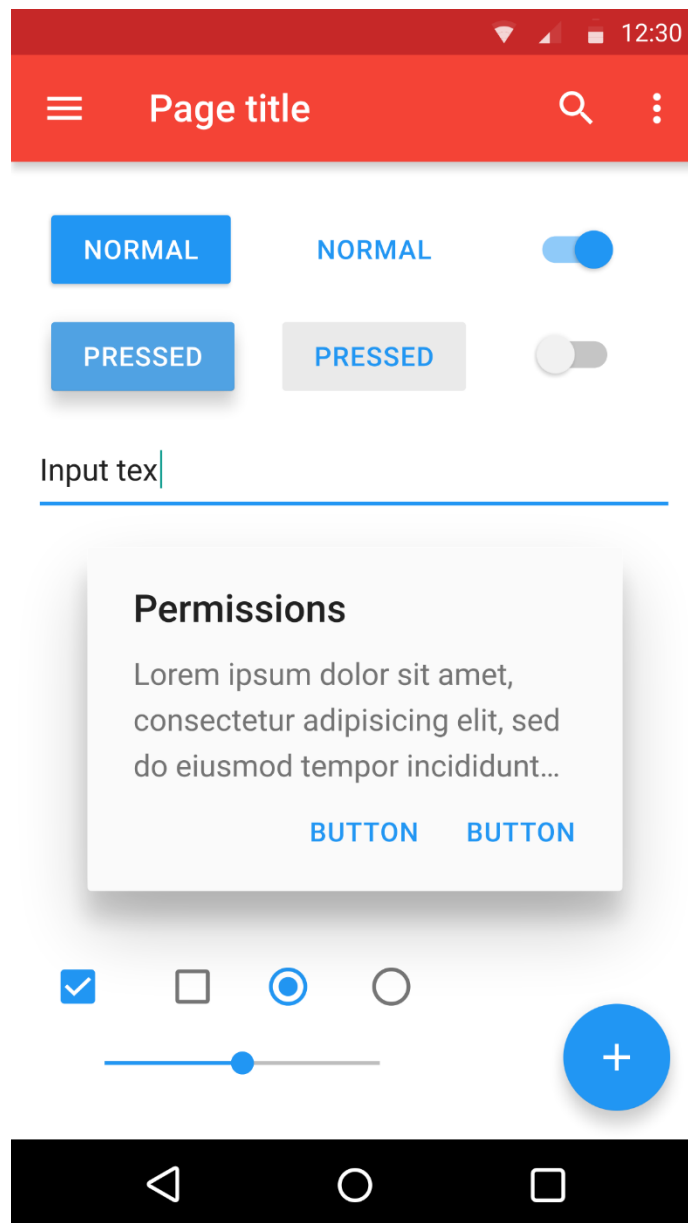


Рис. 1.2. Елементи інтерфейсу Material

РОЗДІЛ 2. ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ

2.1. Зовнішнє проектування

2.1.1. Опис функціональних характеристик

Мобільний додаток для статистичного аналізу та дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів призначений для надання користувачу повної інформації про обрану серверну технологію у вигляді аналітичних даних. Користувач може переглядати актуальну інформацію після виконання обраного запиту. Отримувати статистичні дані результату запиту у вигляді діаграм та графіків. Перевіряти швидкість, правильність та актуальність запитів після їх виконання. Переглядати всю необхідну інформацію завдяки зручному інтерфейсу.

Додаток повинен надавати наступні можливості:

- авторизація за допомогою логіну та паролю через виділений сервер;
- авторизація за допомогою профілю соціальних мереж через хмарні сервіси;
- реєстрація за допомогою електронної адреси та паролю через хмарні сервіси або виділений сервер;
- відновлення паролю через хмарні сервіси;
- перегляд інформації отриманої з серверу;
- налаштування критеріїв пошуку за типом бекенду;
- отримання великих об'ємів даних на з обраного серверу;
- створення користувачем списку даних на прикладі координат місцезнаходження та передача даних на сервер;
- відображення характеристик обраної серверної технології;
- запис отриманих статистичних даних у лог-файл;
- запис отриманих статистичних даних у базу даних, що знаходиться на пристрої користувача;

- відправка висновків отриманих за статистичними даними через соціальні мережі;
- формування висновків роботи з даними у обраних технологіях бекенду;
- аналіз переваг та недоліків обраної технології бекенду;
- перегляд графіків залежності часу від швидкості виконання запиту;
- перегляд статистичних діаграм після виконання запиту;
- перегляд профілю користувача;
- редагування детальної інформації профіля користувача.

2.1.2. Вхідні дані

Приведемо опис вхідних даних, джерелом яких є користувач. Вхідними даними будемо вважати будь-які дані користувача відносно системи, які він вводить власноруч або обирає з представлених варіантів.

Вхідні дані, які користувач вводить власноруч:

- логін, адреса електронної пошти, пароль, фото, ім'я, прізвище, день народження, місто (при реєстрації користувача у системі та редагуванні облікового запису);
- логін та пароль користувача, або ідентифікатор при авторизації за допомогою соціальної мережі (при авторизації користувача);
- ключове слово або назва серверу (при пошуку серверів);
- адреси електронної скриньки при відновленні паролю користувача;
- ім'я, адреса, номер телефону при реєстрації, редагуванні профілю у системі;
- логін при пошуку користувачів для додавання їх списку приятелів.

Вхідні дані, які користувач обирає з представлених варіантів:

- варіант отримання запиту від серверу;
- вибір серверу або серверної технології для передачі запитів;
- список з переліком інформації отриманої з обраного серверу;

- вибір діаграм, яку система змодельює з отриманих даних після запиту;
- вибір типів графіків та їх варіацій, що змодельює система після аналізу зіставлення обраних серверних технологій;
- вибір місця збереження лог-файлу, у системі, базі даних, чи відправлення через соціальну мережу;
- перелік облікових записів приятелів при виборі приятеля зі списку.

2.1.3. Вихідні дані

Приведемо опис вихідних даних для кожного вхідного повідомлення у табл. 2.1.

Таблиця 2.1.

Опис вихідних даних.

Назва вхідного повідомлення	Періодичність надходження	Вихідне повідомлення
1	2	3
Запит на реєстрацію користувача у системі	Після введення даних користувача, які необхідні для реєстрації	Результат реєстрації користувача у системі (повідомлення про успішну реєстрацію або помилку)
Запит на авторизацію користувача у системі	Після введення необхідних для авторизації даних користувача	Результат авторизації користувача (повідомлення про успішну авторизацію або помилку)
Запит на пошук серверу	Після завершення операції пошуку	Перелік серверів, які відповідають критеріям пошуку
Запит на відновлення паролю користувача	Після введення адреси електронної скриньки користувача	За умови що користувач з вказаною адресою електронної скриньки зареєстрований у системі – повідомлення про відправку листа з посиланням на сторінку для зміни пароля, інакше – повідомлення про помилку

Продовження таблиці 2.1

1	2	3
Запит на перегляд великих об'ємів даних отриманих з обраного серверу	Після вибору обраного серверу	Відображення тестових даних у вигляді таблиці та списку
Запит на редагування інформації користувача	Після введення даних користувача, які є обов'язковими для редагування його даних у системі	Результат редагування інформації користувача (повідомлення про успішну зміну інформації або помилку)
Запит на пошук користувача	Після ведення логіну користувача, якого поточний користувач намагається знайти	Результат пошуку користувача (профіль користувача або повідомлення, що такого користувача не існує)
Запит детальної інформації про серверну технологію	Після вибору певної серверної технології зі серверів у системі	Результат інформації про обраний сервер (назва, результат запиту короткий опис серверу)
Запит побудови графіку після виконання одного із запитів	Після вибору певної серверної технології зі серверів у системі та виконання обраного запиту	Результат побудови графіку залежності обраних параметрів
Запит на побудову діаграми залежності та відмінностей обраних технологій	Після вибору кількох технологій та виконання ними однакового запиту	Результат побудови діаграми залежності та відмінності обраних технологій
Запит на отримання висновків про обрану серверну технологію	Після вибору серверу та виконання обраного запиту	Результат інформації після аналізу системи у вигляді висновків
Запит списку доступних серверних технологій	Після вибору критерію фільтрації, а саме типу мережі	Результат завантаження списку серверів
Запит надсилання висновків після аналізу через соціальні мережі	Після вибору списку мереж	Результат надсилання приятелям висновків

Продовження таблиці 2.1

1	2	3
Запит детальної інформації про користувача	Після вибору певного свого профілю з меню	Результат завантаження інформації користувача (Ім'я, по батькові, логін, зображення профілю)

2.1.4. Формалізація задачі

Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми варіантів використання. Варіанти використання системи зображені на рис. 2.1.

Як можна побачити з діаграми, користувач при вході в систему повинен зареєструватися або авторизуватися, користувач може переглядати доступні технології організації бекенду (хмарну чи виділену), для кожної передбачене окреме меню з інформацією, здійснювати запити на пошук з тестової інформації, запит на отримання або відправку даних на обраний сервер, будувати графік залежності та інші варіанти, які наведено далі на діаграмі.



Рис. 2.1. Варіанти використання системи

2.2. Проектування бази даних

База даних знаходиться на виділеному сервері, а також копія знаходиться в меню розробника системи Firebase, та призначена для мобільного додатку, який допомагає користувачам у наданні інформації про обрану серверну технологію у вигляді аналітичних даних.

2.2.1. Специфікація вимог до бази даних

Головне завдання бази даних (БД) – гарантоване збереження значних обсягів інформації (так звані записи даних) та надання доступу до неї користувачеві або ж прикладній програмі.

База даних програми повинна зберігати тестову інформацію для роботи з сервером на прикладі даних про місця харчування, дані про користувача, список друзів користувача, список замовлень користувача. База даних буде зберігатись як на виділеному сервері, так і на хмарному, в однаковому вигляді. Заклади мають містити меню страв з описом та ціною. Користувач може створювати власні списки місць які він бажає відвідати або вже відвідував. Користувач має змогу додавати друзів до свого профілю та редагувати список друзів.

2.2.2. Визначення типів сутностей та зв'язків між ними

Визначення типів сутностей та зв'язків між ними відображає концептуальна модель БД. На цьому етапі визначаються об'єкти, зв'язки між об'єктами, сутності, атрибути, ключові атрибути.

При проектуванні бази даних «Заклади харчування» були виділені наступні сутності:

- сервер;
- користувач;
- замовлення;
- меню серверу;
- розділи;
- резервування;
- розташування;
- працівники;
- додаток до замовлення;
- друзі.

Далі приведено опис типів сутностей у табл. 2.2:

Таблиця 2.2.

Опис типів сутностей.

Ім'я типу сутності	Опис
1	2
Заклад	Містить дані про заклад харчування, а саме: назву, адресу та телефон та ін.
Користувач	Містить дані про користувача, а саме: ПІБ, телефон, список друзів
Замовлення	Містить дані про замовлення, а саме: назву страви, одиницю виміру, ціну, дату, та ін.
Меню закладу	Містить дані про меню страв, що пропонує певний заклад, а саме: назву страви, коротку інформацію про страву, одиницю виміру, ціну
Розділи	Містить дані про тип закладу харчування та тип кухні закладу
Резервування	Містить дані про резервування місць певного закладу,

	а саме: номер місця, дату, ім'я на яке було зарезервовано місце
Розташування	Містить дані про розташування закладу за координатами, а саме: ім'я закладу, координати x, координати y
Працівники	Містить дані про працівників конкретного закладу, а саме: ПІБ, телефон, заклад, в якому працює
Додаток до замовлення	Містить додаткові дані про страву в замовленні, а саме: страва, кількість порцій, замовлення
Друзі	Містить дані про друзів користувача

Для кожного типу сутностей визначимо основні характеристики, які будуть атрибутами сутностей. Перелік атрибутів і відомостей про них приведено в табл. 2.3:

Опис атрибутів сутностей.

Ім'я сутності	Атрибути	Опис атрибута	Тип	Обмеження
1	2	3	4	5
Заклад	* ЗакладПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Назва	Назва закладу	Рядок	До 60 символів
	Адреса	Адреса закладу	Рядок	До 300 символів
	Телефон	Телефон закладу	Рядок	До 25 символів
	РозділВК	Вказує на тип закладу	Ціле число	Зовнішній ключ
	РозташуванняВК	Вказує на розташування закладу	Ціле число	Зовнішній ключ
	Меню закладуВК	Вказує на меню обраного закладу	Ціле число	Зовнішній ключ
Користувач	* КористувачПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	ПІБ	Прізвище, ім'я, по-батькові користувача	Рядок	До 150 символів
	Телефон	Телефон користувача	Рядок	До 25 символів
	ДрузіВК	Вказує на друзів певного користувача	Ціле число	Зовнішній ключ

Продовження таблиці 2.3

1	2	3	4	5
Замовлення	* ЗамовленняПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Кількість товарів	Кількість товарів в замовленні	Ціле число	Більше за 0
	Дата	Дата замовлення	Дата та час	–
	Загальна Сума	Сума замовлення	Гроші	Більша за 0
	Меню закладуВК	Вказує на меню закладу в замовленні	Ціле число	Зовнішній ключ
	КористувачВК	Вказує на користувача який оплатив замовлення	Ціле число	Зовнішній ключ
Меню закладу	* Меню закладуПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Назва	Назва старви	Рядок	До 60 символів
	Одиниця вимірювання	Одиниця вимірювання	Рядок	До 10 символів
	Опис страви	Короткий опис страви	Рядок	До 60 символів
	Ціна	Ціна старви	Гроші	Більша за 0
Розділи	* РозділПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Назва категорії	Назва категорії	Рядок	До 60 символів
Резервування	* РезервуванняПК	Первинний ключ	Ціле число	Унікальне, не нульове значення

Продовження таблиці 2.3

1	2	3	4	5
	Дата	Дата резервування	Дата та час	–
	Номер резервування	Номер резервування	Ціле число	Більше за 0
	КористувачВК	Вказує на користувача, який робив резерв	Ціле число	Зовнішній ключ
	ЗакладВК	Вказує на заклад в якому зроблений резерв	Ціле число	Зовнішній ключ
Розташування	* РозташуванняПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Назва	Назва місця на карті	Рядок	До 60 символів
	Коор Х	Координати Х	Дійсне число	Більше за 0
	Коор У	Координати У	Дійсне число	Більше за 0
	ЗакладВК	Вказує на певний заклад	Ціле число	Зовнішній ключ
Додаток до замовлення	*ДодатокПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Кількість порцій	Кількість порцій у замовленні	Ціле число	Більше за 0, не порожній
	ЗамовленняВК	Вказує на замовлення	Ціле число	Зовнішній ключ
	Меню закладуВК	Вказує на меню закладу	Ціле число	Зовнішній ключ

Продовження таблиці 2.3

1	2	3	4	5
	Страва	Назва страви	Рядок	До 60 символів, не порожній
Працівники	*ПрацівникПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	ПІБ	Прізвище, ім'я, по-батькові працівника	Рядок	До 150 символів, не порожній
	Телефон	Телефон працівника	Рядок	До 25 символів
	Адреса	Адреса проживання	Рядок	До 300 символів
	ЗакладВК	Вказує на заклад, в якому працює	Ціле число	Зовнішній ключ
	Посада	Посада працівника	Рядок	До 60 символів
	Ставка	Зарплата працівника	Дійсне число	Більше за 0, не обов'язкове
Друзі	* ДрузіПК	Первинний ключ	Ціле число	Унікальне, не нульове значення
	Статус	Статус друга	Переліченн я	('0', '1', '2'), не порожній
	КористувачВК	Вказує на користувача	Ціле число	Зовнішній ключ

Приведемо детальний опис зв'язків між сутностями:

- а) зв'язок «Заклад – Розділ». Кожен заклад має декілька категорій розділів. Кожна категорія належить тільки одному закладу. Зв'язок 1 до N.

- б) зв'язок «Заклад – Розташування». Кожен заклад має місце розташування на карті, яке складається з координат осі. Зв'язок 1 до N.
- в) зв'язок «Користувач – Друзі». Кожен користувач має декілька друзів. Кожен друг може бути другом іншому користувачу. Зв'язок 1 до N.
- г) зв'язок «Замовлення – Меню». Кожне замовлення складається зі страми, яке знаходиться в меню. Зв'язок 1 до N.
- д) зв'язок «Замовлення – Користувач». Користувач може скласти одне замовлення. Зв'язок 1 до N.
- е) зв'язок «Заклад – Меню». Кожен заклад має власний список меню. Зв'язок 1 до N.
- ж) зв'язок «Резервування – Заклад». Кожне резервування робиться за певним закладом. Зв'язок 1 до N.
- з) зв'язок «Резервування – Користувач». Кожне резервування робиться за певним користувачем. Зв'язок 1 до N.
- и) зв'язок «Додаток до замовлення – Замовлення». Кожне замовлення має один додаток. Зв'язок 1 до N.
- к) зв'язок «Додаток до замовлення – Меню закладу». Додаток до замовлення складається з інформації про страви з сутності «Меню закладу». Зв'язок 1 до N.
- л) зв'язок «Заклад – Працівник». В кожному закладі працює кілька працівників. Окремий працівник працює лише в одному закладі. Зв'язок 1 до N.

Отже можна побачити, що всі зв'язки між сутностями мають відношення, що спростовує перетворення концептуальної схеми бази даних в логічну.

Дані про кожен зв'язок між сутностями занесемо в табл. 2.4 та зобразимо концептуальну схему.

Зв'язки між сутностями.

Ім'я сутності	Зв'язок	Ім'я сутності	Тип зв'язку
Заклад	має	Категорію	1 до N
Заклад	має	Розташування	1 до N
Користувач	має	Друзів	1 до N
Замовлення	складається	Страв з меню	1 до N
Замовлення	замовляє	Користувач	1 до N
Заклад	має	Меню страв	1 до N
Резервування	робиться за	Закладом	1 до N
Резервування	робить	Користувач	1 до N
Замовлення	має	Додаток до замовлення	1 до N
Меню закладу	має	Додаток до замовлення	1 до N
Заклад	утримує	Працівник	1 до N

Концептуальна схема моделі приведена на рисунку 2.2.

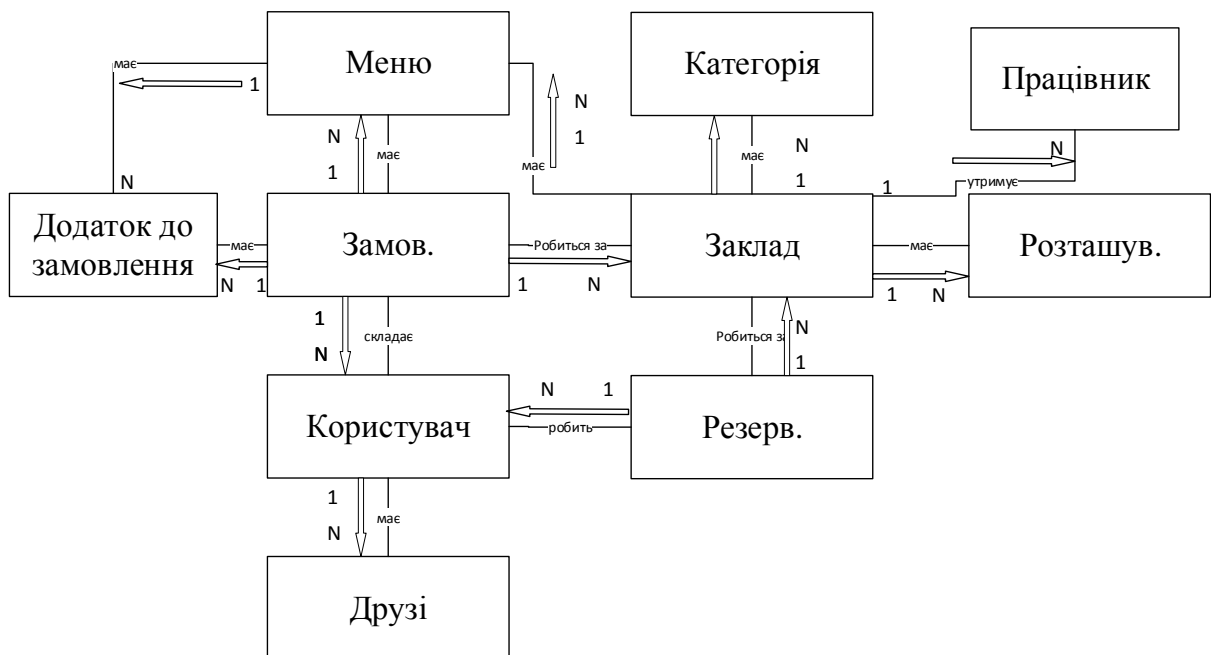


Рис. 2.2. Концептуальна схема моделі

2.2.3. Логічне проектування бази даних

Логічне проектування – перетворення концептуального представлення в логічну структуру бази даних, включаючи проектування зв'язків.

Дії, що виконуються на даному етапі проектування, залежать від обраної моделі даних. Для реляційної моделі перехід від концептуальної до логічної моделі складається з наступних дій:

- а) приведення ER-діаграми до виду, сумісного з реляційною моделлю: дія полягає у видаленні двосторонніх і рекурсивних зв'язків «багато до багатьох», видаленні складних зв'язків, видаленні багатозначних атрибутів. В зазначеній концептуальній моделі такі зв'язки відсутні;
- б) відображення концептуальної моделі на логічну, а саме визначення набору відносин (таблиць), ключових полів і зв'язків між таблицями: як правило, сутність стає таблицею, її первинний ключ стає первинним ключем таблиці; головна сутність передає підлеглий сутності копію свого первинного ключа для використання його в якості зовнішнього ключа;
- в) перевірки відносин за допомогою правил нормалізації на відповідність першим трьом формам нормалізації:
 - відношення знаходиться в 1НФ, якщо воно не містить повторюваних груп (кожен його елемент є атомарним);
 - відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і всі його не первинні атрибути повністю залежать від первинного ключа. (видалені часткові залежності);
 - відношення знаходиться в 3НФ, якщо воно знаходиться в 2НФ і кожен його не первинний атрибут нетранзитивно залежить від ключа (видалені транзитивні залежності);
- г) перевірки відповідності відносин вимогам користувацьких транзакцій;
- д) визначення вимог підтримки цілісності даних; обов'язковість даних, обмеження для доменів атрибутів, цілісність сутностей, посилальна цілісність, користувацькі обмеження.

Так як вибір первинного ключа сутності здійснюється з урахуванням сумарної довжини атрибутів, мінімальної кількості необхідних атрибутів в

ключі, а також наявності гарантій унікальності його значень в поточний момент часу і в осяжному майбутньому, то в якості первинних ключів для кожної сутності необхідно додати поле ID – унікальний ідентифікатор.

Результатам побудови логічної моделі буде логічна схема бази даних. Виконаємо розробку попередньої логічної схеми бази даних, на основі відображення концептуальної схеми моделі даних. Приведемо результати розробки на рис. 2.3. При логічному проектуванні бази даних «Заклади харчування» були виконані наступні дії:

- а) виключені особливості, які не сумісні з реляційною моделлю даних, а саме – зв'язки «Багато до багатьох».
- б) відображена концептуальна схема на логічній – визначено набори відношень згідно логічної моделі, які приведено у табл. 2.5.
- в) нормалізовано відношення БД до 3 нормальної форми.
- г) уточнено первинні і зовнішні ключі.
- д) визначено вимоги підтримки цілісності даних.
- е) побудована логічна схема моделі – визначені набори відношень згідно логічної моделі, які приведено у табл. 2.4. Приведено повний опис сутностей (табл. 2.5).

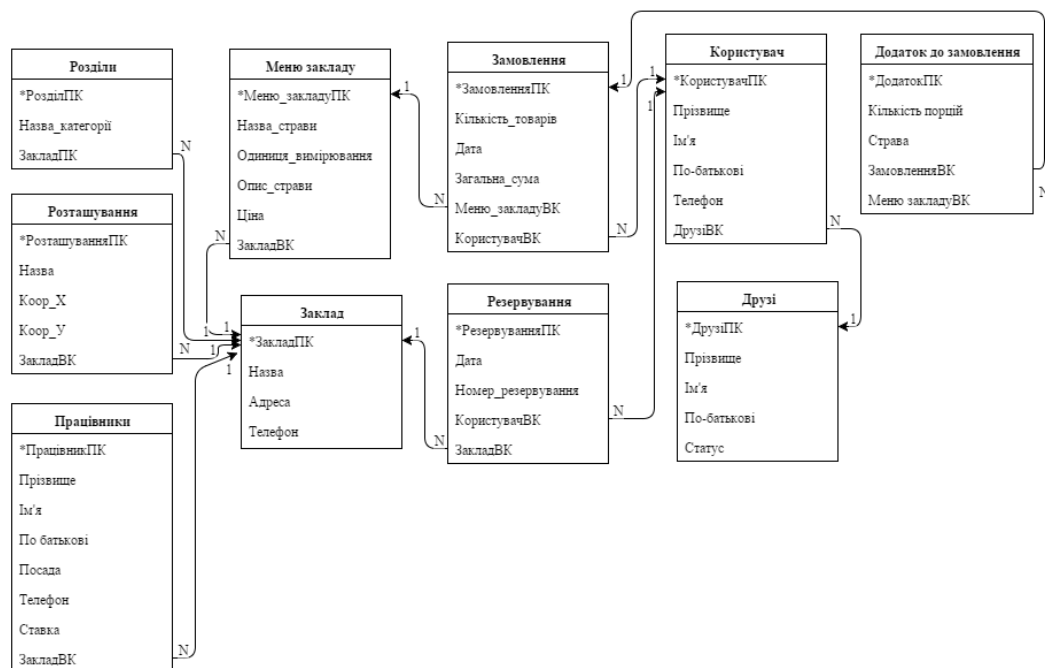


Рис. 2.3. Попередня логічна схема бази даних «Заклади харчування»

Таблиця 2.5.

Опис атрибутів сутностей.

Ім'я сутності	Атрибути	Опис атрибута	Тип	Обмеження
1	2	3	4	5
Заклад	* ЗакладПК	Первинний ключ	int	NOT NULL
	Назва	Назва закладу	nvarchar(60)	До 60 символів, NOT NULL
	Адреса	Адреса закладу	nvarchar(300)	До 300 символів
	Телефон	Телефон закладу	nvarchar(25)	До 25 символів
Користувач	* КористувачПК	Первинний ключ	int	NOT NULL
	Прізвище	Прізвище користувача	nvarchar(150)	До 150 символів, NOT NULL
	Ім'я	Ім'я користувача	nvarchar(60)	До 60 символів, NOT NULL
	По-батькові	По-батькові користувача	nvarchar(60)	До 60 символів
	Телефон	Телефон кор.	nvarchar(25)	До 25 симв.
	ДрузіВК	Зовнішній ключ	int	NOT NULL
Замовлення	* ЗамовленняПК	Первинний ключ	int	NOT NULL
	Кількість_товарів	Кільк.товарів в замовленні	int	Більше за 0
	Дата	Дата замовлення	datetime	–
	Загальна_Сума	Сума замовлення	money	Більша за 0

Продовження таблиці 2.5

1	2	3	4	5
	Меню_закладуВК	Зовнішній ключ	int	NOT NULL
	КористувачВК	Зовнішній ключ	int	NOT NULL
Меню закладу	*Меню_закладуПК	Первинний ключ	int	NOT NULL
	Назва_страви	Назва старви	nvarchar(60)	До 60 симв, NOT NULL
	Одиниця_вимірювання	Одиниця вимірювання старви	nvarchar(10)	До 10 символів
	Опис_страви	Короткий опис страви	nvarchar(60)	До 60 символів
	Ціна	Ціна старви	money	Більша за 0
	ЗакладВК	Зовнішній ключ	int	NOT NULL
Розділи	* РозділПК	Первинний ключ	int	NOT NULL
	Назва_категорії	Назва категорії	nvarchar(60)	До 60 симв.
	ЗакладВК	Зовнішній ключ	int	NOT NULL
Резервування	* РезервуванняПК	Первинний ключ	int	NOT NULL
	Дата	Дата резервування	datetime	–
	Номер_резервування	Номер резервування	int	NOT NULL
	КористувачВК	Зовнішній ключ	int	NOT NULL
	ЗакладВК	Зовнішній ключ	int	NOT NULL
Розташування	* РозташуванняПК	Первинний ключ	int	NOT NULL

1	2	3	4	5
	Назва	Назва місця на карті	nvarchar(60)	До 60 символів, NOT NULL
	Коор_Х	Координати Х	float	NOT NULL
	Коор_У	Координати У	float	NOT NULL
	ЗакладВК	Зовнішній ключ	int	NOT NULL
Додаток до замовлення	*ДодатокПК	Первинний ключ	int	NOT NULL
	Кількість порцій	Кількість порцій у замовленні	int	NOT NULL
	ЗамовленняВК	Зовнішній ключ	int	NOT NULL
	Меню_закладуВК	Зовнішній ключ	int	NOT NULL
	Страва	Назва страви	nvarchar(60)	До 60 симв. NOT NULL
Працівники	*ПрацівникПК	Первинний ключ	int	NOT NULL
	Прізвище	Прізвище працівника	nvarchar(150)	До 150 символів, NOT NULL
	Ім'я	Ім'я працівника	nvarchar(60)	До 60 символів, NOT NULL
	По-батькові	По-батькові працівника	nvarchar(60)	До 60 символів NOT NULL
	Телефон	Телефон працівника	nvarchar(60)	До 25 символів NOT NULL
	Адреса	Адреса проживання	nvarchar(60)	До 300 символів NOT NULL
	ЗакладВК	Зовнішній ключ	int	NOT NULL
	Посада	Посада працівника	nvarchar(60)	До 60 символів NOT NULL

1	2	3	4	5
	Ставка	Зарплата працівника	float	Більше за 0 NOT NULL
Друзі	* ДрузіПК	Первинний ключ	int	NOT NULL
	Статус	Статус друга	enum	('0','1','2') NOT NULL
	Прізвище	Прізвище друга	nvarchar(150)	До 150 символів, NOT NULL
	Ім'я	Ім'я друга	nvarchar(60)	До 60 символів, NOT NULL
	По-батькові	По-батькові друга	nvarchar(60)	До 60 символів

2.2.4. Схема бази даних

Схема бази даних – її структура, описана на формальній мові, що підтримується системою управління базами даних. В реляційних базах даних схема визначає таблиці, поля в кожній таблиці, а також відношення між полями і таблицями.

Схема бази даних була розроблена в середовищі MySQL, що являє собою компактний багатопотоковий сервер баз даних. Характеризується високою швидкістю, стійкістю і простотою використання.

Схема бази даних приведена на рис. 2.4, вона відображає таблиці і зв'язки між ними.

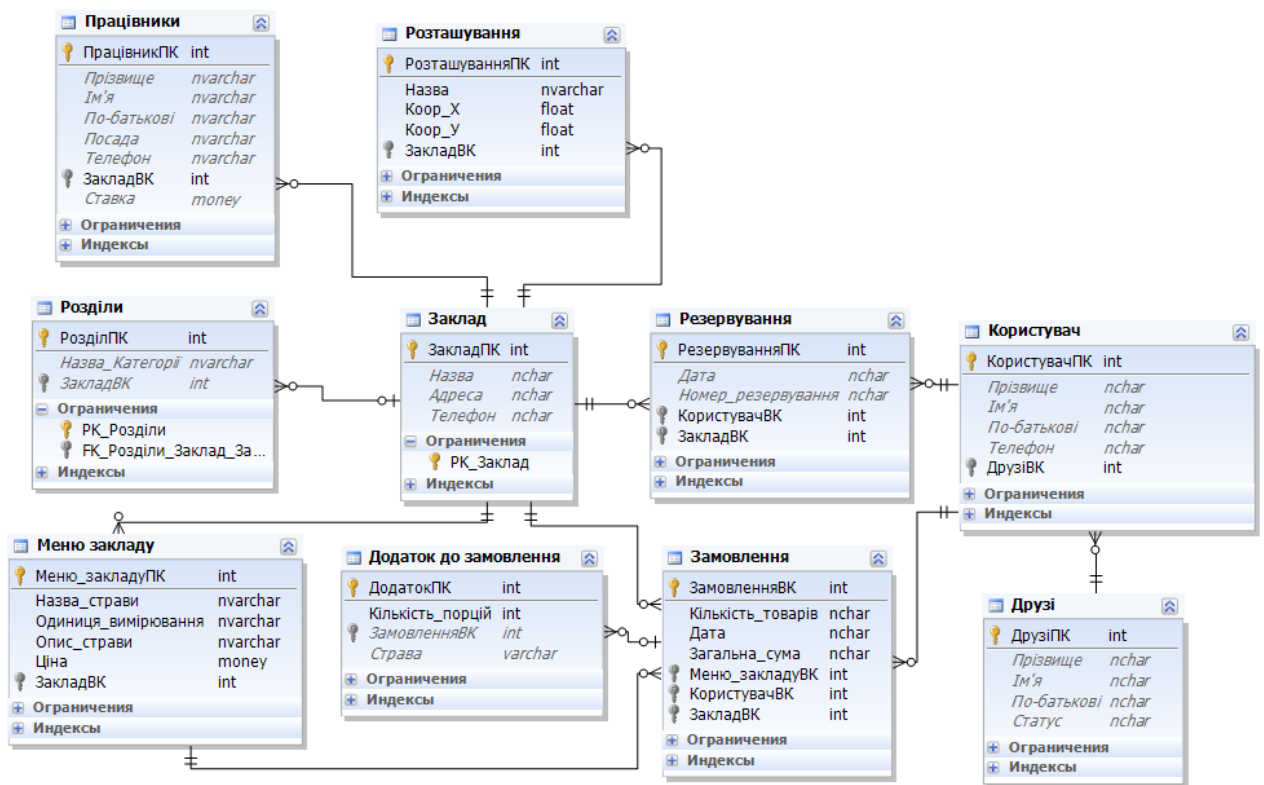


Рис. 2.4. Структура таблиць бази даних

РОЗДІЛ 3. РОЗРОБКА МОБІЛЬНОГО ДОДАТКА ДЛЯ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ХМАРНИХ ТА ВИДІЛЕНИХ СЕРВЕРІВ

3.1. Вибір мови програмування

Обираючи мову програмування необхідно проаналізувати задачі, що ставляться перед розробником та можливі проблеми, що можуть виникнути під час розробки.

Згідно з вимогами програма представляє собою мобільний додаток, так як операційна система Android на даний момент є найпопулярнішою у світі необхідно обрати мову програмування, що буде повністю підтримувати дану систему. Проаналізувавши зазначені особливості мовою програмування було обрано Java. Середовищем розробки під операційну систему Android є Android Studio 2.0. Середовище є безкоштовним та має широкі можливості для розробки, рефакторінгу, відлагодження та профілювання мобільних додатків.

Мобільний додаток підтримує з'єднання з сервером, на якому розміщена база даних з необхідною інформацією. Базу даних для серверу було спроектовано та розроблено в СКБД MS SQL Server.

3.1.1. Мова програмування Java

При розробці додатків Android використовується Java – одна з найбільш поширених мов програмування. Використання Java стало логічної вибором для платформи Android, тому що це потужна, вільна і відкрита мова, відома мільйонам розробників. Досвідчені програмісти Java можуть швидко освоїти Android-програмування, використовуючи інтерфейси Google Android API (Application Programming Interface) та інші розробки незалежних фірм. Мова Java є об'єктно-орієнтована, надає розробникам доступ до потужних бібліотек класів, що прискорюють розробку програм [10]. Програмування графічного інтерфейсу користувача є керованою подією. Мова програмування Java має наступні характеристики:

- простота мови входить в ключові характеристики Java: розробник не повинен тривалий час вивчати мову, перш ніж він зможе на ній програмувати. Фундаментальні концепції мови Java швидко схоплюються, і програмісти з самого початку можуть вести продуктивну роботу. Розробниками Java було прийнято до уваги, що багато програмістів добре знайомі з мовою C++, тому Java, наскільки це можливо, наближений до C++;
- об'єктно-орієнтованість. Мова Java із самого початку проектувалась як об'єктно-орієнтована. Завданням розподілення систем клієнт-сервер відповідає об'єктно-орієнтована парадигма: використання концепцій інкапсуляції, успадкування та поліморфізму. Java надає ясну і дієву об'єктно-орієнтовану платформу розробки. Програмісти на Java можуть використовувати стандартні бібліотеки об'єктів, що забезпечують роботу з пристроями введення/виводу, мережеві функції, методи створення графічних користувацьких інтерфейсів;
- надійність. Платформа Java розроблена для створення високонадійного прикладного програмного забезпечення. Велику увагу приділено перевірці програм на етапі компіляції, за якою слідує другий рівень – динамічна перевірка (на етапі виконання);
- незалежність від архітектури. Java розроблена для підтримки додатків, впроваджуваних в гетерогенні мережеві середовища. У подібних середовищах додатки повинні виконуватися на різних апаратних архітектурах, під керуванням різних операційних систем і у взаємодії з інтерфейсами різних мов програмування. Для забезпечення незалежності від платформи програм компілятор Java генерує байт-код;
- архітектурно-нейтральний проміжний формат програми, створюваний для ефективною передачі коду на різні апаратні і програмні платформи. При виконанні програми байт-код

інтерпретується виконуючою машиною Java. Один і той же Java-байткод буде виконуватися на будь-якій платформі;

- багатопоточність. Більшості сучасних мережових додатків зазвичай необхідно здійснювати кілька дій одночасно. У Java реалізований механізм підтримки легковагих процесів-потоків (ниток). Нить Java надає засоби створення додатків з безліччю одночасно активних потоків;
- інтерпретованість. Java-інтерпретатор може виконувати Java байткод на будь-якій машині, на якій встановлено інтерпретатор та система виконання. На інтерпретованій платформі фаза збірки програми є простою і покроковою, тому процес розробки істотно прискорюється і прощується, відсутні традиційні важкі етапи компіляції, збірки, тестування;
- висока продуктивність. Продуктивність завжди заслуговує особливої уваги. Java досягає високої продуктивності завдяки спеціально оптимізованому байт-коду, легко перекладному в машинний код. Автоматична збірка сміття виконується як фоновий потік з низьким пріоритетом, забезпечуючи високу ймовірність доступності необхідної пам'яті, що веде до збільшення продуктивності. Додатки, що вимагають великих обчислювальних ресурсів, можуть бути спроектовані так, щоб ті частини, які вимагають інтенсивних обчислень, були написані мовою асемблера і взаємодіяли з Java платформою.

Сфера застосування Java являє собою наступну картину:

- створення десктопів і аплетів. Даний продукт створюється на замовлення;
- створення мобільних додатків. Програмування для мобільних пристроїв на мові Java;
- різні серверні додатки, які в основному орієнтовані на роботу з мережею.

Крім безпосереднього написання коду додатків, можна скористатися середовищами розробки Eclipse і Android Studio, що дозволяють збирати графічний інтерфейс з готових об'єктів, таких як кнопки та текстові поля, перетягуючи їх в певні місця екрану, додаючи підписи і змінюючи їх розміри. Ці середовища розробки дозволяють швидко і зручно створювати, тестувати і налагоджувати програми при розробці на Android.

3.1.2. Опис середовища розробки Android Studio

При розробці даного програмного проекту було обрано середовище розробки Android Studio. Нове середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки додатків для платформи Android. У тому числі в середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування додатків, що працюють на пристроях з різними дозволами екрану (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема збірки, тестування і розгортання додатків, заснована на складальному інструментарії Gradle і підтримуюча використання коштів безперервної інтеграції.

Для прискорення розробки додатків представлена колекція типових елементів інтерфейсу і візуальний редактор для їх компонування, що надає зручний передперегляд різних станів інтерфейсу додатку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середу вбудовані функції завантаження типових прикладів коду з GitHub. До складу також включені враховують особливості платформи Android розширені інструменти рефакторінга, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю,

моніторингу споживання пам'яті та оцінки зручності використання. У редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови.

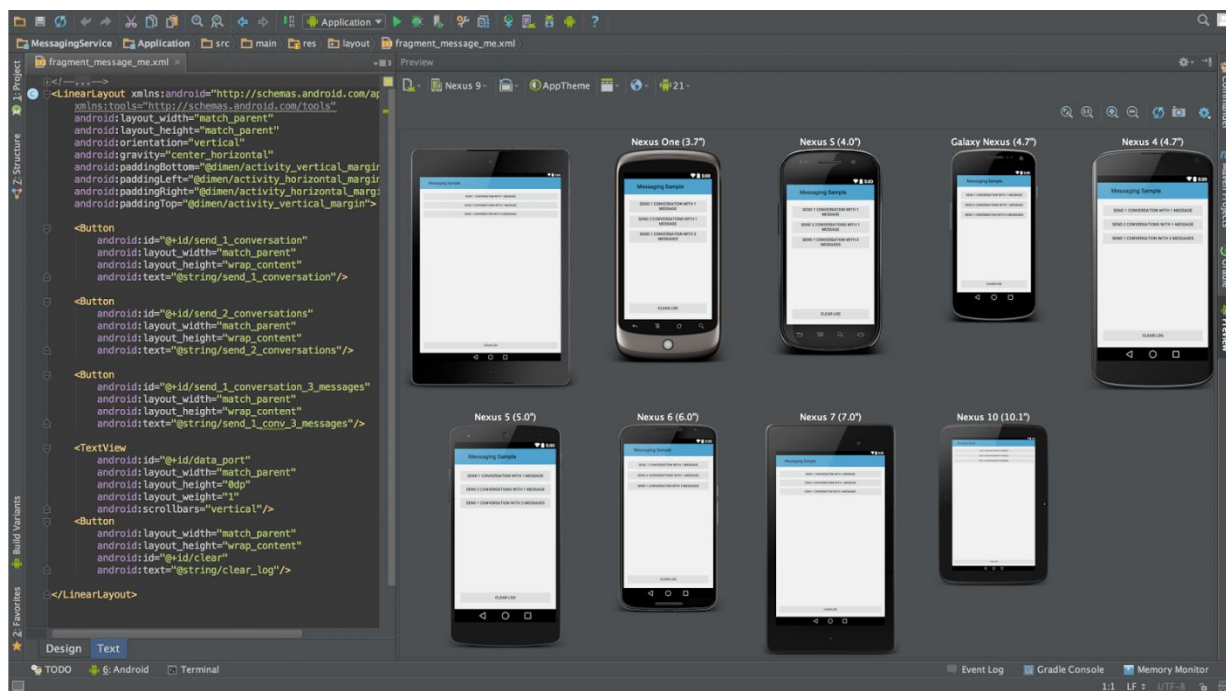


Рис. 3.1. Середовище розробки Android Studio

3.1.3. Опис СКБД MS SQL Server

Microsoft SQL Server – одна з найбільш потужних СКБД архітектури клієнт-сервер. Ця СКБД дозволяє задовольняти таким вимогам, що застосовуються до систем розподіленої обробки даних, як тиражування даних, паралельна обробка, підтримка великих баз даних на відносно недорогих апаратних платформах при збереженні простоти управління і використання. MS SQL Server не призначений безпосередньо для розробки користувальницьких додатків, а виконує функції керування базою даних. Сервер має засоби віддаленого адміністрування і керування операціями, організовані на базі об'єктно-орієнтованої розподіленого середовища управління.

Під час розробки використовувалась версія Microsoft SQL Server 2012, яка є новою і потужною системою управління базами даних. Крім стандартних для СУБД функцій, SQL Server 2012 містить великий набір інтегрованих служб з аналізу даних. Доступ до даних, розташованих на SQL Server можуть отримати будь-які додатки, розроблені на .Net, VisualStudio, а також додатки пакета Microsoft Office 2007.

SQL Server 2012 забезпечує найвищу у своєму класі масштабованість, продуктивність і безпеку.

3.2. Об'єктно-орієнтоване проектування

Для розробки програмного продукту був обраний метод об'єктно-орієнтованого проектування.

Підставами до вибору даної парадигми є:

- 1) швидкість модифікації та розширення програмного коду;
- 2) наявність якісного середовища проектування;
- 3) широкі можливості з повторного використання коду;
- 4) зручність при розробці архітектури системи;
- 5) наявність загальноприйнятих стратегій щодо вирішення типових проблем.

Для створення об'єктно-орієнтованої моделі системи використовуються діаграми класів UML.

Під час об'єктно-орієнтованого проектування для вирішення деяких типових проблем були використані шаблони проектування.

Патерн, який породжує об'єкти – одинак (Singleton). Призначення – гарантує, що у класі є тільки один екземпляр, і надає до нього глобальну точку доступу. Для деяких класів важливо, щоб існував тільки один екземпляр. Наприклад, важливо, щоб після авторизації, в системі був тільки один поточний користувач (тільки один екземпляр класу «Користувач» з даними про поточного користувача, рис. 3.2.

Реалізація: зазвичай такий патерн реалізують за допомогою приватного конструктора, статичного поля, яке буде містити єдиний екземпляр класу та статичного методу, який буде повертати це статичне поле.

Результат: після використання такого патерну можна з впевненістю гарантувати, що в кожен момент часу в системі існує максимум один екземпляр класу ParseUser.

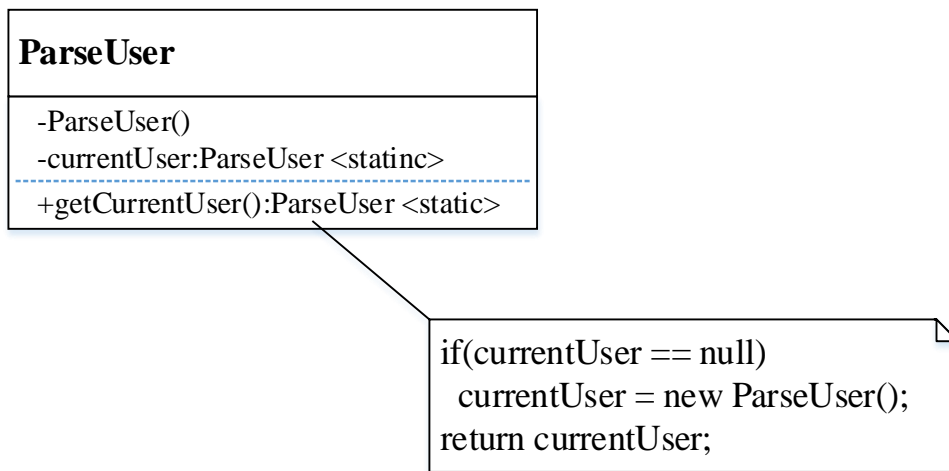


Рис. 3.2. Клас ParseUser

Патерн, який породжує об'єкти – будівельник (Builder).

Призначення: відокремлює конструювання складного об'єкта від його подання, так що в результаті одного і того ж процесу конструювання можуть виходити різні представлення. Використання цього патерну дозволяє приховати, спростити процес конструювання складних об'єктів. Наприклад, в системі необхідно здійснювати складний аналіз сторінок. Для здійснення такого аналізу іноді необхідно задати досить велику кількість параметрів. Для спрощення таких операцій використовувався патерн Builder. Діаграма класів з прикладом використання такого патерну приведена на рис. 3.3.

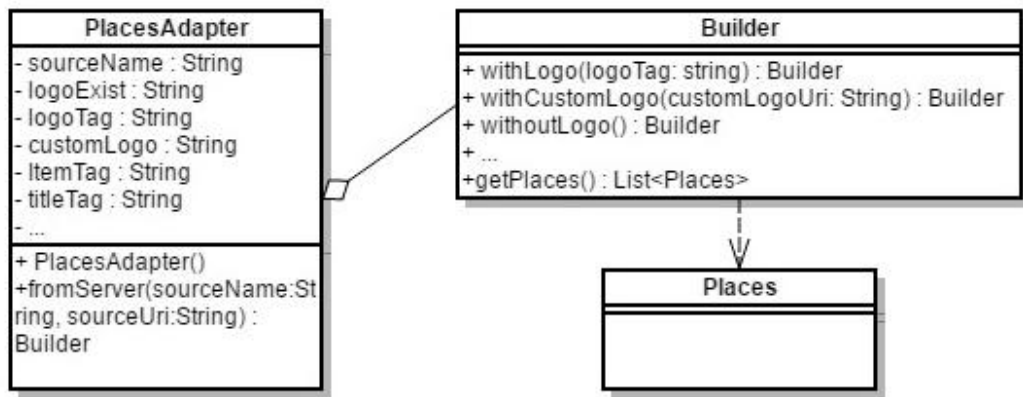


Рис. 3.3. Клас PlacesAdapter та inner-клас Builder

Приклад використання такого підходу:

```

...
new PlacesAdapter("Casta")
    .builder()
    .customLogo("https://ps.twimg.com/profile_images/Casta_logo_400x400.png"
)
    .withDescription("content:encoded")
    .withLogo()
    .getPlaces()
  
```

Патерн поведінки об'єктів – спостерігач (Observer).

Призначення: визначає залежність типу «один до багатьох» між об'єктами таким чином, що при зміні стану одного об'єкта всіх залежних від нього сповіщуються про це і автоматично оновлюються. У результаті розбиття системи на безліч спільно працюючих класів з'являється необхідність підтримувати узгоджений стан взаємопов'язаних об'єктів. Але не хотілося б, щоб за узгодженість треба було платити жорсткою пов'язаністю класів, так як це в деякій мірі зменшує можливості повторного використання.

Необхідно реалізувати клас компоненту FloatingActionButton (кнопка), яка буде автоматично змінювати свої параметри видимості при взаємодії користувача з компонентом ScrollView, тобто при перелистуванні тексту на компоненті кнопка буде з'являтися/зникати рис. 3.4.

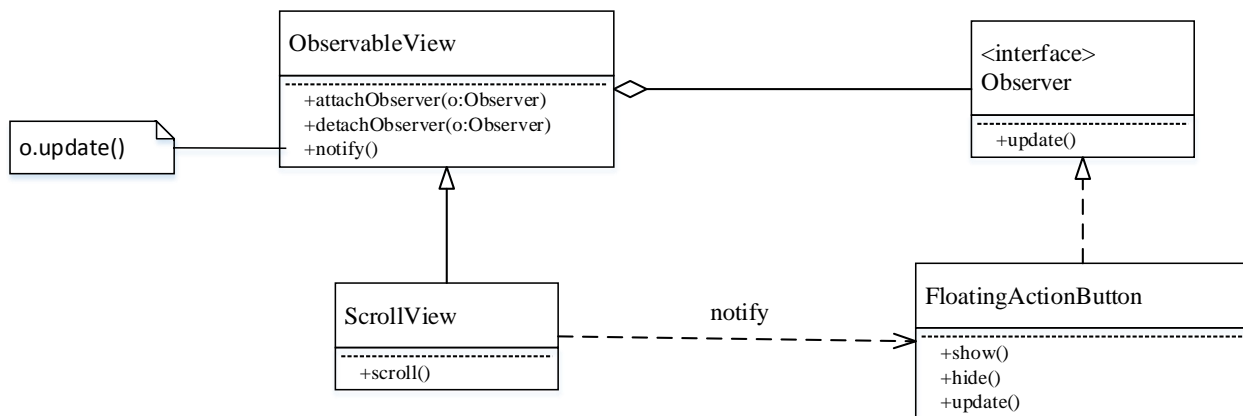


Рис. 3.4. Реалізація патерну Observer

Таким чином, кожного разу, коли користувач буде взаємодіяти з `ScrollView`, цей клас буде сповіщати об'єкт класу `FloatingActionButton` про зміни, викликаючи метод `update()`.

Поведінковий патерн – стан (State). Призначення: дозволяє об'єкту змінювати свою поведінку в залежності від внутрішнього стану, рис 3.5.

Використання: слід використовувати шаблон стан у випадках, коли: поведінка об'єкта залежить від його стану та повинна змінюватись під час виконання програми; у коді операцій зустрічаються складні умовні оператори, у котрих вибір гілки залежить від стану. Зазвичай у такому разі стан представлено константами, що перелічуються. До того ж часто одна й та ж структура умовного оператора повторюється у декількох операціях. Шаблон стан пропонує замінити кожен гілку окремим класом. Це дозволить трактувати стан об'єкта як самостійний об'єкт, котрий може змінитися незалежно від інших.

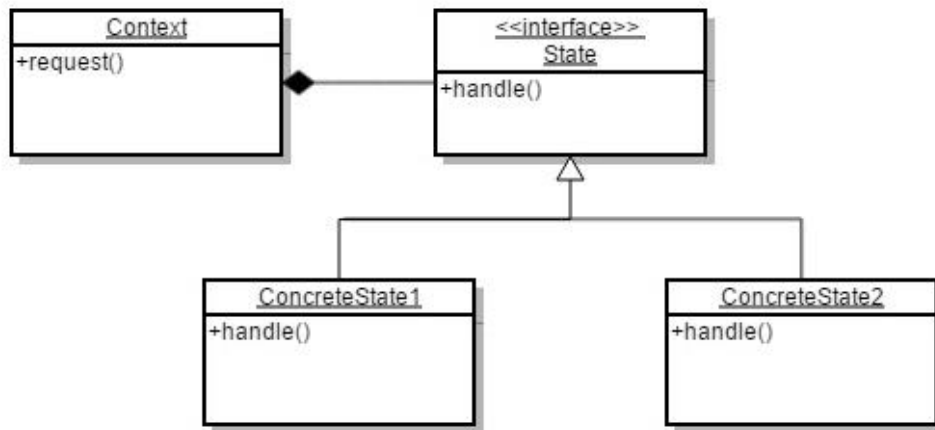


Рис. 3.5. Реалізація патерну State

Приклад використання такого підходу:

```

@Override
public Fragment getItem(int position) {
    return mFragments.get(position);
}
@Override
public int getCount() {
    return mFragments.size();
}
@Override
public CharSequence getPageTitle(int position) {
    return mFragmentTitles.get(position);
}
  
```

Діаграми UML проекту представлені на рис. 3.6 – 3.10.

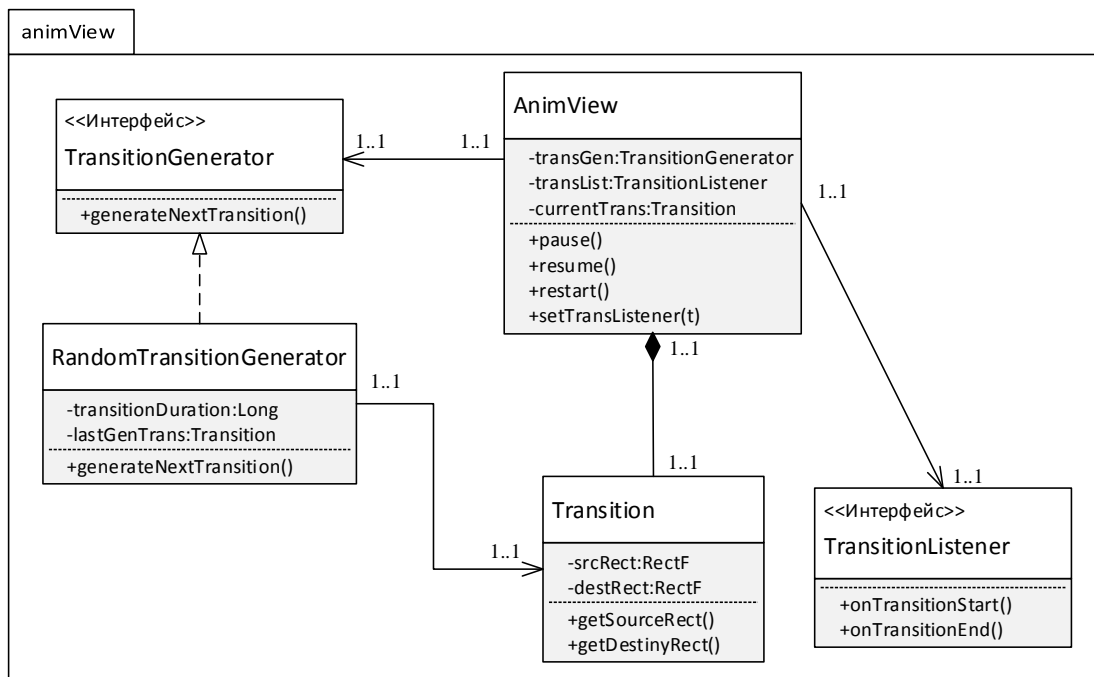


Рис. 3.6. Діаграма взаємодії та ієрархії класів пакету AnimView

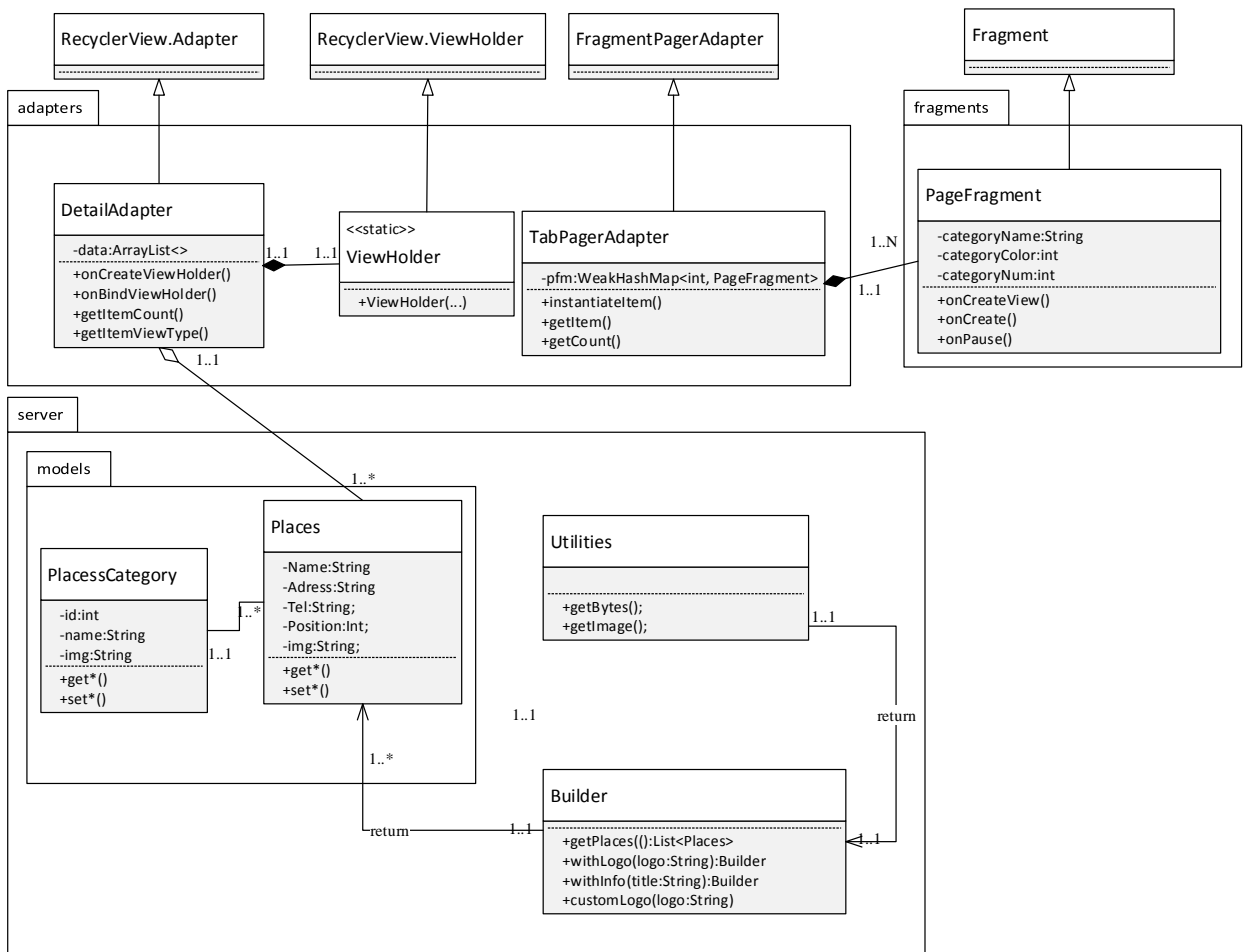


Рис. 3.9. Діаграма взаємодії та ієрархії класів пакетів Adapters та Server

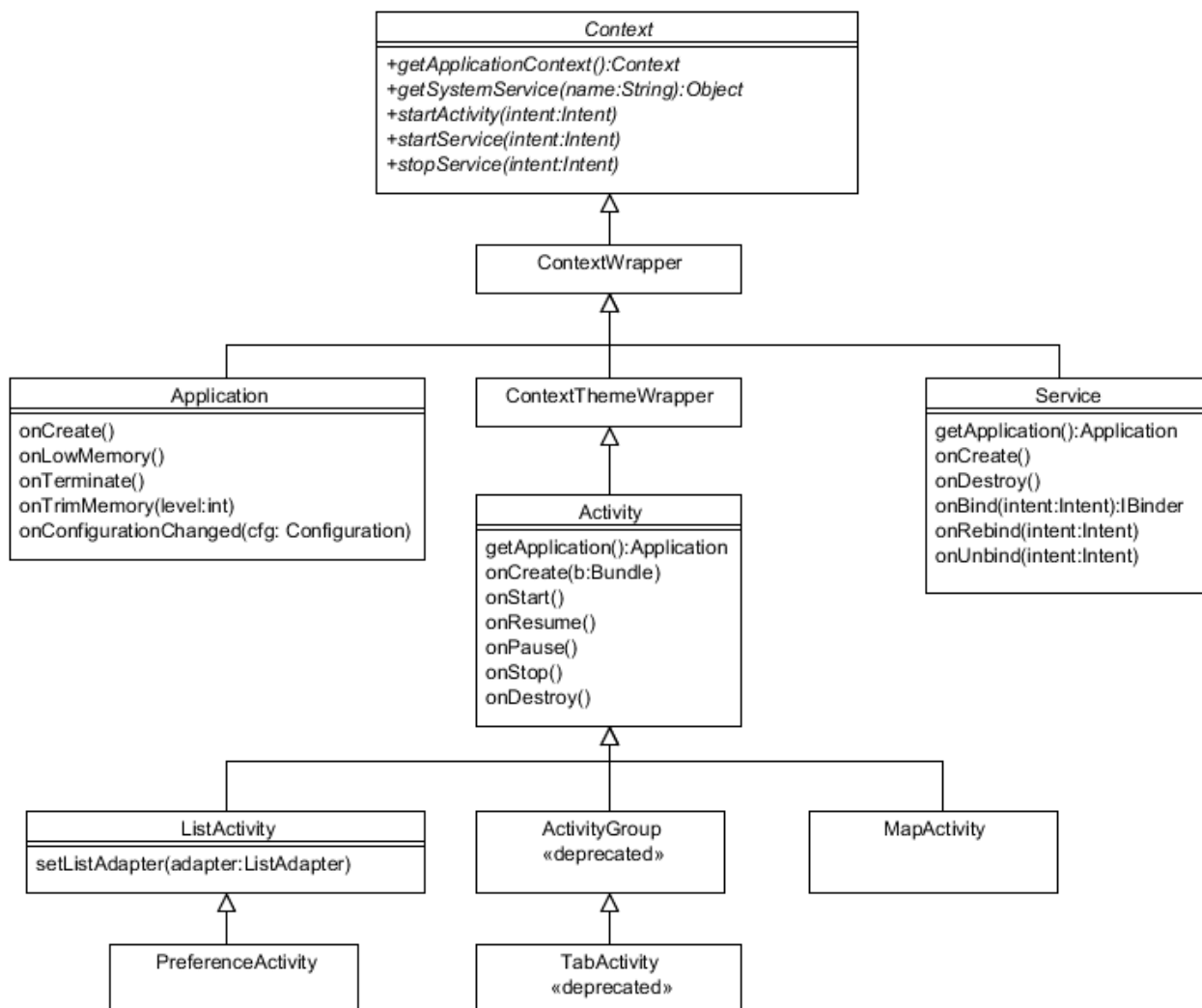


Рис. 3.10. Діаграма взаємодії та ієрархії класів пакету ContentTabs

3.3. Розробка інтерфейсу

В об'єктно-орієнтованих мовах, програмний інтерфейс зазвичай включає в себе опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це абстрактне поняття пов'язане з реальними функціями, які надані або надаватимуться, класами, які реалізуються в методах класу. Прикладний програмний інтерфейс в даному випадку можна розглядати як сукупність всіх методів, які публічно доступні в класах (зазвичай званий інтерфейс класу). Це означає, що прикладний програмний інтерфейс вказує методи, за допомогою яких взаємодіє з об'єктами, отриманими з визначень класів і обробляє їх.

У більш загальному плані можна визначити програмний інтерфейс як сукупність усіх видів об'єктів, які можна вивести з визначення класу, і пов'язаних з ними можливих варіантів поведінки.

Через те, що мова програмування Java працює на віртуальній машині, вона може ділитися програмними інтерфейсами. У цьому випадку віртуальна машина дозволяє мові взаємодії завдяки спільному знаменнику віртуальної машини, що абстрагується від конкретної мови, використовувати проміжний байт-код і його мову.

Графічний інтерфейс в операційній системі Android представляє собою ієрархією об'єктів `android.view.View` і `android.view.ViewGroup`. Кожен об'єкт `ViewGroup` представляє контейнер, який містить і впорядковує дочірні об'єкти `View`. Об'єкти `View` є елементами управління та інші віджети, наприклад, кнопки, текстові поля і т.д., через які користувач взаємодіє з програмою, рис. 3.11.

Основу обміну інформацією користувача із системою складають діалоги.

Інтерфейс програми має бути зручним та інтуїтивно зрозумілим, не викликати роздратованості. Ніякі дії користувача не повинні призвести до помилок в системі, у разі виникнення непередбачуваних станів, користувач повинен отримати вичерпне повідомлення про помилку, можливі причині її виникнення та рекомендовані подальші дії для її усунення.

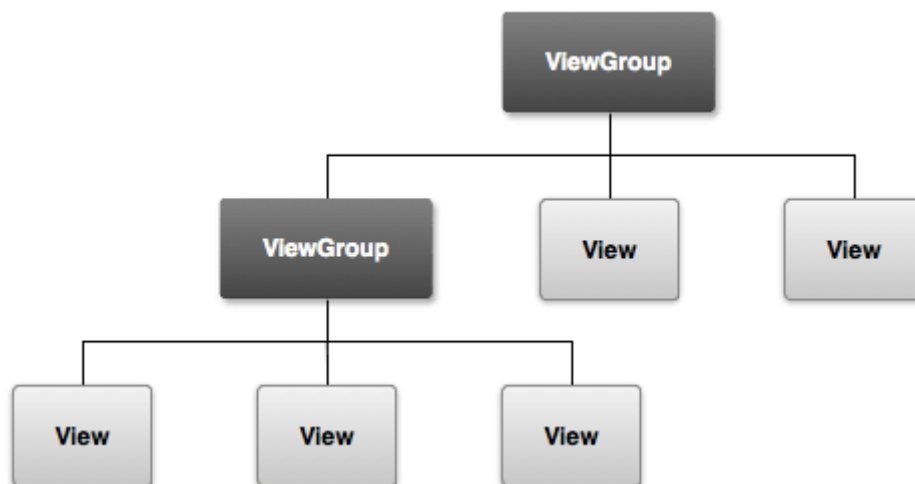


Рис. 3.11. Ієрархія об'єктів `android.view.View`

Для того, щоб мати уявлення про інтерфейс користувача на етапі проектування були розроблені макети інтерфейсу, за допомогою дизайнерської платформи «UxPin» які наведені на рис. 3.12.

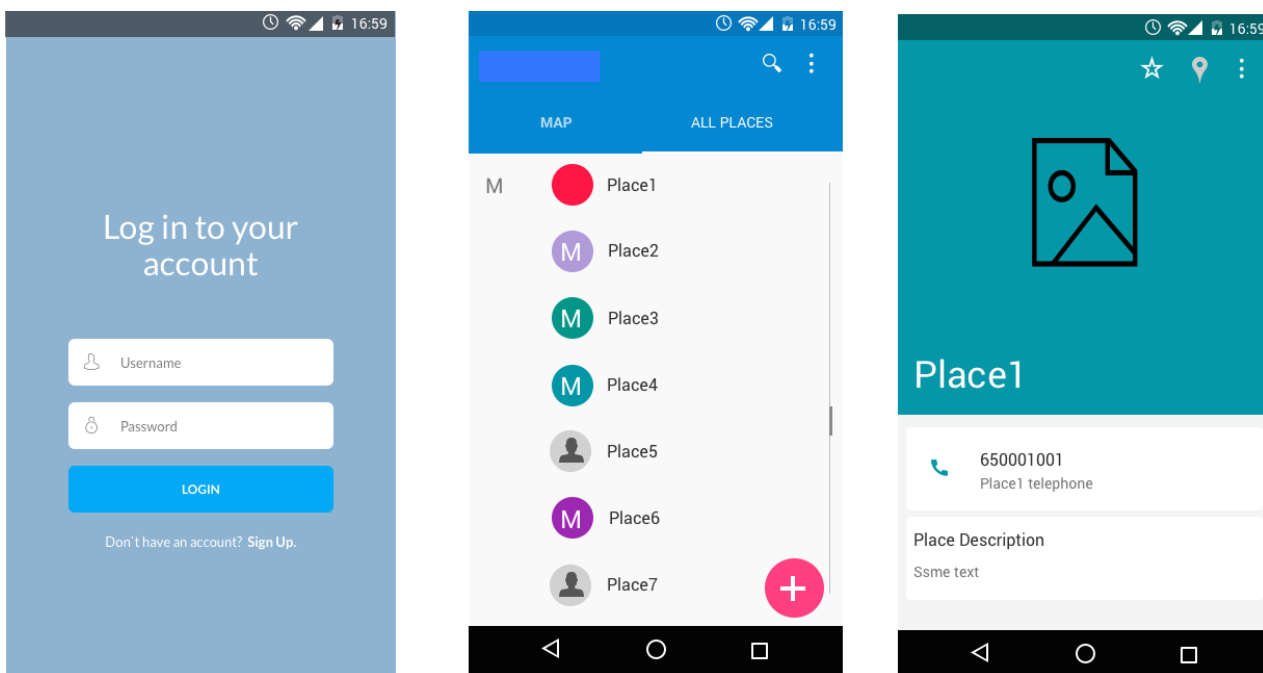


Рис. 3.12. Макети інтерфейсу

На рис. 3.12 схематично наведено три вікна додатка. На першому макеті наведено вікно входу у систему, можна побачити текстове поле TextView з підказкою для входу, два поля для введення інформації (логіна та пароля) EditText, та дві кнопки Button (для входу та створення акаунту). На наступному макеті зображено дві вкладинки TabView, на першій повинна бути мапа даних, які отримані після виконання всіх запитів у системі, на іншій наведено перелік серверів за допомогою компоненту ListView. Також внизу розміщено кнопку Button для додання серверу до списку обраного. На останньому макеті наведено детальний опис обраного серверу, який складає: зображення марки серверу ImageView, назва серверу, та коротка інформація. Перелік використаних компонентів з їх описом та властивостями наведено у табл. 3.1.

Таблиця 3.1.

Перелік використаних компонентів.

Назва компоненту	Опис	Використані властивості
1	2	3
LinearLayout	LinearLayout представляє об'єкт ViewGroup, який впорядковує всі дочірні елементи в одному напрямку: по горизонталі або по вертикалі.	layout_width="match_parent" layout_height="match_parent" orientation="horizontal"
ImageView	Призначений для перегляду зображення.	layout_width="wrap_content" layout_height="wrap_content" src="@drawable/my_comps"
Button	Кнопка, відповідає на натиснення, або іншу подію.	text="@string/button_text" onClick="sendMessage"
ScrollView	Макет контейнера для перегляду ієрархії, можна прокручувати користувачем, що дозволяє йому бути більше, ніж фізичною частиною дисплея.	layout_width="match_parent" layout_height="match_parent"
EditText	Текстове поле з можливістю введення інформації.	layout_weight="1" layout_width="0dp" layout_height="wrap_content" hint="@string/edit_message"
RelativeLayout	Представляє об'єкт ViewGroup, який має в своєму розпорядженні дочірні елементи відповідно до їх координат.	paddingLeft="16dp" paddingRight="16dp"
TextView	Відображає текст без можливості редагування.	text="@string/display_message" layout_centerHorizontal="true" textSize="16sp"
ListView	Представляє собою класичний список.	layout_width="match_parent" layout_height="wrap_content"

Приведемо екрани програми, які ілюструють інтерфейс системи при вході або реєстрації користувача на рис. 3.13 – 3.14.

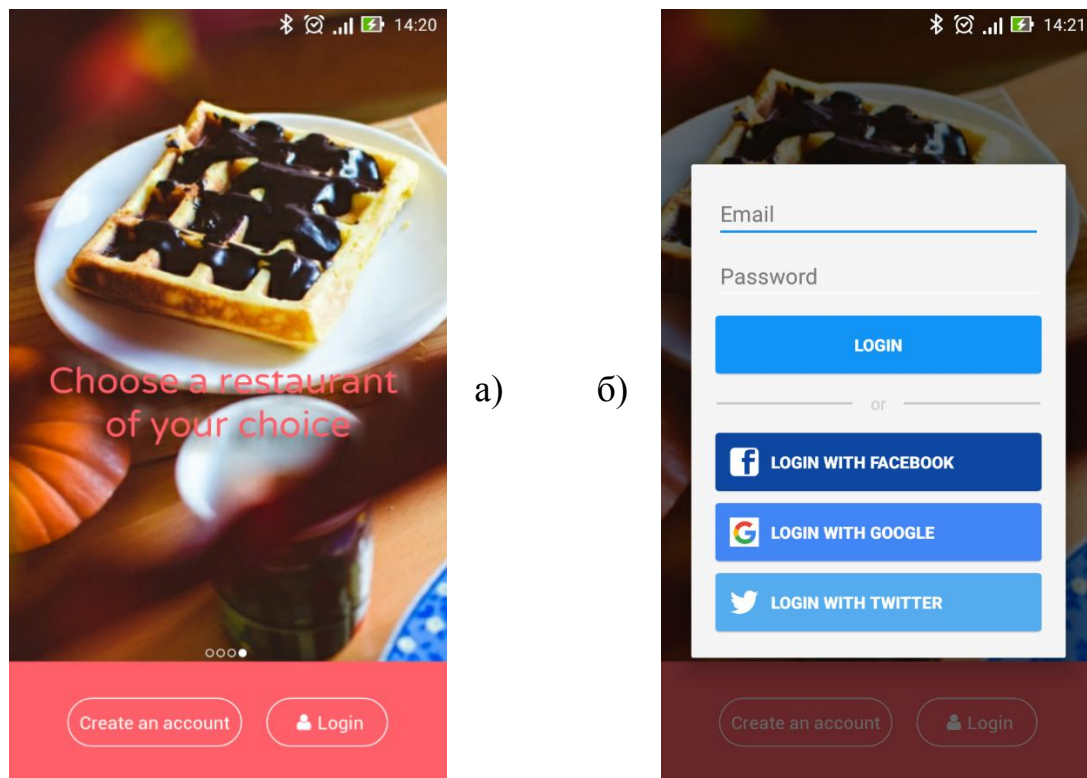


Рис. 3.13. Вхід в систему: (а) – головне вікно входу; (б) – вікно авторизації

На рис. 3.13а наведено вікно входу до програми, користувач може обрати створення нового облікового запису, або увійти до існуючого, на задньому плані можна побачити слайдер з коротким описом додатку. При перелистуванні зображення та написи опису змінюються. Якщо користувач обрав вхід до існуючого облікового запису (рис. 3.13б) він може обрати декілька типів авторизації. Авторизація за допомогою акаунту Google наведена на рис. 3.14.

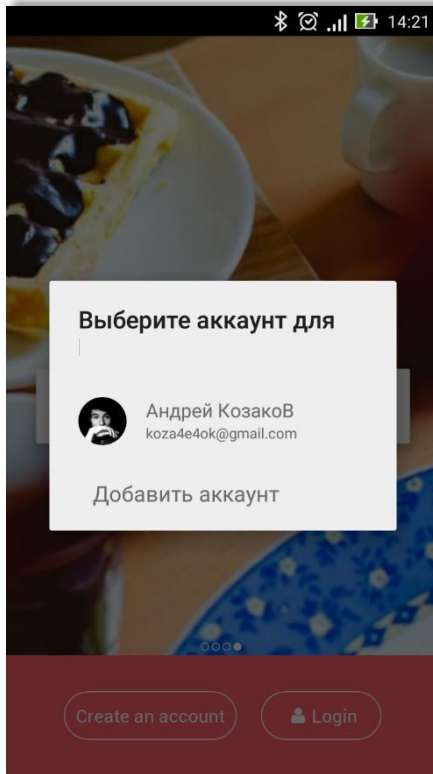


Рис. 3.14. Авторизация за допомогою акаунту Google

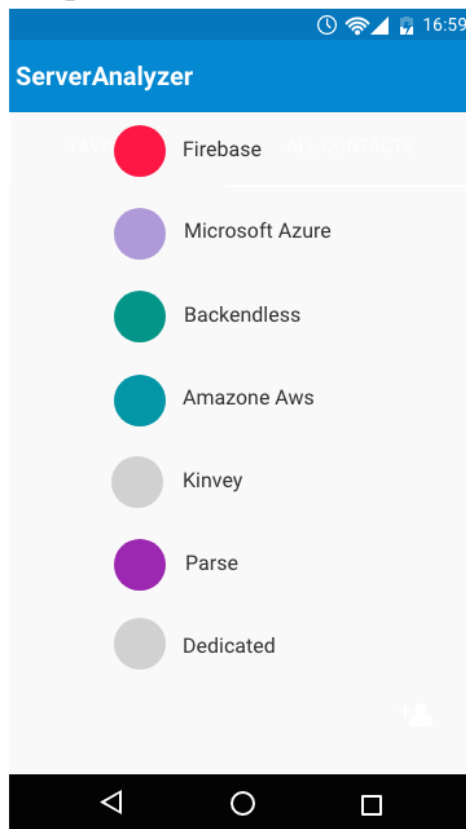


Рис. 3.15. Головне вікно програми

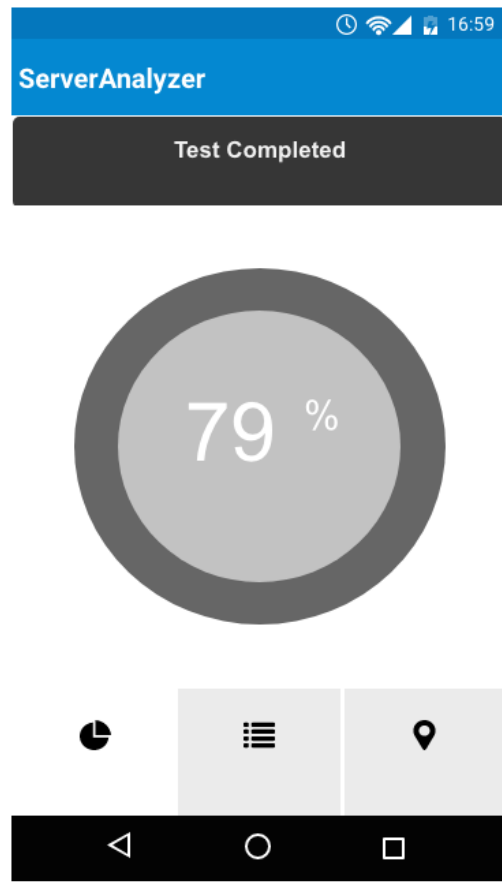
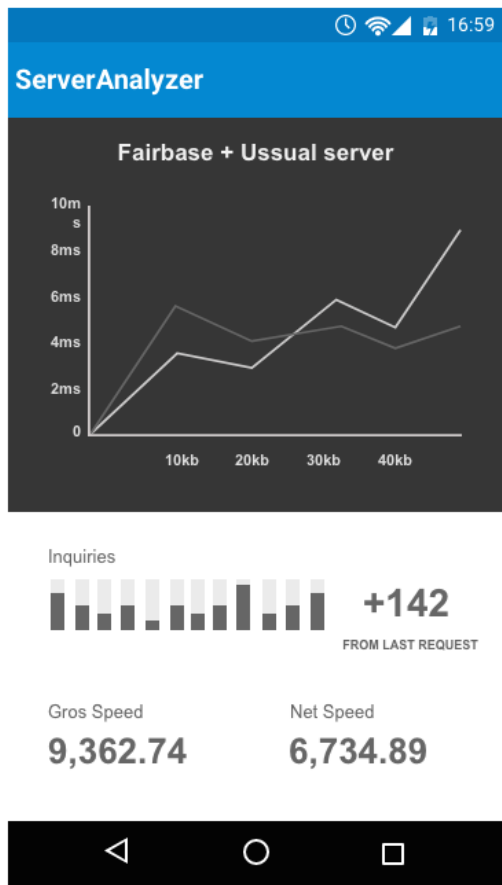


Рис. 3.16. Вкладки додатку: (а) – вкладка з інформацією про технологію; (б) – вкладка із тестом

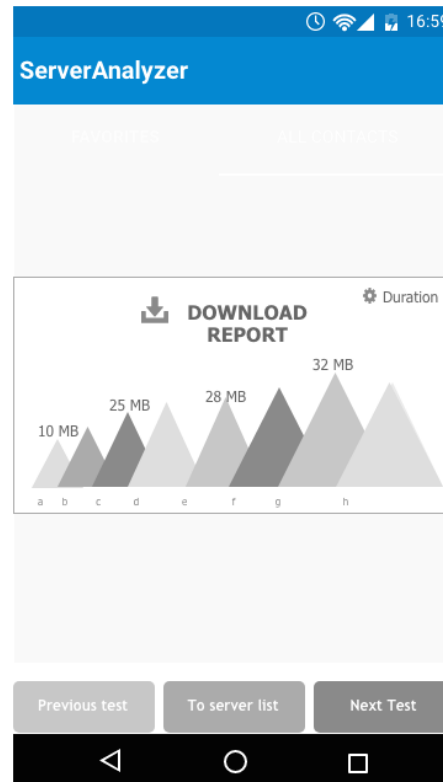
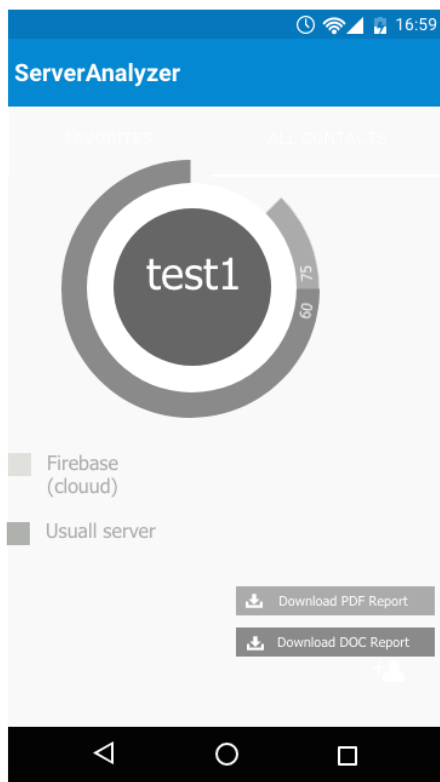


Рис. 3.17. Вкладка з виконаним тестом та діаграмою

Коли користувач увійшов до облікового запису, перед ним з'являється головна вкладинка, рис. 3.15, на якій зображено список доступних серверів, на цьому екрані користувач може обрати доступний сервер та виконати на ньому один з запитів, для отримання даних та інформації про технологію його розробки.

На рис. 3.16а знаходиться інформація після виконання запиту, також система здійснює аналіз після отримання даних та на їх основі будує діаграму залежності швидкості від об'єму отриманої інформації. Також користувач може завантажити отриманій лог-файл з усіма вииконаними операціями даної сесії.

Обравши певний сервер, користувач переходить на екран з його детальним описом (рис 3.17), на ньому зображено фото та назву серверу, інформація про сервер, коментарі до характеристик, на цьому екрані також можна лишити коментар, або переглянути існуючі запити.

3.4. Вибір стратегії тестування

Для досягнення максимальної якості тестування під кожен метод або модуль програми, що тестується необхідно обрати найбільш вдалий метод або набір методів, що забезпечать необхідний результат. При цьому необхідно підібрати найкраще співвідношення між часом, що буде витрачено на тестування, та якістю тестування. Набір тестів повинен мати мінімальну збитковість, але при цьому максимально охоплювати функціональність системи [10].

Під час тестування програми буде використано декілька різних методів тестування. Умовно ці методи можна поділити на 2 типи: методи білого ящика та методи чорного ящика. Різниця між цими групами методів полягає в тому, що при використанні методів чорного ящика, тестування відбувається без доступу до коду програми, в наявності тестувальника є тільки ті можливості, що матиме користувач програми. Під час використання методів

білого ящика, тестувальник використовує код програми для досягнення необхідного результату.

Для методів, що мають лінійну структуру, можна застосувати тестування «чорною скринькою» з використанням їхніх специфікацій методом еквівалентних розбиттів та припущення про помилку.

Тестування методів класів, які мають нелінійну структуру, тобто мають умови, цикли, рекурсивні виклики необхідно провести з використанням тестування методом покриття рішень («білою скринькою»). Також при тестуванні особливо складних методів класів необхідно додатково застосувати метод «чорної скриньки», а саме висування гіпотези про помилку. Це дозволить підвищити надійність особливо важливих для роботи системи методів та функцій. Даний метод досить добре підходить для тестування інтерфейсної частини програм.

І найголовніше – тестування зібраної програми, тобто після того як будуть протестовані всі методи класів, тому що помилки могли і не виникнути при незалежних тестуваннях окремих класів.

Отже, для тестування розробленого програмного продукту була розроблена наступна стратегія:

- для тестування були відібрані методи припущення про помилку, та покриття умов;
- тестування головних процедур має бути проведений всіма вибраними методами;
- тестування інших процедур допускається лише методом припущення про помилку;
- напрямок тестування – знизу вгору, тобто спочатку тестуються всі підмодулі, а далі головний модуль, який використовує всі підмодулі.

Середовище розробки Android Studio передбачає створення декількох типів тестів:

- створення складних модульних тестів – побудова більш складних модульних тестів, для Android залежностей, які не можуть бути легко заповнені за допомогою фіктивних об'єктів.
- автоматичні тести користувацького інтерфейсу – створення тестів, щоб переконатися, що призначений для користувача інтерфейс поводить себе правильно для взаємодії з користувачем в межах однієї програми або для взаємодії між декількома додатками.
- тестування компонентів системи – перевірка поведінки компонентів, з якими користувачі не взаємодіють безпосередньо, наприклад, сервісами або контентом постачальника.

3.4.1. Тестування функціональності

Для даного додатка було створено модульний тест, який перевіряє правильність введеної користувачем email адреси. Далі наведено лістинг класу модульного тесту з результатами тесту (рис. 3.18).

```
import android.test.suitebuilder.annotation.SmallTest;
import org.junit.Test;
import java.util.regex.Pattern;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

/**
 * Unit tests for the EmailValidator logic.
 */
@SmallTest
public class EmailValidatorTest {

    @Test
    public void emailValidator_CorrectEmailSimple_ReturnsTrue() {
        assertTrue(EmailValidator.isValidEmail("name@email.com"));
    }

    @Test
    public void emailValidator_CorrectEmailSubDomain_ReturnsTrue() {
        assertTrue(EmailValidator.isValidEmail("name@email.co.ua"));
    }

    @Test
    public void emailValidator_InvalidEmailNoTld_ReturnsFalse() {
        assertFalse(EmailValidator.isValidEmail("name@email"));
    }

    @Test
    public void emailValidator_InvalidEmailDoubleDot_ReturnsFalse() {
        assertFalse(EmailValidator.isValidEmail("name@email..com"));
    }
}
```

```

    }

    @Test
    public void emailValidator_InvalidEmailNoUsername_ReturnsFalse() {
        assertFalse(EmailValidator.isValidEmail("@email.com"));
    }

    @Test
    public void emailValidator_EmptyString_ReturnsFalse() {
        assertFalse(EmailValidator.isValidEmail(""));
    }

    @Test
    public void emailValidator_NullEmail_ReturnsFalse() {
        assertFalse(EmailValidator.isValidEmail(null));
    }
}

```

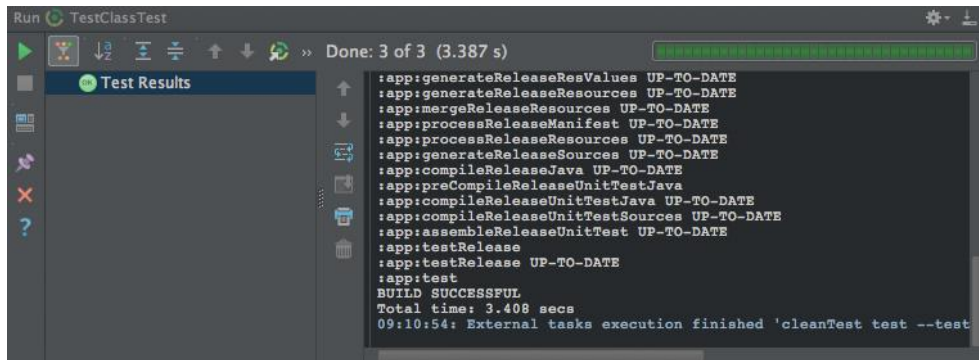


Рис. 3.18. Результат тесту EmailValidator

3.4.2. Тест користувацького інтерфейсу

Для перевірки реакції на зміну компонентів інтерфейсу користувача було написано окремий тест, який перевіряє зміну тексту під час введення в текстове поле. Далі наведено лістинг класу тесту з результатами тесту (рис 3.19).

```

package com.example.android.testing.espresso.texttest;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;

import android.support.test.rule.ActivityTestRule;
import android.support.test.runner.AndroidJUnit4;
...

@RunWith(AndroidJUnit4.class)
@LargeTest
public class ChangeTextBehaviorTest {

    private String mStringToBetyped;

    @Rule
    public ActivityTestRule<MainActivity> mActivityRule = new
    ActivityTestRule<>(

```

```

        MainActivity.class);

@Before
public void initValidString() {
    // Specify a valid string.
    mStringToBetyped = "Espresso";
}

@Test
public void changeText_sameActivity() {
    // Type text and then press the button.
    onView(withId(R.id.editTextUserInput))
        .perform(typeText(mStringToBetyped), closeSoftKeyboard());
    onView(withId(R.id.changeTextBt)).perform(click());

    // Check that the text was changed.
    onView(withId(R.id.textToBeChanged))
        .check(matches(withText(mStringToBetyped)));
}
}

```

Наведений тест імітує дії користувача, натискає на текстове поле, вводить текст з клавіатури, закриває клавіатуру та натискає на кнопку обробки тексту.

Element	Class, %	Method, %	L...
com.comp...	100% (2/2)	100% (12/12)	...
com.comp...	100% (1/1)	100% (3/3)	...
com.comp...	100% (1/1)	100% (9/9)	...

Рис. 3.19. Результат тесту ChangeTextBehaviorTest

3.4.3. Тестування методами чорної шухляди

Для розробленої програмної системи було написано ряд тестів для перевірки виконаного користувачем запиту в обраному сервері зі списку. Запит включає в себе перелік отриманих даних, швидкість з можливістю збереження інформації. Сервер включає в себе назву, інформацію, дату останнього запиту. Здійснювати перевірку введених даних за правилами:

- назва – будь-який не пустий рядок символів;
- швидкість – будь-яке дійсне число, більше за 0;
- дата виконання запиту – будь-яка дата, менша або рівна поточній.

Далі наведено лістинг тесту, з результатами тесту (рис.3.20).

```

import org.testng.annotations.Test;
import static org.testng.Assert.*;
/**
 * Created by Andrey on 26.03.2017.
 */

```

```

public class UserAccountTest {
    @Test
    public void testGetDiscount() throws Exception {
        assertEquals(UserAccount.getDiscount(999f), 0);
        assertEquals(UserAccount.getDiscount(1_000f), 1);
        assertEquals(UserAccount.getDiscount(4_999f), 1);
        assertEquals(UserAccount.getDiscount(5_000f), 5);
        assertEquals(UserAccount.getDiscount(5_001f), 5);
        assertEquals(UserAccount.getDiscount(9_999f), 5);
        assertEquals(UserAccount.getDiscount(10_000f), 10);
        assertEquals(UserAccount.getDiscount(10_001f), 10);
        assertEquals(UserAccount.getDiscount(20_000f), 20);
        assertEquals(UserAccount.getDiscount(20_001f), 20);
    }
}
import org.testng.annotations.Test;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import static org.testng.Assert.*;
/**
 * Created by Andrey on 26.03.2017.
 */
public class ServerTest {

    @Test
    public void testAddServer() throws Exception {
        Server = new Server();
        assertEquals(Server.setName(""), false);
        assertEquals(Server.setName("Server1"), true);
        assertEquals(Server.setCost(0), false);
        assertEquals(Server.setCost(-0.0001), false);
        assertEquals(Server.setCost(0.0001), true);

        assertEquals(Server.setDate(new Date(2010, 01, 01), false)); //year,
month, date
        assertEquals(Server.setDate(new Date(2017, 01, 03), false);
        assertEquals(Server.setDate(new Date(2010, 01, 02), true);
        assertEquals(Server.setDate(new Date(2017, 01, 02), true);
        assertEquals(Server.addServer(Server), true);
    }
    @Test
    public void testGetAverageSum() throws Exception {
        List<Server> Servers = Arrays.asList(new Server("Server1",10,new Date()),
            new Server("Server2", 11, new Date()));
        assertEquals(Server.getAverageSum(Servers), 10.5f, 0.0001f);
    }
}
import org.testng.annotations.Test;
import java.util.Date;
import static org.testng.Assert.*;
/**
 * Created by Andrey on 26.11.2017.
 */
public class AccountsManagerTest {

    @Test
    public void testMakePurchase() throws Exception {
        AccountsManager.addUser(new UserAccount("Jacob"));
        AccountsManager.addUser(new UserAccount("Emma"));
        AccountsManager.addUser(new UserAccount("William"));

        Server p1 = new Server("Server1", 1001f, new Date());

```



```

Server p2 = new Server("Server2", 5000f, new Date());
Server p3 = new Server("Server3", 6000f, new Date());

    assertEquals(AccountsManager.makePurchase("William", p1), 1001f,
0.0001f);
    assertEquals(AccountsManager.makePurchase("William", p2), 4950f,
0.0001f);
    }
}

```

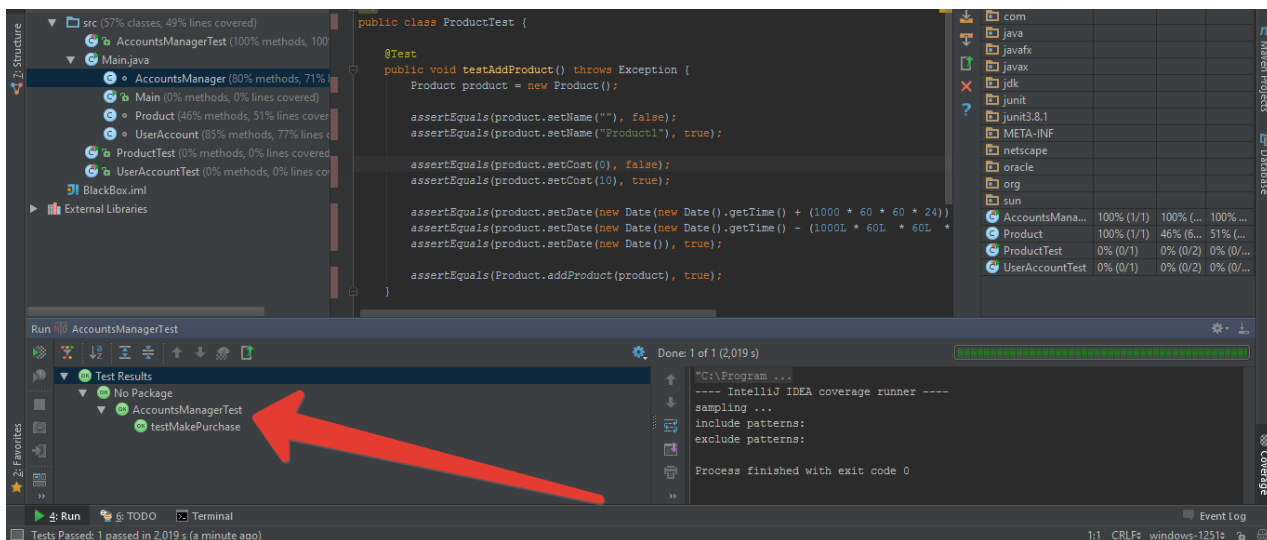


Рис. 3.20. Результати виконання Unit – тестів

Як бачимо на рис. 3.20, усі Unit-тести були виконані успішно.

Перш ніж виконати тестування методів методами чорної шухляди, приведемо опис кожного з них у табл. 3.2.

Таблиця 3.2.

Опис методів системи.

Прототип методу	Опис
1	2
Клас UserAccount: public static int getSpeed(double totalSpeed)	Повертає розмір швидкості за виконання запиту користувача користувача у байтах за секунду Приймає загальну швидкість, які користувач зробив раніше
Клас AccountsManager: public static float makeRequest(String userName, Server Server) throws Exception	Повертає дані з серверу, які необхідно відобразити клієнту, враховуючи швидкість. Додає інформацію про обраний сервер “Server” до відомостей про усі виконані запити користувача “username”. Повертає

	похибку якщо користувача не було знайдено в базі.
--	---

Продовження таблиці 3.2

1	2
Клас Server: public boolean setName(String name)	Встановлює назву серієру, якщо рядок не порожній. Повертає “true”, якщо зміни зроблені, інакше – “false”.
Клас Server: public boolean setCost(float cost)	Встановлює коштовність сервнру, якщо вона більше ніж 0. Повертає “true”, якщо зміни зроблені, інакше – “false”.
Клас Server: public boolean setDate(Date date)	Встановлює дату виконання запиту, якщо вона менше або дорівнює поточній даті. Повертає “true”, якщо зміни зроблені, інакше – “false”.

3.4.4. Метод еквівалентних розбиттів

Для функції `UserAccount.getServerSpeed(float totalSpeed)` приведемо наступні класи еквівалентності (рис. 3.21).

Розіб’ємо загальну суму замовлення на наступні класи:

1. $totalSpeed < 1000.0$;
2. $1000.0 \leq totalSpeed < 5000.0$;
3. $5000.0 \leq totalSpeed < 10000.0$;
4. $10000.0 \leq totalSpeed < 20000.0$;
5. $totalSpeed \geq 20000.0$.

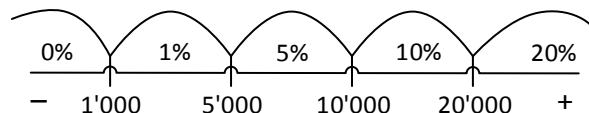


Рис. 3.21. Класи еквівалентності функції `getDiscount()`

Результати тестування подані у табл. 3.3.

Таблица 3.3.

Результати тестування методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	<code>totalCost = 0</code>	0

2	totalCost = 1001	1
3	totalCost = 5001	5
4	totalCost = 10001	10
5	totalCost = 20001	20

Для функції `Server.setName(String name)` приведемо наступні класи еквівалентності:

1. name = пустий рядок (неправильний клас);
2. name = не пустий рядок (правильний клас).

Результати тестування подані у табл. 3.4.

Таблиця 3.4.

Результати тестування `Server.setName()` методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	name = ""	Помилка: назва серверу не може бути пустим рядком. Результат = false
2	name = "Server1"	true

Для функції `Server.setSpeed(float cost)` приведемо наступні класи еквівалентності:

1. speed ≤ 0 (неправильний клас);
2. speed > 0 (правильний клас).

Результати тестування подані у табл. 3.5.

Таблиця 3.5.

Результати тестування `UserAccount.setCost()` методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	cost = 0	Помилка: коштовність серверу не може бути менше або рівна 0. Результат = false
2	cost = 100	true

Для функції `Server.setDate(Date date)` приведемо наступні класи еквівалентності:

1. `date >` поточної дати (неправильний клас);

Результати тестування подані у табл. 3.5.

Таблиця 3.6.

Результати тестування `UserAccount.setDate()` методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	<code>date = "01.01.2100"</code>	Помилка: запит повинен бути виконаний не пізніше поточної дати Результат = <code>false</code>
2	<code>date = "01.01.1800"</code>	Помилка: запит повинен бути виконаний не раніше, ніж за 5 днів до поточної дати та не пізніше поточної дати Результат = <code>false</code>
3	<code>date = "26.06.2017"</code>	<code>true</code>

Для функції `double Server.getAverageSum(List<Server> Servers)` приведемо наступні класи еквівалентності:

1. пустий список серверів (невірний клас);
2. не пустий список серверів (вірний клас).

Результати тестування функції `double Server.getAverageSum(List<Server> Servers)` подані у табл. 3.7.

Таблиця 3.7.

Результати тестування функції `double Server.getAverageSum(List<Server> servers)` методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	2	3
1	<code>servers = { }</code>	Помилка: неможливо підрахувати середню швидкість виконання

		запиту
2	servers = {<Server1, 200, "02.02.2002">, <Server12, 400, "03.03.2003">}	300, вірно

Продовження таблиці 3.7

1	2	3
3	servers = {< Server2, 500, "11.05.2005">, < Server2, 700, "09.03.2003">}	600, вірно
4	servers = {< Server2, 700, "09.03.2003">, <null>}	Помилка: неможливо підрахувати середню швидкість виконання запиту з неповним списком

Для функції AccountManager.makePurchase(String userName, Server Server) throws Exception приведемо наступні класи еквівалентності:

1. заданого користувача не існує – не вірний;
2. заданий користувач існує, користувач здійснює перший запит або загальна швидкість усіх попередніх запитів користувача менша 1000 б/с. – вірний;
3. заданий користувач існує, користувач здійснює не перший запит та загальна сума всіх попередніх запитів користувача більша або дорівнює 1000 б/с. – вірний.

Результати тестування функції AccountManager.makeRequest(String userName, Server Server) throwsException, що робить запит, приймає ім'я користувача, список серверів, та повертає швидкість запиту подані у табл. 3.8.

Таблиця 3.8.

Результати тестування Server.getAverageSum() методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	2	3
1	username = "Unknown"	Помилка:

	Server = <Server1, 200, "02.02.2002">	заданого користувача не існує
2	username = "User1" Server = <Server1, 1000, "02.02.2002">	1000

Продовження таблиці 3.8

1	2	3
3	Виклик 1: username = "User1" Server = <Server1, 1000, "02.02.2002"> Виклик 2: username = "User1" Server = <Server1, 1000, "02.02.2002">	Виклик 1: 1000 Виклик 2: 990.
4	username = "User1" Server = <Unknown>	Помилка: списку серверів не зазначено

3.4.5. Тестування методом граничних умов

Виконаємо тестування кожної з вище зазначених функцій методом граничних умов там, де це необхідно.

Граничні умови – це ситуації, які виникли на вищих та нижчих границях вхідних класів еквівалентності. Наприклад, якщо вхідні значення повинні бути в інтервалі [-1.0; 1.0], то використовуючи метод граничних умов, необхідно перевірити роботу програми при вхідних значеннях -1.0, 1.0, -1.001, 1.001.

Для наступних класів еквівалентності протестуємо такі вхідні значення:

- [1000; 5000]: 1000, 5000, 999, 5001;
- [5000; 10000]: 5000, 10000, 4999, 10001;
- [10000; 20000]: 10000, 20000, 9999, 20001.

Результати тестування функції UserAccount.getDiscount() подані у табл. 3.9.

Таблиця 3.9.

Результати тестування UserAccount.getDiscount() методом граничних умов.

№ тесту	Вхідні дані	Вихідні дані
1	2	3
1	totalCost = 500	0
2	totalCost = 999	0
3	totalCost = 1000	1

Продовження таблиці 3.9

1	2	3
4	totalCost = 4999	1
5	totalCost = 5000	5
6	totalCost = 5001	5
7	totalCost = 9999	5
8	totalCost = 10000	10
9	totalCost = 10001	10
10	totalCost = 20000	20
11	totalCost = 20001	20

Для функції Server.setName(String name) тести повністю співпадають з тестами методу еквівалентних розбиттів.

Для функції Server.setSpeed(float cost) тести подані у табл. 3.10.

Таблиця 3.10.

Результати тестування Server.setSpeed() методом граничних умов.

№ тесту	Вхідні дані	Вихідні дані
1	speed = 0	Помилка: швидкість запиту не може бути менше або рівна 0. Результат = false
2	speed = 0.0001	true
3	speed = 99.01	true
4	speed = -0.0001	Помилка: швидкість запиту не може бути менше або рівна 0. Результат = false

Для функції Server.setDate(Date) приведемо тести у табл. 3.11. Для прикладу, припустимо, що поточна дата – 2 лютого 2017 року.

Результати тестування `Server.setDate()` методом граничних умов.

№ тесту	Вхідні дані	Вихідні дані
1	2	3
1	date = "02.01.2010"	true
2	date = "02.01.2017"	true

Продовження таблиці 3.11

1	2	3
3	date = "01.01.2010"	Помилка: запит повинен бути виконаний не раніше, ніж за 5 днів до поточної дати та не пізніше поточної дати Результат = false
4	date = "03.01.2017"	Помилка: запит повинен бути виконаний не раніше, ніж за 5 днів до поточної дати та не пізніше поточної дати Результат = false

3.4.6. Тестування методом припущення помилки

Процедура методу припущення про помилку в значній мірі заснована на інтуїції. Основна його ідея полягає в тому, щоб перерахувати в деякому списку можливі помилки або ситуації, в яких вони можуть з'явитися, а потім на основі цього списку скласти тести.

1. Спираючись на власний досвід, припустимо, що метод `Server.setDate()` працює некоректно при введенні дати у не вірному форматі. Наприклад, прописом «перше січня тисяча дві тисячі першого року».
2. Припустимо, що метод `Server.setDate()` працює некоректно якщо користувач переплутає формат написання дати. Наприклад, «12.31.2017».

3. Припустимо, що метод `Server.setSpeed()` працює некоректно при введенні швидкості запиту у невірному форматі. Наприклад, «двадцять дев'ять».
4. Припустимо, що метод `makeRequest()` працює неправильно при спробі зробити користувачем запиту з серверу, якого немає в БД програми.

Результати тестування системи методом припущення помилки приведені у табл. 3.12.

Таблиця 3.12.

Результати тестування методом припущення помилки

№ тесту	Вхідні дані	Вихідні дані
1	date = “перше січня тисяча дві тисячі першого року”	Помилка: дата виконання запиту задана не вірно. false
2	date = “12.31.2017”	Помилка: дата виконання запиту задана не вірно. false
3	cost = “двадцять дев'ять”	Помилка: швидкість запиту введена не вірно. false
4	username = “Unknown” Server = <Server1, 200, “02.02.2002”>	Помилка: заданого користувача не існує

3.5. Висновки на основі отриманих даних після тестування серверних технологій в мобільному додатку

Після отриманих результатів з мобільного додатка, можна зробити наступні висновки. Найдешевшою та найпростішою у використанні, для мобільного розробника є система Firebase. Проте головний недолік цієї системи полягає в кастомізації користувацької бази. Виділений сервер має

переваги в налаштуванні, проте час на розробку серверу занадто більший від хмарних.

	Формат передачі даних	Доступ до соціальних мереж	Оповіщення	Аналітика	Вартість	Тип бази даних	Швидкість реагування
Parse	Ключ-значення	+	+	+	0	no SQL	30 req/s
Firebase	json	-	+	+	5\$/міс	Cloud SQL	1000 req/s
Backendless	SQL-style	-	+	+	8\$/міс	SQL	50 req/s
Amazon AWS	Ключ-значення	+	+	+	10\$/міс	RDS	47 req/s
Microsoft Azure	SQL-style	-	+	+	10\$/міс	SQL	50 req/s
Kinvey	Ключ-значення	+	+	+	7\$/міс	no SQL	50 req/s
Власний сервер	Налаштовується	Налаштовується	+	Налаштовується	В залежності від витрат	Налаштовується	В залежності від розташування

РОЗДІЛ 4. ЕКОНОМІЧНИЙ РОЗДІЛ

Дані для розрахунків:

- 1) передбачуване число операторів – 1640;
- 2) коефіцієнт складності програми – 1.6;
- 3) коефіцієнт корекції програми в ході її розробки – 0.11;
- 4) годинна заробітна плата програміста, грн/год – 24.0;
- 5) вартість машинного часу – 8;

4.1. Розрахунок трудомісткості і вартості розробки програмного продукту

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{omi} + t_d, \text{ людино-годин,} \quad (4.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі, (приймається $t_o = 50$);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

t_{oml} – витрати праці на налагодження програми на ЕОМ;

t_{∂} – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів в програмі обчислюється за формулою:

$$Q = qC(1 + p), \quad (4.2)$$

де q – передбачуване число операторів, $q = 1640$;

C – коефіцієнт складності програми, $C = 1.6$;

p – коефіцієнт корекції програми в ході її розробки, $p = 0.11$.

Звідси:

$$Q = 1640 \times 1.6 \times (1 + 0.11) = 2913$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{QB}{(75 \dots 85)K} \text{ ,людино-годин} \quad (4.3)$$

де B – коефіцієнт збільшення витрат праці унаслідок недостатнього опису завдання. Він може мати значення від 1,2 до 1,5, в нашому випадку він складає 1,4.

K – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. Він складає при стажі роботи, задану кількість років. Оскільки стаж дорівнює від двох до трьох років, $K = 1,0$.

$$t_u = \frac{(2913 \times 1,4)}{(80 \times 1,0)} = 51 \text{ , люд.-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)K}, \quad (4.4)$$

$$t_a = \frac{2913}{(24 \times 1,0)} = 121,37, \text{ люд.-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25)K}, \quad (4.5)$$

$$t_n = \frac{2913}{(23 \times 1,0)} = 127, \text{ люд.-годин.}$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)K}, \quad (4.6)$$

$$t_{отл} = \frac{2913}{(4 \times 1,0)} = 728,25, \text{ люд.-годин.}$$

Витрати праці на налагодження програми за умови комплексного налагодження завдання:

$$t^k_{отл} = 1,5 \cdot t_{отл}, \quad (4.7)$$

$$t^k_{отл} = 1,5 \times 728 = 1092, \text{ люд.-годин.}$$

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}, \quad (4.8)$$

де $t_{\partial p}$ – трудомісткість підготовки матеріалів і рукопису.

$$t_{\partial p} = \frac{Q}{(15..20)K}, \quad (4.9)$$

$$t_{\partial p} = \frac{2913}{(17 \times 1,0)} = 171, \text{ люд.-годин.}$$

$t_{\partial o}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{\partial o} = 0,75 \cdot t_{\partial p}, \quad (4.10)$$

$$t_{\partial o} = 0,75 \times 171 = 128,25, \text{ люд.-годин.}$$

Звідси: $t_{\partial} = 171 + 128,25 = 299,25, \text{ люд.-годин}$

В підсумку отримуємо, що трудомісткість розробки ПЗ становить:

$$t = 50 + 51 + 121.37 + 127.25 + 728 + 128.25 = 1205.87, \text{ люд.-годин}$$

Таким чином, трудомісткість розробки програмного забезпечення складає 1205,87 люд.-годин.

4.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ K_{no} включають витрати на заробітну плату виконавця програми Z_{zn} і витрат машинного часу, необхідного на налагодження програми на ЕОМ

$$K_{no} = Z_{zn} + Z_{mv} \quad (4.11)$$

Заробітна плата виконавців визначається за формулою:

$$Z_{zn} = t \cdot C_{pp}, \quad (4.12)$$

де: t – загальна трудомісткість, людино-годин;

C_{pp} – середня годинна заробітна плата програміста, грн/година;

$$C_{pp} = 35,0 \text{ грн/год.}$$

$$\text{Отже: } Z_{zn} = 1205.87 \times 35 = 42175 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$Z_{mv} = t_{oml} \cdot C_{mch}, \quad (4.13)$$

де t_{oml} – трудомісткість налагодження програми на ЕОМ, год.

C_{mch} – вартість машино-години ЕОМ, грн/год (приблизно 20грн/год).

Звідси:

$$Z_{mv} = 728,25 \times 20 = 14565 \text{ грн.}$$

З отриманих даних витрати на створення програмного забезпечення складатимуть:

$$K_{no} = 42175 + 14565 = 56740 \text{ грн.}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.}, \quad (4.14)$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні $F_p = 176$ годин).

Звідси:

$$T = \frac{1205.87}{(1 \times 176)} \approx 6 \text{ міс.}$$

Таким чином, період розробки програми складає приблизно 6 місяців.

4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту

В сучасних умовах розробники ПП повинні перш за все турбуватися про підвищення якості програмних продуктів та покращення споживацьких властивостей товару. Маркетингові дослідження при розробці ПП дозволяють виявити споживацькі властивості ПП, потрібну функціональність, визначити величину попиту на ПП та сформулювати основні вимоги до ПП, який повинен бути розроблений. Розглянемо деякі програмні аналоги.

Додаток Amazon Web Services (AWS) Console для iOS і Android дозволяє швидко і просто переглядати і управляти існуючими інстанси EC2, балансувальник навантаження, зонами хостингу Route 53, інстанси RDS, групами Auto Scaling, додатками AWS Elastic Beanstalk, таблицями Amazon DynamoDB, стеками AWS OpsWorks і попередженнями CloudWatch зі свого смартфона або планшета.

Ця програма добре доповнює повнофункціональні можливості роботи з Консоллю управління через Інтернет, дозволяючи в мобільному режимі виконувати такі операції: перегляд зон хостингу і відомостей про зони хостингу, наборів записів і відомостей про набори записів; перегляд

відомостей про групи, метрик, політик і попереджень. Редагування мінімального, максимального і запитаного кількості інстансів. Перегляд відомостей про конфігурацію стеків, шарів, інстанси і додатків. Перегляд логів інстанси, перезавантаження інстанси і отримання статусу про розгортання.

Додаток Mobile Dashboard.iQ забезпечує збір різних даних про бізнес-діяльність організації згідно з потребами користувача і відображає їх у форматі звітів для подальшого аналізу і прийняття рішень. Аналітичні дані надаються в режимі реального часу відповідно до змісту і графіку, складеним користувачем системи. Передбачені наступні функції системи: доступ до всіх типів аналітичної інформації та узгодження використовуваних організацією систем; оптимізація управління активами організації; можливість створення персоналізованих звітів; моніторинг бізнес-процесів в режимі реального часу; можливість підписатися на інформацію про контрольні події; обмеження на отримання даних згідно з правами доступу до системи.

Додаток QualityTime може зібрати інформацію про щоденне використання мобільного гаджета і окремих додатків. На основі цієї статистики можна зробити висновок, наскільки далеко зайшло виконання певного запиту з серверу і, в разі необхідності, обмежити його використання.

Grapholite – це універсальний редактор діаграм, призначений для створення різних типів ділової і технічної графіки від найпростіших чорнових начерків до складних документів.

Незважаючи на те, що продукт був розроблений з урахуванням обмежень планшетних комп'ютерів, він включає в себе всі можливості сучасних редакторів графіки типу Visio: систему стилів і тем, складний алгоритм автоматичного розташування зв'язків і з'єднань, динамічну сітку для вирівнювання об'єктів і вбудований редактор кривих і сплайнів .

Прості елементи діаграм і з'єднання між ними можуть створюватися дуже швидко за допомогою інструменту для малювання з автоматичним розпізнаванням геометричних фігур. Для більш складних зображень ви

можете використовувати сотні вбудованих візуальних об'єктів з набору трафаретів. Багато з цих об'єктів не просто статичні зображення, а «розумні» фігури, чия форма гнучко настроюється в процесі редагування. Також реалізована стилізація малюнка «під чернетку»: в один дотик для будь-якого об'єкта включається імітація відтворення «від руки».

Маркетинг супроводжує будь-який товар, в тому числі програмний продукт, на всіх стадіях життєвого циклу. Одним з перших стратегічних рішень, які приймає підприємство-розробник ПП, є визначення ринку, на якому воно буде вести конкурентну боротьбу. Базовою посилкою при формулюванні вимог до розроблюваного програмного продукту повинна бути відповідність техніко-економічних характеристик ПП потребам користувачів. Для орієнтації діяльності підприємства-розробника ПП на задоволення потреб конкретної аудиторії, ринок збуту програмних продуктів повинен бути проаналізований та оцінений.

Усі зазначені вище програмні засоби-аналоги потребують версію операційної системи Android не нище версії 4.0. Вимоги до апаратного забезпечення залежать від встановленої операційної системи. Деякі програмні аналоги підтримують роботу лише з смартфонами, та не підтримують мобільні планшети. Проте головним недоліком зазначених додатків є те, що вони не підтримують великої кількості технологій хмарної розробки серверів.

Аналіз існуючого ринку показує, що більшість виробників спеціалізується на декількох напрямках діяльності. Для країн з «перехідною економікою», зокрема для України, характерне прагнення багатьох підприємців працювати з різними товарами на будь-яких ринках через дефіцитність багатьох товарів і послуг. Це дає змогу успішно діяти на тому чи іншому ринку, навіть не знаючи добре особливостей самого ринку, специфіки товару і покупців, перспектив розвитку і т. п. В умовах ринку його вибір є ключовим.

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

Основна мета економічної ефективності – дати фінансову оцінку передбачуваних витрат та одержуваного корисного результату, а також оцінити прибутковість проекту і, в кінцевому підсумку, економічну доцільність його розробки та впровадження.

Нова техніка, технологія, засоби автоматизації, що розробляються і впроваджуються у виробництво, повинні приносити певний корисний результат – ефект. Ефект може проявлятися у поліпшенні умов праці працюючих (соціальний), в зниженні шкідливого впливу виробництва на навколишнє середовище (екологічний), та в економії витрат підприємства на виробництво продукції та збільшенні його прибутку (економічний).

Абсолютна величина економічного ефекту без співставлення його з витратами підприємства не дозволяє однозначно оцінити, наскільки вдалим виявився відповідний інноваційний проект. Таку оцінку дають показники економічної ефективності (прибутковості) проекту.

При впровадженні інвестиційного проекту підприємство несе разові витрати, пов'язані з розробкою проекту, а також з придбанням і налагодженням необхідного обладнання, засобів програмного забезпечення і таке інше.

Такі разові витрати називають капітальними витратами або інвестиціями. При використанні інновацій підприємство отримує певний ефект, що зазвичай виражається приростом прибутку. Величина щорічного прибутку, додатково одержуваного підприємством за рахунок впровадження інвестиційного проекту, повинна бути достатньо високою у порівнянні з капітальними витратами підприємства та у порівнянні з іншими можливими варіантами вкладення коштів у розвиток виробництва.

Розроблений програмний проєкт буде поширюватись в рамках “ Open Source Software”, що передбачає вільне розповсюдження, доступний у відкритому доступі текст програми, пов'язані з відкритим ПО дані, повинні

бути застосовні до всіх користувачів програми без укладення додаткових угод, наприклад, угоди про нерозголошення та технологічно нейтральною ліцензією. Головною метою програмного продукту є отримання великої кількості аудиторії та числа користувачів, тому економічна ефективність у даного продукту може враховуватись лише у випадку монетизації додатку (поширення реклами), а на передній план виходить соціальна ефективність від впровадження, що характеризує покращення навичок у користувачів та розробників програмного забезпечення.

Таблиця 4.1.

Розрахунок чистих грошових надходжень від розробки ПЗ.

Показники, грн	За роками						Усього за 5 років	Середнє за 5 років
	0	1	2	3	4	5		
1. Інвестиції на ПЗ	56740	-	-	-	-	-	56740	11348
2. Витрати до впровадження ПЗ,	-	9700	3700	3700	3700	3700	24500	5300
- на придбання ліцензійного ПЗ	-	3000	-	-	-	-	3000	600
- на покупку пристрою Android	-	3000	-	-	-	-	3000	600
- на придбання хостингу	-	2000	2000	2000	2000	2000	10000	2000
- на щорічну перевірку серверів	-	500	500	500	500	500	2500	500
- на електроенергію	-	1200	1200	1200	1200	1200	6000	1200
3. Витрати після впровадження ПО	-	1240	1240	1240	1240	1240	9200	1840
- на продовження хостингу	-	1000	1000	1000	1000	1000	5000	1000
- на щорічну перевірку серверів	-	200	200	200	200	200	1000	200
- на електроенергію	-	240	240	240	240	240	1200	240
4. Економія	-	8460	2460	2460	2460	2460	15300	2660
5. Амортизація	-	11348	11348	11348	11348	11348	56740	11348
6. Чисті грошові надходження	-	19808	13808	13808	13808	13808	72040	14408
7. Коефіцієнт дисконтування	-	0,870	0,756	0,658	0,572	0,497	-	-
Дисконтові грошові надходження	-	19103	11683	9428	7326	6366	62740	12548

Коефіцієнти економічної ефективності

Чиста поточна вартість доходів:

$$NPU = 62740 - 56740 = 6000 \text{ грн. } > 0$$

Строк окупності:

$$T = 56740 / 12548 = 4 \text{ роки}$$

Індекс прибутковості:

$$ІД = 62740 / 56740 = 1.10$$

Показник економічної ефективності (NPU – чиста поточна вартість доходів за роки реалізації впровадження (3-5 років) складе 6000 грн тобто він відповідає умовам ефективності, тому що $NPU > 0$. Середній строк окупності капітло-вкладень складе 4 роки. Індекс прибутковості за 5 років складе 1.10, тобто $ІД > 1$, проект варто прийняти. Таким чином, показник ефективності свідчить про те, що дане впровадження є майже економічно вигідним.

4.5. Висновки

Витрати на створення програмного додатка для статистичного аналізу та дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів складають 56740 грн.

Період розробки програми складає приблизно 6 місяців.

ВИСНОВКИ

В результаті виконаної роботи, було спроектовано та розроблено Android додаток для статистичного аналізу та дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів. Програмний продукт складений з двох частин – серверної та клієнтської. Серверна частина містить базу даних для обліку користувачів, переліку та надання тестових даних, логів тощо. Клієнтська частина має інтерактивний інтерфейс для перегляду списку серверів, виконання запитів на них, складання графіків, виконання аналізу результатів та ін.

В роботі виконано аналіз існуючих серверних технологій для розробки мобільних додатків, виявлення їх переваг та недоліків. Спроектовано та розроблено мобільний додаток для операційної системи Android, для надання та зображення отриманих результатів.

Під час аналізу предметної області були освітлені основні проблеми використання інтернет-технологій в сфері розробки програмного забезпечення.

В результаті дослідження встановлено певні недоліки та переваги використання технологій хмарного та виділеного серверів, які описані в роботі.

Для того, щоб додаток максимально відповідав поставленим вимогам, був зручним та багатофункціональним, перед його проектуванням та розробкою були розглянуті та проаналізовані існуючі методології та літературні джерела. В розглянутих методах розробки аналогічних додатків були виявлені недоліки та переваги. Деякі з рішень, що були знайдені в цих методологіях, були використанні під час проектування та розробки мобільного додатка.

Проектування та розробка програми проводилась з використанням сучасних технологій та підходів до вирішення цих питань, також була спроектована база даних серверної частини.

Проектування системи відбувалося з використанням об'єктно-орієнтованого підходу. Використання шаблонів проектування та окремі дизайнерські рішення розробника дозволили зробити систему гнучкою, що легко піддається модифікації, зрозумілою та надійною.

Використання сучасних гнучких технологій розробки дозволить і надалі розширювати функціональність системи або вносити зміни до її поточної функціональності.

Інтерфейс програми було спроектовано з урахуванням потреб цільового користувача. Було досягнуто основної мети зробити інтерфейс зручним та зрозумілим. Інтерфейс мав бути спроектований так, щоб будь які дії займали у користувача мінімальну кількість часу, адже планується велика інтенсивність використання додатка.

Велику увагу було приділено тестуванню та відлагодженню системи. Використання сучасних інструментів тестування та відлагодження та використання різних підходів та методів тестування, що найкраще підходять до тестування кожного окремого методу дозволили зробити систему надійною та стійкою до можливих помилок.

Була проаналізована економічна ефективність розроблюваної системи. Проведені розрахунки свідчать про економічну доцільність розробки та провадження нового програмного продукту. А також було проведено визначення трудомісткості розробки програмного забезпечення та розраховані витрати на створення програмного забезпечення

Після закінчення розробки, була написана вся необхідна програмна документація. Наявність якісного керівництва користувача дозволить користувачам, що будуть використовувати додаток, швидко вивчити можливості програми та почати її використовувати. Задokumentований опис та текст програми спростить подальше супроводження системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1 Герберт Шилдт. Java. Полное руководство. –М.:Вильямс, 2012. – 1104 с.
- 2 Рето Майер. Android 4. Программирование приложений для планшетных компьютеров и смартфонов. – М.: ЕКСМО, 2013. – 816 с.
- 3 Документація Realm (Електрон. ресурс) / Спосіб доступу: URL: <https://realm.io/> – Загол. з екрана.
- 4 Документація Git (Електрон. ресурс) / Спосіб доступу: URL: <https://git-scm.com/>. – Загол. з екрана.
- 5 Матеріал з Вікіпедії - вільної енциклопедії (Електрон. ресурс) / Спосіб доступу: URL:https://ru.wikipedia.org/wiki/Android_Studio
- 6 Документація Dagger (Електрон. ресурс) / Спосіб доступу: URL: <https://google.github.io/dagger/>. – Загол. з екрана.
- 7 Девід Гриффитс. Head First. Программирование для Android. – М.:Питер, 2016. – 704 с.
- 8 Пол Дейтел. Android для разработчиков. –М.:Питер, 2016. – 512 с.
- 9 Матеріал з Вікіпедії - вільної енциклопедії (Електрон. ресурс) / Спосіб доступу: <https://ru.wikipedia.org/wiki/Android>
- 10 Матеріал з Вікіпедії - вільної енциклопедії (Електрон. ресурс) / Спосіб доступу: URL: <https://ru.wikipedia.org/wiki/Git>
- 11 Habrahabr – «Структура приложения для Android» (Електрон. ресурс) / Спосіб доступу: URL: habrahabr.ru/company/ncloudtech/blog/274025/
- 12 Habrahabr – «Dagger 2. Часть первая» (Електрон. ресурс) / Спосіб доступу: URL: <https://habrahabr.ru/post/279125/>
- 13 Habrahabr – «Dagger 2. Часть вторая» (Електрон. ресурс) / Спосіб доступу: <https://habrahabr.ru/post/279641/>
- 14 Скотт Чакон. Git для профессионального программиста. –М.:Питер, 2016. – 496 с.
- 15 Чад Фаулер. Программист-фанатик. –М.:Питер, 2016. – 208 с.
- 16 Стив Макконнелл. Совершенный код. Мастер-класс. – М.:Питер, 2016. – 896 с.

- 17 Эрих Гамма. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.:Microsoft, 2016. – 366 с.
- 18 Роберт К. Мартин. Чистый код: создание, анализ и рефакторинг.. – М.:Питер, 2016. – 464 с.
- 19 Документація Java (Електрон. ресурс) / Спосіб доступу: URL: <https://docs.oracle.com/javase/7/docs/api/>. – Загол. з екрана.
- 20 Документація Android (Електрон. ресурс) / Спосіб доступу: URL: <https://developer.android.com>. – Загол. з екрана.
- 21 Шилдт, Г. Java. Полное руководство, 8-е изд. [Текст] / Г. Шилдт; пер. с англ. – М. : ООО “И.Д. Вильямс”, 2012. – 1104 с. : ил. – Парал. тит. англ. ISBN 978-5-8459-1759-1 (рус).
- 22 Etheridge David. Java, Graphical User Interfaces. David Etheridge & Ventus Publishing ApS, 2009. – 101 p.
- 23 Evans B.J., Verburg M. The Well-Grounded Java Developer: Vital techniques of Java 7 and polyglot programming. Manning Publications, 2012. - 496 P.
- 24 Goncalves A. Beginning Java EE 7. Apress, 2013. – 608 p.
- 25 Higuera-Toledano M.T., Wellings A.J. (eds.) Distributed, Embedded and Real-time Java Systems. Springer, 2012, -389 p.
- 26 Jendrock E. The Java EE 6 Tutorial: Basic Concepts. Prentice Hall, 2010. – 600 p.
- 27 Coward D. Java EE 7: The Big Picture: Master the Code, Applications, and Frameworks of Java Platform, Enterprise Edition 7. McGraw-Hill, 2015. – 512 p.
- 28 Gupta A. Java EE 7 Essentials. O'Reilly Media, 2013.– 362 p.
- 29 Pilgrim P.A. Java EE 7 Developer Handbook: Develop professional applications in Java EE 7 with this essential reference guide. Packt Publishing, 2013. – 634 p.
- 30 Эванс Б., Вербург М. Java. Новое поколение разработки. СПб.: Питер, 2014. – 560 с.

- 31 Сети М. Программирование игр. – М.: Вильямс, 100 книг, 2007. – 384 с.
- 32 Шилдт Г. Полный справочник по Java. – М.: Вильямс, 2009. - 1040 с.
- 33 Блинов И.Н. Java 2: практическое руководство / И.Н. Блинов, В.С. Романчик. – Мн.: УниверсалПресс, 2005. – 400 с.
- 34 Грофф Дж. Энциклопедия SQL. 3-е издание / Дж. Грофф, П. Вайнберг – М.: Вильямс, 2003. - 896 с.
- 35 Гарсиа-Молина Г. Системы баз данных. Полный курс / Г. Гарсиа-Молина Дж. Ульман , Дж. Уидом – М.: Вильямс, 2003. – 1088 с.
- 36 Дашко Ю.В. Основы разработки компьютерных игр / Ю.В. Дашко, А.А. Заика. – М.: Форум, 2009. - 350с.
- 37 Роллингз Э. Проектирование и архитектура игр: пер. с англ. / Э. Роллингз, Д. Моррис. – М.: Вильямс, 2006.- 1040с.
- 38 Adamatzky A. (Ed.) Advances in Unconventional Computing. Volume 2. Prototypes, Models and Algorithms. – Springer International Publishing, 2017. – 812 p.
- 39 Albers V., Still B. (Eds.) Usability of Complex Information Systems: Evaluation of User Interaction. – CRC Press, 2011. – 392 p.
- 40 Arlow Jim, Neustadt Ila. Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML. – Addison–Wesley Professional, 2004. 528 p.
- 41 Aslaksen E.W. Designing Complex Systems: Foundations of Design in the Functional Domain. – Auerbach Publications; 1 edition (October 27, 2008). – 176 p.
- 42 Baskerville R., De Marco M., Spagnoletti P. (Eds.) Designing Organizational Systems: An Interdisciplinary Discourse. – Springer, 2013 – 342 p.
- 43 Braun A.W. Enterprise Software Delivery. – Addison–Wesley, 2012. – 291 p.
- 44 Camp O., Filipe J.B.L., Hammoudi S., Piattini M. Enterprise Information Systems V. – Kluwer, 2005. –339 p.

- 45 Cheng B.H.C., de Lemos R., Paola H.G., Magee I.J. (eds.) Software Engineering for Self-Adaptive Systems. – Springer, 2009. –270 p.
- 46 Cruz-Cunha M.M. Social, Managerial, and Organizational Dimensions of Enterprise Information Systems. – IGI Global, 2010. –604 p.
- 47 Cruz-Cunha M.M., Varajago J. (Eds.) Enterprise Information Systems Design, Implementation and Management: Organizational Applications. – Information Science Reference, 2010. – 622 p.
- 48 Эккель Б. Философия Java. Библиотека программиста. 4-е изд.; пер. с англ. – СПб.: Питер, 2009. – 640 с.: ил.
- 49 Duggan D. Enterprise Software Architecture and Design: Entities, Services, and Resources. – Wiley, 2012. – 512 p
- 50 Evans Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison Wesley, 2003. – 560 p.

Призначення модулів програми.

Клієнтський додаток складається з наступних модулів:

Залежності системи:

- файл “build.gradle” – використовується для автоматичної збірки проектів та підключення зовнішніх бібліотек.

Ресурси системи:

- папка “values” – містить файли констант додатку;
- папка “layout” – містить layout-файли (файли розмітки вікон) додатку;
- папка “drawable” – містить файли зображень додатку;

Пакети системи:

- пакет “activity” – містить Activity – класи, що відповідають за відображення даних;
- пакет “adapter” – містить адаптери даних;
- пакет “auth” – містить класи що відповідають за автентифікацію з соціальними мережами;
- пакет “dialog” – містить класи діалогових вікон;
- пакет “fragment” – містить Fragment – класи, що відповідають за відображення даних;
- пакет “server” – містить класи, що відповідають за спілкування з сервером;
- пакет “utilities” – містить класи-утиліти;
- пакет “widget” – містить класи візуальних компонентів системи.

Текст програми

Залежності системи

Зміст файлу “build.gradle”

```
apply plugin: 'com.android.application'
android {
    //задаємо версії засобів розробки
    compileSdkVersion 23
    buildToolsVersion "23.0.3"
    defaultConfig {
        applicationId "com.example.koza4.ServerAnalyzer"
        minSdkVersion 21
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
    packagingOptions {
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/NOTICE'
        exclude 'META-INF/NOTICE.txt'
    }
}
//підключення зовнішніх бібліотек
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile project(':myapplication')
    compile 'com.android.support:design:24.0.0-beta1'
    compile 'com.android.support:cardview-v7:24.0.0-beta1'
    compile 'com.github.bumptech.glide:glide:3.6.0'
    compile 'de.hdodenhof:circleimageview:1.3.0'
    compile 'com.android.support:appcompat-v7:24.0.0-beta1'
    compile 'com.flaviofaria:kenburnsview:1.0.6'
    compile 'com.google.android.gms:play-services-auth:9.0.2'
    compile 'com.firebase:firebase-client-android:2.3.1'
    compile 'com.android.support:recyclerview-v7:24.0.0-beta1'
    compile 'com.facebook.android:facebook-android-sdk:4.6.0'
    compile 'com.viewpagerindicator:library:2.4.1'
    compile 'com.github.medyo:fancybuttons:1.5@aar'
    compile 'com.google.android.gms:play-services-identity:9.0.2'
    compile 'com.google.android.gms:play-services:9.0.2'
    compile 'com.google.android.gms:play-services-ads:9.0.2'
    compile 'com.google.android.gms:play-services-gcm:9.0.2'
}
```

Ресурси системи

Папка “values”

Вміст файлу strings.xml:

```
<resources>
<!-- Задаємо значення строковим ресурсам додатку -->
  <string name="app_name">ServerAnalyzer</string>
  <string name="splash_bg">qq</string>
  <string name="fontello_logo">&#xe800;</string>
  <string name="fontello_search">&#xe816;</string>
  <string name="fontello_x_mark">&#xe81f;</string>
  <string name="fontello_x_mark_masked">&#xe806;</string>
  <string name="fontello_microfon">&#xe80f;</string>
  <string name="fontello_user">&#xe81b;</string>
  <string name="fontello_password">&#xe823;</string>
  <string name="fontello_upload">&#xe81a;</string>
  <string name="fontello_play">&#xe813;</string>
  <string name="fontello_heart_empty">&#xe80b;</string>
  <string name="fontello_heart_full">&#xe80a;</string>
  <string name="fontello_drag_and_drop">&#xe821;</string>
  <string name="welcome">Welcome!</string>
  <string name="facebook_app_id">377505602442747</string>
  <string name="twitter_app_key">vvvcQrekKC6dlr9FNhhxxW10K</string>
  <string
name="twitter_app_secret">hSgApoxqojbfEH13yub3B8QlVs2pd7KQooyQHlrwELa6Lryrqh<
/string>
  <string name="action_settings">Settings</string>
  <string name="app_overview_1st_item_text"> </string>
  <string name="app_overview_2nd_item_text"> </string>
  <string name="app_overview_3rd_item_text"> </string>
  <string name="app_overview_4th_item_text"> </string>
  <string name="icon_user">&#xf007;</string>
  <string name="icon_add">&#xf067;</string>
  <string name="icon_readmore">&#xf178;</string>
  <string name="icon_answer">&#xf095;</string>
  <string name="icon_mute">&#xf131;</string>
  <string name="icon_like">&#xf087;</string>
  <string name="icon_share">&#xf064;</string>
  <string name="icon_follow">&#xf005;</string>
  <string name="icon_android">&#xf17b;</string>
  <string name="icon_dropbox">&#xf16b;</string>
  <string name="icon_soundcloud">&#xf1be;</string>
  <string name="icon_spotify">&#xf1bc;</string>
  <string name="icon_vine">&#xf1ca;</string>
  <string name="icon_twitter">&#xf099;</string>
  <string name="icon_tumblr">&#xf173;</string>
  <string name="icon_stackoverflow">&#xf16c;</string>
  <string name="icon_gplus">&#xf0d5;</string>
  <string name="icon_facebook">&#xf09a;</string>
  <string name="icon_entypo_facebook">&#59140;</string>
  <string name="icon_behance">&#xf1b4;</string>
  <string name="icon_dribbble">&#xf17d;</string>
  <string name="icon_send">&#xf1d8;</string>
```

```

<string name="icon_download">&#xf0ed;</string>
<string name="icon_upload">&#xf0ee;</string>
<string name="icon_config">&#xf085;</string>
<string name="icon_creditcard">&#xf09d;</string>
<string name="ghost_button_title">Ghost Button</string>
<string name="icon_envelope">✉</string>
<string name="checkin">Checkin</string>
<string name="menu_settings">Settings</string>
<string name="title_activity_maps">Map</string>
</resources>

```

Папка “layout”

Зміст файлу activity_login.xml:

```

<!-- Задаємо розмітку для вікна входу у систему -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/tools"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    xmlns:fancy="http://schemas.android.com/apk/res-auto"
    android:fillViewport="true">
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginBottom="95dp">
            <com.flaviofaria.kenburnsview.KenBurnsView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:id="@+id/icon_image1"
                android:scaleType="centerCrop"/>
            <com.flaviofaria.kenburnsview.KenBurnsView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:id="@+id/icon_image2"
                android:scaleType="centerCrop" />
        </FrameLayout>
        <android.support.v4.view.ViewPager
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/intro_view_pager"/>
        <com.viewpagerindicator.CirclePageIndicator
            android:id="@+id/indicator"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal|bottom"
            app:fillColor="@color/material_grey_800"
            app:pageColor="@color/material_grey_400"
            app:radius="3dp"
            app:strokeWidth="1dp"
            app:strokeColor="#ff838b"
            android:layout_marginBottom="84dp" />
    </RelativeLayout>

```

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#ff5f69"
    android:gravity="center"
    android:paddingBottom="30dp"
    android:paddingTop="30dp"
    android:layout_gravity="center_vertical|bottom">
<mehdi.sakout.fancybuttons.FancyButton
    android:id="@+id/btn_create_account"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="20dp"
    android:padding="10dp"
    fancy:fb_borderColor="#FFFFFF"
    fancy:fb_borderWidth="1dp"
    fancy:fb_defaultColor="#ff5f69"
    fancy:fb_focusColor="#ff838b"
    fancy:fb_radius="30dp"
    fancy:fb_text="Create an account"
    fancy:fb_textColor="#FFFFFF" >
</mehdi.sakout.fancybuttons.FancyButton>
<mehdi.sakout.fancybuttons.FancyButton
    android:id="@+id/btn_login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingBottom="10dp"
    android:paddingLeft="20dp"
    android:paddingRight="20dp"
    android:paddingTop="10dp"
    fancy:fb_borderColor="#FFFFFF"
    fancy:fb_borderWidth="1dp"
    fancy:fb_defaultColor="#ff5f69"
    fancy:fb_focusColor="#ff838b"
    fancy:fb_fontIconResource="@string/icon_user"
    fancy:fb_iconPosition="left"
    fancy:fb_radius="30dp"
    fancy:fb_text="Login"
    fancy:fb_textColor="#FFFFFF" />
</LinearLayout>
</FrameLayout>
</RelativeLayout>

```

Папка “drawable”

Зміст файлу side_naw_bar.xml

```

<?xml version="1.0" encoding="utf-8"?><!-- Малюємо панель навігації -->
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:angle="135"
        android:endColor="@color/colorPrimaryDark"
        android:startColor="@color/colorPrimary"
        android:type="linear"/>
</shape>

```

Пакети системи

Пакет “activity”

Лістинг класу LoginActivity.java:

```
/*
 * Даний клас передбачає вікно входу у систему, де користувач може
 скористатися
 * або логіном та паролем, або увійти за допомогою профілю соціальної мережі.
 * Також у наведеному класі передбачено слайдер, завдяки якому користувач
 може
 * познайомитись з коротким описом функцій додатку.
 * Якщо користувач вже був авторизований у системі, вікно входу не
 з'являється
 */

package com.example.koza4.ServerAnalyzer;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.design.widget.NavigationView;
import android.support.v4.view.PagerAdapter;
import android.support.v4.view.ViewPager;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import com.androidquery.AQuery;
import com.example.koza4.myapplication.ui.auth.core.AuthProviderType;
import
com.example.koza4.myapplication.ui.auth.core.FirebaseLoginBaseActivity;
import com.example.koza4.myapplication.ui.auth.core.FirebaseLoginError;
import com.firebase.client.AuthData;
import com.firebase.client.Firebase;
import com.firebase.client.Query;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.viewpagerindicator.CirclePageIndicator;
import com.viewpagerindicator.PageIndicator;
import mehdi.sakout.fancybuttons.FancyButton;

public class LoginActivity extends FirebaseLoginBaseActivity {
    private ViewPager viewPager;
    private PageIndicator indicator;
    private ImageView topImage1;
    private ImageView topImage2;
```



```

private int lastPage = 0;
private int[] icons;
private int[] messages;
private Firebase mRef;
public static String TAG = "FirebaseUI.chat";
private String mName;
private FirebaseAuth mAuth;
private FirebaseAuth.AuthStateListener mAuthListener;
//функція, що з'являється під час створення Activity
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    mAuth = FirebaseAuth.getInstance();
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    Firebase.setAndroidContext(this); //зв'язуємось з сервером Firebase
    setContentView(R.layout.activity_login);
    //mLoginButton = (Button) findViewById(R.id.btn_login);
    FancyButton startMessagingButton = (FancyButton)
findViewById(R.id.btn_login);
    startMessagingButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Runnable runnable = new Runnable() { //створюємо новий потік
                public void run() {
                    showFirebaseLoginPrompt();
                }
            };
            Thread thread = new Thread(runnable);
            thread.start();
        }
    });
    FancyButton CreateAccButton = (FancyButton)
findViewById(R.id.btn_create_account);
    CreateAccButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent mainIntent = new Intent(LoginActivity.this,
                ThreeActivity.class);
            LoginActivity.this.startActivity(mainIntent);
            /* Finish splash activity so user cant go back to it. */
            LoginActivity.this.finish();
        }
    });
    mRef = new Firebase("https://glaring-torch-7273.firebaseio.com");
    AQuery aq = new AQuery(this); //задаємо слайдер
    aq.id(R.id.icon_image1).image(R.drawable.app_overview_1_buy);
    icons = new int[]{
        R.drawable.app_overview_1_buy,
        R.drawable.app_overview_2_smartphone,
        R.drawable.app_overview_3_save_money,
        R.drawable.app_overview_4_coupons
    };
    messages = new int[] { //повідомлення слайдера
        R.string.app_overview_1st_item_text,

```

```

        R.string.app_overview_2nd_item_text,
        R.string.app_overview_3rd_item_text,
        R.string.app_overview_4th_item_text
    };
    viewPager = (ViewPager) findViewById(R.id.intro_view_pager);
    topImage1 = (ImageView) findViewById(R.id.icon_image1);
    topImage2 = (ImageView) findViewById(R.id.icon_image2);
    indicator = (CirclePageIndicator) findViewById(R.id.indicator);
    topImage2.setVisibility(View.GONE);
    viewPager.setAdapter(new IntroAdapter());
    viewPager.setPageMargin(0);
    viewPager.setOffscreenPageLimit(1);
    indicator.setViewPager(viewPager);
    indicator.setOnPageChangeListener(new
ViewPager.OnPageChangeListener() {
    @Override
    public void onPageScrolled(int position, float positionOffset,
int positionOffsetPixels) { //при прокручуванні сторінки
        if (lastPage != viewPager.getCurrentItem()) {
            lastPage = viewPager.getCurrentItem();
            final ImageView fadeoutImage;
            final ImageView fadeinImage;
            if (topImage1.getVisibility() == View.VISIBLE) {
                fadeoutImage = topImage1;
                fadeinImage = topImage2;
            } else {
                fadeoutImage = topImage2;
                fadeinImage = topImage1;
            }
            fadeinImage.bringToFront();
            fadeinImage.setImageResource(icons[lastPage]);
            fadeinImage.clearAnimation();
            fadeoutImage.clearAnimation();
            Animation outAnimation =
AnimationUtils.loadAnimation(LoginActivity.this, R.anim.icon_anim_fade_out);
            outAnimation.setAnimationListener(new Animation.AnimationListener() {
                @Override
                public void onAnimationStart(Animation animation) {
                }
                @Override
                public void onAnimationEnd(Animation animation) {
                    fadeoutImage.setVisibility(View.GONE);
                }
                @Override
                public void onAnimationRepeat(Animation animation) {
                }
            }); //анімація при перелистуванні
            Animation inAnimation =
AnimationUtils.loadAnimation(LoginActivity.this, R.anim.icon_anim_fade_in);
            inAnimation.setAnimationListener(new
Animation.AnimationListener() {
                @Override
                public void onAnimationStart(Animation animation) {
                    fadeinImage.setVisibility(View.VISIBLE);
                }
            }

```

```

        @Override
        public void onAnimationEnd(Animation animation) {
        }
        @Override
        public void onAnimationRepeat(Animation animation) {
        }
    });
    fadeoutImage.startAnimation(outAnimation);
    fadeinImage.startAnimation(inAnimation);
    }
}
@Override
public void onPageSelected(int position) {
}
@Override
public void onPageScrollStateChanged(int state) {
}
});
 mAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth
firebaseAuth) {
        FirebaseUser user = firebaseAuth.getCurrentUser();
        if (user != null) {
            // User is signed in
            Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
        } else {
            // User is signed out
            Log.d(TAG, "onAuthStateChanged:signed_out");
        }
    }
};
}
private class IntroAdapter extends PagerAdapter { //клас для зображення
слайдера
    @Override
    public int getCount() {
        return 4;
    }
    @Override
    public boolean isViewFromObject(View view, Object object) {
        return view.equals(object);
    }
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        View view = View.inflate(container.getContext(),
R.layout.intro_view_layout, null);
        TextView messageTextView = (TextView)
view.findViewById(R.id.message_text);
        container.addView(view, 0);
        messageTextView.setText(AndroidUtilities.replaceTags(getString(messages[posit
ion]), getApplicationContext()));
        return view;
    }
}

```

```

        @Override
        public void destroyItem(ViewGroup container, int position, Object
object) {
            ((ViewPager) container).removeView((View) object);
        }
    }
    @Override
    protected void onStart() {
        super.onStart();
        setEnabledAuthProvider(AuthProviderType.FACEBOOK);
        setEnabledAuthProvider(AuthProviderType.TWITTER);
        setEnabledAuthProvider(AuthProviderType.GOOGLE);
        setEnabledAuthProvider(AuthProviderType.PASSWORD);
        mAuth.addAuthStateListener(mAuthListener);
    }
    @Override
    public void onStop() {
        super.onStop();
        if (mAuthListener != null) {
            mAuth.removeAuthStateListener(mAuthListener);
        }
        // ...
    }
    @Override //при успішній авторизації
    public void onFirebaseLoggedIn(AuthData authData) {
        Log.i(TAG, "Logged in to " + authData.getProvider().toString());
        switch (authData.getProvider()) {
            case "password":
                mName = (String) authData.getProviderData().get("email");
                break;
            default:
                mName = (String)
authData.getProviderData().get("displayName");
                break;
        }
        Toast.makeText(getApplicationContext(), mName,
            Toast.LENGTH_LONG).show();
        invalidateOptionsMenu();
        Intent mainIntent = new Intent(LoginActivity.this,
            ThreeActivity.class);
        LoginActivity.this.startActivity(mainIntent);
        /* Finish splash activity so user cant go back to it. */
        LoginActivity.this.finish();
    }
    @Override
    public void onFirebaseLoggedOut() { //при виході з облікового запису
        Log.i(TAG, "Logged out");
        mName = "";
        invalidateOptionsMenu();
    }
    @Override //при помилці входу
    public void onFirebaseLoginProviderError(FirebaseLoginError
firebaseError) {
        Log.e(TAG, "Login provider error: " + firebaseError.toString());
        resetFirebaseLoginPrompt();
    }

```

```

    }
    @Override //при помилці входу
    public void onFirebaseLoginUserError(FirebaseLoginError firebaseError) {
        Log.e(TAG, "Login user error: " + firebaseError.toString());
        resetFirebaseLoginPrompt();
    }
    @Override
    public Firebase getFirebaseRef() {
        return mRef;
    }
}

```

Лістинг класу OverviewActivity.java:

```

package com.example.koza4.splashscreen;
import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.NavigationView;
import android.support.design.widget.Snackbar;
import android.support.design.widget.TabLayout;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.GravityCompat;
import android.support.v4.view.ViewPager;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import com.firebase.client.AuthData;
import com.firebase.client.Firebase;
import com.google.firebase.auth.FirebaseAuth;
import java.util.ArrayList;
import java.util.List;

/* Клас який зображує вікно з панеллю навігації, на цьому вікні користувач
 * може переходити від виконання запитів до списку доступних серверів, або
 * обраного списку серверу з можливостями, на навігаційній панелі він може
 курувати своїм обліковим записом
 */
public class OverviewActivity extends AppCompatActivity {
    private DrawerLayout mDrawerLayout;
    private NavigationView navigationView, navigation_view;
    private String userName;
    private Firebase ref;
    private TextView txtProfileName, txtEmail;
    private FirebaseAuth mAuth;
    private FirebaseAuth.AuthStateListener mAuthListener;

```

```

public static String TAG = "FirebaseUI.chat";
private int[] imageResId = {
    R.drawable.ic10,
    R.drawable.ic2,
    R.drawable.ic4,
    R.drawable.ic3
};
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_three);
    Firebase.setAndroidContext(this);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    ref = new Firebase("https://glaring-torch-7273.firebaseio.com");
    mAuth = FirebaseAuth.getInstance();
    final ActionBar ab = getSupportActionBar();
    ab.setHomeAsUpIndicator(R.drawable.ic_menu);
    ab.setDisplayHomeAsUpEnabled(true);
    mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    navigationView = (NavigationView) findViewById(R.id.nav_view);
    if (navigationView != null) { // задаємо навігаційну панель
        setupDrawerContent(navigationView);
    }
    navigationView.setNavigationItemSelectedListener(new
NavigationView.OnNavigationItemSelectedListener() {
        // This method will trigger on item Click of navigation menu
        @Override
        public boolean onNavigationItemSelectedListener(MenuItem menuItem) {
            //Checking if the item is in checked state or not, if not
            //make it in checked state
            if (menuItem.isChecked()) menuItem.setChecked(false);
            else menuItem.setChecked(true);
            //Closing drawer on item click
            mDrawerLayout.closeDrawers();

            //Check to see which item was being clicked and perform
            appropriate action
            switch (menuItem.getItemId()) {
                //Replacing the main content with ContentFragment Which
                is our Inbox View;
                case R.id.nav_home:
                    Toast.makeText(getApplicationContext(), "Inbox
                    Selected", Toast.LENGTH_SHORT).show();
                    ContentFragment fragment = new ContentFragment();
                    android.support.v4.app.FragmentTransaction
                    fragmentTransaction = getSupportFragmentManager().beginTransaction();
                    fragmentTransaction.replace(R.id.frame, fragment);
                    fragmentTransaction.commit();
                    return true;
                // For rest of the options we just show a toast on click
                case R.id.nav_messages:
                    Toast.makeText(getApplicationContext(), "Stared
                    Selected", Toast.LENGTH_SHORT).show();
                    return true;
            }
        }
    });
}

```

```

        case R.id.nav_friends:
            Toast.makeText(getApplicationContext(), "Send
Selected", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.nav_discussion:
            Toast.makeText(getApplicationContext(), "Drafts
Selected", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.dashboard:
            Toast.makeText(getApplicationContext(), "All Mail
Selected", Toast.LENGTH_SHORT).show();
            return true;
        case R.id.forum:
            Toast.makeText(getApplicationContext(), "Loggin out",
Toast.LENGTH_SHORT).show();// повідомлення про вихід з профілю
            ref.unauth();
            Intent mainIntent = new Intent(ThreeActivity.this,
                LoginActivity.class); // перехід на вікно
входу
            ThreeActivity.this.startActivity(mainIntent);
            ThreeActivity.this.finish();
            return true;
        default:
            Toast.makeText(getApplicationContext(), "Somethings
Wrong", Toast.LENGTH_SHORT).show();
            return true;
    }
    });
    ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);
    if (viewPager != null) {
        setupViewPager(viewPager);
    }
    FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Snackbar.make(view, "Here's a Snackbar",
Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        }
    });
    TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);
    tabLayout.setupWithViewPager(viewPager);
    tabLayout.getTabAt(0).setIcon(imageResId[0]);
    tabLayout.getTabAt(1).setIcon(imageResId[1]);
    tabLayout.getTabAt(2).setIcon(imageResId[2]);
    tabLayout.getTabAt(3).setIcon(imageResId[3]);
    View header = LayoutInflater.from(this).inflate(R.layout.nav_header,
null);
    navigationView.addView(header);
    txtProfileName = (TextView) header.findViewById(R.id.username);
    txtEmail = (TextView) header.findViewById(R.id.textView);
    ref.addAuthStateListener(new FirebaseAuthStateListener() {
        @Override

```

```

        public void onAuthStateChanged(AuthData authData) {
            if (authData != null) {
                userName =
authData.getProviderData().get("displayName").toString();
//зображення даних користувача на навігаційній панелі
                txtProfileName.setText(userName);
                txtEmail.setText("test_mail@gmail.com");
            } else {
                Toast.makeText(getApplicationContext(), "Somethings
Wrong", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
@Override
public boolean onCreateOptionsMenu(Menu menu) { //меню налаштувань
    getMenuInflater().inflate(R.menu.sample_actions, menu);
    return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            mDrawerLayout.openDrawer(GravityCompat.START);
            return true;
    }
    return super.onOptionsItemSelected(item);
}
private void setupViewPager(ViewPager viewPager) {
    Adapter adapter = new Adapter(getSupportFragmentManager());
    adapter.addFragment(new CheeseListFragment(), "Hand picked");
    adapter.addFragment(new LocationFragment(), "Map");
    adapter.addFragment(new CardContentFragment(), "Places");
    adapter.addFragment(new TileContentFragment(), "My list");
    viewPager.setAdapter(adapter);
}
private void setupDrawerContent(NavigationView navigationView) {
    navigationView.setNavigationItemSelectedListener(
        new NavigationView.OnNavigationItemSelectedListener() {
            @Override
            public boolean onOptionsItemSelected(MenuItem
menuItem) {
                menuItem.setChecked(true);
                mDrawerLayout.closeDrawers();
                return true;
            }
        }
    );
}
static class Adapter extends FragmentPagerAdapter { //зображення вкладок
переходу
    private final List<Fragment> mFragments = new ArrayList<>();
    private final List<String> mFragmentTitles = new ArrayList<>();
    public Adapter(FragmentManager fm) {
        super(fm);
    }
    public void addFragment(Fragment fragment, String title) {

```



```

        mFragments.add(fragment); // додання вкладки
        mFragmentTitles.add(title);
    }
    @Override
    public Fragment getItem(int position) {
        return mFragments.get(position);
    }
    @Override
    public int getCount() {
        return mFragments.size();
    }
    @Override
    public CharSequence getPageTitle(int position) {
        return mFragmentTitles.get(position);
    }
}
}

```

Пакет “adapter”

Лістинг класу FirebaseListAdapter.java:

```

package com.example.koza4.ServerAnalyzer.ui;
import android.app.Activity;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import com.firebase.client.DataSnapshot;
import com.firebase.client.Firebase;
import com.firebase.client.Query;
/*
 * Даний клас реалізує підтримку компоненту ListView з сервером Firebase
 */
public abstract class FirebaseListAdapter<T> extends BaseAdapter {
    private final Class<T> mModelClass;
    protected int mLayout;
    protected Activity mActivity; //містить ListView
    FirebaseArray mSnapshots;
    public FirebaseListAdapter(Activity activity, Class<T> modelClass, int
modelLayout, Query ref) {
        mModelClass = modelClass;
        mLayout = modelLayout; // представлення однієї строки в ListView
        mActivity = activity;
        mSnapshots = new FirebaseArray(ref); //змінна що представляє Firebase
        mSnapshots.setOnChangedListener(new FirebaseArray.OnChangedListener()
{
            @Override
            public void onChanged(EventType type, int index, int oldIndex) {
                notifyDataSetChanged();
            }
        });
    }
    public FirebaseListAdapter(Activity activity, Class<T> modelClass, int
modelLayout, Firebase ref) {
        this(activity, modelClass, modelLayout, (Query) ref);
    }
}

```

```

    public void cleanup() {
        // We're being destroyed, let go of our mListener and forget about
all of the mModels
        mSnapshots.cleanup();
    }
    @Override
    public int getCount() {
        return mSnapshots.getCount();
    }
    @Override
    public T getItem(int position) {
        return parseSnapshot(mSnapshots.getItem(position));
    } // Цей метод аналізує отримані дані та повертає в необхідний тип.
    protected T parseSnapshot(DataSnapshot snapshot) {
        return snapshot.getValue(mModelClass);
    }
    public Firebase getRef(int position) {
        return mSnapshots.getItem(position).getRef();
    }
    @Override
    public long getItemId(int i) {
        // http://stackoverflow.com/questions/5100071/whats-the-purpose-of-
item-ids-in-android-listview-adapter
        return mSnapshots.getItem(i).getKey().hashCode();
    }
    @Override
    public View getView(int position, View view, ViewGroup viewGroup) {
        if (view == null) {
            view = mActivity.getLayoutInflater().inflate(mLayout, viewGroup,
false);
        }
        T model = getItem(position);
        // Call out to subclass to marshall this model into the provided view
        populateView(view, model, position);
        return view;
    }
    abstract protected void populateView(View v, T model, int position);
}

```

Пакет “auth”

Лістинг класу AuthProvider.java:

```

package com.example.koza4.ServerAnalyzer.ui.auth.core;
import android.content.Context;
import android.content.Intent;
import android.util.Log;
import com.firebase.client.AuthData;
import com.firebase.client.Firebase;
import com.firebase.client.FirebaseError;
import java.util.HashMap;
import java.util.Map;
/*
 * Даний абстрактний клас передбачає вибір способу авторизації
 */
public abstract class FirebaseAuthProvider {

```

```

private static final String TAG = "FirebaseAuthProvider";
private final Context mContext;
private final AuthProviderType mAuthProviderType;
private final String mProviderName;
private final Firebase mRef;
private final TokenAuthHandler mHandler;
public abstract void logout();
public Context getContext() {
    return mContext;
}
//отримуємо спосіб авторизації
public AuthProviderType getProviderType() {
    return mAuthProviderType;
}
//отримуємо ім'я сервісу
public String getProviderName() {
    return mProviderName;
}

public Firebase getFirebaseRef() {
    return mRef;
}
public TokenAuthHandler getHandler() {
    return mHandler;
}
//повертаємо дані необхідні для авторизації
protected FirebaseAuthProvider(Context context, AuthProviderType
providerType, String providerName, Firebase ref, TokenAuthHandler handler) {
    mContext = context;
    mAuthProviderType = providerType;
    mProviderName = providerName;
    mRef = ref;
    mHandler = handler;
}
//перезавантаження функції
public void login() {
    Log.w("FirebaseAuthProvider", "Login() is not supported for provider
type " + getProviderName());
}
//перезавантаження функції
public void login(String email, String password) {
    Log.w("FirebaseAuthProvider", "Login(String email, String password)
is not supported for provider type " + getProviderName());
}
//при відправленні токена
public void onFirebaseTokenReceived(FirebaseOAuthToken token,
TokenAuthHandler handler) {
    authenticateRefWithOAuthFirebaseToken(token, handler);
}

public void onActivityResult(int requestCode, int resultCode, Intent
data) {
}

private void authenticateRefWithOAuthFirebaseToken(FirebaseOAuthToken

```

```

token, final FirebaseAuthHandler handler) {
    Firebase.AuthResultHandler authResultHandler = new
Firebase.AuthResultHandler() {
    @Override
    public void onAuthenticated(AuthData authData) {
        handler.onSuccess(authData);
    }

    @Override
    public void onAuthenticationError(FirebaseError firebaseError) {
        Log.e(TAG, "Authentication failed: " +
firebaseError.toString());
        handler.onProviderError(new
FirebaseLoginError(FirebaseResponse.PROVIDER_NOT_ENABLED, "Make sure " +
getProviderName() + " login is enabled and configured in your Firebase. (" +
firebaseError.toString() + ")"));
    }
};

    if (token.mode == FirebaseAuthToken.SIMPLE) {
        // Simple mode is used for Facebook and Google auth
        getFirebaseRef().authWithOAuthToken(token.provider, token.token,
authResultHandler);
    } else if (token.mode == FirebaseAuthToken.COMPLEX) {
        // Complex mode is used for Twitter auth
        Map<String, String> options = new HashMap<>();
        options.put("oauth_token", token.token);
        options.put("oauth_token_secret", token.secret);
        options.put("user_id", token.uid);
        getFirebaseRef().authWithOAuthToken(token.provider, options,
authResultHandler);
    }
}
}

```

Пакет “dialog”

Лістинг класу LoginDialog.java:

```

package com.example.koza4.ServerAnalyzer.ui.auth.core;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.EditText;
import com.firebase.client.AuthData;
import com.firebase.client.Firebase;
import com.example.koza4.myapplication.R;
import com.example.koza4.myapplication.ui.auth.google.GoogleAuthProvider;
import java.util.HashMap;
import java.util.Map;
/*Даний абстрактний клас передбачає з'єднання з сервером та діалогове вікно

```

```

*авторизації
*/
public class FirebaseLoginDialog extends DialogFragment {
    Map<AuthProviderType, FirebaseAuthProvider> mEnabledProvidersByType = new
    HashMap<>();
    TokenAuthHandler mHandler;
    AuthProviderType mActiveProvider;
    Firebase mRef;
    Context mContext;
    View mView;
    @Override
    public void onStop() {
        super.onStop();
        cleanUp();
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        cleanUp();
    }
    @Override
    public void onPause() {
        super.onPause();
        cleanUp();
    }
    public void cleanUp() {
        if (getGoogleAuthProvider() != null)
        getGoogleAuthProvider().cleanUp();
    }
    public void onActivityResult(int requestCode, int resultCode, Intent
    data) {
        for (FirebaseAuthProvider provider :
        mEnabledProvidersByType.values()) {
            provider.onActivityResult(requestCode, resultCode, data);
        }
    }
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        mView = inflater.inflate(R.layout.fragment_firebase_login, null);
        for (AuthProviderType providerType : AuthProviderType.values()) {
            if (mEnabledProvidersByType.keySet().contains(providerType)) {
                showLoginOption(mEnabledProvidersByType.get(providerType),
                providerType.getViewId());
            } else {
                mView.findViewById(providerType.getViewId()).setVisibility(View.GONE);
            }
        }
        if (mEnabledProvidersByType.containsKey(AuthProviderType.PASSWORD) &&
        ! (mEnabledProvidersByType.containsKey(AuthProviderType.FACEBOOK) ||
        mEnabledProvidersByType.containsKey(AuthProviderType.GOOGLE) ||
        mEnabledProvidersByType.containsKey(AuthProviderType.TWITTER))) {

```

```

        mView.findViewById(R.id.or_section).setVisibility(View.GONE);
    }
    mView.findViewById(R.id.loading_section).setVisibility(View.GONE);
    builder.setView(mView);
    this.setRetainInstance(true);
    return builder.create();
}
public FirebaseLoginDialog setRef(Firebase ref) {
    mRef = ref;
    return this;
}
public FirebaseLoginDialog setContext(Context context) {
    mContext = context;
    return this;
}
public void reset() { //перезавантаження вікна
    if (mView != null) {

mView.findViewById(R.id.login_section).setVisibility(View.VISIBLE);

mView.findViewById(R.id.loading_section).setVisibility(View.GONE);
    }
    }
    public void logout() { //при виході з акаунту
        for (FirebaseAuthProvider provider :
mEnabledProvidersByType.values()) {
            provider.logout();
        }
        mRef.unauth();
    }

public FirebaseLoginDialog setHandler(final TokenAuthHandler handler) {
    mHandler = new TokenAuthHandler() {
        @Override
        public void onSuccess(AuthData auth) {
            dismiss();
            handler.onSuccess(auth);
        }
        @Override
        public void onUserError(FirebaseLoginError err) {
            handler.onUserError(err);
        }
        @Override
        public void onProviderError(FirebaseLoginError err) {
            handler.onProviderError(err);
        }
    };
    return this;
}
public FirebaseLoginDialog setEnabledProvider(AuthProviderType provider)
{
    if (!mEnabledProvidersByType.containsKey(provider)) {
        mEnabledProvidersByType.put(provider,
provider.createProvider(mContext, mRef, mHandler));
    }
}

```

```

        return this;
    }
    private void showLoginOption(final FirebaseAuthProvider helper, int id) {
        mView.findViewById(id).setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View view) {
                if (AuthProviderType.getTypeForProvider(helper) ==
                AuthProviderType.PASSWORD) {
                    EditText emailText = (EditText) mView.findViewById(R.id.email);
                    EditText passwordText = (EditText)
                    mView.findViewById(R.id.password);
                    helper.login(emailText.getText().toString(),
                    passwordText.getText().toString());
                    passwordText.setText("");
                } else {
                    helper.login();
                }
                mActiveProvider = helper.getProviderType();

                mView.findViewById(R.id.login_section).setVisibility(View.GONE);
                mView.findViewById(R.id.loading_section).setVisibility(View.VISIBLE);
            }
        });
    }
    public GoogleAuthProvider getGoogleAuthProvider() {
        return (GoogleAuthProvider)
        mEnabledProvidersByType.get(AuthProviderType.GOOGLE);
    }
}

```

Пакет “fragment”

Лістинг класу CardContentFragment.java:

```

package com.example.koza4.splashscreen;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.Toast;
/**
 * Provides UI for the view with Cards.
 */
public class CardContentFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,

```

```

        Bundle savedInstanceState) {
RecyclerView recyclerView = (RecyclerView) inflater.inflate(
        R.layout.recycler_view, container, false);
ContentAdapter adapter = new ContentAdapter();
recyclerView.setAdapter(adapter);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new
LinearLayoutManager(getActivity()));
return recyclerView;
}
public static class ViewHolder extends RecyclerView.ViewHolder {
public ViewHolder(LayoutInflater inflater, ViewGroup parent) {
super(inflater.inflate(R.layout.item_card, parent, false));
itemView.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Context context = v.getContext();
Intent intent = new Intent(context, FourActivity.class);
context.startActivity(intent);
}
});
// Adding Snackbar to Action Button inside card
Button button = (Button)
itemView.findViewById(R.id.action_button);
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Snackbar.make(v, "Action is pressed",
Snackbar.LENGTH_LONG).show();
}
});
ImageButton favoriteImageButton =
(ImageButton)
itemView.findViewById(R.id.favorite_button);
favoriteImageButton.setOnClickListener(new View.OnClickListener()
{
@Override
public void onClick(View v) {
Snackbar.make(v, "Added to Favorite",
Snackbar.LENGTH_LONG).show();
}
});
ImageButton shareImageButton = (ImageButton)
itemView.findViewById(R.id.share_button);
shareImageButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Snackbar.make(v, "Share article",
Snackbar.LENGTH_LONG).show();
}
});
}
}
/**

```



```

    * Adapter to display recycler view.
    */
    public static class ContentAdapter extends
RecyclerView.Adapter<ViewHolder> {
        // Set numbers of Card in RecyclerView.
        private static final int LENGTH = 18;
        @Override
        public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)
{
            return new ViewHolder(LayoutInflater.from(parent.getContext()),
parent);
        }
        @Override
        public void onBindViewHolder(ViewHolder holder, int position) {
            // no-op
        }
        @Override
        public int getItemCount() {
            return LENGTH;
        }
    }
}

```

Пакет “server”

Лістинг класу AuthProviderType.java:

```

package com.example.koza4.myapplication.ui.auth.core;
import android.content.Context;
import com.firebase.client.Firebase;
import com.example.koza4.myapplication.R;
import java.lang.reflect.InvocationTargetException;
/**
 * Вибір методу входу до системи, з'єднання з соц. мережами
 */

public enum AuthProviderType {
    GOOGLE("google", "google.GoogleAuthProvider", R.id.google_button),
    FACEBOOK("facebook", "facebook.FacebookAuthProvider",
R.id.facebook_button),
    TWITTER("twitter", "twitter.TwitterAuthProvider", R.id.twitter_button),
    PASSWORD("password", "password.PasswordAuthProvider",
R.id.password_section);
    private final static String AUTH_PACKAGE =
"com.example.koza4.myapplication.ui.auth.";
    private final String mName;
    private final String mProviderName;
    private final int mViewId;
    AuthProviderType(String name, String providerName, int viewId) {
        this.mName = name;
        this.mProviderName = providerName;
        this.mViewId = viewId;
    }
    public String getName() {
        return mName;
    }
}

```

```

    }
    public int getViewId() {
        return mViewId;
    }
    public FirebaseAuthProvider createProvider(Context context, Firebase ref,
    TokenAuthHandler handler) {
        try {
            Class<? extends FirebaseAuthProvider> clazz = (Class<? extends
    FirebaseAuthProvider>) Class.forName(AUTH_PACKAGE + mProviderName);
            return clazz.getConstructor(Context.class,
    AuthProviderType.class, String.class, Firebase.class,
    TokenAuthHandler.class).newInstance(context, this, this.getName(), ref,
    handler);
        } catch (NoSuchMethodException e) {
            throw new RuntimeException(e);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        } catch (InstantiationException e) {
            throw new RuntimeException(e);
        } catch (InvocationTargetException e) {
            throw new RuntimeException(e);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
    public static AuthProviderType getTypeForProvider(FirebaseAuthProvider
    provider) {
        for (AuthProviderType type : AuthProviderType.values()) {
            if (provider.getProviderName() == type.getName()) {
                return type;
            }
        }
        throw new IllegalArgumentException("The provider you specified is not
    of a known type");
    }
}

```

Пакет “utilites”

Лістинг класу AndroidUtilities.java:

```

/*
 * Інтерфейс для обробки та стилізації тексту
 */

package com.example.koza4.ServerAnalyzer;
import android.content.Context;
import android.graphics.Color;
import android.text.Spannable;
import android.text.SpannableStringBuilder;
import android.text.Spanned;
import android.text.style.ForegroundColorSpan;
import java.util.ArrayList;

```

```

public class AndroidUtilities {
    public static float density = 1;
    public static int dp(float value, Context c) {
        density = c.getResources().getDisplayMetrics().density;
        return (int) Math.ceil(density * value);
    }
    public static Spannable replaceTags(String str, Context context) {
        try {
            int start = -1;
            int startColor = -1;
            int end = -1;
            StringBuilder stringBuilder = new StringBuilder(str);
            while ((start = stringBuilder.indexOf("<br>")) != -1) {
                stringBuilder.replace(start, start + 4, "\n");
            }
            while ((start = stringBuilder.indexOf("<br/>")) != -1) {
                stringBuilder.replace(start, start + 5, "\n");
            }
            ArrayList<Integer> bolds = new ArrayList<>();
            ArrayList<Integer> colors = new ArrayList<>();
            while ((start = stringBuilder.indexOf("<b>")) != -1 ||
(startColor = stringBuilder.indexOf("<c>")) != -1) {
                if (start != -1) {
                    stringBuilder.replace(start, start + 3, "");
                    end = stringBuilder.indexOf("</b>");
                    stringBuilder.replace(end, end + 4, "");
                    bolds.add(start);
                    bolds.add(end);
                } else if (startColor != -1) {
                    stringBuilder.replace(startColor, startColor + 2, "");
                    end = stringBuilder.indexOf(">", startColor);
                    int color =
Color.parseColor(stringBuilder.substring(startColor, end));
                    stringBuilder.replace(startColor, end + 1, "");
                    end = stringBuilder.indexOf("</c>");
                    stringBuilder.replace(end, end + 4, "");
                    colors.add(startColor);
                    colors.add(end);
                    colors.add(color);
                }
            }
            SpannableStringBuilder spannableStringBuilder = new
SpannableStringBuilder(stringBuilder);
            for (int a = 0; a < colors.size() / 3; a++) {
                spannableStringBuilder.setSpan(new
ForegroundColorSpan(colors.get(a * 3 + 2)), colors.get(a * 3), colors.get(a *
3 + 1), Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            }
            return spannableStringBuilder;
        } catch (Exception e) {
        }
        return new SpannableStringBuilder(str);
    }
}

```

Пакет “widget”

Лістинг класу FontelloTextView.java:

```
package com.example.koza4.splashscreen;
import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.TextView;
//Віджет для зображення тексту у компоненті TextView
public class FontelloTextView extends TextView {
    private static Typeface sFontello;
    public FontelloTextView(Context context) {
        super(context);
        if (isInEditMode()) return; //Won't work in Eclipse graphical layout
        setTypeface();
    }

    public FontelloTextView(Context context, AttributeSet attrs) {
        super(context, attrs);
        if (isInEditMode()) return;
        setTypeface();
    }

    public FontelloTextView(Context context, AttributeSet attrs, int defStyle)
    {
        super(context, attrs, defStyle);
        if (isInEditMode()) return;
        setTypeface();
    }

    private void setTypeface() {
        if (sFontello == null) {
            sFontello = Typeface.createFromAsset(getContext().getAssets(),
"fonts/Fontello.ttf");
        }
        setTypeface(sFontello);
    }
}
```

Відгук

на дипломну роботу магістра на тему:

«Дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів у мобільному додатку»

студента групи 121м-16-1 Козакова Андрія Олександровича

1. Мета дипломного проекту – проаналізувати ефективність обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів, їх зіставлення, виділення переваг та недоліків, опису відмінностей з роботою над великими об'ємами даних.

2. Актуальність роботи зумовлена наявністю значних відмінностей у традиційному підході до проектування мобільних додатків з використанням хмарних технологій та виділених серверів, такі як різні затрати часу, коштів та функцій.

3. Тема дипломної роботи безпосередньо пов'язана з об'єктом діяльності магістра спеціальності 121 «Інженерія програмного забезпечення» галузі знань 12 «Інформаційні технології» – створення, дослідження і реалізація моделей і програмних засобів.

4. Наукова новизна полягає в реалізації та використанні власної системи аналізу даних серверних технологій в мобільних додатках, та розробки мобільного додатка для тестування описаної методики.

5. Оригінальність технічних рішень при розробці програмного засобу полягає в використанні зв'язки таких інструментальних засобів як Java, Android, Cloud services.

6. Практична цінність полягає в розробці системи аналізу отриманих даних на прикладі різних серверних технологій, та побудові висновків з використанням діаграм на мобільному пристрої.

7. Оформлення дипломної роботи магістра виконано на сучасному рівні і відповідає вимогам, що пред'являються до робіт даної кваліфікації. Ступінь самостійності виконання досить висока.

8. Дипломна робота магістра в цілому заслуговує оцінки «відмінно», а сам автор - присвоєння кваліфікації «інженер-програміст».

Керівник дипломного проекту

д.т.н., проф. к

В.М. Куваєв

Рецензія

на дипломну роботу магістра на тему:

«Android додаток для статистичного аналізу та дослідження ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарного серверів»

студента групи 121м-16-1 Козакова Андрія Олександровича

Дипломна робота присвячена дослідженню ефективності обробки великих об'ємів даних на прикладі використання виділеного та хмарних серверів в мобільних додатках.

Актуальність роботи зумовлена наявністю значних відмінностей у традиційному підході до проектування мобільних додатків з використанням хмарних технологій та виділених серверів, такі як різні затрати часу, коштів та функцій.

В роботі виконано аналіз існуючих серверних технологій для розробки мобільних додатків, виявлення їх переваг та недоліків. Спроектовано та розроблено мобільний додаток для операційної системи Android, для надання та зображення отриманих результатів.

В результаті дослідження встановлено певні недоліки та переваги використання технологій хмарного та виділеного серверів, які описані в роботі.

Беручи до уваги вище викладене, можна зробити висновок, що дана робота цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт рівня магістра.

З огляду на наукову новизну і ступінь опрацювання компонентів даної роботи, в цілому автор заслуговує оцінки «відмінно», а також присвоєння кваліфікації 2132.2 «Інженер програміст».

Рецензент

Підпис