Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Комп'ютерних інтелектуальних технологій та систем

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

Обробка великих обсягів даних при застосуванні хмарного та виділеного серверів для мобільного застосунку
(тема)

Виконав:

студент 2 курсу, групи КІТм-21-2
Пащенко Георгій Ігорович

Спеціальність 123 Комп'ютерна інженерія

Тип програми освітньо–професійна

Освітня програма Комп'ютерні інтелектуальні технології

Керівник проф. Безсонов О.О

Допускається до захисту _____

Зав. кафедри _____ О.Г. Руденко
(підпис)

2022 р

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерної інженерії та управління
Кафедра	Комп'ютерних інтелектуальних технологій та систем
Рівень вищої освіти	другий (магістерський)
Спеціальність	123 Комп'ютерна інженерія
Тип програми	освітньо–професійна
Освітня програма	Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пащенку Георгію Ігоровичу
(прізвище, ім'я, по батькові)

1.Тема роботи Обробка великих обсягів даних при застосуванні хмарного та виділеного серверів для мобільного застосунку

затверджена наказом по університету від 07 листопада 2022р. № 1455Ст

2. Термін подання студентом роботи до екзаменаційної комісії

3.Вхідні дані до роботи Тестові дані з використання хмарних та фізичних серверів Мультисерверні додатки

4. Перелік питань, що потрібно опрацювати в роботі

Аналіз предметної області.

Функціональна схема додатку

Програмне забезпечення та мобільний додаток

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Презентація 14 слайдів

скриншоти застосунку

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Збір вимог до програмного забеспечення	08.11 – 13.11	виконано
2	Огляд сучасної літератури за напрямком	13.11 – 20.11	Виконано
	магістерської роботи		
3	Зовнішнє та логічне проектування	21.11 – 23.11	Виконано
4	Огляд та вибір комплектуючих майбутньої	24.11 – 26.11	Виконано
	системи		
5	Розробка програмного забезпечення та	26.11 – 30.12	Виконано
	мобільного додатку		
6	Оформлення пояснювальної записки	01.12 – 07.12	Виконано
7	Оформлення графічного матеріалу	08.12 – 10.12	виконано

Дата видачі завдання 07 листопада 2022 р.

Студент _____
(підпис)

_____ проф. Безсонов О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної магістерської роботи: 145 с., 30 рис., 18 табл., 4 додатки, 42 джерела.

Ключові слова: ANDROID ДОДАТОК, BAAS, JAVA, ANDROID STUDIO, BACKENDLESS, FIREBASE, SQL, ХМАРНИЙ СЕРВІС, MATERIAL DESIGN

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

Метою роботи є розробка програмного засобу Android додатоку 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 технології в мобільних додатках, та розробки мобільного додатка для тестування описаної методики.

Практична цінність полягає в розробці системи аналізу отриманих даних на прикладі різних серверних технологіях, 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.

ABSTRACT

Explanatory note: 145 pp., 30 figures, 18 tables, 4 appendices, 42 sources. Object of research: technology of cloud implementation of the backend, development of a system for analyzing data from the received results of performing operations of requests to the server.

Keywords: ANDROID APP, BAAS, JAVA, ANDROID STUDIO, BACKENDLESS, FIREBASE, SQL, CLOUD SERVICE, MATERIAL DESIGN

The purpose of the work is the development of a software tool Android application for statistical analysis and research of the efficiency of processing large volumes of data on the example of the use of dedicated and cloud servers, which is designed to evaluate the use of cloud technologies and dedicated servers, their comparison, highlighting advantages and disadvantages, describing the differences with working on large volumes of data.

Research methods. Scientific achievements in the fields of mobile application and software development were used to solve the task.

The scientific novelty consists in the implementation and use of a proprietary data analysis system of server technology in mobile applications, and the development of a mobile application for testing the described methodology.

The practical value lies in the development of a system for analyzing the received data using the example of various server technologies, and drawing conclusions using diagrams on a mobile device.

Field of application. The developed software application can be used to solve a wide range of tasks, in particular, to create software, design information systems. Value of work and conclusions.

The improved technique allows designing information systems with a significant reduction of both material costs and time, which is confirmed by the developed software product in this master's thesis.

Forecasts regarding the development of 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 the "Economics" section, the calculations of the labor intensity of software development, the costs of creating software and the duration of its development, as well as conducting marketing research of the sales market of the created software product, were carried out.

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерної інженерії та управління
Кафедра Комп’ютерних інтелектуальних технологій та систем

АНОТАЦІЯ

КВАЛІФІКАЦІЙНОЇ РОБОТИ

рівень вищої освіти другий (магістерський)

Обробка великих обсягів даних при застосуванні хмарного та виділеного серверів для мобільного застосунку

Виконав:
студент 2 курсу, групи:КІТм-21-2
Пащенко Г.І.
Спеціальність 123 Комп’ютерна інженерія
Тип програми освітньо–професійна
Освітня програма Комп’ютерні інтелектуальні технології
Керівник проф. Безсонов О.О.

2022р.

АНОТАЦІЯ

Пащенко Г.І. Обробка великих обсягів даних при застосуванні хмарного та виділеного серверів для мобільного застосунку – Магістерська кваліфікаційна робота.

Актуальність теми дослідження. В наше життя все дедалі частіше втручаються різні технічні пристрої, завдяки яким людина спілкується, працює, відпочіває**.** Для роботи цих пристроів потрібне щось, що дає змогу їх контролювати. Створювати такі додатки – це робота програмістів. Для комфортноы роботи з пристроэм або сайтом програмісти створюють мобільні додатки, аби користуватись благами цивілізації з мобільного пристрою. 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 різноманітнимисервісами.

В магістерській роботі досліджено проблеми сучасної розробки додатків та створено невеликий додаток для аналізу великих даних.

Для отримання необхідних результатів були сформульовані наступні задачі:

- аналіз існуючих серверних технологій для розробки мобільних додатків, виявлення їх переваг та недоліків;
- 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.

У першому розділі розібрано вимоги до програмного забеспечення, які визначають розробку додатку, проаналізовано область застосування та існуючі рішення, визначено та описано підібрані технології.

У другому розділі зазначено логічну частину додатку та сформульовано чітке завдання з розробки додатку.

У третьому розділі додатково описано технології, що було застосовано для розробки мобільного додатку та описан його функціонал. В кінці аналізу застосованих технологій наведено висновки щодо доречності тих чи інших технологій.

У четвертому розділі наведено економічну частину розробки з витраченими людино–годинами на створення та розписано економічні показники розробленого додатку.

Ключові слова: ANDROID ДОДАТОК, BAAS, JAVA, ANDROID STUDIO, BACKENDLESS, FIREBASE, SQL, ХМАРНИЙ СЕРВІС, MATERIAL DESIGN

ПЕРЕЛІК ПУБЛІКАЦІЙ КЕРІВНИКА ТА СПІВРОБІТНИКІВ КАФЕДРИ, ВИКОРИСТАНИХ В РОБОТІ:

1. Rudenko O. et al. Developing a Multi-Step Recurrent Algorithm to Maximize the Criteria of Correntropy. *Eastern-European Journal of Enterprise*

Technologies. Vol. 1. No. 4. 2021. 109.

2. Rudenko O. et al. Robust identification of non-stationary objects with nongaussian interference. *Eastern-european Journal of enterprise Technologies.* Vol. 5. No. 4. 2019. 44-52.

ЗМІСТ

АНОТАЦІЯ 11
ЗМІСТ 14
ПЕРЕЛІК СКОРОЧЕНЬ 16
ВСТУП 17
1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19
1.1 Призначення та область застосування 19
1.2 Постановка задачі 20
1.3 Аналіз предметної області 21
1.3.1 Опис проблеми 21
1.3.2 Актуальність дослідження 21
1.3.3 Аналіз останніх досліджень і публікацій 22
1.3.4 Мета 22
1.4 Огляд літературних та інтернет джерел 22
1.4.1 Технологія MBaa 23
1.4.2 Хмарний сервіс Firebase 24
1.4.3 Інтерфейс Google Maps Android API 26
1.4.4 Інтерфейс Google Places Android API 27
1.4.5 Використання бази даних на мобільному пристрої 27
1.4.6 Використання Material Design 28
1.5 Виклад основного матеріалу 30
1.6 Висновки до першого розділу 31
2 ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ 32
2.1 Зовнішнє проектування 32
2.1.1 Опис функціональних характеристик 32
2.1.2 Вхідні дані 33
2.1.3 Вихідні дані 34
2.2 Формалізація задачі 36
2.3 Проектування бази даних 37
2.4 Специфікація вимог до бази даних 37
2.5 Визначення типів сутностей та зв’язків між ними 38
2.6 Логічне проектування бази даних 47
2.7 Схема бази даних 49
3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКА 51
3.1 Вибір мови програмування 51
3.1.1 Мова програмування «Java» 51
3.1.2 Опис середовища розробки Android Studio 54

3.1.3 Опис СКБД MS SQL Server 55
3.2 Об’єктно–орієнтоване проектування 56
3.3 Розробка інтерфейсу 63
3.4 Вибір стратегії тестування 70
3.5 Тестування функціональності 71
3.5.1 Тест користувацького інтерфейсу 73
3.5.2 Тестування методами чорної шухляди 74
3.5.3 Метод еквівалентних розбитків 79
3.5.4 Тестування методом граничних умов 83
3.5.5 Тестування методом припущення помилки 86
3.6 Висновки до третього розділу 87
4. ЕКОНОМІЧНИЙ РОЗДІЛ 89
4.1 Розрахунок трудомісткості і вартості розробки програмного продукта . 89
4.2 Витрати на створення програмного забезпечення 92
4.3 Маркетингові дослідження ринку збуту розробленого програмного продукту 93
4.4 Оцінка економічної ефективності впровадження програмного забезпечення 96
4.5 Коефіцієнти економічної ефективності 99
4.6 Висновки до четвертого розділу 100
ВИСНОВКИ 101
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 103
ДОДАТОК А. Текст програми клієнтського додатку ..... **Ошибка! Закладка не определена.**
ДОДАТОК Б. Презентація……..136

ПЕРЕЛІК СКОРОЧЕНЬ

ПК – персональний компьютер;

ОС – операційна система;

ПЗ – програмне забеспечення;

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.

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 технологіях, та побудові висновків з використанням діаграм на мобільному пристрої.

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 різнимисерверними 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.

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.

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

Спроектувати 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 переглядуданих 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. Було проаналізовано роботу керівника диплому та внесено правки до алгоритму порівня серверів.

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.

1.4 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 геоданних;
– використання нового дизайну Material.

Під час розробки додатків Android використовується Java – одна знайбільш поширених мов програмування. Використання Java стало логічним вибором для платформи Android, тому що це потужна, вільна і відкрита мова програмування, 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 та надає розробникам доступ до потужних бібліотек класів, що прискорюють розробку програм. Програмування графічного інтерфейсу користувача є керованою подією.

Далі детально наведений опис використаних засобів розробки.

1.4.1 Технологія MBaa

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. 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 ігроші.

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 [3].

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

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 рисунку 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.

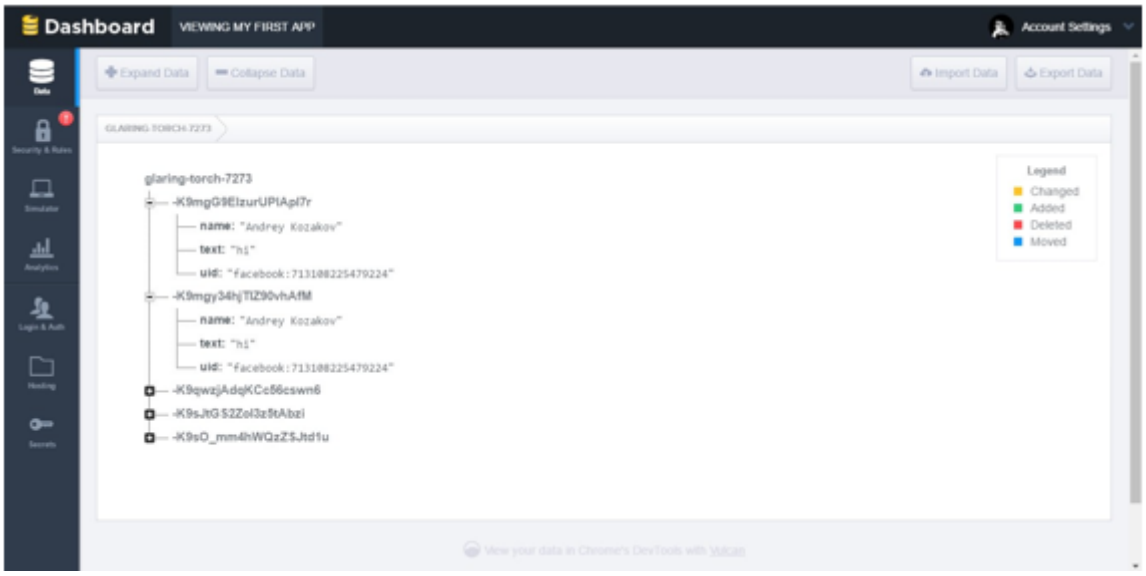


Рисунок 1.1 – Панель управління додатком

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

За допомогою Google Maps Android API можна додавати в свій додаток карти на основі даних Google Карт. Цей API–інтерфейс автоматично управляєдоступом 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

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 помірі введення тексту.

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

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, потрібно 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 дляможливості автономної роботи.

1.4.6 Використання Material Design

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.

Приклад використання Material design у проекті наведено далі у вигляді 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: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 рисунку 1.2.

Рисунок 1.2 – Елементи інтерфейсу Material

1.5 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, налаштовування 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 IT–інфраструктури.

1.6 Висновки до першого розділу

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.

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 виділених сервер;
- авторизація за допомогою профілю соціальних мереж через хмарнісервіси;
- реєстрація за допомогою електронної адреси та паролю через хмарнісервіси 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 знаходитьсяна 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

Наведемо 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 змоделює з отриманих даних після

запиту;

- вибір типів графіків та їх варіацій, що змоделює 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
Запит на реєстрацію користувача усистемі	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

Продовження таблиці 2.1

Назва вхідного повідомлення	Періодичність надходження	Вихідне повідомлення
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

Продовження таблиці 2.1

Назва вхідного повідомлення	Періодичність надходження	Вихідне повідомлення
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

2.2 Формалізація задачі

Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми варіантів використання, що зображено на рисунку 2.1.

Рисунок 2.1 – Варіанти використання системи

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.

2.3 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.

2.4 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.

БД програми повинна зберігати тестову інформацію для роботі 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.

2.5 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 (табл. 2.2).

Таблиця 2.2 – Опис типів сутностей

Ім’я типу сутності	Містить наступні дані
Заклад	назва, адреса, телефон та ін.
Користувач	ПІБ, телефон, список друзів
Замовлення	назва страви, одиниця виміру, ціна, дата, та ін.
Меню закладу	Меню страв, що пропонує певний заклад, а саме: назву страви, коротку інформацію про страву, одиницю виміру, ціну
Розділи	тип закладу харчування та тип кухні закладу

Продовження таблиці 2.2

Ім’я типу сутності	Містить наступні дані
Резервування	номер місця, дату, ім’я на яке було зарезервовано місце
Розташування	ім’я закладу, координати x, координати y
Працівники	ПІБ, телефон, заклад, в якому працює
Додаток до замовлення	страва, кількість порцій, замовлення
Друзі	друзі користувача

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

Таблиця 2.3 – Опис атрибутів сутностей.

Ім’я сутності	Атрибути	Опис атрибута	Тип	Обмеження
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
Користувач	*КористувачПК	Первинний ключ	int	NOT NULL
	Прізвище	Прізвище користувача	nvarchar(150)	До 150 символів

Продовження таблиці 2.3

Ім’я сутності	Атрибути	Опис атрибута	Тип	Обмеження
Користувач	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
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

Продовження таблиці 2.3

Ім’я сутності	Атрибути	Опис атрибута	Тип	Обмеження
Замовлення	* Замовлення ПК	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
	Меню_закладу ВК	Зовнішній ключ	int	NOT NULL
	Користувач ВК	Зовнішній ключ	int	NOT NULL
Меню закладу	*Меню_закладу ПК	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

Продовження таблиці 2.3

Ім’я сутності	Атрибути	Опис атрибута	Тип	Обмеження
Розділи	* Розділ ПК	Первинний ключ	int	NOT NULL
	Назва_категорії	Назва категорії	nvarchar(60)	До 60 симв.
	Заклад ВК	Зовнішній ключ	int	NOT NULL
Резервування	* Резервування ПК	Первинний ключ	int	NOT NULL
	Дата	Дата резервування	datetime	–
	Номер_резервування	Номер резервування	int	NOT NULL
	Користувач ВК	Зовнішній ключ	int	NOT NULL
	Заклад ВК	Зовнішній ключ	int	NOT NULL
Розташування	* РозташуванняПК	Первинний ключ	int	NOT NULL
	Назва	Назва місця накарті	nvarchar(60)	До 60 символів, NOT NULL
	Коор_Х	Координати Х	float	NOT NULL
	Коор_У	Координати У	float	NOT NULL
	Заклад ВК	Зовнішній кл.	int	NOT NULL

Продовження таблиці 2.3

Ім’я сутності	Атрибути	Опис атрибута	Тип	Обмеження
Додаток до замовлення	*Додаток ПК	Первинний ключ	int	NOT NULL
	Кількість порцій	Кількість порцій у замовленні	int	NOT NULL
	Замовлення ВК	Зовнішній ключ	int	NOT NULL
	Меню_закладу ВК	Зовнішній ключ	int	NOT NULL
	Страва	Назва страви	nvarchar(60)	До 60 симв. NOT NULL
Працівники	*Працівник ПК	Первинний ключ	int	NOT NULL
	Прізвище	Прізвище працівника	nvarchar(150)	До 150 символів,
	Ім’я	Ім’я працівника	nvarchar(60)	До 60 символів,
	По–батькові	По–батькові працівника	nvarchar(60)	До 60 символів
	Телефон	Телефон працівника	nvarchar(60)	До 25 символів
	Адреса	Адреса проживання	nvarchar(60)	До 300 символів
	Заклад ВК	Зовнішній ключ	int	NOT NULL
	Посада	Посада працівника	nvarchar(60)	До 60 символів
	Ставка	Зарплата працівника	float	Більше за 0 NOT NULL

Приведемо опис зав’язків між сутностями:

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

– зв’язок «Заклад – Розташування». Кожен заклад має місце розташування на карті, яке складається з координат осі. Зв’язок 1 до N.

– зв’язок «Користувач – Друзі». Кожен користувач має декілька друзів.

– кожен друг може бути другом іншому користувачу. Зв’язок 1 до N.

– зв’язок «Замовлення – Меню». Кожне замовлення складається зі страми, яке знаходиться в меню. Зв’язок 1 до N.

– зв’язок «Замовлення – Користувач». Користувач може складати одне замовлення. Зв’язок 1 до N.

– зв’язок «Заклад – Меню». Кожен заклад має власний список меню.

– зв’язок 1 до N.

– зв’язок «Резервування – Заклад». Кожне резервування робиться за певним закладом. Зв’язок 1 до N.

– зв’язок «Резервування – Користувач». Кожне резервування робиться за певним користувачем. Зв’язок 1 до N.

– зв’язок «Додаток до замовлення – Замовлення». Кожне замовлення має один додаток. Зв’язок 1 до N.

– зв’язок «Додаток до замовлення – Меню закладу». Додаток до замовлення складається з інформації про страви з сутності «Меню закладу». Зв’язок 1 до N.

– 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 влогічну.

Дані про кожен зв’язок між сутностями занесемо в табл. 2.4 та зобразимо

концептуальну схему на рисунку 2.2.

Таблиця 2.4 – 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

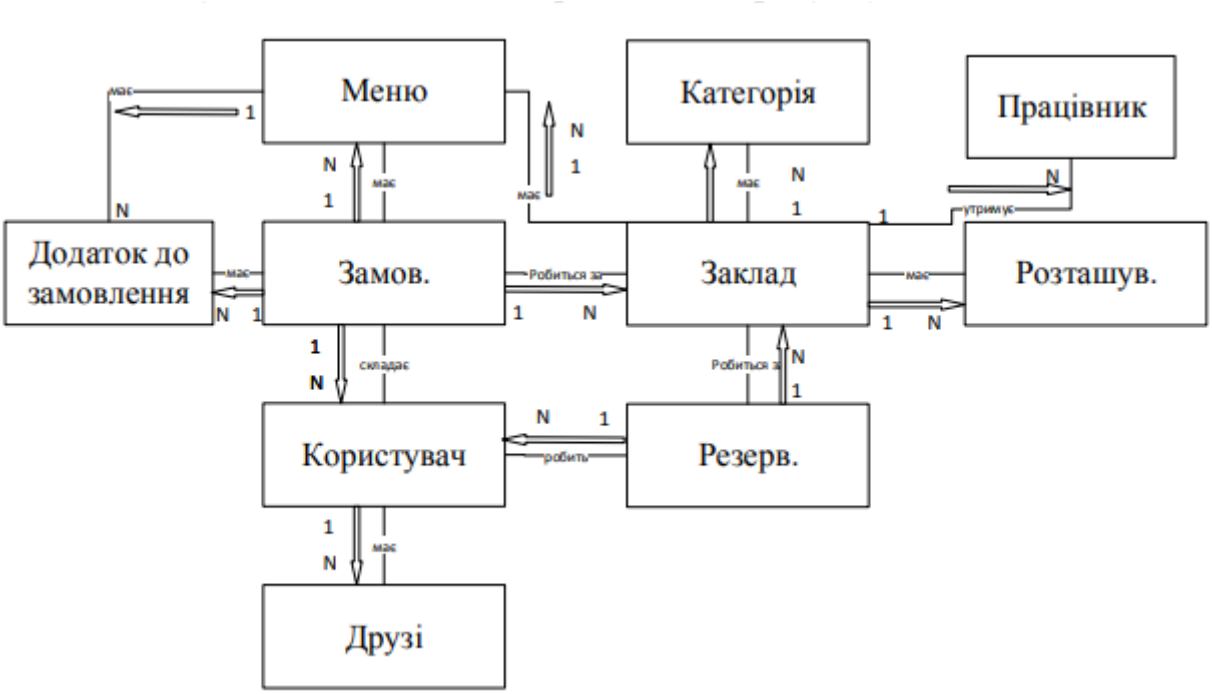


Рисунок 2.2 – Концептуальна схема моделі

2.6 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:

- 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;
- 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 підлеглій сутності копію свого первинного ключа для використання його в якості зовнішнього ключа;
- 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);
- перевірки відповідності відносин вимогам користувацьких транзакцій;
- визначення вимог підтримки цілісності даних; обов’язковість даних, обмеження для доменів атрибутів, цілісність сутностей, посилальна цілісність, користувальницькі 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.

При логічному проектуванні бази даних «Заклади харчування» були виконані наступні дії:

- виключені особливості, які не сумісні з реляційною моделлю даних,а саме – зв’язки «Багато до багатьох».
- відображена концептуальна схема на логічній – визначено набори відношень згідно логічної моделі, які приведено у табл. 2.2.
- нормалізовано відношення БД до 3 нормальної форми. г) уточнено первинні і зовнішні ключі.
- визначено вимоги підтримки цілісності даних.
- побудована логічна схема моделі – визначені набори відношень згідно логічної моделі, які приведено у табл. 2.4. Приведено повний опис сутностей (табл. 2.3). Результати розробки приведено на рисунку 2.3.

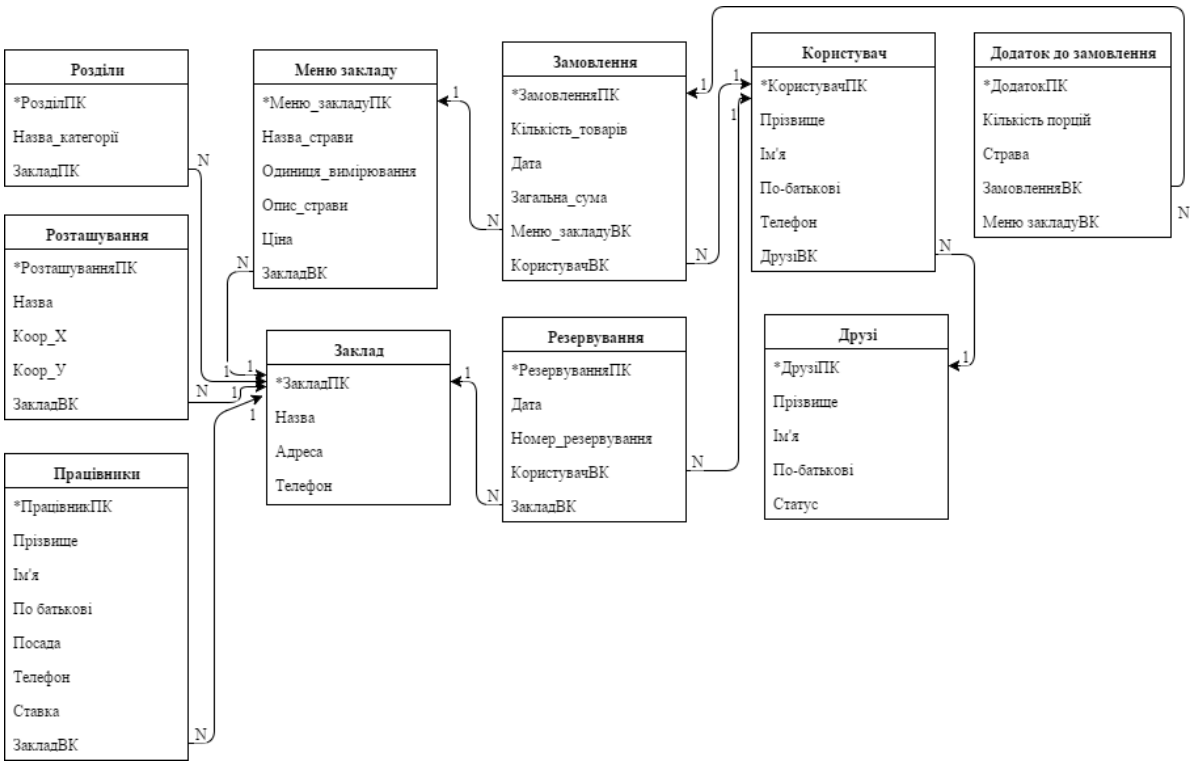


Рисунок 2.3 – Попередня логічна схема бази даних «Заклади харчування»

2.7 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 міжполями і таблицями.

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

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

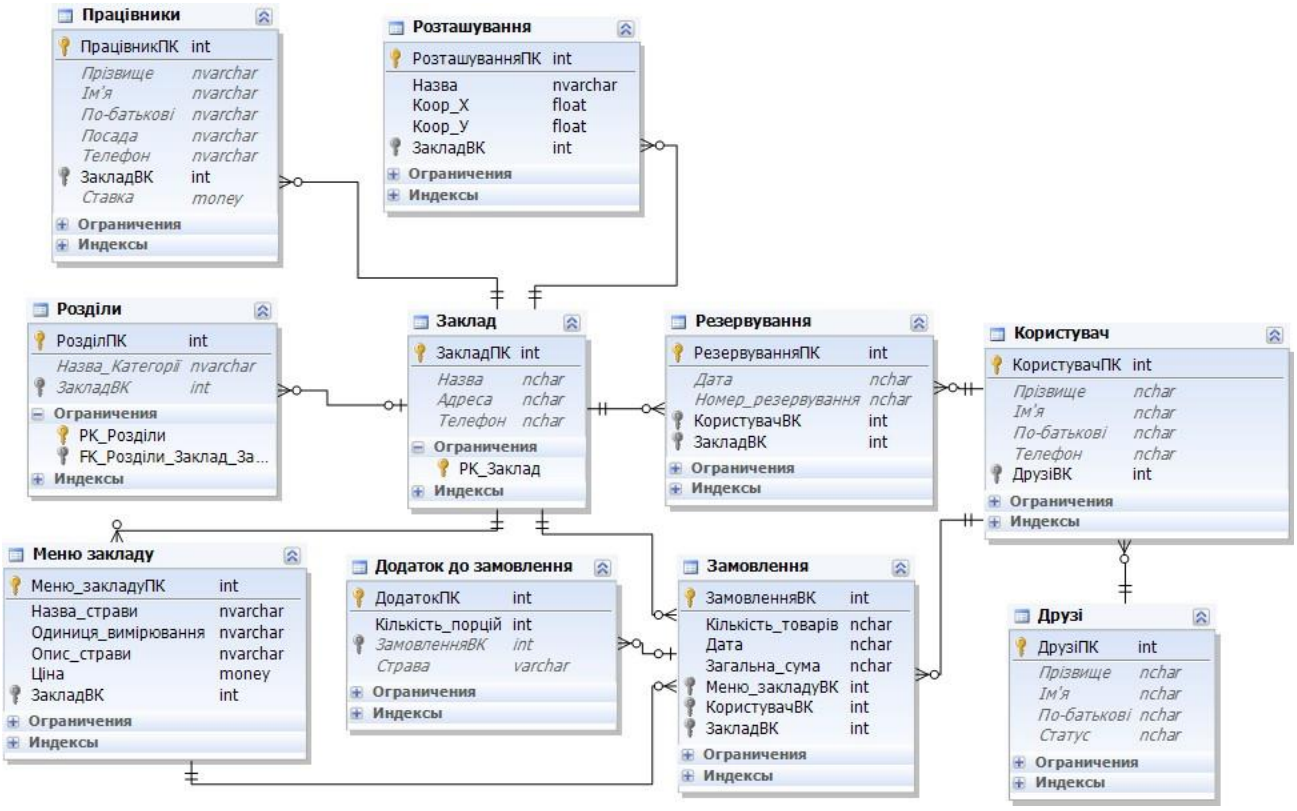


Рисунок 2.4 – Структура таблиць бази даних

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКА

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 Adroid Studio 2.0. Середовище є безкоштовним та має широкі можливості для розробки, рефакторінгу, відлагодження та профілювання мобільних додатків.Мобільний додаток підтримує з’єднання зсервером, на якому розміщена база даних з необхідною інформацією. Базу даних для серверу було спроектовано та розроблено в MS SQL Server.

3.1.1 Мова програмування «Java»

Під час розробки 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 [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, наскількице 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 ефективної 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 мережевих 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 байт–код на будь–якій машині, на якій встановлено інтерпретатор та система виконання. На інтерпритованій платформі фаза збірки програми є простою і покрокової, 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 під час розробки на Android.

3.1.2 Опис середовища розробки Android Studio

Для розробки даного проекту було обрано середовище програмної розробки Android Studio.Адаптоване для виконання типових завдань, що вирішуються в процесі розробки додатків дляплатформи 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 длярізних 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.

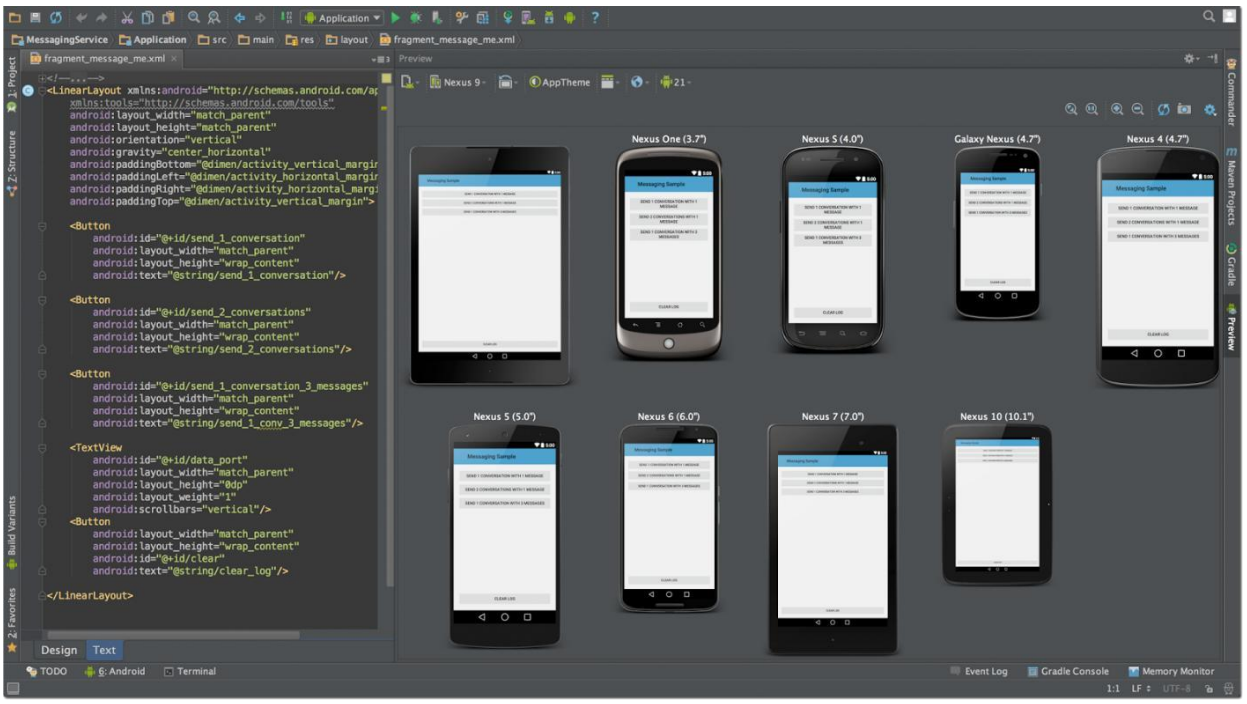


Рисунок 3.1 – Середовище розробки Android Studio

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 користувальних 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, розташованим 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–орієнтовного проектування.

Підставами до вибору даної парадигми є:

– швидкість модифікації та розширення програмного коду;

– наявність якісного середовища проектування;

– широкі можливості з повторного використання коду;

– зручність під час розробки архітектури системи;

– наявність загальноприйнятих стратегій щодо вирішення типовихпроблем.

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» зданими про поточного користувача, на рисунку 3.2.

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

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

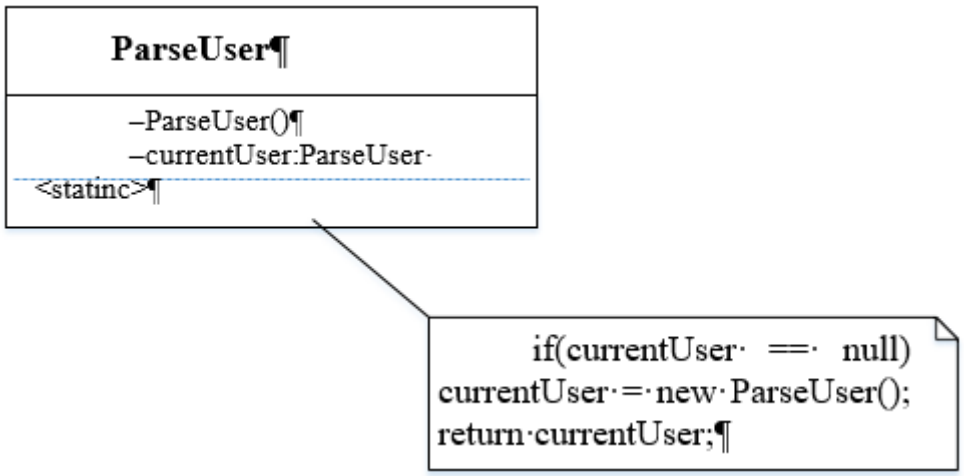


Рисунок 3.2 – Клас ParseUser

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 операції використовувався патерн Builder. Діаграма класів з прикладом використання такого патерну приведена на рисунку 3.3.

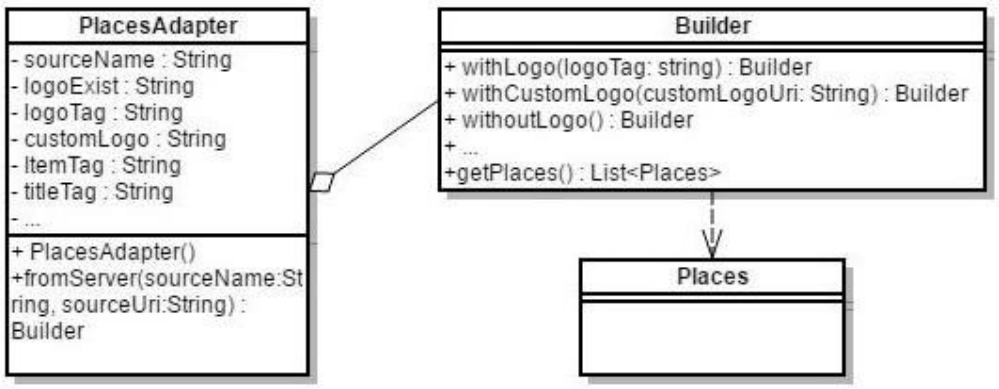


Рисунок 3.3 – Клас PlacesAdapter та inner–клас Builder

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.

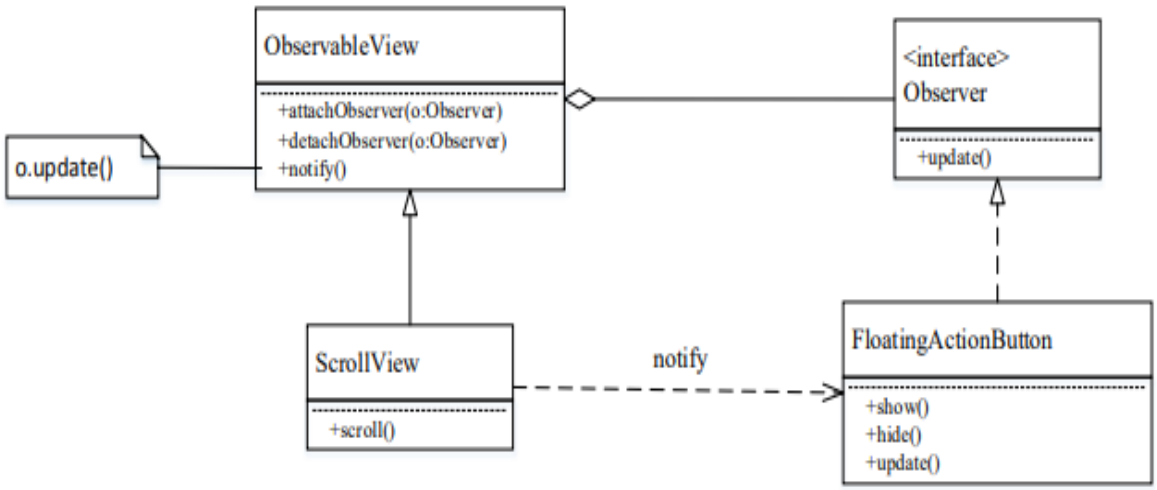


Рисунок 3.4 – Реалізація патерну Observer

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.

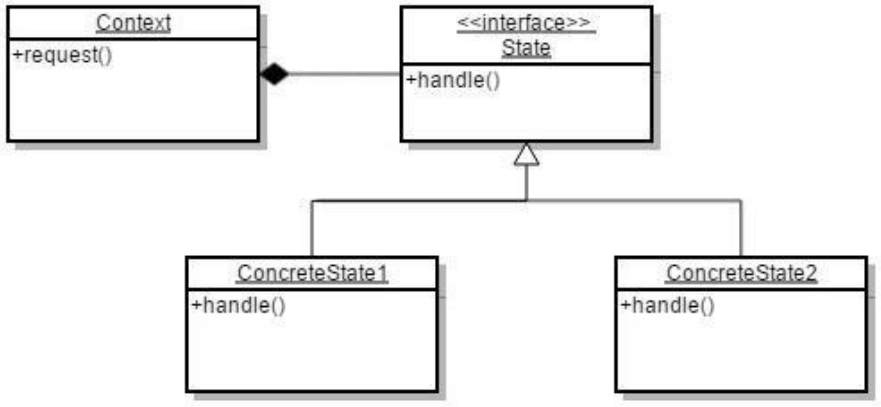


Рисунок 3.5 – Реалізація патерну State

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 оператору повторюється у декількох операціях. Шаблон стан пропонує замінити кожну гілку окремим класом. Це дозволить трактувати стан об'єкта як самостійний об'єкт, котрий може змінитися незалежно від інших.

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 рисунках 3.6 – 3.10.

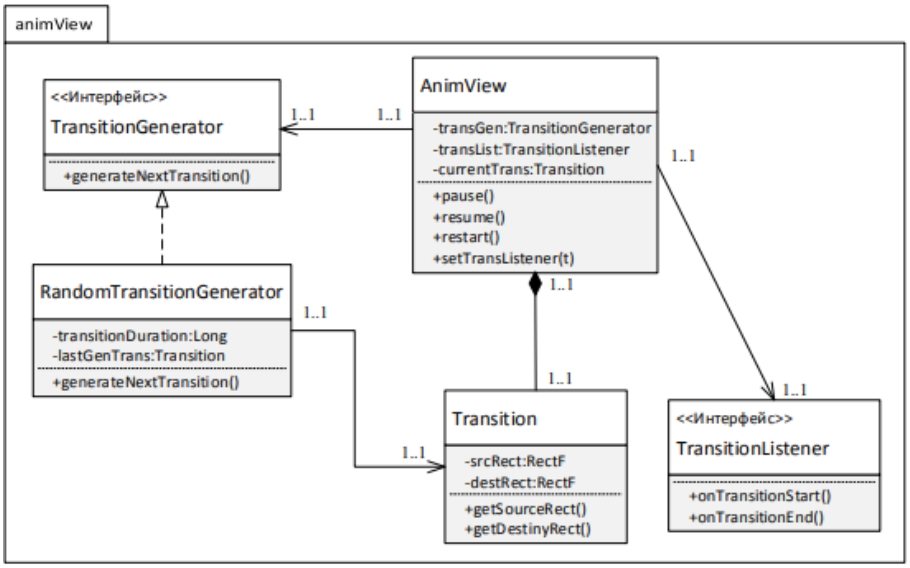


Рисунок 3.6 – Діаграма взаємодії та ієрархії класів пакету AnimView

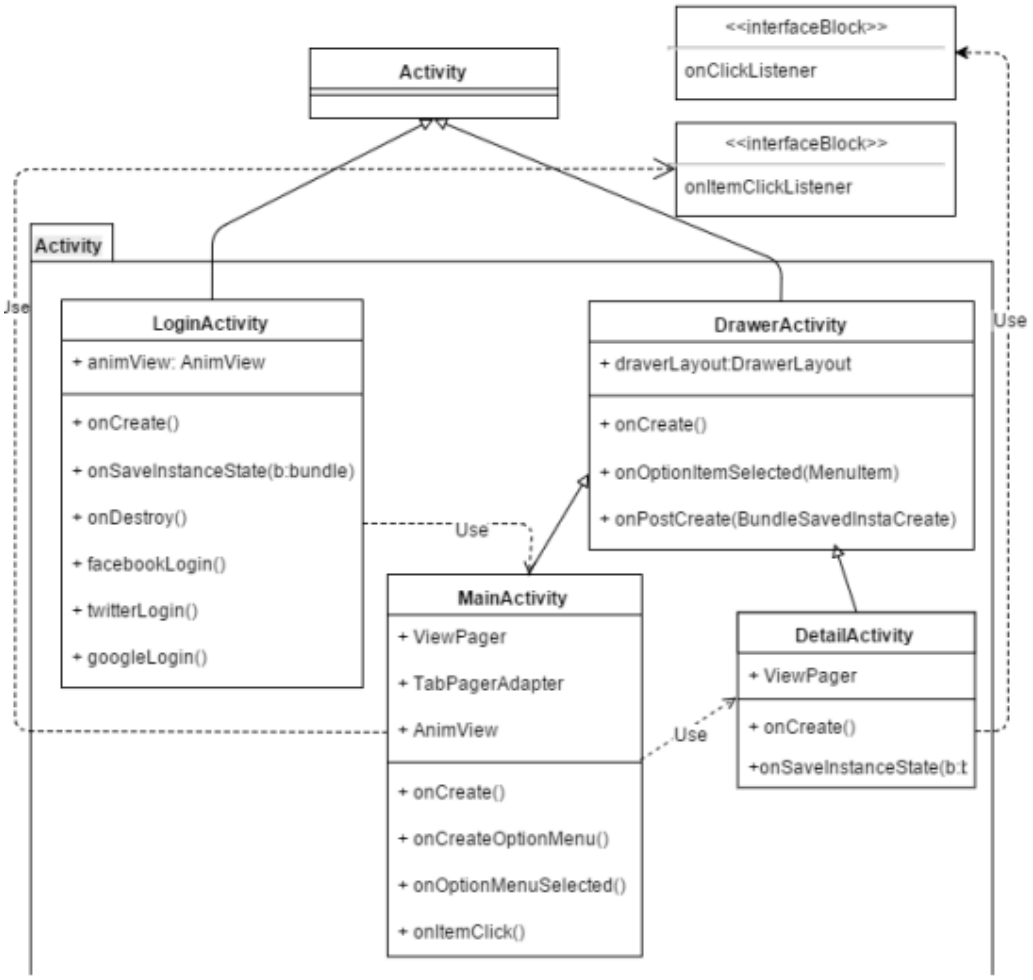


Рисунок 3.7 – Діаграма взаємодії та ієрархії класів пакету Activity

Рисунок 3.8 – Діаграма класів пакету Auth згенерована в середовищі програмування

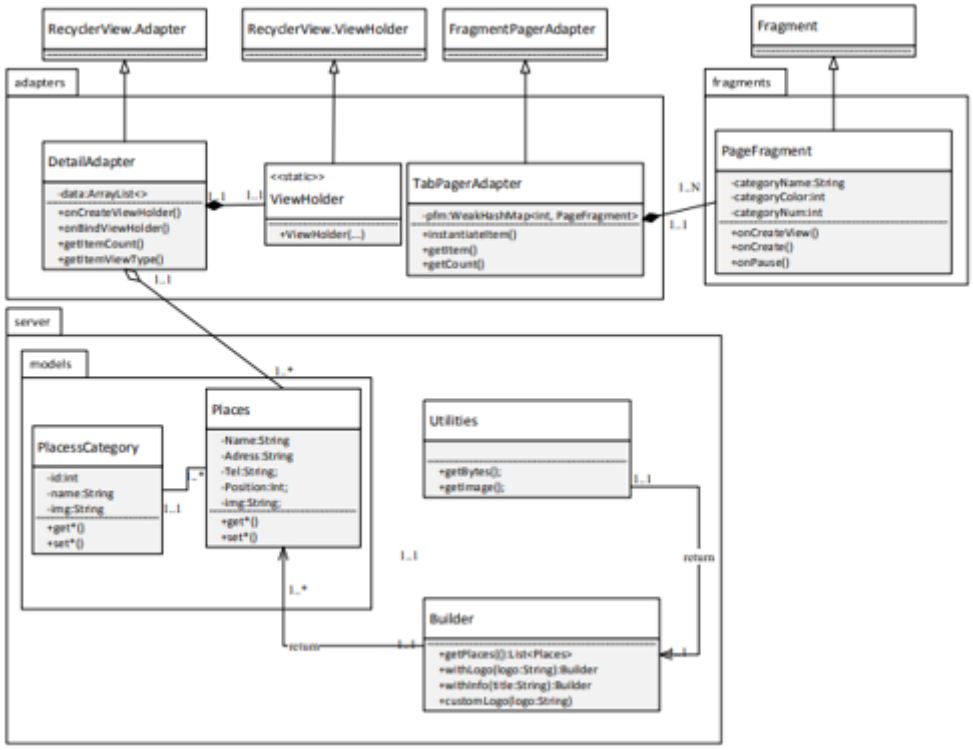


Рисунок 3.9 – Діаграма взаємодії та ієрархії класів пакетів Adapters та Server

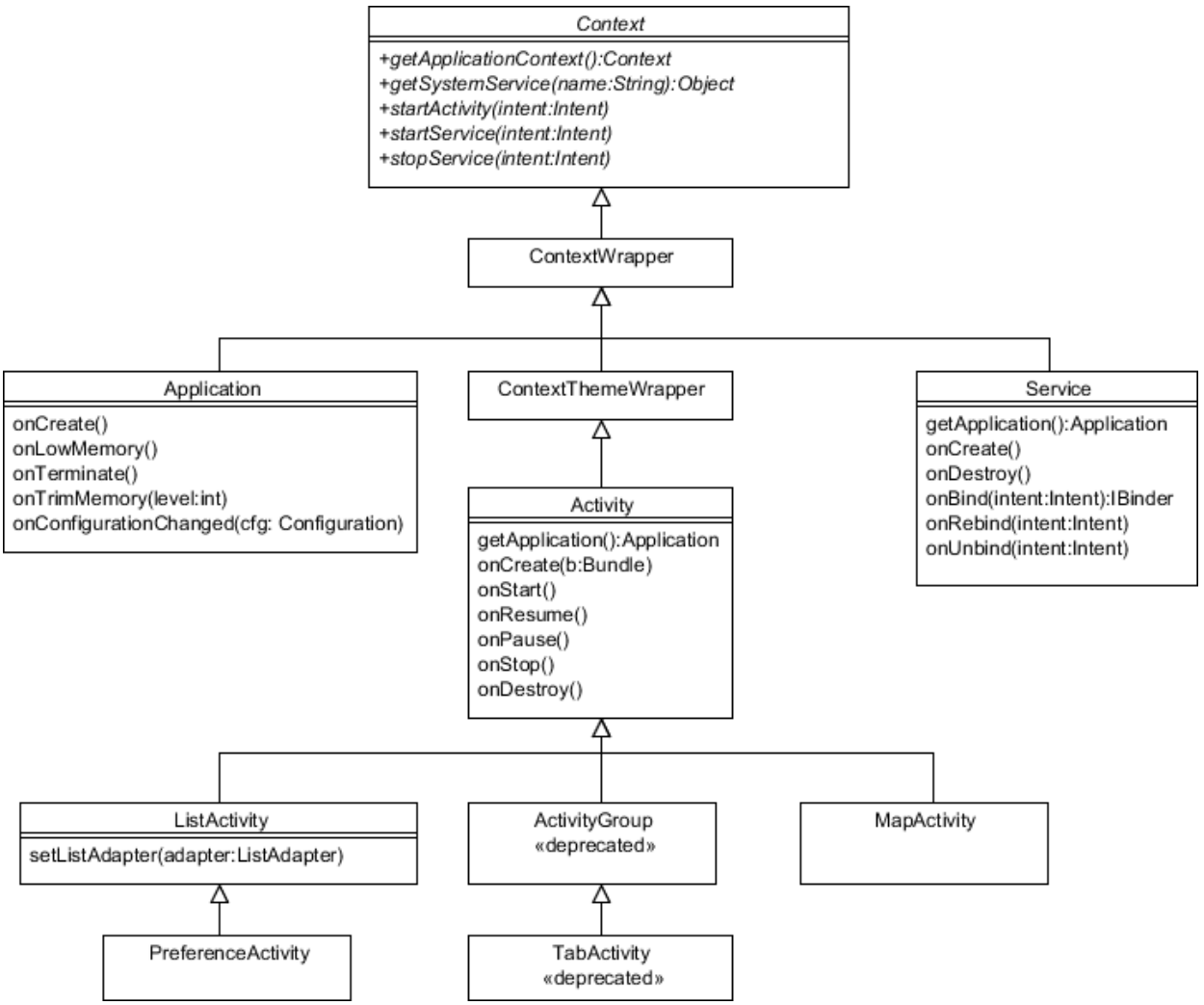


Рисунок 3.10 – Діаграма взаємодії та ієрархії класів пакету ContentTabs

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.

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, рисунок 3.11.

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.

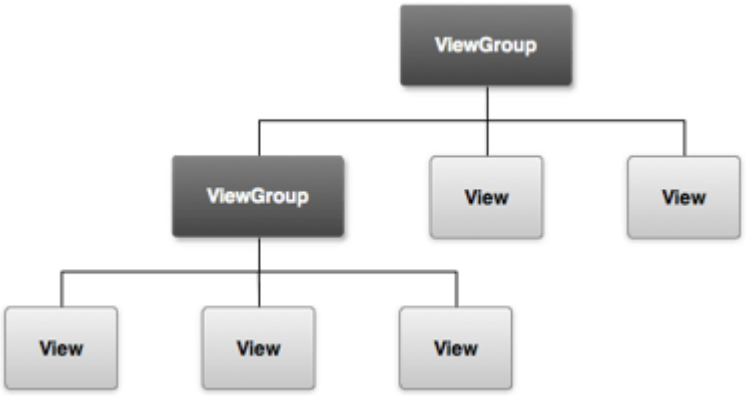


Рисунок 3.11 – Ієрархія об’єктів android.view.View

Для того, щоб мати уявлення про інтерфейс користувача на етапі проектування за допомогою дизайнерської платформи «UxPin» були розроблені макети інтерфейсу (рис. 3.12). На рисунку 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 виконнання всіх запитів у системі, на іншій наведено перелік серверів за допомогою компоненту ListView.

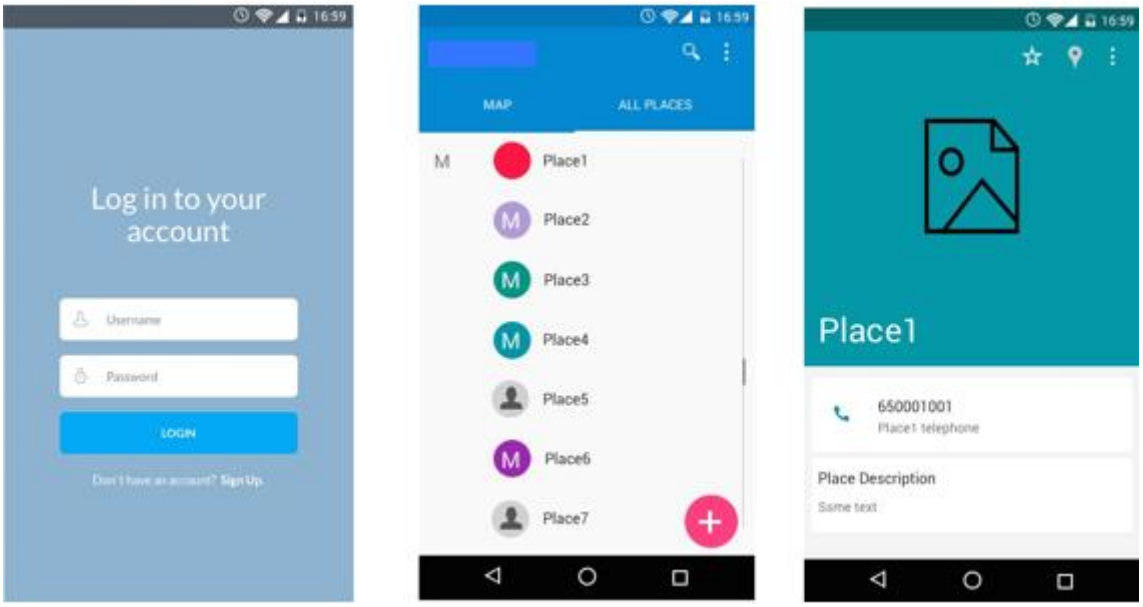


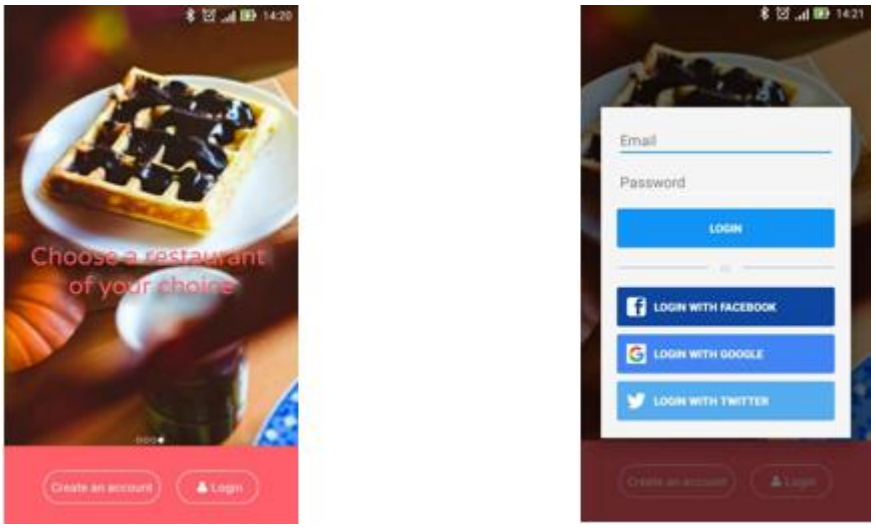
Рисунок 3.12 – Макети інтерфейсу

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
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"

Приведемо екрани програми, які ілюструють інтерфейс системи при вході або реєстрації користувача на рисунках 3.13 – 3.17.



(а) (б)

Рисунок 3.13 – (а) головне вікно входу; (б) вікно автори зації

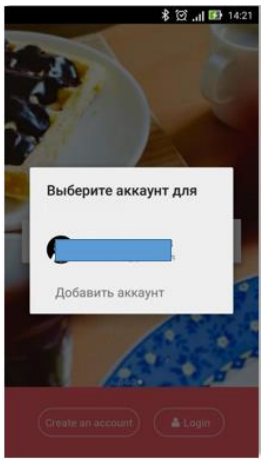


Рисунок 3.14 – Авторизація за допомогою акаунту Google

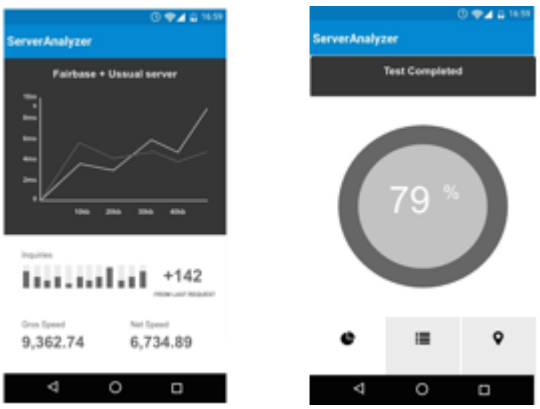
На рисунку 3.13а наведено вікно входу до програми, користувач може обрати створення нового облікового запису, або увійти до існуючого, на задньому плані можна побачити слайдер з коротким описом додатку. При перелистуванні зображення та написи опису змінюються. Якщо користувач обрав вхід до існуючого облікового запису (рисунок 3.13б) він може обрати

декілька типів авторизації.

Коли користувач увійшов до облікового запису, перед ним з’являється головна сторінка, рисунок 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.

Рисунок 3.15 – Головне вікно програми

На рисунку. 3.16 (а) знаходиться інформація після виконання запиту.



(а) (б)

Рисунок 3.16 – Вкладка з: (а) інформацією про технологію; (б) – тестом

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 усіми виконаними операціями даної сесії.

Обравши певний сервер, користувач переходить на екран з його детальним описом (рисунок 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.

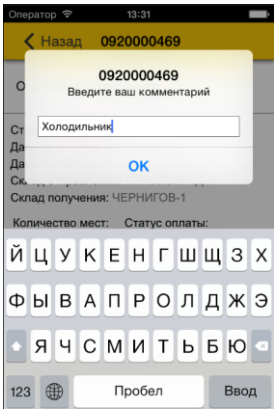


Рисунок 3.17 – Опис сервера та можливість залишити коментар

На рисунках 3.18а та 3.18б зазначено тестування серверу.

а)

б)

Рисунок 3.18 – Вкладка з: а) діаграмою виконаного теста; б) звітом

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 припущенняпро помилку;
- напрямок тестування – знизу вверх, тобто спочатку тестуються всіпідмодулі, 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 легкозаповнені за допомогою фіктивних об'єктів.
- автоматичні тести користувацького інтерфейсу – створення тестів, щоб переконатися, що призначений для користувача інтерфейс поводиться 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.

3.5 Тестування функціональності

Для даного додатка було створено модульний тест, який перевіряє правильність введеної користувачем email адреси. Далі наведено лістинг класу “EmailValidator”:

```
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));
    }  }
```

Результати модульного тесту классу “EmailValidator” (рисунок 3.18) .

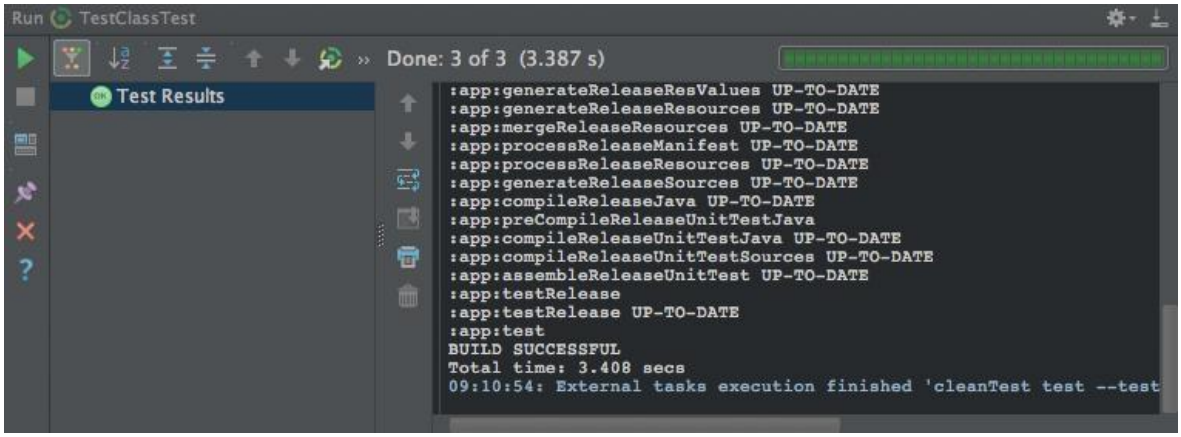


Рисунок 3.18 – Результат тесту EmailValidator

3.5.1 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.

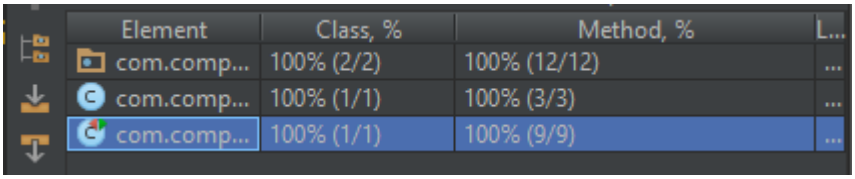


Рисунок 3.19 – Результат тесту ChangeTextBehaviorTest

3.5.2 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 (рисунок 3.20).

```
import org.testng.annotations.
Test; import static org.testng.Assert.*;
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.*;
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);
    }

    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);}}
```

В табл.3.2 наведено скорочений опис методів системи

Таблиця 3.2 – Опис методів системи

Прототип методу	Опис
Клас UserAccount: Public static int getSpeed(double totalSpeed)	Повертає розмір швидкості виконання запиту 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 помилку якщо користувача не було знайдено в базі.

Клас Server: Public boolean setName(String name)	Встановлює назву серверу, якщо рядок не порожній. Повертає “true”, якщо зміни зроблені, інакше – “false”.

Продовження таблиці 3.2

Прототип методу	Опис
Клас Server: Public boolean setCost(float cost)	Встановлює коштовність серверу, 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”.

Результати тестування методів системи за принципом чорної шухляди наведено на рисунку 3.20. Всі Unit-тести були виконані успішно.

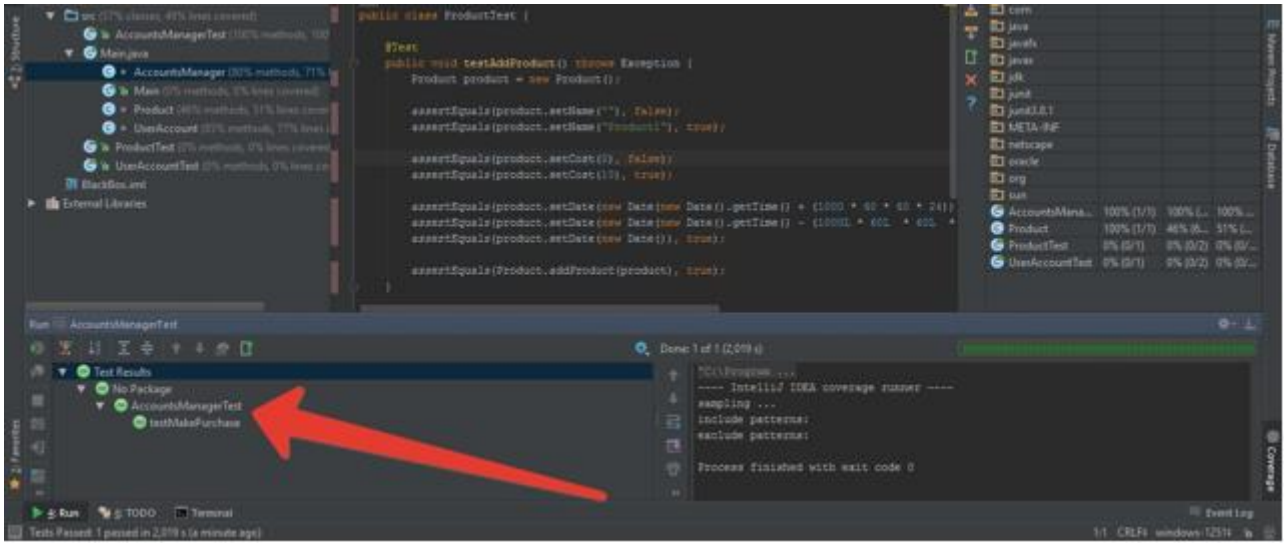


Рисунок 3.20 - Результати виконання Unit-тестів

3.5.3 Метод еквівалентних розбитків

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

Розіб’ємо загальну суму замовлення на наступні класи, що наведені в таблиці 3.3:

1. totalSpeed < 1000.0;
2. 1000.0 ≤ totalSpeed < 5000.0;
3. 5000.0 ≤ totalSpeed < 10000.0;
4. 10000.0 ≤ totalSpeed < 20000.0;
5. totalSpeed ≥ 20000.0.

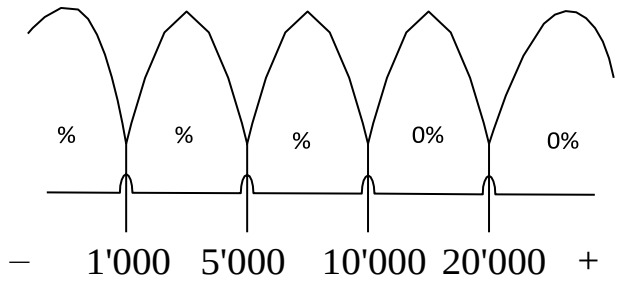


Рисунок 3.21 – Класи еквівалентності функції getDiscount()

Таблиця 3.3. – Результати тестування методом еквівалентних розбиттів

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	totalCost = 0	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: name = пустий рядок (неправильний клас); 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 = “”	Результат = false
2	name = “Server1”	true

Для функції Server.setSpeed (float cost) приведемо наступні класи еквівалентності, що подано у таблиці 3.5:

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

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 наступний клас еквівалентності, що подано у таблиці 3.6:

– date > поточної дати (неправильний клас).

Таблиця 3.6 – Результати тестування UserAccount.setDate() методом еквівалентних розбиттів.

№ класу еквівалентності	Вхідні дані	Вихідні дані
1	date = “01.01.2100”	Помилка: запит повинен бути виконнанийне пізніше поточної дати Результат = false
2	date = “01.01.1800”	Помилка: запит повинен бути виконнанийне раніше, ніж за 5 днів до поточної дати та не пізніше поточної дати Результат = false
3	date = “26.06.2017”	true

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

– пустий список серверів (невірний клас);

– 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	servers = { }	Помилка: неможливо підрахувати середню швидкість виконання запиту
2	servers = {<Server1, 200, “02.02.2002”>,<Server12, 400, “03.03.2003”>}	300, вірно
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:

– заданого користувача не існує – не вірний;

– заданий користувач існує, користувач здійснює перший запит або загальна швидкість усіх попередніх запитів користувача менша 1000 б/с. – вірний;

– 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	username = “Unknown”	Помилка:
	Server = <Server1, 200, “02.02.2002”>	Заданого користувача не існує
2	username = “User1” Server = <Server1, 1000, “02.02.2002”>	1000
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: серверівне зазначено

3.5.4 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.

В таблиці 3.9 наведено результати тестування UserAccount.getDiscount() методом граничних умов з вхідними значеннями.

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.

Таблиця 3.9 – Результати тестування UserAccount.getDiscount() методом граничних умов

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

Таблиця 3.11 – Результати тестування Server.setDate() методом граничних умов (поточна дата – 2 лютого 2017 року)

№ тесту	Вхідні дані	Вихідні дані
1	Date = 02.01.2010”	true
2	Date =“02.01.2017”	true
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

3.5.5 Тестування методом припущення помилки

Процедура методу припущення про помилку в значній мірі заснованана 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, результати яких наведено в таблиці 3.12.

– 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».

– припустимо, що метод Server.setSpeed() працює некоректно при введенні швидкості запиту у невірному форматі. Наприклад, «двадцять дев’ять».

– припустимо, що метод makeRequest() працює неправильно при спробі зробити користувачем запиту з серверу, якого немає в БД програми.

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

3.6 Висновки до третього розділу

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.

Нижче, в таблиці 3.13, стисло наведені існуючі технології та їх характеристики.

Таблиця 3.13 – Короткий огляд характеристик існуючих технологій

	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
Amazone 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 розташува

							ння

4. ЕКОНОМІЧНИЙ РОЗДІЛ

Дані для розрахунків:

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

4.1 Розрахунок трудомісткості і вартості розробки програмного продукта

Нормування праці в процесі створення ПЗ суттєво 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 4.1.

$$t = t_o + t_u + t_a + t_n + t_{\text{отл}} + t_\partial, \text{люд./год.}, \quad (4.1)$$

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;

$t_{\text{отл}}$ – 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 4.2.

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

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 задачі визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u \frac{QB}{(75...85)K} \text{ люд./год}, \quad (4.3)$$

де B = [1 . 2 ÷ 1 . 5] – коефіцієнт збільшення витрат праці внаслідок недостатнього опису завдання, в нашому випадку B = 1,4.

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності. Він складає певну кількість років. Оскільки стаж дорівнює від двох до трьох років, K – 1,0: t_u=51 люд./год.

$$t_u = \frac{(2913 \times 1.4)}{(80 \times 1.0)} = 51 \text{ люд./годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)K} = \frac{2913}{(24 \times 1.0)} = 121.37 \text{ люд./год.}, \quad (4.4)$$

Витрати на складання програми по готовій блок–схемі:

$$t_a = \frac{Q}{(20...25)K} \quad (4.5) \qquad t_a = \frac{2913}{(23 \times 1.0)} = 1.27 \text{ люд./год.}$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання можна розрахувати по формулі 4.6.

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

$$t_n = \frac{2913}{(4 \times 1.0)} = 728.25, \text{люд./год.}$$

Витрати праці на налагодження програми за умови комплексного налагодження завдання по формулі 4.7:

$$t^k{}_{\text{отл}} = 1.5\, t_{\text{отл}}, \quad (4.7)$$

$$t^k{}_{\text{отл}} = 1.5 \times 728 = 1092, \text{люд./год.}$$

Витрати праці на підготовку документації по формулі 4.8:

$$t_д = t_{дp} + t_{до}, \quad (4.8)$$

де $t_{дp}$– трудомісткість підготовки матеріалів і рукопису по формулі 4.9,

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації по формулі 4.10.

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

$$t_{дp} = \frac{2913}{(17 \times 1.0)} = 171, \text{люд./год.}$$

$$t_{до} = 0.75 t_{дp}, \quad (4.10)$$

$$t_{до} = 0{,}75 \times 171 \ = \ 128.25, \text{люд./годин.}$$

Звідси: $t_{д}$ = 171 + 128.25 = 299.25, люд./годин

В підсумку отримуємо, що трудомісткість розробки ПЗ становить:

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

4.2 Витрати на створення програмного забезпечення

Витрати на створення ПЗ $K_{по}$, за формулою 4.11, включають витрати на заробітну плату виконавця програми $З_{з/п}$ (формула 4.12) і витрат машинного часу, необхідного на налагодження програми на ЕОМ (формула 4.13).

$$K_{по} = З_{зп} + З_{МВ}, \qquad (4.11)$$

Заробітна плата виконавців визначається за формулою 4.12:

$$З_{зп} = tC_{пр}, \qquad (4.12)$$

Таким чином, трудомісткість розробки програмного забезпечення склада, є 1205,87 люд./годин.

де: t – загальна трудомісткість, люд./годин;

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

$C_{пр}$= 35,0 грн/год.

Отже: зп = 1205.87 × 35 = 42175 грн.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ за формуло 4.13:

$$З_{МВ} = t_{отл}C_{мч}, \qquad (4.13)$$

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

$C_{\text{мч}}$– вартість машино–години ЕОМ, грн/год (приблизно 20грн/год).

Звідси:

$$З_{МВ} = 728{,}25 \times 20 = 14565 \text{ грн.}$$

З отриманих даних витрати на створення програмного забезпечення складатимуть:

$$K_{по} = 42175 + 14565 = 56740 \text{ грн.}$$

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

Очікуваний період створення ПЗ за формулою 4.14:

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

де B_k– число виконавців;

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

Звідси:

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

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 під час розробкиПП 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 цієїстатистики можна зробити висновок, наскільки далеко зайшло виконанняпевного 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 стратегічнихрішень, 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 напрямах діяльності. Для країн з «перехідною економікою», зокрема для України, характерне прагнення багатьохпідприємців 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, розрахунок надходжень наведено в таблиці 4.1.

Розроблений програмний проукт буде поширюватись в рамках “Open Source Software”, що передбачає вільне розповсюдження, доступний у відкритому доступі тескт 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 дількості аудиторії та числа користувачів, тому економчна ефективніс уданого продукту може враховуватись лише у випадку монетизації додатку (поширення реклами), а на передній план виходить соціальна ефективність від вровадження, що характеризує покращення навичок у користувачів та розробників програмного забезпечення.

Таблиця 4.1 – Розрахунок чистих грошових надходжень від розробки ПЗ

Показники, грн	За роками						За 5 років	
	0 (підго товка)	1	2	3	4	5	Усього	Середнє
1. 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

Продовження таблиці 4.1

Показники, грн	За роками						За 5 років	
	0 (підготовка)	1	2	3	4	5	Усього	Середнє
– 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
2.Витрати після впровадження ПО	–	1240	1240	1240	1240	1240	9200	1840
– на подовження 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
– на щорічну перевірку серверів	–	200	200	200	200	200	1000	200

Продовження таблиці 4.1

Показники, грн	За роками						За 5 років	
	0 (підготовка)	1	2	3	4	5	Усього	Середнє
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

4.5 Коефіцієнти економічної ефективності

Чиста поточна вартість доходів:

NPU = 62740 − 56740 = 6000 грн. > 0

Строк окупності:

T = 56740 /12548 = 4 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 капітпло–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.

4.6 Висновки до четвертого розділу

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 преваг та недоліків. Спроектовано та розроблено мобільний додаток для операційної системи Android, для наданнята 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 відлагодженю 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. Задокументований опис та текст програми спростить подальше супроводження системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

2 Tripathy P. An IoT-enabled smart greenhouse for sustainable agriculture. *IEEE Consumer Electronics Magazine*. 10.4. 2021. 57-62.

3 Rayhana R., Xiao G., Liu Z. Internet of things empowered smart greenhouse. Farming. *IEEE Journal of Radio Frequency Identification*. 4.3. 2020. 195-211.

4 Майер, Рето. Android 2. Программирование приложений для планшетных компьютеров и смартфонов. *Litres*. 2011.

5 Документація Git. *GitHub*. URL: https://gitscm.com.

6 Android Studio. *Матеріал з Вікіпедії*. URL: https://ru.wikipedia.org/wiki/Android_Studio.

7 Документація Dagger. *GitHub*. URL: https://google.github.io/dagger.

8 Д. Гриффитс. Head First. Программирование для Android. *СПб.Питер*. 2016.

9 Дейтел Харви, Дейтел Эбби. Android для разработчиков. *Издательский дом Питер*. 2014.

10 Android. *Матеріал з Вікіпедії*. URL: https://ru.wikipedia.org/wiki/Android.

11 Git. *Матеріал з Вікіпедії*. URL: https://ru.wikipedia.org/wiki/Git.

12 *Habrahabr* – Структура додатків для Android» URL: https://habrahabr.ru/company/ncloudtech/blog/274025.

13 Dagger 2. Частина перша. *Habrahabr*. URL: https://habrahabr.ru/post/279125.

14 Dagger 2. Частина друга. *Habrahabr*. URL: https://habrahabr.ru/post/279641.

15 Чакон, Скотт, and Б. Штрауб. Git для профессионального программиста. *СПб.: Питер*. 2016.

16 М. Роберт. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. *Издательский дом Питер*. 2013.

17 Леонтьев, В. К., Г. Л. Мовсисян, Ж. Г. Маргарян. Совершенные коды в аддитивных каналах. Vol. 411. No. 3. *Федеральное государственное бюджетное учреждение "Российская академия наук"*. 411.3. 2006.

18 Duggan, Dominic.Enterprise Software Architecture and Design: Entities, Services, and Resources. *John Wiley & Sons*. Vol. 10. 2012.

19 Документація Java URL: https://docs.oracle.com/javase/7/docs/api.

20 Документація Android URL: https://developer.android.com.

21 Герберт Шилдт. Полное руководство. *John Wiley & Sons*. Vol. 8. 2011.

22 Evans, Ben, and David Flanagan. Java in a Nutshell: A Desktop Quick Reference. *O'Reilly Media*. 2018.

23 Goncalves, Antonio, and Massimo Nardone. Beginning Java EE 7. *New York: Apress*. 2013.

24 Higuera-Toledano, M. Teresa, and Andy J. Wellings. Distributed, embedded and real-time java systems. *Springer Science & Business Media*. 2012.

25 Jendrock, Eric, et al. The Java EE 6 tutorial: advanced topics. *Addison-Wesley*. 2013.

26 Eric Armstrong. The J2EE 1.4 tutorial. *Sun Microsystems. Vol. 17*. 2004.

27 Gupta, Arun. Java EE 6 Pocket Guide: A Quick Reference for Simplified Enterprise Java Development. *O'Reilly Media*. 2012.

28 Pilgrim, Peter A. Java EE 7 Developer Handbook. *Packt Publishing Ltd*. 2013.

29 Verburg, Peter H. Beyond land cover change: towards a new generation of land use models. Current Opinion in Environmental Sustainability. *O'Reilly Media. Vol. 38*. 2019. 77-85.

30 Evans, Eric, and Eric J. Evans. Domain-driven design: tackling complexity in the heart of software. *Addison-Wesley Professional*. 2004.

31 Шилдт, Герберт. C++: Руководство для начинающих. *Издательский дом Вильямс*. Vol. 2. 2005.

32 Cruz-Cunha, Maria Manuela. Handbook of research on ICTs and Management Systems for Improving Efficiency in healthcare and social care. *IGI*

global. Vol. 2. 2013.

33 Цехановский, Владислав, Борис Советов, and Владимир Чертовской. Базы данных. Учебник для прикладного бакалавриата. *Litres*. 2022.

34 Гарсиа-Молина Г., Ульман Д. Д., Уидом Д. Системы баз данных: Полный курс. *Вильямс*. 2003.

35 Adamatzky A. Advances in unconventional computing: Theory. *Springer*. Vol. 2. No. 22. 2016.

36 Albers M., Still B. Usability of complex information systems. Evaluation of user interaction. *Litres*. 2010. 3-16.

37 Arlow J., Neustadt I. Enterprise patterns and MDA: building better software with archetype patterns and UML. *Addison-Wesley Professional*. 2004.

38 Aslaksen E. W. Designing Complex Systems: Foundations of design in the functional domain. *Auerbach publications*. 2016.

39 Chen C. et al. The impacts of knowledge sharing-based value co-creation on user continuance in online communities. *Information Discovery and Delivery*. 2017.

40 Romero D., Vernadat F. Enterprise information systems state of the art: Past, present and future trends. *Computers in Industry*. Vol. 79. 2016. 3-13.

41 Fairley R. Software engineering concepts. *McGraw-Hill, Inc*. 1985.

42 Rudenko O. et al. Developing a Multi-Step Recurrent Algorithm to Maximize the Criteria of Correntropy. *Eastern-European Journal of Enterprise Technologies*. Vol. 1. No. 4. 2021. 109.

43 Rudenko O. et al. Robust identification of non-stationary objects with nongaussian interference. *Eastern-european Journal of enterprise Technologies.* Vol. 5. No. 4. 2019. 44-52.