МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних техноло-
гій

Бакалаврська кваліфікаційна робота

на тему: Розробка клієнт-серверного додатку для валідації квитків

Виконав студент 4 курсу групи
К-45
Спеціальності 122 комп'ютерні
науки,
Небож Віталій Ігорович

Керівник к.т.н., доцент
Трегубова Ірина Анатоліївн

Консультант

Рецензент к.ф-м.н., доцент
Вітавецька Лариса Анатоліївна

Одеса 2019

ЗМІСТ

Скорочення та умовні познаки 7
Вступ 8
1 Характеристика предметної області і постановка завдання 9
1.1 Опис предметної області 9
1.2 Огляд функціональних можливостей існуючих систем 13
1.3 Визначення вимог до системи 17
2 Вибір програмних засобів реалізації проекту 18
2.1 Серверний сервіс 18
2.1.1 Система керування базами даних PostgreSQL 18
2.1.2 СередарозробкиJava Virtual Machine 19
2.1.3 Огляд фреймворку SpringFramework 20
2.1.4 Стандарт взаємодії з базами даних Java DataBase Connectivity 21
2.2 Клієнтський сервіс 22
2.2.1 Особливості мови програмування JAVA 22
2.2.2 Мобільна операційна система Android 25
2.2.3 Середа розробки Android SDK 29
2.2.4 Середа розробки Android Studio 29
2.2.5 Об'єктно-орієнтоване програмування 31
2.2.6 Бібліотека Retrofit для роботи з мережею 32
2.2.7 Бібліотека для роботи з SQLite базою даних ROOM 32
2.2.8 Бібліотека Zxing для опрацювання баркодів 32
3 Реалізація клієнтського програмногозабезпечення для валідації квитків 34
3.1 Моделювання додатку 34
3.2 Проектування діаграми прецедентів 35
3.3 Проектування діаграми потоків даних 36
3.4 Функціональна схема додатку 38
3.5 Алгоритм роботи додатку 41
3.6 Розробка зовнішнього вигляду додатку 42

3.7 Розробка логіки додатку 46
4 Реалізація серверної частини додатку 49
4.1 Опис таблиць бази даних і звязків між ними 49
Висновки 53
Перелік джерел посилання 54

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Баркод– спосіб запису даних, зручний для зчитування машиною.

БД (База даних) – упорядкований набір логічно взаємопов'язаних даних, що використовується спільно, та призначений для задоволення інформаційних потреб користувачів.

СКБД (Система керування базами даних)– набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних.

Таблиці БД– це набір елементів даних (значень), що використовує модель вертикальних стовпців (мають унікальне ім'я) і горизонтальних рядків.

API– (програмний інтерфейс програми, інтерфейс прикладного програмування) – опис способів (набір класів, процедур, функцій, структурних або константних), які можуть одна комп'ютерна програма може взаємодіяти з іншою програмою.

Id– унікальна ознака об'єкта, що дозволяє відрізняти його від інших об'єктів, тобто ідентифікувати.

Java– об'єктно-орієнтована мова програмування.

JDBC (JavaDataBaseConnectivity) –прикладний програмний інтерфейс Java, який визначає методи, з допомогою яких програмне забезпечення на Java здійснює доступ до бази даних.

XML (ExtensibleMarkupLanguage) – запропонований консорціумом WorldWideWebConsortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет.

ВСТУП

Актуальність теми. Майже неможливо уявити життя сучасної людини без розваг. Будь то похід у кінотеатр, на концерт, на виставку картин або ж на поетичний вечір.

Такі події відбуваються скрізь та чим більше місто, у якому людина проживає, тим концентрація таких подій на одиницю часу більша. Проте, багато з цих подій мають певні проблеми організаційного характеру. Люди або повинні пред'являти паперові квитки, або забов'язані купувати їх на місті.

Часто це буває дуже незручно. Звичайно, якщо говорити про великих монополістів на ринку розваг (наприклад concert.ua), або ж просто про величезні компанії надання послуг (наприклад Укрзалізниця) то там з перевіркою білетів все гаразд. Володіючи закритими корпоративними системами та складним технічним обладненням цей процес стає простим та зручним.

Якщо ж якась невелика громада вирішить організувати якусь суспільну подію, у цьому разі нема інструментів які б допомоглм їй це зробити. Просто нема таких сервісів, які б допомогли перевірити раніше згенеровані квитки у людей.

Ця тема є актуальною тому, що дозволяє людям значно облегшити процес організації суспільних заходів та процес перевірки квитків.

Метою роботи є розробка клієнт-серверного додаку для валідації квитків, який являє собою інструмент що забезпечує етапи :

– перевірки занесених у серверну базу даних квитків;

– фільтрації діючих та недіючих квитків.

Для досягнення мети необхідно вирішити наступні завдання:

– вивчити та проаналізувати предметну область;

– створити серверний додаток;

– створити клієнтський додаток.

Розробка подібної системи є дуже актуальним у наш час і особливо в нашому регіоні, багатому на різноманітні розваги.

1ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

В даному розділі проводиться перший етап розробки інформаційної системи «Клієнт-серверний додаток для валідації електронних білетів», а саме розбір предметної області, виявлення актуальності та постановки концептуальних питань щодо структури системи та їх реалізації.

1.1Опис предметної області

Поняд 12 років тому, в вересні 2007 року корпорація google презентувала світу мобільну операційну систему Android, а вже через рік компанія HTC розпочинає продаж першого в світі смартфону під керуванням цієї системи. Однією з головних переваг нової системи перед іншими на ряду із можливістю адаптування її до будь-яких вимог, зручним інтерфейсом та політикою безкоштовного розповсюдження стає вбудований у кожний пристрій магазин додатків, публікація власного програмного продукту у якому не потребує а ні проходження технічної модерації а ні фінансових витрат.

Та саме завдяки такому “ліберальному” підходу до діла численна кількість самостійних розробників програмного забезпечення всього за декілька років наповнили магазин сотнями тисяч додатків на будь-якій смак та колір.

Фантазія програмних архітекторів у купі з численними апаратними особливостями компактних комп’ютерів під керуванням Android (маються на увазі такі апаратні модулі як : вбудованій сенсорний екран з великою розподільною здатністю, модуль wifi, bluetooth, модуль телефонного зв’язку, мікрофон, ік-порт, гіроскоп, камера, люксметр, динаміки, тощо) перетворювала їх на дещо більше ніж просто телефон.

Завдяки такому величезному набору модулів можливості звичайних смартфонів стають майже безмежними. Завантаживши необхідний додаток

будь-який телефон легко перетворюється у ір камеру, gps трекер, сканер баркодів, зі смартфона стає можливим керування автомобіля, медіацентру чи системи «розумний дім», тощо. Нема сенсу навіть продовжувати цей спісок бо будь-хто доставши свій телефон із кишені і завітавши до Play Store або App Store або ж до Windows Mobile Store може переконатися що додаток знайдеться на будь-який випадок.

Майже неможливо уявити життя сучасної людини без розваг. Будь то похід у кінотеатр, на концерт, на виставку картин або ж на поетичний вечір.

Такі події відбуваються скрізь та чим більше місто у якому людина проживає тим концентрація таких подій на одиницю часу більша. Проте багато з цих подій мають певні проблеми організаційного характеру. Люди або повинні пред’являти паперові квитки або забов’язані купувати їх на місті. Часто це буває дуже незручно. Звичайно якщо говорити про великих монополістів на ринку розваг (наприклад concert.ua), або ж просто про величезні компанії надання послуг (наприклад Укрзалізниця) то там з перевіркою білетів все гаразд. Володіючи закритими корпоративними системами та складним технічним обладненням цей процес стає простим та зручним. Але для невеликих подій нема зручних інструментів контролю. Моя дипломна робота якраз і вирішує проблему організації перевірки квитків для заходів.

Ця тема є актуальною тому що дозволяє людям значно облегшити процес організації суспільних заходів та процес перевірки квитків.

Квиток (фр. Billet, від середньовічного billetus - записка, лист, свідоцтво, посвідчення) - документ, що засвідчує наявність якогось права у будь-якого певного особи або у пред'явника квитка. Дія квитка може поширюватися на конкретний час або не мати термінів.

Наприклад, квитки використовуються для проїзду на громадському транспорті (в тому числі багаторазові проїзні квитки), відвідування різних заходів (в тому числі безкоштовно за запрошеннями).

Квитки зазвичай не є грошима і не мають вільного ходіння. Це не універсальний еквівалент вартості інших товарів або послуг, і їх обмін їх на

гроші або товари в необмеженій кількості є незаконним. В обмеженому ж кількості квитки можна купувати, продавати, обмінювати або дарувати безоплатно.

Квитки є предметом колекціонування. Колекціонування квитків на проїзд в громадському транспорті називається «Перидромофілія».

Раніше повсюдно використовувались паперові квитки. При поході у кіно, чи у театр. В музей чи на музичну виставу крізь необхідно було брати квитки. Мабуть у кожного були випадки коли забутий вдома чи загублений по дорозі квиток ставив хрест на подальші плани. Незручно у ті часи було всім : і продавцям (через те що білет необхідно було фізично передати покупцю) і покупцям (їм то необхідно було за квитком поїхати та подбати про його подальше збереження), а ще це було незручно тим хто квиток цей перевіряв (наприклад через погане освітлення приміщення). Тобто недоліків було маса (рис. 1):

Рисунок 1– Приклад перевірки білетів зразка 1942 року

З розвиненням технологій процедура значно спростилась. Клієнт купує квиток зі смартфону, отримуючи при цьому його електронну версію. Дані про купівлю з усіма даними покупця (телефон, ім’я, прізвище тощо) та придбаного квитка зберігаються на сервері. Пізніше в день проведення події квиток пред’являється охоронцю який стоїть на вході та він, використовуючи спеціальне дороге обладнення (сканери наподобі тих що використовуються в супермаркетах на касі) сканують квиток, і запросивши дані про квиток з сервера перевіряють чи все добре та якщо все ок - пропускають людину. Найсерйозніша реалізація з якою я особисто мав справу це реалізація перевірок від Укрзалізниці. По перше вона а ні жодного разу не виходила з ладу а по друге сканування даних відбувається за допомогою сканерів Motorola які є надійними, потужними, зручними та дуже дорогими (рис. 2):

Рисунок 2 – Приклад сучасної системи перевірки квитків

Існують і інші подібні системи. Як завжди - за допомогою додатка або на сайті квиток купуємо, отримаємо його потім пред’являємо наш білет охороннику (білет знаходяться в скритому вигляді, в виді QR коду чи баркоду).

1.2 Огляд функціональних можливостей існуючих систем

Система перевірки квитків від компанії «Укрзалізніця»

«Українська залізниця» – державне акціонерне товариство залізничного транспорту загального користування, національний перевізник вантажів та пасажирів, метою діяльності якого є задоволення потреб у безпечних та якісних залізничних перевезеннях у внутрішньому та міжнародному сполученні, забезпечення ефективного функціонування та розвитку залізничного транспорту та ін.

Якісний перехід від традиційного формату паперових квитків до новітньої цифрової системи продажу та перевірки компанія здійснила в 2014 році (до речі значно пізніше за своїх європейських товаришів).

Зараз якщо людині необхідно придбати квиток – існує два шляхи це зробити.

Шлях перший – приїхати до місцевого залізничного вокзалу, вистояти у черзі, отримати паперовий квиток і не загубити його потім. Погодьтеся що не дуже цей варіант є привабливим особливо для людини у якої у зв’язку із складним робочим графіком нема вільного часу навіть для зустрічі з друзями. Не те що для всієї цієї епопеї.

Шлях другий –встановити офіційний мобільний додаток на смартфон, обрати необхідний маршрут, заповнити пункти з особистими даними (ФІО, номер телефону і т.д), оплатити через один з сервісів онлайн банкінгу і отримати на електронну пошту свій квиток, який і є підставою для посадки на поїзд. Тобто отриманий поштою електронний квиток вже є підставою для посадки. Другий варіант як для мене є привабливишим. Попри економію часу грошей та нервів, економиться такоє і папір а через це природні ресурси.

Процес пошуку квитків та відображення вже придбаних є зручнім та простім (рис.3):

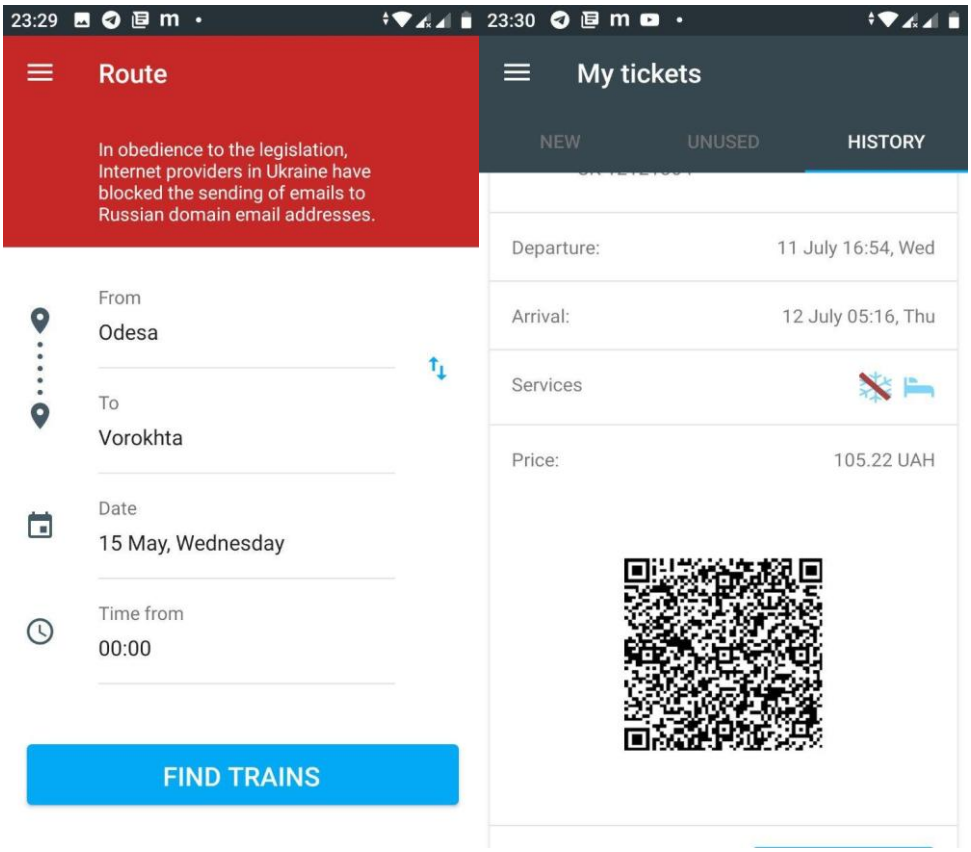

Рисунок 3– Процес вибору маршруту у додатку Укрзалізниця

Як можемо бачити інтерфейс доволі простий та інтуїтивно зрозумілий. Обираємо станцію відправлення, станцію прибуття та час, тиснемо на кнопку знайти потяг і знаходимо його.

Перевагами системі від «Укрзалізниця» є:

– зручний та зрозумілий інтерфейс;
– зручний спосіб для перегляду придбаних квитків.
– зручне керування за допомогою електронної пошти.
– надійність системи.

Серед недоліків цього додатку можливо зазначити:

– система перевіркі квитків є закритим корпоративним продуктом товариства «Укрзалізниця», тому її неможливо використовуваті для створення власніх білетів;

– перевіряючі не завжди погоджуються сканувати квиток – потребують паперовий;

– іноді трапляються збої у системі.

Ukrticket.com.ua – найбільший український сайт для продажу електронних квитків.

Укртікет є офіційним партнером всіх великих концертів чи подій що проходять в нашій країні. У 2018 року Ukrticket продав більше двох мільйонів квитків Головна сторінка ресурсу приведена нижче (рис. 4):

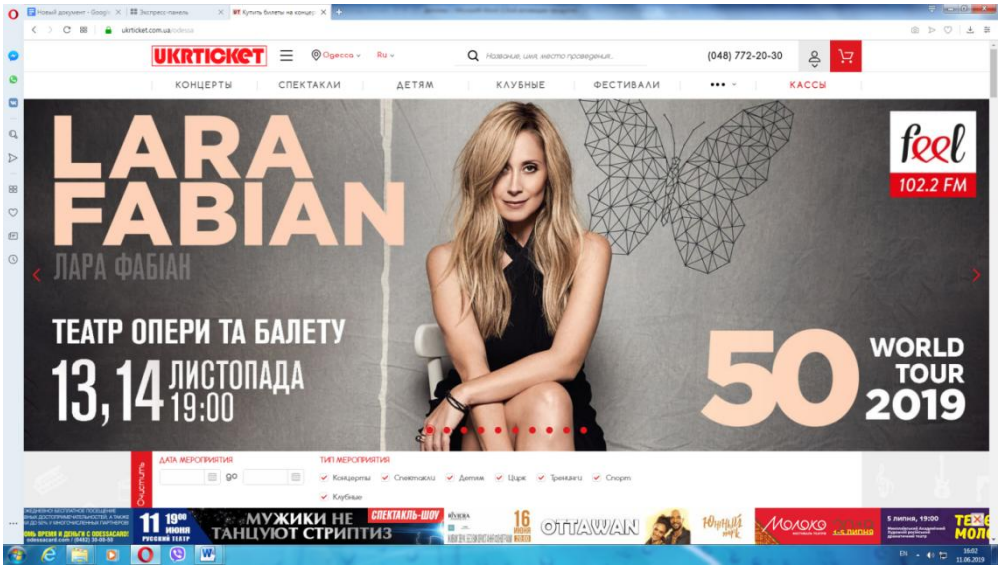

Рисунок 4 – Головний сторінка сайту Ukrticket.com.ua

Перевагами додатку Ukrticket є:

– перевірена роками надійність у роботі.

– зручність використання.

– користувачі отримають розгорнуту інформацію у вигляді.

– повідомлень та діаграм.

– ресурс має консультаційні точки по всій країні

– служба підтримки клієнтів

Серед недоліків цього додатку можливо зазначити:

– система немає повноцінного додатку.

– неможливість використання під час організації невеликих заходів.

– завелика ціна контраку для організаторів.

MCSTicketValidator– є простим та безкоштовним програмним комплексом який складається з сайту та мобільного додатку. Має сама такий набір функцій який є необхідним під час організації невеликих заходів (рис. 5):

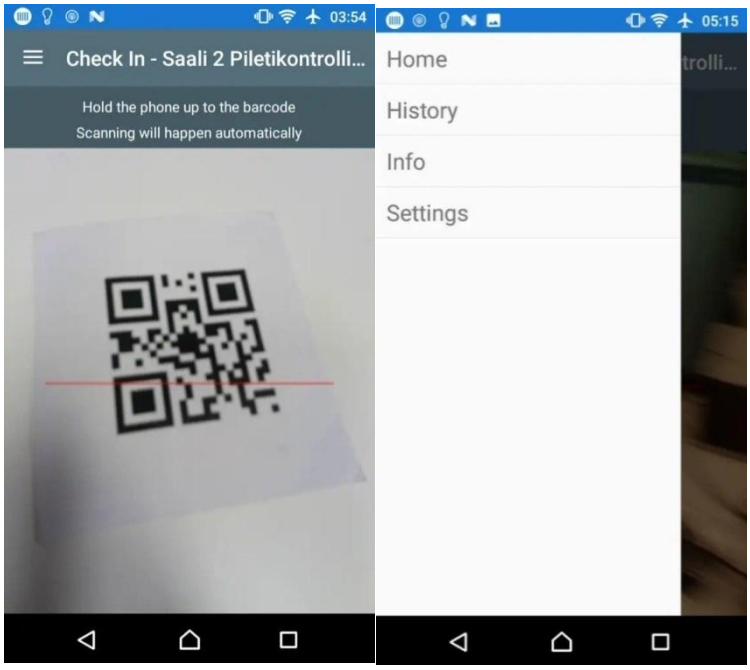

Рисунок 5 – Інтерфейс додатку MCSTicketValidator

ПеревагамидодаткуMCSTicketValidatorє:

– сервіс не перевантажен зайвими функціями.

– має додаток для мобільних пристроїв.

– має відкритий сирцевий код.

– Розповсюджується безкоштовно

Серед недоліків цього додатку можливо зазначити:

– проект не відновлювався вже 3 роки.

– сайт не працює.

– дизайн не відповідає сучасним трендам у розробці мобільних додатків.

– служба підтримки не відповідає на запити.
– під час тестування виявилися деякі програмні недоліки.

Із виявлених недоліків можна зазначити що велика кількість систем не задовольняють певні потреби користувача так як є продуктами корпоративними та спрямовані на взаємодію з конкретними сервісами, або ж для їх використання необхідно заключати контракти із власниками що є досить великою перешкодою на шляху їх використання. Окремі особливості усіх систем є досить вдалими проте варто зазначити що всі вони не містяться в дному проекті.

1.3 Визначення вимог до системи

Мета системи валідації квитків дозволити користувачу або команді користувачів автоматизувати перевірку раніше згенерованих квитків на певну подію, звести рівень безбілетників до нуля.

Для цього повинні бути реалізовані наступні можливості.

Загальні можливості системи:

– можливість здійснювати зчитування даних з квитка.
– можливість відмічення від сканованого квитка як використаного.
– можливість перегляду історії від сканованих квитків.

Функціонал перевірки квитків

– користувач перевіряючий повинен мати змогу увійти у додаток використавши свої регістраційні дані;
– під час сканування працівник повинен бачити ФІО власника квитка та інші дані пов язані з квитком;
– після збереження даних про від сканований квиток квиток позначається на сервері як використаний

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ПРОЕКТУ

2.1 Серверний сервіс

Даний розділ описує програмні засоби які були використані при розробці клієнтської частини проекту

2.1.1 Система керування базами даних PostgreSQL

PostgreSQL створена на основі некомерційної СУБД Postgres, розробленої як open-source проект в Каліфорнійському університеті в Берклі. До розробки Postgres, що почалася в 1986 році, мав безпосереднє відношення Майкл Стоунбрейкер, керівник більш раннього проекту Ingres, на той момент вже придбаного компанією Computer Associates. Назва розшифровується як «Post Ingres», і при створенні Postgres були застосовані багато вже раніше зроблені напрацювання[1][1)].

Стоунбрейкер і його студенти розробляли нову СУБД протягом восьми років з 1986 по 1994 рік. За цей період в синтаксис були введені процедури, правила, призначені для користувача типи і інші компоненти. У 1995 році розробка знову розділилася: Стоунбрейкер використовував отриманий досвід у створенні комерційної СУБД Illustra, яку просуває його власної однойменної компанією (придбаної згодом компанією Informix), а його студенти розробили нову версію Postgres – Postgres95, в якій мова запитів POSTQUEL – спадщина Ingres – був замінений на SQL.

Розробка Postgres95 була виведена за межі університету і передана команді ентузіастів. Нова СУБД отримала ім'я, під яким вона відома і розвивається в поточний момент – PostgreSQL.

[1)][1]Ткаченко, В. А. Системы управления базами данных и экспертные системы В. А. Ткаченко. 2016. 1024стр.

PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає йому деякі переваги над іншими SQL базами даних з відкритим вихідним кодом, такими як MySQL, MariaDB і Firebird.

Фундаментальна характеристика об'єктно-реляційної бази даних – це підтримка об'єктів і їх поведінки, включаючи типи даних, функції, операції, домени і індекси. Це робить Постгрес неймовірно гнучким і надійним. Серед іншого, він вміє створювати, зберігати та видавати складні структури даних. У деяких прикладах нижче ви побачите вкладені і складові конструкції, які не підтримуються стандартними РСКБД.

2.1.2 СередарозробкиJava Virtual Machine

Java Virtual Machine (скорочено Java VM, JVM) – віртуальна машина Java – основна частина виконує системи Java, так званої Java Runtime Environment (JRE). Віртуальна машина Java виконує байт-код Java, попередньо створений з вихідного тексту Java-програми компілятором Java (javac). JVM може також використовуватися для виконання програм, написаних на інших мовах програмування. Наприклад, вихідний код на мові Ada може бути откомпилирован в байт-код Java, який потім може виконатися за допомогою JVM [2][2)].

JVM є ключовим компонентом платформи Java. Так як віртуальні машини Java доступні для багатьох апаратних і програмних платформ, Java може розглядатися і як сполучна програмне забезпечення, і як самостійна платформа.

Використання одного байт-коду для багатьох платформ дозволяє описати Java як «скомпільовано одного разу, запускається всюди»

[2)][2] Эккель Б. Философия Java уч. пособние / Эккель.Б:СПБ: Издательский дом «Питер», 2000. 1168 с.

2.1.3 Огляд фреймворку SpringFramework

Spring Framework (або коротко Spring) – універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Також існує форк для платформи .NET Framework, названий Spring.NET.

Перша версія була написана Родом Джонсоном, який вперше опублікував її разом з виданням своєї книги «Expert One-on-One Java EE Design and Development» (Wrox Press, жовтень 2002 року).

Фреймворк був вперше випущений під ліцензією Apache 2.0 license вчервні 2003 року. Першастабільнаверсія 1.0 булавипущенавберезні 2004. Spring 2.0 буввипущенийвжовтні 2006, Spring 2.5 –влистопаді 2007, Spring 3.0 вгрудні 2009, і Spring 3.1 вгрудні 2011. Поточна версія – 5.1.2.

Незважаючи на те, що Spring не забезпечував якусь конкретну модель програмування, він став широко поширеним в Java-співтоваристві головним чином як альтернатива і заміна моделі Enterprise JavaBeans. Spring надає велику свободу Java-розробникам в проектуванні; крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу.

Тим часом, особливості ядра Spring застосовні в будь-якому Java-додатку, і існує безліч розширень і удосконалень для побудови веб-додатків на Java Enterprise платформі. З цих причин Spring набув великої популярності і визнається розробниками як стратегічно важливий фреймворк[3][1)].

Spring забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники і організації, які хочуть створити інформаційну систему, засновану на платформі Java. Через широку функціональності важко визначити найбільш значущі структурні елементи, з яких він складається. Spring не по-

---

[1)][3] Карпов Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001.304 с.

вністю пов'язаний з платформою Java Enterprise, незважаючи на його масштабну інтеграцію з нею, що є важливою причиною його популярності.

Spring, ймовірно, найбільш відомий як джерело розширень (features), потрібних для ефективної розробки складних бізнес-додатків поза великовагових програмних моделей, які історично були домінуючими в промисловості. Ще одне його достоїнство в тому, що він ввів раніше невикористовувані функціональні можливості в сьогоднішні панівні методи розробки, навіть поза платформи Java.

Цей фреймворк пропонує послідовну модель і робить її придатною до більшості типів додатків, які вже створені на основі платформи Java. Вважається, що Spring реалізує модель розробки, засновану на кращих стандартах індустрії, і робить її доступною в багатьох областях Java.

2.1.4 Стандарт взаємодії з базами даних Java DataBase Connectivity

JDBC (англ. Java DataBase Connectivity – з'єднання з базами даних на Java) – переносних незалежний промисловий стандарт взаємодії Java-додатків з різними СУБД, реалізований у вигляді пакета java.sql, що входить до складу Java SE.

JDBC заснований на концепції так званих драйверів, що дозволяють отримувати з'єднання з базою даних по спеціально описаного URL. Драйвери можуть завантажуватися динамічно (під час роботи програми). Завантажившись, драйвер сам реєструє себе і викликається автоматично, коли програма вимагає URL, що містить протокол, за який драйвер відповідає[4][1)].

Sun Microsystems випустила JDBC як частина JDK 1.1 19-го лютого 1997 року. З тих пір він є частиною JSE. Класи JDBC знаходяться в пакетах

[1)] [4]Информационное издание Devcolibri. URL: https://devcolibri.com. (дата звернення 09.03.2019).

java.sql і javax.sql. Починаючи з версії 3.1, JDBC розроблявся в рамках Java Community Process (JCP), який включає в себе стандартизовані специфікації для Java розробників.

Перевагами JDBC вважають:

- легкість розробки: розробник може не знати специфіки бази даних, з якою працює;
- код практично не змінюється, якщо компанія переходить на іншу базу даних (кількість змін залежить виключно від відмінностей між діалектами sql);
- не потрібно встановлювати громіздку клієнтську програму;
- до будь-якої бази можна під'єднатися через легко описуваний url.

2.2 Клієнтський сервіс

Даний розділ описує програмні засоби які були використані при розробці клієнтської частини проекту

2.2.1 Особливості мови програмування JAVA

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи де-

яких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Цього можна досягти, компілюючи початковий Java код у байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми[5][1)].

Основна перевага використання байт-коду – це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Про-

---

[1)][5] Шилдт Гебрерт. Java 8 полное руководство. уч. Пособие:М:Вильямс, 2016. 720 с.

те, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just-in-time компіляція (JIT, що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його. У результаті така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про завантажені класи. До того ж він може виявляти так звані гарячі точки (англ. hot spots) – частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція ahead-of-time (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

На противагу C++, Java є більш об'єктно-орієнтованою. Всі дані і дії групуються в класи об'єктів. Виключенням з повної об'єктності є примітивні

типи (int, float тощо). Це було свідомим рішенням проектувальників мови задля збільшення швидкості. Через це Java не вважається повністю об'єктно-орієнтовною мовою.

У Java всі об'єкти є похідними від головного об'єкта (він називається просто Object), з якого вони успадковують базову поведінку і властивості.

Хоча у C++ вперше стало доступне множинне успадкування, але у Java можливе тільки одинарне успадкування, завдяки чому виключається можливість конфліктів між членами класу (методи і змінні), які успадковуються від базових класів.

Java використовує автоматичний збирач сміття для керування пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті. Проте, витік пам'яті все ж може статися, якщо код, написаний програмістом, має посилання на вже непотрібні об'єкти, наприклад на об'єкти, що зберігаються у діючих контейнерах.

Збирання сміття дозволене у будь-який час. В ідеалі воно відбувається під час бездіяльності програми. Збірка сміття автоматично форсується при нестачі вільної пам'яті в купі для розміщення нового об'єкта, що може призводити до кількасекундного зависання.

Java не має підтримки вказівників у стилі C/C++. Це зроблено задля безпеки й надійності, аби дозволити збирачу сміття переміщувати вказівникові об'єкти.

2.2.2 Мобільна операційна система Android

Android, Inc. була заснована в Пало-Альто, у Каліфорнії, у жовтні 2003 року Енді Рубіном (співзасновник компанії Danger), Річардом Майнером (співзасновник Wildfire Communications, Inc.), Ніком Сірсом (колишній віце-

президент компанії T-Mobile) і Крісом Уайтом (очолював дизайн і розробку інтерфейсу в WebTV) для розробки, за словами Рубіна «більш розумних мобільних пристроїв, які краще знають про місце перебування власника і його вподобання». Ранні наміри компанії полягали в тому, щоб розробити вдосконалену операційну систему для цифрових фотоапаратів, але було зрозуміло, що ринок пристроїв не був достатньо великим, і вони спрямували свої зусилля на розробку операційної системи для смартфонів, щоб конкурувати з Symbian і Windows Mobile (на той момент Apple ще не випустила iPhone). Незважаючи на минулі досягнення засновників і ранніх співробітників, Android, Inc. працювала таємно, показуючи тільки те, що вона працювала над програмним забезпеченням для мобільних телефонів. Цього ж року Рубін залишився без грошей. Стів Перлман, близький друг Рубіна, приніс йому 10 тис. доларів готівкою у конверті та відмовився від своєї частки в компанії.

Перша версія Android була випущена 23 вересня 2008 року і носила назву 1.0 Astroboy, а наступна – 1.1 Bender. Від назв у честь відомих роботів згодом довелося відмовитися через розбіжності з правовласниками. З 2008 року Android пережив численні оновлення, які поступово покращували операційну систему, додаючи нові функції, та виправляли помилки у попередніх випусках. І тепер кодове ім'я кожного великого релізу Android, починаючи з версії 1.5, являє собою назву якого-небудь десерту[6][1)].

Основні характеристики системи :

Інструментарій. Платформа легко пристосовується для використання VGA, бібліотек двовимірної і тривимірної графіки, розроблених на основі OpenGLES 1.0-3.1 специфікації, традиційних інструментаріїв для смартфонів.

Бази даних. SQLite для структурованих даних.

Технології зв'язку. Android підтримує багато технологій, що забезпе-

---

[1)][6] Информационный портал Habr. URL: https://habr.com/ru/.(дата звернення 09.03.2019).

чують зв'язок, у тому числі: GSM, Bluetooth, EDGE, 3G та WiFi.

Обмін повідомленнями .Для обміну повідомленнями доступні як SMS, так і MMS сервіси, у тому числі й потокові повідомлення.

Веб-браузери. На Android доступний веб-браузер, розроблений на основі WebKit application framework.

Java Virtual Machine. Програми, написані на Java, можна скомпілювати в Dalvik байткод і виконувати на Dalvik virtual machine, яка являє собою розроблену спеціально для використання на мобільних пристроях віртуальну машину, незважаючи на те, що не є стандартною Java Virtual Machine.

Підтримка медіа. Android підтримує такі формати для аудіо/відео даних та зображень: MPEG-4, H.264, MP3, та AAC, AMR, JPG, PNG, GIF.

Підтримка нестандартного обладнання. Android підтримує відеокамери, фотоапарати, дотикові екрани, GPS, компаси, акселерометри, та прискорювачі 3D графіки.

Середовище розробки. Офіційним середовищем розробки є Android Studio, створене на базі Intellij IDEA. Містить емулятор, засоби відлагодження, профілювання пам'яті та швидкодії. Також доступні плагіни для Intellij IDEA, Eclipse та NetBeans.

Сирцевий код. 21 жовтня 2008 альянс OHA опублікував сирцевий код платформи Android. До релізу увійшов весь стек Android: і операційна система, і проміжне ПЗ (middleware), і основні кінцеві застосунки, написані на Java. Загальний обсяг сирцевого коду Android склав 2,1 Гб.

В травні 2010 року був досягнутий рубіж у 100 тисяч активацій в день. В грудні 2010 в день активувалося вже 300 тисяч апаратів. У травні на конференції Google I/O була озвучена статистика, згідно з якою щодня активується близько 400 тисяч нових пристроїв на базі платформи Android. В липні 2011 Енді Рубін (Andy Rubin), віце-президент Google, який відповідає за розробку платформи Android, повідомив про подолання нової межі – 500 тисяч активацій на день при зростанні поширення платформи в 4,4 % на тиждень. У статистику активацій включена тільки інформація про першу реєстрацію но-

вих пристроїв, що поставляються з набором сервісів Google. Апарати з прошивками без набору застосунків Google, що випускаються деякими азійськими виробниками, у статистиці не враховані.

Продаж перших смартфонів на платформі Android в Україні розпочався 15 січня 2009 р. Першим офіційним смартфоном на платформі Android в Україні вважається HTC Hero, хоча зустрічаються різні дати його появи в Україні: 28 вересня 2009 р., 19 жовтня 2009 р., листопад 2009 р. (в той же час офіційною датою власного виходу на український ринок корпорація HTC назвала 25 травня 2010 року, коли було представлено Android-смартфони HTC Legend, HTC Desire, HTC Wildfire). Також HTC Hero присутній на українському ринку як операторське рішення щонайменше з 1 березня 2010 р. Другим офіційним смартфоном вважається Samsung Galaxy Spica, що був представлений 19 листопада 2009 р. і надійшов у продаж у грудні-місяці. Першим офіційним Android-смартфоном з підтримкою технологій CDMA/EV-DO став операторський Huawei IDEOS C8150.

Появу Google Play на офіційних пристроях, що ввозяться до України (з початку виробництва Samsung, потім HTC), компанія «Андроїд Україна» прогнозувала на кінець січня 2010 р. 13 січня 2010 р. Samsung оголосила про початок поставок в Україну смартфону Galaxy Spica i5700 з доступом до Android Market для українців. 8 грудня 2010 поширено інформацію про те, що платна версія Android Market для України знаходиться в стадії публічного тестування.

12 травня 2011 можливість купувати застосунки в Google Play стала доступною мешканцям України. Всі ціни платних застосунків для українців вказані в гривнях. Для здійснення оплати необхідна платіжна картка, зареєстрована в системі Google Checkout.

Загалом станом на липень 2011 продано вже понад 100 млн Android-пристроїв, випущених 36 виробниками, їх розповсюджували у мережах 215 операторів зв'язку. Загальне число моделей пристроїв на базі платформи Android досягло 310. Всього було продано більш ніж 200 млн Android-

пристроїв. Каталог Google Play подолав позначку в 200 тисяч застосунків. Всього з Google Play встановлено близько 4.5 мільярдів копій програм.

За даними Google на вересень 2015 року активовано 1.4 мільярдів «Андроїдів»

2.2.3 Середа розробки Android SDK

Android SDK – універсальний засіб розробки мобільних додатків для операційної системи Android. Відмінною рисою від звичайних редакторів для написання кодів є наявність широких функціональних можливостей, що дозволяють запускати тестування і налагодження вихідних кодів, оцінювати роботу програми в режимі сумісності з різними версіями ОС Android і спостерігати результат в реальному часі (опціонально). Підтримує велику кількість мобільних пристроїв, серед яких виділяють: мобільні телефони, планшетні комп'ютери, розумні окуляри (в тому числі Google Glass), сучасні автомобілі з бортовими комп'ютерами на ОС Андроїд, телевізори з розширеним функціоналом, особливі види наручних годинників і багато інших мобільні гаджети, габаритні технічні пристосування[7][1)].

2.2.4 Середа розробки Android Studio

Android Studio прийшло на зміну плагіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA Community Edition, що розвивається компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0.

Бінарні складання підготовлені для Linux (для тестування використа-

---

[1)][7] Библиотека Proglib. URL: https://proglib.io. (дата звернення 08.03.2019).

ний Ubuntu), Mac OS X і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для носимих пристроїв на базі Android Wear, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto). Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio[8][1)].

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. У тому числі у середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції.

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. У середовище вбудовані функції завантаження типових прикладів коду з GitHub.

[1)][8] Форум Medium. URL: https://medium.com/androiddevelopers/. (дата звернення 08.03.2019).

2.2.5 Об'єктно-орієнтоване програмування

Об'єктно-орієнтоване програмування – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua) або ж є цілком об'єктно-орієнтованими (зокрема, Java, C#, C++, Python, Ruby і Objective-C, ActionScript 3, Swift, Vala).

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-тих роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Разом із тим, як ускладнювалось апаратне та програмне забезпечення, було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт – своєрідний незалежний автомат з окремим призначенням та відповідальністю.

2.2.6 Бібліотека Retrofit для роботи з мережею

Багато сайтів мають власні API для зручного доступу до своїх даних. На даний момент найпоширеніший варіант – це JSON. Також можуть зустрічатися дані у вигляді XML і інших форматів.

Бібліотека Retrofit спрощує взаємодію з REST API сайту, беручи на себе частину рутинної роботи.

Бібліотекою зручно користуватися для запиту до різних веб-сервісів з командами GET, POST, PUT, DELETE. Може працювати в асинхронному режимі, що позбавляє від зайвого коду.

2.2.7 Бібліотека для роботи з SQLite базою даних ROOM

Робота з базою даних SQLite в Android не відрізнялася зручністю, доводилося писати багато рядків коду, перш ніж приступити безпосередньо до програмування своєї логіки. Сторонні розробники раніше пропонували власні варіанти для зменшення рутинної роботи програміста. Найбільш популярними стали такі бібліотеки як Realm, ORMLite, GreenDao, DBFlow, Sugar ORM.Гугл запропонувала своє рішення – Room, який входить до складу Android Architecture Components і є обгорткою над SQLiteOpenHelper, тобто ORM (Object-relational mapping) між класами Java і SQLite.

2.2.8 Бібліотека Zxing для опрацювання баркодів

ZXing ("зебра") – це бібліотека обробки зображень штрих-кодів, реалізована на Java, з портами на інші мови. Вона підтримує стандарти 1D, і 2D штрих-кодів.

2.2.9 Розши́рювана мо́ва розмі́тки Extensible Markup Language

Розши́рювана мо́ва розмі́тки (англ. Extensible Markup Language, скорочено XML) – запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Є спрощеною підмножиною мови розмітки SGML. XML-документ складається із текстових знаків, і придатний до читання людиною.

Стандарт XML (англ. Recommendation, перше видання від 10 лютого 1998, останнє, четверте видання 29 вересня 2006) визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів.

3 РЕАЛІЗАЦІЯ КЛІЄНТСЬКОГО ПРОГРАМНОГОЗАБЕЗПЕЧЕННЯ ДЛЯ ВАЛІДАЦІЇ КВИТКІВ

3.1 Моделювання додатку

UML (англ. Unified Modeling Language – уніфікована мова моделювання) – мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення, для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

UML є мовою широкого профілю, це – відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, званої UML-моделлю. UML був створений для визначення, візуалізації, проектування та документування, в основному, програмних систем. UML не є мовою програмування, але на підставі UML-моделей можлива генерація коду.

Модель представляється у вигляді сутностей і відношень між ними, які показуються на діаграмах.

Сутності – це абстракції, які є основними елементами моделей. Є чотири типи сутностей – структурні (клас, інтерфейс, компонент, варіант використання, кооперація, вузол), поведінкові (взаємодія, стан), групуються (пакети) і анотаційні (коментарі). Кожен вид сутностей має своє графічне представлення. Суті будуть детально розглянуті при вивченні діаграм.

При створенні діаграм використання спочатку визначаються виконавці (ролі, користувачі). Виконавець (actor) є зовнішнім по відношенню до системи поняттям, яке певним чином участь в процесі, описуваному прецедентом. Актор є якийсь ідеалізацією намірів користувача, а не самого користувача. Один реальний користувач може використовуватися декількома акторам, а один актор може представляти один і той же намір відразу декількох користувачів.

3.2 Проектування діаграми прецедентів

Поведінка, розробляється (тобто функціональність, що забезпечується системою) описується за допомогою моделі прецедентів, яка відображає системні прецеденти (use cases), системне оточення (дійових осіб або акторів – actors) і зв'язку між прецедентами і акторами (діаграми прецедентів – use cases diagrams) . Головною метою даної моделі є реалізація собою єдиного засобу для взаємодії розробника з замовником і кінцевим користувачем для обговорення функціональної складової та поведінки системи.

На основі цього типу діаграм проводиться ітераційний цикл загальної постановки завдання разом із замовником. Оскільки замовник ніколи не буде точно знати чого він хоче, то діаграми прецедентів якраз і є основою для досягнення взаєморозуміння між програмістами-професіоналами, які розробляють проект, і "бізнесменами"– замовниками проекту. Ці діаграми описують функціональність ІС, яка буде видна користувачам системи, основні функції, які повинні бути включені в систему (use case), їх оточення (actors). "Кожна функціональність" зображується у вигляді "прецедентів використання" (use case) або просто прецедентів. Прецедент – це типова взаємодія користувача з системою, при цьому:

- описує видиму користувачем функцію;
- може представляти різні рівні деталізації;
- забезпечує досягнення конкретної мети, важливої для користувача.

Актори не є частиною системи, вони використовують систему. Актори можуть:

- тільки постачати інформацією систему;
- тільки отримувати інформацію з системи;
- забезпечувати інформацією і отримувати інформацію з системи.

Актор, який представляє людини-користувача, характеризується роллю в даному прецеденті. На діаграмі зображується тільки один актор, проте, реальних користувачів може бути багато. Актори визначаються на основі скла-

деного завдання або в процесі обговорення з замовником.

Спроектована діаграма прецедентів згідно додатку, що розроблюється, зображена на рис. 6:

Рисунок 6– Діаграма прецедентів для перевіряючого

Як можна побачити перевіряючий має досить невеликий набір забов’язань.

3.3 Проектування діаграми потоків даних

Діаграма потоків даних – модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма потоків даних також може використовуватись для візуалізації процесів обробки даних.

З точки зору розробки ПЗ, зазвичай, спочатку потрібно креслити

діаграму потоків даних рівня контексту, завдяки чому буде показано взаємодію системи із зовнішніми модулями. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати детально всю розроблювану систему.

Діаграми потоків даних містять чотири типи графічних елементів:

– сховища даних (репозиторії);

– зовнішні по відношенню до системи сутності;

– потоки даних між елементами трьох попередніх типів.

– процеси - являють собою трансформацію даних в рамках описуваної системи;

Зовнішня сутність позначається прямокутником з тінню, всередині якого вказується її ім'я. При цьому в якості імені рекомендується використовувати іменник в називному відмінку. Іноді зовнішню сутність називають також термінатором.

Спроектована діаграма потоків даних наведена на рис. 7:

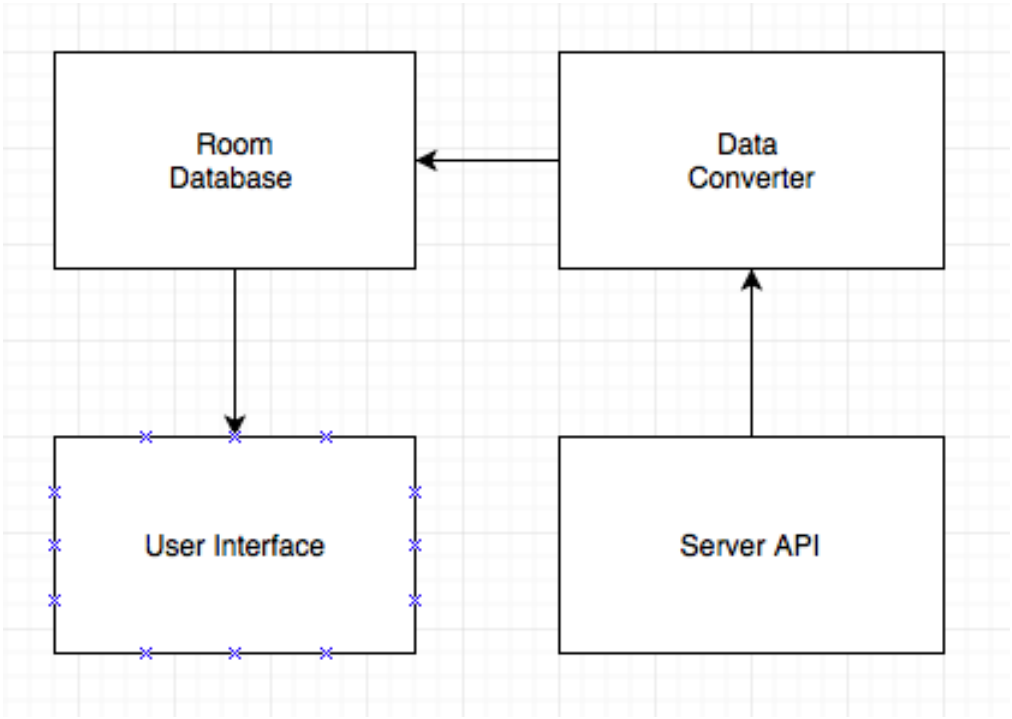

Рисунок 7– Діаграма потоків даних

Дана схема демонструє загальних принцип надходження даних у додаток. Server API надає дані у вигляді Json об'єктів які, пройшовши конвертацію потрапляють у базу даних, із якої вже загружаються в адаптер і займається безпосередньо їх відображенням у користувацькому інтерфейсі.

3.4 Функціональна схема додатку

Додатки для Android складаються з компонентів, які система може запускати і керувати так, як їй необхідно. Головна особливість платформи Android полягає в тому, що один додаток може використовувати елементи інших додатків (за умови, що ці програми дозволяють їх використовувати). При цьому ваш додаток не включає код іншого додатка або посилання на нього, а просто запускає потрібний елемент іншої програми.

Для реалізації такого використання компонентів інших додатків система повинна бути в змозі запустити процес для додатка, в якому знаходиться необхідний компонент, і форматувати потрібні їй об'єкти. Тому, на відміну від додатків в більшості інших систем, у додатків Android немає єдиної точки входу для запуску всього програми, аналогічної, наприклад, функції main () в подібних мовах програмування. Android-додатки складаються з компонентів, які система може ініціювати і запустити при необхідності.

Всього в Android додатках існує чотири типи компонентів:

– діяльність (Activity);
– служба (Service);
– приймач широкомовних намірів (Broadcast Receiver);
– контентпровайдер (Content Provider).

Не кожен додаток повинен містити всі чотири блоки, але він буде використовувати деяку їх комбінацію.

Діяльність є візуальний інтерфейс для програми – вікно. Як правило, вікно повністю заповнює екран мобільного пристрою, але може мати розміри менше, ніж у екрану. Діяльність може також використовувати додаткові вікна, наприклад спливаюче діалогове вікно, яке запитує користувальницький відповідь для основної діяльності, або вікно повідомлення про какомлибо подію в додатку або системі.

Візуальне інформаційне наповнення вікна визначається ієрархією уявлень. Кожна вистава управляє заданим прямокутним простором в межах вік-

на. Батьківські уявлення містять і організовують схему розміщення своїх чірніх уявлень. Уявлення також можуть реагувати на призначене для користувача взаємодія з інтерфейсом програми. У Android є багато готових уявлень для використання в додатках – кнопки, текстові поля, лінійки прокрутки, пункти меню, прапорці і т. Д.

Все діяльності реалізуються як підклас базового класу Activity.

Додаток може містити кілька діяльностей. Кожна діяльність незалежна від інших. При відкритті нової діяльності робота попередньої діяльності призупиняється, а сама вона вноситься і зберігається в стек діяльностей.

Клас View являє собою основний будівельний блок для компонентів для користувача інтерфейсу. Вид займає прямокутну область на екрані і відповідає за малювання і обробку подій. Вид - це базовий клас для віджетів, які використовуються для створення компонентів інтерактивного інтерфейсу (кнопки, текстові поля і т. Д.).

ViewGroup – це спеціальне подання, яке може містити інші уявлення (звані ViewGroup). Група уявлень є базовим класом для контейнерів макетів і уявлень. Цей клас також визначає клас ViewGroup.LayoutParams який слугує базовим класом для параметрів макетів.

Принципова схема роботи ViewGroup показана на рис. 8:

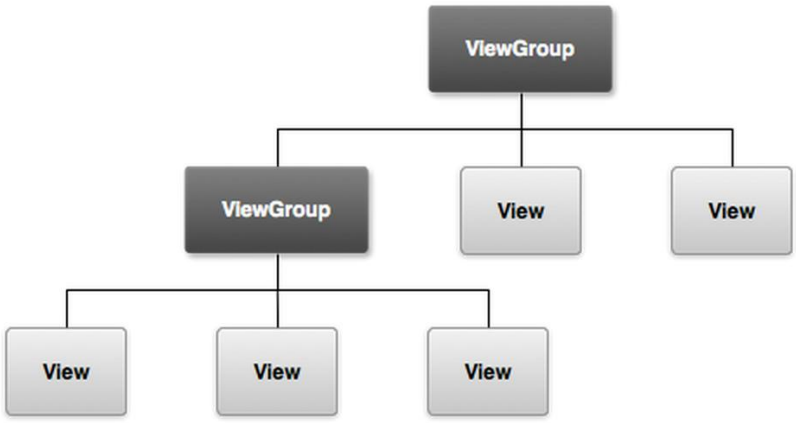

Рисунок 8 – Структура інтерфейсу Android

Контентпровайдер робить певний набір даних, використовуваних додатком, доступним для інших додатків.

Дані можуть бути збережені в файлової системі, в базі даних SQLite або будь-яким способом. Контентпровайдери для безпечного доступу до даних використовують механізм дозволів. Це означає, що ви можете конфігурувати власні контентпровайдери, щоб дозволити доступ до своїх даних з інших додатків і використовувати контентпровайдери інших додатків для звернення до їх сховищ даних.

Служба не має візуального інтерфейсу користувача і виконується в фоновому режимі протягом невизначеного періоду часу. Додатки можуть підключатися до служби або запускати її, якщо вона не запущена, а також завершувати роботу запущених служби. Підключившись до служби, ви можете звертатися до її функцій через інтерфейс, що надається цією службою.

Додаток використовує такі активності та фрагменти:

- «LogInActivity» – активність яка на яку потрапляє користувач для того щоб авторизуватись у додатку;
- «PremierActivity» – активність де запрошується дозвіл на використання необхідних залежностей та відбувається ініціалізація трьох основних фрагментів додатку;
- «ScannerFragment» – фрагмент в рамках якого безпосередньо відбувається сканування квитка на який відображає останні вісім відсканованих квітків;
- «HistoryFragment» – фрагмент який має повну історію відсканованих квитків;
- «SettingsFragment» – фрагмент з налаштуваннями. Містить такі елементи як : вихід з облікового запису, про додаток та використані бібліотек;

До цих активностей та фрагментів додаток містить відповідні макети:

- «activity_registration» – макет, що відповідає за формування авторизаційної сторінки;

– «activity_premier» – макет, що відповідає за формування сторінки «PremierActivity»;

– «scanner_fragment» – макет, що відповідає за формування «ScannerFragment»;

– «history_fragment» – макет, що відповідає за формування сторінки «HistoryFragment »

– «settings_fragment» – макет, що відповідає за формування сторінки «SettingsFragment »

3.5 Алгоритм роботи додатку

Користувач перший раз відкриває додаток. Перше що він бачить це екран входу у свій обліковий запис.

Для доступу до основної функції необхідно авторизуватися. Вписавши свої дані користувач потрапляє на головний екран – екран зі сканером.

При наступних входах у додаток вже не треба вводити свої авторизаційні дані – вони вже збережені. Перед початком роботи треба надати додатку необхідні для його роботи залежності. В даному випадку це доступ до камери та доступ до зберігання даних на внутрішній носій. Після надання необхідних залежностей додаток стає повністю придатним для використання.

Для реалізації сканера використана бібліотека ZXing. Ця бібліотека дозволяє зчитувати дані з баркодів та QR-кодів легко та надійно. Наступний екран – історія з усіма перевіреними квитками. Дані для заповнення цього екрану беруться з бази даних Room. Наступний екран це єкран інформації та налаштувань. Тут користувач може видалити всі збережені дані та вийти з облікового запису, подивитися основні відомості про додаток та знайти інформацію щодо використаних при розробці додатку бібліотек.

3.6 Розробка зовнішнього вигляду додатку

Розробка зовнішнього вигляду додатку відбулася по принципам проектування Material Design, з використанням однойменної бібліотеки.

Material Design випливає з ідеї вільного переміщення в потрібному користувачеві напрямку і базується на тактильній реальності.

Спочатку його розробники надихалися звичайним чорнилом і папером, але з тих пір він, звичайно, далеко пішов від цієї фундаментальної особливості, набуваючи рис справжнього чаклунства.

Іншими словами, удосконалюючи візуально-функціональну частину, фахівці Google об'єднав принципи гарного дизайну з інноваціями і можливостями нових технологій.Крім того, Material Design використовує принципи поліграфічного дизайну для ефектної розстановки акцентів (тобто фокусування уваги на потрібному елементі), спрощення навігації в інтерфейсі, інтуїтивної передачі сенсу його елементів.

Для Material Design характерні насичені, рівні кольори, різкі, окреслені краю, велика типографіка і чималі відступи між елементами. У сукупності ці елементи не просто складаються в приємну для очей картинку, а створюють нову реальність з концептуальним змістом і безліччю функцій, які дарують користувачеві унікальний UX.

У випадку з цим видом дизайну дії користувача знаходяться в центрі уваги. Всі взаємодії відбуваються в одному оточенні, інтерактивні об'єкти без переривання послідовності переходять з однієї середу в іншу.

Матеріальний дизайн за допомогою системи управління тінями створює візуальну ілюзію простору між додатком і екраном пристрою.

Даний додаток був розроблений відповідно до вимог зручності та функціональності, грунтуючись на UX-дизайні і естетики зовнішнього вигляду

Приклад компонентів матеріального дизайну наведені на рис. 9:

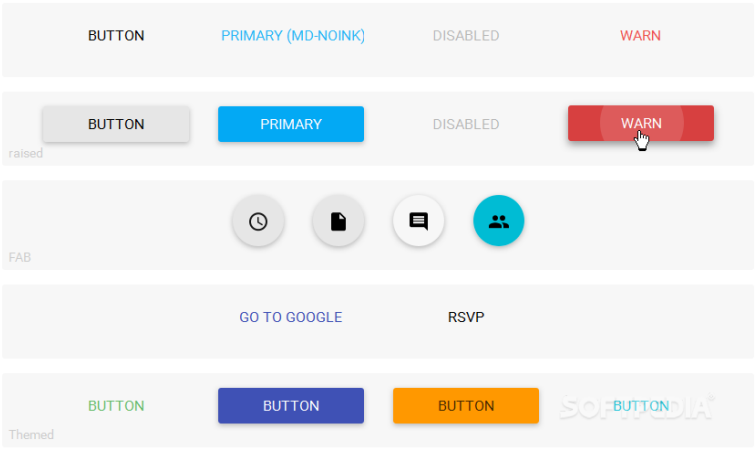

Рисунок 9– Material Design

Матеріальний дизайн за допомогою системи управління тінями створює візуальну ілюзію простору між додатком і екраном пристрою.

Даний додаток був розроблений відповідно до вимог зручності та функціональності, грунтуючись на UX-дизайні і естетики зовнішнього вигляду.

Нижче будуть наведені скріншоти екранів додатка.

На рис. 10 зображена сторінка авторизації у додаток, на яку потрапляє користувач відразу після відкриття додатку:

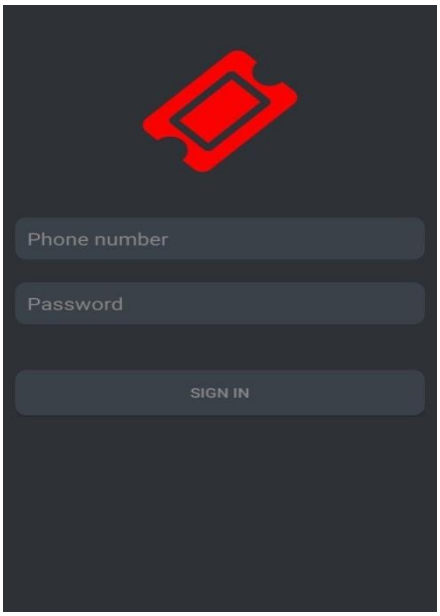

Рисунок 10– Єкран входу у додаток

На рис.11 показан екран на який потрапляє користувач відразу після авторизації:

Рисунок 11– Екран зі сканером та переліком відсканованих раніше квитків

Тут також можна бачити список який складається з нещодавно відсканованих білетів.

На наступному рисунку – рис. 12 показан діалог який показується перевіряючому коли він сканує квиток:

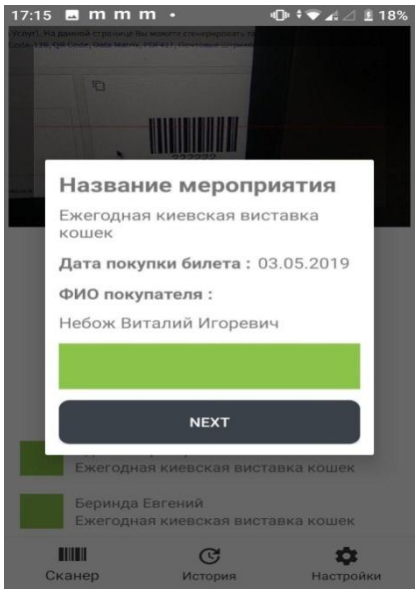

Рисунок 12– Екран з даними про квиток

Даний діалог показує необхідні дані такі як назву заходу, дата придбання квитка, ФІО покупця та колір червоний чи зелений в залежності від того чи проходить квиток валідацію чи ні.

На рис. 13 демонструє розгорнутий список історії відсканованих квитків:

Рисунок 13– Екран з розширеною історією

3.7Розробка логіки додатку

Дана частина присвячена розробці логіки додатку та насамперед отриманню та обробці даних з серверу. Представлені фрагменті сирцевого коду є найголовнішими функціональніми частинами програми.

Для роботи додатка потрібно створити APIінтерфейс із запитами до серверу. Реалізація цього класу представлена нижче.

```
public interface VerifierApi {
    @POST("ticket/setEntered/{ticketHash}")
    Single<EnteredResponse>   setEntered(@Path("ticketHash")
String ticketHash);
    @GET("user/check")
    Single<String> checkCredentials();
}
```

Наступний крок це строкення саме об єкту API для здійснення запитів. Сирцевий код приведено нижче.

```
private VerifierApi createAuthApi(String username, String
 password) {
List<Interceptor> interceptors = new
 ArrayList<Interceptor>() {{
add(new BasicAuthInterceptor(username, password));
}};
return RetrofitService.createApi(VerifierApi.class,
 endpoint,
true, interceptors);
}
```

Наступний крок полягає у ініціалізації та конфігуванні бази даних ROOM.Сирцевий код наведено нижче.

```
@Database(entities   =   {Ticket.class},   version   =   1,
 exportSchema = true)
public abstract class AppDatabase extends RoomDatabase {
    public abstract TicketDao ticketDao();
}
```

Далі необхідно вирішити питання ініціалізації всіх основних програмних модулів, а саме, проініціалізувати їх до того моменту як вони будуть ви-

користані в UI. Ця задача була вирішена шляхом додавання необхідних трукцій у модуль App[9][1)].

Реалізація цього класу наведена нижче.

```
public class App extends Application {

public static App instance;
private AppDatabase database;
private VerifierApi api;
    @Override
public void onCreate() {
super.onCreate();
instance = this;
database  =  Room.databaseBuilder(this,  AppDatabase.class,
"database")
                  .allowMainThreadQueries().build();
    }
public static App getInstance() {}
public AppDatabase getDatabase() {}
public VerifierApi getApi() {}
public void setApi(VerifierApi api) {}

}
```

Нижче буде представлений клас головної активності додатку яка керує поведінкою усіх причетних но неї фрагментів.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.premier_activity);
    BottomNavigationView navigationView =
findViewById(R.id.bottom_navigation);
navigationView.setOnNavigationItemSelectedListener(navigati
onListener);
getSupportFragmentManager().beginTransaction().replace(R.id.prem
ier_container, new ScannerFragment()).commit();}
BottomNavigationView.OnNavigationItemSelectedListener
navigationListener = menuItem -> {
    Fragment currentFragment = null;
    switch (menuItem.getItemId()) {
        case R.id.navigation_scan:
            currentFragment = new ScannerFragment();
            break;
        case R.id.navigation_history:
            currentFragment = new HistoryFragment();
     break;
        case R.id.navigation_settings:
            currentFragment = new SettingsFragment();
```

[1)][9] Уроки программирования от StartAndroid. URL :https://startandroid.ru/ru/.(дата звернення 08.03.2019).

```
                break;
```

В залежності від обороного користувачем єкрану, екземпляр цього єкрану і використовуються для будування інтерфейсу. Такий підхід дозволяє значно економити розмір використаної ОЗУ.

Також приводиться код діалогу який бачить користувач після від сканування квитку. Діалог не є активністю а складає частину окремого підвиду View.

```
        public TicketDialog (Activity parent, Handler
reponseHandler, Ticket ticket){
            mResponseHandler = reponseHandler;
            Dialog dialog = new Dialog(parent);
            dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
            dialog.setContentView(R.layout.alert_dialog);
            findViews(dialog);
            fillFields(ticket);
            dialogButton.setOnClickListener(v -> {
                mResponseHandler.sendEmptyMessage(dlgResultOk);
                dialog.dismiss();
            });
            dialog.show();
        }
        private void fillFields(Ticket ticket){
            eventName.setText(ticket.getEvent().getName());

buyDate.setText(Utils.dateConverter(ticket.getBuyDate()));
            holderName.setText(ticket.getHolder().getName());
        }
```

Дані для відображення надходять з головного єкрану шляхом передачі у конструктор класу об'єкту Ticket.

Для відображення даних з бази даних використовується адаптер, код якого наведений нижче. Адаптер отримує дані, строює та заповнює на основі цих даних складові інтерфейсу та повертає їх у Activity.

```
        public class TicketsRecyclerAdapter extends
RecyclerView.Adapter<RecyclerView.ViewHolder> {
        private List<Ticket> tickets = new ArrayList<>();
        private Context context;
        public TicketsRecyclerAdapter(Context context,
List<Ticket> tickets) {
        this.context = context;
        this.tickets = tickets;}

        PublicRecyclerView.ViewHolder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
```

4 РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ДОДАТКУ

4.1 Опис таблиць бази даних і звязків між ними

База даних для проекту «Кліент – серверний додаток для валідації квитків» використовуються в клієнт-серверній архітектурі та знаходиться на стороні серверу. Користувач буде слати запити до сервера, які будуть оброблятися використовуючи сервером використовуючи базу даних.

Для виконання системи управління проектами була обрана база даних SQL[10][1)].

Дані для авторизації зберігаються у двох таблицях. Пароль при цьому не зберігається у відкритому вигляді а у вигляді хеш коду. Завдяки цим двум таблицям була реалізована система аутентифікації (рис. 14).

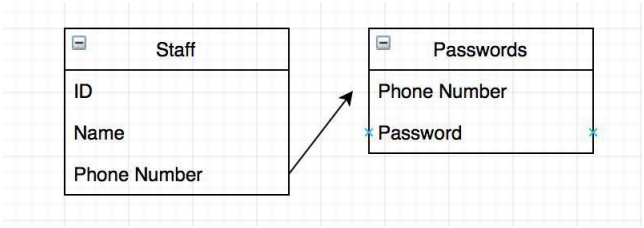

Рисунок 14 – Зв’язок для таблиці перевіряючих

Опис таблиць.

Таблиця 1– Сутність «аккаунт перевіряючого»

№	Назва таблиці	Назва колонки	Тип даних	Призначення
1	Staff	ID	int	Ідентифікатор користувача
2		Name	varchar	Логін користувача для авторизації
3		Phone Number	varchar	Номер телефону користувача

[1)][10] Справочник Stackoverflow. URL: https://stackoverflow.com/.(дата звернення 08.03.2019).

Таблиця 2 – Сутність «сховище паролів»

№	Назва таблиці	Назва колонки	Тип даних	Призначення
1	Passwords	Phone Number	int	Ідентифікатор користувача
2		Password	varchar	Логін користувача для авторизації

Також була створена таблиця для реалізації об' єкту «Білет»

Так як об' єкт «Білет» зв'язаний з іншими таблицями, цей зв'язок був показаний нижче (рис. 15):

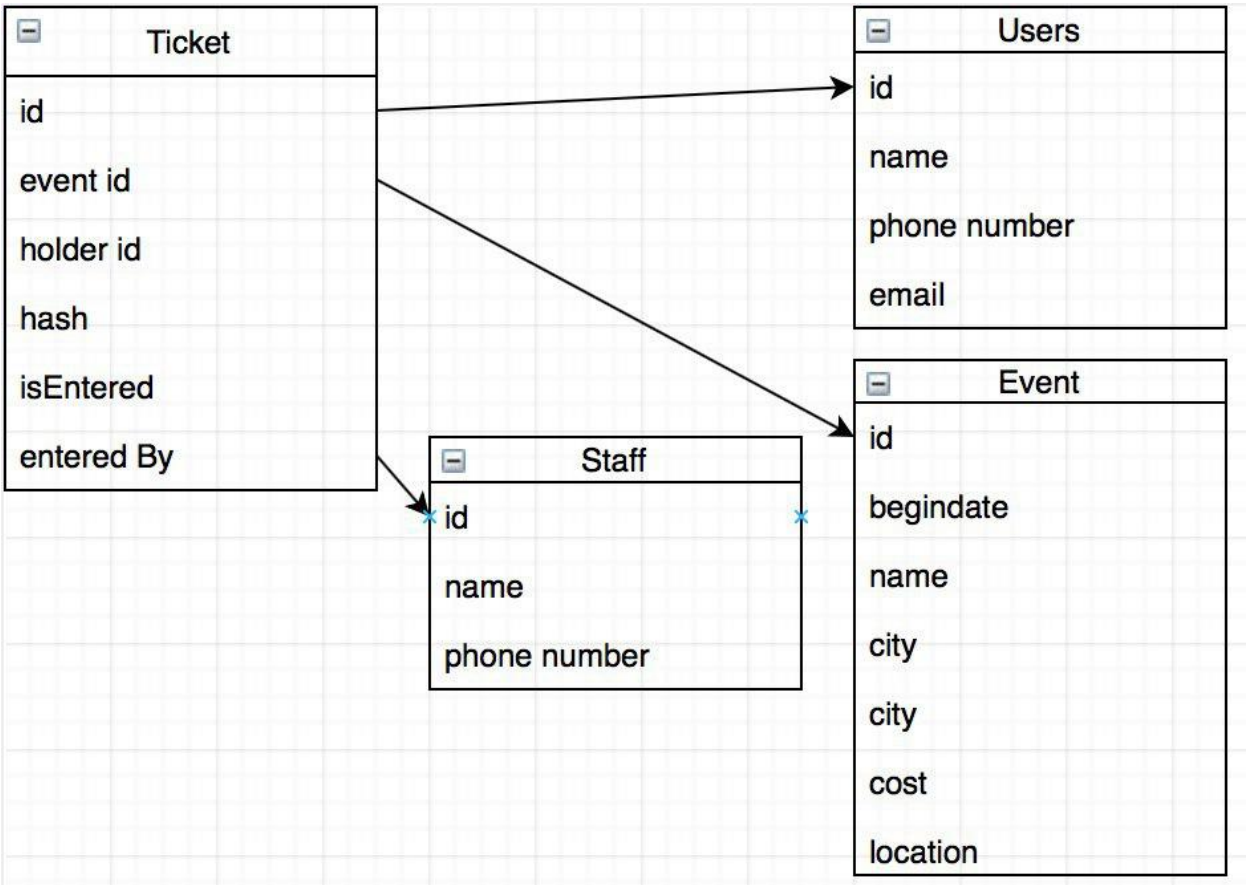

Рисунок 15 – Зв’язок для таблиці перевіряючих

Опис таблиці білету та допоміжних таблиць (табл. 3 – 5).

Таблиця 3 – Таблиця квитків

№	Назва таблиці	Назва колонки	Тип даних	Призначення
1	Ticket	id	int	id квитка
2		event id	int	Ідентифікаторзаходу
3		holder id	int	Ідентифікаторкористувача
4		hash	varchar	хєш квитка
5		isEntered	boolean	Відображає використаний квиток чи ні
6		Entered By	int	Ідентифікатор який відображає ким був перепірен квиток

Таблиця 4 – Таблиця користувачів

№	Назва таблиці	Назва колонки	Тип даних	Призначення
1	Users	id	int	id користувача
2		name	int	Ім'я користувача
3		Phone number	int	Телефон користувача
4		email	varchar	email користувача

Таблиця 5 – Таблиця подій

№	Назва таблиці	Назва колонки	Тип даних	Призначення
1	Event	id	int	Ідентифікатор події
2		begindate	date	Дата початку події
3		name	varchar	Назва події
4		city	varchar	Місто події
5		cost	int	Ціна білета на подію
6		location	date	Координати події

Усі взаємовідносини які формують функціонал системи для валідації квитків відображені на (рис. 16):

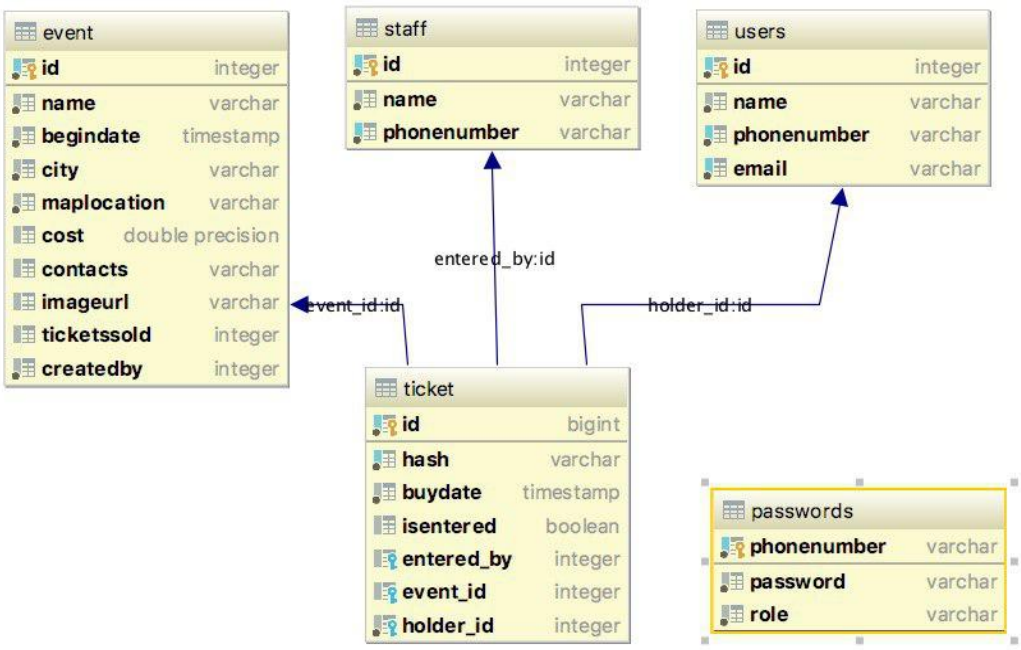

Рисунок 16 – Структура бази даних

У результаті проведеного проектування системі валідації квитків, спроектована база даних системи, яка представлена в виде діаграм «сутність-зв’язок», яка дозволяє реалізувати усі функціональні можливості яка потребує система валідації квитків, та закласти в них потенціал для розширення у майбутньому.

ВИСНОВКИ

В даній дипломній роботі було виконано проектування та розробку кліент-серверної системи для перевірки електронних квитків.

Даний програмний продукт має наступну функціональність : він забезпечує можливість автоматизованої перевірки електронних квитків.
Наявність такого продукту значно полегшує організацію невеликих публічних заходів, конференцій, концертів, тощо.

Доступність системи і можливість роботи з даними без використання спеціальних знань в програмуванні та адмініструванні дозволяє швидко імплементувати цю систему у будь-яку організацію, значно облегшивши деякі процеси, підвищити продуктивність при цьому не зазнавши будь-яких фінансових витрат.

Розроблений продукт значно відрізняється від усіх знайдених мною аналогів своєю гнучкістю, простим та збалансованим функціоналом, політикою безкоштовного розповсюдження та самим сценарієм його використання. Як вже було раніше написано, усі аналоги це або величезні приватні корпоративні продукти, розраховані на використання у рамках однієї організації (наприклад система перевірки білетів в Укрзалізниці), або агрегатори (наприклад ресурс ukrticket), які надають свої послуги на умовах контракту та відрахування проценту від обсягу продажів.

Тестування даного програмного продукту показало, що система виконує покладені на неї функції, стабільно працює та загалом придатна для використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ткаченко, В. А. Системы управления базами данных и экспертные системы В. А. Ткаченко. 2016. 1024стр.
2. Эккель Б. Философия Java уч. пособние / Эккель.:СПБ: Издательский дом «Питер», 2000. 1168 с.
3. Карпов Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001.304 с.
4. Информационное издание Devcolibri. URL: https://devcolibri.com. (дата звернення 09.03.2019).
5. Шилдт Гебрерт. Java 8 полное руководство. уч Пособие:М:Вильямс, 2016. 720 с.
6. Информационный портал Habr. URL: https://habr.com/ru/.(дата звернення 09.03.2019).
7. Библиотека Proglib. URL: https://proglib.io. (дата звернення 08.03.2019).
8. ФорумMedium. URL: https://medium.com/androiddevelopers/. (дата звернення 08.03.2019).
9. Уроки программирования от StartAndroid. URL :https://startandroid.ru/ru/.(дата звернення 08.03.2019).
10. Справочник Stackoverflow. URL: https://stackoverflow.com/.(дата звернення 08.03.2019).