МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Інститут післядипломної освіти

Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка мобільного застосунку для навігації користувачів в Одеському державному екологічному університеті

Виконав студент групи КН-5
спеціальності 122 «Комп'ютерні науки»
Пушкаренко Кирило Григорович

Керівник к.геогр.н., доцент
Кузніченко Світлана Дмитрівна

Консультант _____

Рецензент к.техн.н., доцент
Гнатовська Ганна Арнольдівна

Одеса 2022

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

ADT – Android Development Tools – Інструменти розробки в ОС Android

API – Application Programming Interface – програмний інтерфейс програми

AR – Augmented Reality – доповнена реальність

IDE – Integrated Development Environment – інтегроване середовище розробки

SDK – Software Development Kit – набір засобів розробки

ВСТУП

Інтернет послуги в сфері геоданих постійно розширюються і технологічно вдосконалюються. Сегодні користувача Інтернет неможливо представити без використання картографічних веб-сервісів. В мережі кожний день збільшується кількість сайтів, що використовують технологію картографічних сервісів (інтерактивних карт) для поширення даних та інформації, а також різних картографічних карт та наборів даних. В тому числі сучасні смартфони втілюють функцію навігації, що значно полегшує орієнтування людини у просторі.

З появою смартфонів картографія вийшла на новий рівень і тепер завдяки супутниковій і мережевій системам позиціювання прийняла абсолютно нову форму. Перебуваючи в будь-якій точці земної кулі, людина може без проблем зорієнтуватися на місцевості за допомогою цифрової картографії. Технології розвивалися протягом тривалого часу і на даному етапі існує можливість їх використання не тільки для навігації по містам та в подорожжах, але і для створення на їх основі різних застосунків і сервісів для орієнтації в приміщеннях, завдяки перенесенню планів будівель на географічні карти.

Подібні розробки є актуальними для різних будівель, навчальних закладів, в тому числі і для Одеського державного екологічного університету. Університет є комплексом, що складається з кількох корпусів та гуртожитків, в яких багато поверхів та аудиторій. Актуальним завданням є створення інтерактивної карти ОДЕКУ, яка би виконувала не лише функції навігації по корпусам, але також являлася би удобним графічним уявленням інформації об навчальних аудиторіях, кабінетах, службових приміщеннях та інших об’єктах університета.

Метою даної кваліфікаційної роботи є розробка навігаційно-інформаційної системи Одеського державного екологічного університету, з використанням вбудованих алгоритмів пошуку оптимального маршруту.

Для досягнення мети кваліфікаційної роботи необхідно вирішити наступні завдання:

- вивчити та описати предметну область;
- провести аналіз аналогів програмного забезпечення, що існує на ринку;
- сформувати функціональні та не функціональні вимоги до програмного продукту:
- виконати дослідження та обґрунтування вибору алгоритмів пошуку оптимального маршруту;
- виконати проектування мобільного застосунку за допомогою нотації UML, а саме: обрати систему управління базою даних; мову програмування; середовище програмування для розробки інтерфейсної частини застосунку;
- обрати програмні та апаратні засоби розробки та виконати розробку навігаційно-інформаційної системи у вигляді мобільного за стосунку;
- виконати теестування та перевірку працездатності системи;
- розробити керівництво користувача.

Структура кваліфікаційної роботи складається з вступу, чотирьох розділів, висновків, переліку посилань на 17 найменувань. Повний обсяг роботи становить 73 сторінок і містить 26 рисунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПРОГРАМНИХ СИСТЕМ

1.1 Аналіз існуючих програмних засобів

Розглянемо програми аналогі, які існують зараз та схожі за своїм функціоналом до мобільного застосунку для навігації відвідувачів ОДЕКУ, що розроблюється.

«ТачІнформ: Навігатор» – це інтерактивний комплекс на базі сенсорного інформаційного кіоску і спеціального програмного забезпечення. Він об'єднує рішення по навігації, інформування та ефективної реклами. Сама система також має бути доопрацьовати, з причини того, що після придбання вона не міститиме необхідних планів будівлі та інформації про магазини, розташовані всередині – для цього необхідно їх вносити вручну або скористатися допомогою програміста. Зовнішній вигляд програмного засобу приведений на рис. 1.1.

Рисунок 1.1 – Зовнішній вигляд програмного засобу «ТачІнформ: Навігатор»

Інтерактивний комплекс має наступні можливості:

- представлення інформації про торгові бренди і компанії торгівельного центру;
- зворотній зв'язок від відвідувачів (опитування та анкетування);
- інформування про спеціальні пропозиції та акції.

Крім того, інтерактивний комплекс має такі недоліки:

- придбання дорогих кіосків обслуговування;
- відсутність мобільності системи;
- необхідність доопрацювання.[1]

2ГИС – широко відома міжнародна картографічна компанія, що випускає однойменні електронні довідники з картами міст з 1999 року.

Вона запустила проект «Этажи» в травні 2014 року, який представляє з себе систему навігації по торгівельних центрах. На кожному поверсі позначені магазини, кафе, банкомати, ліфти та інші об'єкти. Це спрощує навігацію і допомагає швидше знайти потрібне місце [2].

Поки в цьому проекті є торгівельні центри тільки великих міст, але в іншому система виконана на дуже високому рівні.

Зовнішній вигляд програмного аналогу «2Гис: Этажи» наведений нижче на рис. 1.2.

1.2 Огляд алгоритмів пошуку оптимального маршруту

Задача про найкоротший шлях – завдання пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.

Задача про найкоротший шлях є однією з найважливіших класичних задач теорії графів. Сьогодні відомо безліч алгоритмів для її вирішення.

У даної задачі існують і інші назви: завдання про мінімальний шлях або, в застарілому варіанті, завдання про диліжанси.

Рисунок 1.2 – Програмний засіб «2Гис: Этажи»

Значимість даного завдання визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між двома перехрестями, де вершини виступають перехрестями, а дороги – ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.

Завдання пошуку найкоротшого шляху на графі може бути визначене для неорієнтованого, орієнтованого або змішаного графа. Для змішаного і орієнтованого графа додатково повинні враховуватися напрямки ребер.

Граф являє собою сукупність непорожньої безлічі вершин і ребер (наборів пар вершин). Дві вершини на графі суміжні, якщо вони з'єднуються загальним ребром. Шлях в неорієнтованому графі являє собою послідовність вершин. Якщо всі ребра в графі мають одиничну вагу, то задача зводиться до визначення найменшої кількості обхідних ребер.[3]

Існують різні постановки задачі про найкоротший шлях:

– Завдання про найкоротший шлях в заданий пункт призначення. Потрібно знайти найкоротший шлях в задану вершину призначення t, який починається в кожній з вершин графа (крім t). Помінявши напрямок кожного на-

лежного графу ребра, це завдання можна звести до задачі про єдину вихідну вершину (в якій здійснюється пошук найкоротшого шляху з заданої вершини в усі інші).

− Завдання про найкоротший шлях між заданою парою вершин. Потрібно знайти найкоротший шлях із заданої вершини u в задану вершину v.

− Завдання про найкоротший шлях між усіма парами вершин. Потрібно знайти найкоротший шлях з кожної вершини u в кожну вершину v. Це завдання теж можна вирішити за допомогою алгоритму, призначеного для вирішення завдання про одну вихідну вершину, однак зазвичай вона вирішується швидше.

У різних постановках задачі, роль довжини ребра можуть грати не тільки самі довжини, але і час, вартість, витрати, обсяг витрачених ресурсів (матеріальних, фінансових, паливно-енергетичних і т. П.) Або інші характеристики, пов'язані з проходженням кожного ребра. Таким чином, завдання знаходить практичне застосування у великій кількості областей (інформатика, економіка, географія та ін.).

У зв'язку з тим, що існує безліч різних постановок даного завдання, є найбільш популярні алгоритми для вирішення завдання пошуку найкоротшого шляху на графі:

1) Алгоритм Дейкстри (англ. Dijkstra's algorithm) – алгоритм пошуку на графах, який винайшов нідерландський вчений Едсгер Дейкстрой в 1959 році. Знаходить найкоротші шляхи від однієї з вершин графа до всіх інших шляхом складання ваг ребер и порівняння їх сумарних величин.

2) Алгоритм Беллмана – Форда знаходить найкоротші шляхи від однієї вершини графа до всіх інших в підвішеному графі. Вага ребер може бути негативною.

3) Алгоритм пошуку A * знаходить маршрут з найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої), використовуючи алгоритм пошуку по першому найкращому збігу на графі. Порядок обходу вершин визначається евристичної функцією «відстань + вартість» (зазвичай

позначається як f (x)). Ця функція – сума двох інших: функції вартості досягнення даної вершини (x) з початковою (зазвичай позначається як g (x) і може бути як евристичної, так і немає), і функції евристичної оцінки відстані від розглянутої вершини до кінцевої (позначається як h (x)). Функція h (x) повинна бути допустимою евристичною оцінкою, тобто не повинна переоцінювати відстані до цільової вершині. Цей алгоритм здебільшого являє з себе розширення алгоритму Дейкстри. Новий алгоритм досягав більш високої продуктивності (за часом) за допомогою евристики. A * покроково переглядає всі шляхи, що пролягають від початкової вершини до кінцевої, поки не знайде мінімальний. Як і всі поінформовані алгоритми пошуку, він переглядає спочатку ті маршрути, які «здаються» ведуть до мети. Від жадібного алгоритму, який теж є алгоритмом пошуку по першому найкращому збігу, його відрізняє те, що при виборі вершини він враховує, крім іншого, весь пройдений до неї шлях. Складова g(x) – це вартість шляху від початкової вершини, а не від попередньої, як в жадібному алгоритмі. На початку роботи проглядаються вузли, суміжні з початковим; вибирається той з них, який має мінімальне значення f(x), після чого цей вузол розкривається. На кожному етапі алгоритм оперує з безліччю шляхів з початкової точки до всіх ще не розкритих (листових) вершин графа – безліччю приватних рішень, – яке розміщується в черзі з пріоритетом. Пріоритет шляху визначається за значенням f(x) = g(x) + h(x). Алгоритм продовжує свою роботу до тих пір, поки значення f(x) цільової вершини не виявиться меншим, ніж будь-яке значення в черзі, або поки все дерево не буде переглянуто. З безлічі рішень вибирається рішення з найменшою вартістю. A * є повним в тому сенсі, що він завжди знаходить рішення, якщо таке існує.

4) Алгоритм Флойда-Воршелла знаходить найкоротші маршрути між усіма вершинами зваженого орієнтованого графа. Безпосередньо грунтується на тому факті, що в графі з позитивною вагою ребер всякий неелементарний (що містить більше 1 ребра) найкоротший шлях складається з інших найкоротших шляхів. Цей алгоритм більш загальний порівняно з алгоритмом Дей-

кстри, так як він знаходить найкоротші шляхи між будь-якими двома вершинами графа.

5) Алгоритм Джонсона знаходить найкоротші маршрути між усіма парами вершин зваженого орієнтованого графа. В даному алгоритмі використовується алгоритм Беллмана-Форда і алгоритм Дейкстри. При виникненні негативних циклів, алгоритм застосовує зсув значень для їх усунення.

6) Алгоритм Лі (хвильовий алгоритм) заснований на методі пошуку в ширину. Знаходить шлях між вершинами графа, що містить мінімальну кількість проміжних вершин (ребер). Основне застосування – трасування електричних з'єднань на кристалах мікросхем і на друкованих платах. Так само використовується для пошуку найкоротшого відстані на карті в стратегічних іграх [4].

Для розв’язання задачі пошуку оптимального маршруту у випадку прокладання маршруту по будівлі Одеського державного екологічного університету було вирішено представити систему коридорів у вигляді графу, де вершини графу відображують учбові або адміністративні приміщення та перехрестя маршрутів, а ребрами – дистанції між ними. Схема графу першого поверху для прокладання маршруту на основі коридорів навчального закладу представлена на рис. 1.3. Задача пошуку оптимального маршруту в даному випадку приймає вигляд задачі пошуку маршруту між двома точками (вершинами), де початковою точкою позначається поточне місце знаходження користувача системи, а кінцевою, тобто цільовою, – шукане приміщення.

З приведених вище алгоритмів, буде використаний алгоритм A*, оскільки він не лише повний, але й оптимальний. Крім того, він є маловитратним, бо зберігає тільки оптимальний маршрут, видаляючи відразу гілки, що мають менш ефективне значення. Саме це є вирішальним фактором, адже у розробці додатків для мобільних операційних систем дуже важливо враховувати нижчу продуктивність процесору телефона, а також менший об’єм пам’яті.

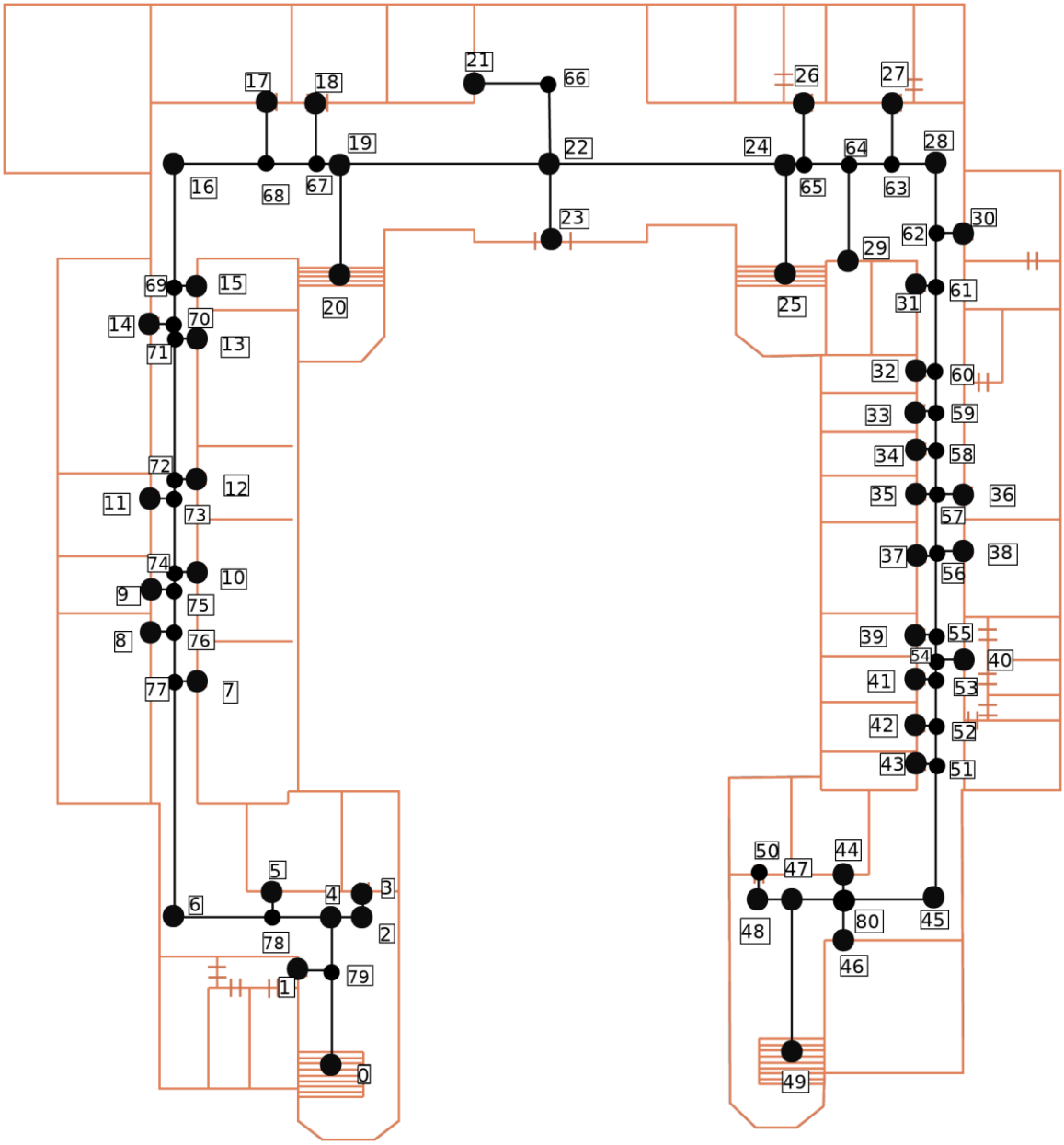

Рисунок 1.3 – Схема графу переміщення по першому поверху навчального закладу

Для реалізації алгоритму пошуку оптимального маршруту на основі алгоритму A*, було власноруч розроблено відповідний алгоритм. Як було сказано раніше, алгоритм працює з графом, що відображує шляхи переміщення по будівлі учбового закладу. Кількість вершин цього графу на чотирьох поверхах дорівнює 308 вузлам. Кожен вузол відповідає конкретній координаті.

Для роботи алгоритму необхідні дві точки, що відповідають початку маршруту та його кінцю. Перша точка відповідає поточному місцю знаходження користувача, а кінцева – цільовій вершині. Знайшовши оптимальне рішення, алгоритм повертає набір координат точок, що ведуть до цільової вершини, і за допомогою інструментів інтерфейсу прикладного програмування Google Maps API, проводиться прорисовка маршруту на карті Google Maps на обраному користувачем поверсі.

2 ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

Операційна система є самою популярною платформою для мобільних пристроїв. Різноманіття та широке розповсюдження смартфонів та планшетів різних виробників, що функціонують під керуванням даної платформи, стимулює ріст ринку мобільних за стосунків та робить навики розробки під Android затребуваними у сучасному світі.

За широтою можливостей платформа Android не поступається операційним системам настільних ПК. Це багаторівневе середовище на основі ядра Linux з багатими функціональними можливостями. У підсистему призначеного для користувача інтерфейсу входять:

- вікна;
- представлення;
- віджети для відображення загальних елементів, таких як редаговані поля, списки і розгортається списки.

Android містить вбудований браузер на базі WebKit – того ж механізму з відкритим вихідним кодом, який лежить в основі браузера Safari мобільного телефону iPhone.

Серед багатого функціоналу операційної системи Android присутні такі можливості, як підключення до Wi-Fi, Bluetooth і протоколи передачі даних через стільникову мережу (GPRS, EDGE, 3G і ін.). В стек програмного забезпечення Android входить і підтримка сервісів, заснованих на визначенні місця розташування (наприклад, GPS), і акселерометрів, хоча не всі пристрої на цій платформі оснащені необхідним обладнанням. Також присутня підтримка відеокамери.

Історично двома областями, де мобільні застосунки відставали від своїх настільних побратимів, були графіка / мультимедіа і способи зберігання даних. Android вирішує проблему графіки завдяки вбудованій підтримці 2-D і 3-D графіки, включаючи бібліотеку OpenGL. Завдяки наявності в платформі Android популярної невимогливої компактної бази даних SQLite, проблема

зберігання даних зовсім спрощується. На рис. 2.1 показана спрощена схема рівнів програмного забезпечення Android.

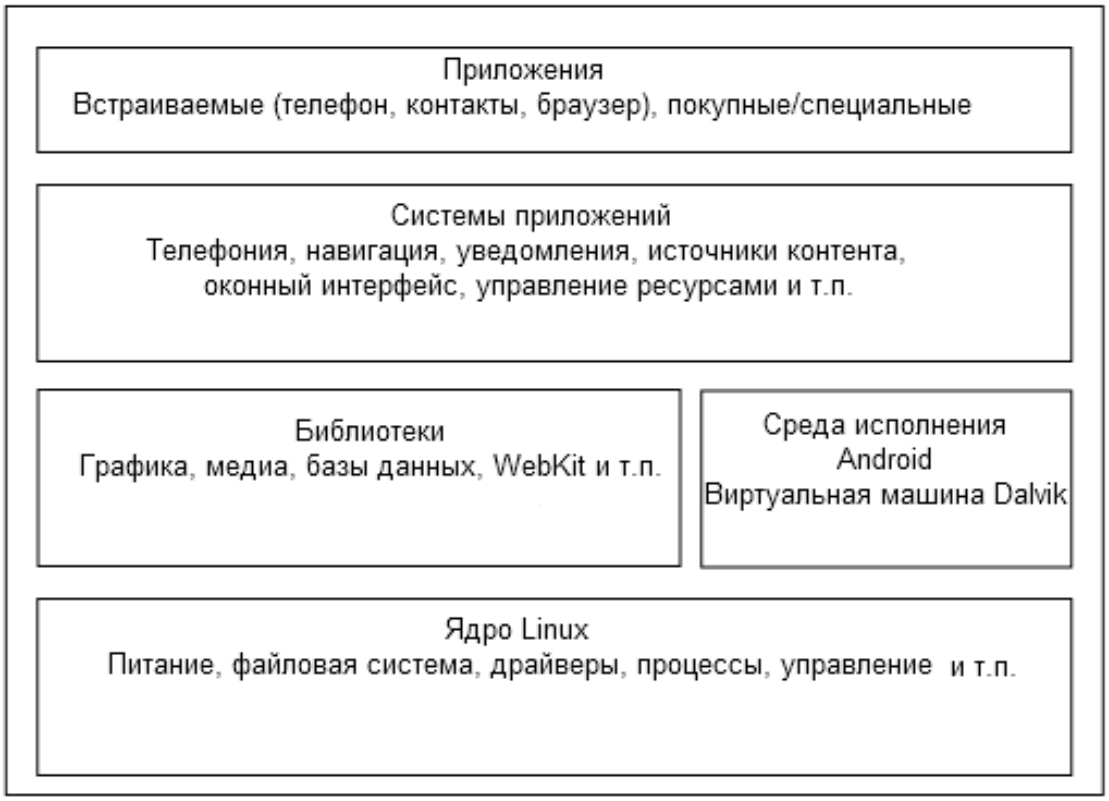

Рисунок 2.1 – Рівні програмного забезпечення Android

Отже, Android базується на основі ядра Linux. Android-додатки пишуться на мові програмування Java і виконуються у віртуальній машині (VM). Важливо відзначити, що віртуальна машина – це не JVM, як можна було б очікувати, а відкрита технологія Dalvik Virtual Machine. Кожна програма Android запускається всередині примірника Dalvik VM, який, в свою чергу укладений в межах керованого ядром Linux процесу, як показано на рис. 2.2.

Android-додаток містить елементи одного або декількох наступних типів:

- дії (Activities);
- сервіси (Services);
- джерела даних (Content providers);
- приймачі (Broadcast receivers).

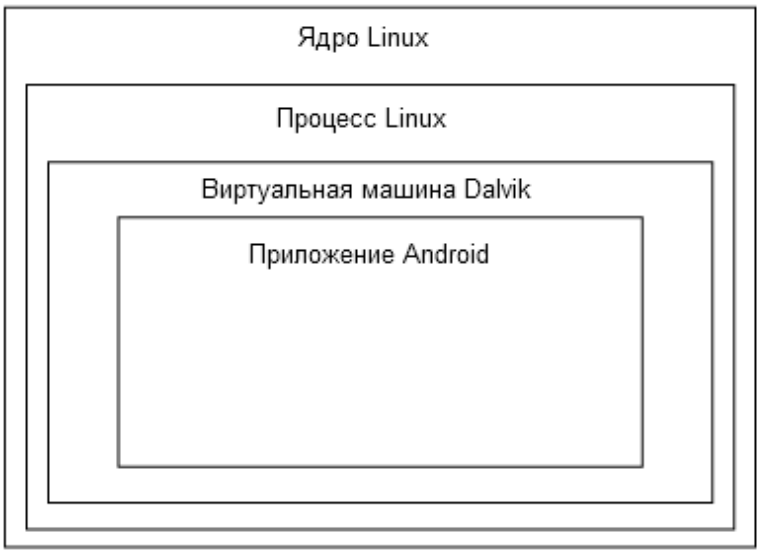

Рисунок 2.2 – Структура «Dalvik VM»

За допомогою дії реалізується графічний інтерфейс. У тому випадку, коли користувач обирає додаток на екрані або екрані запуску додатків, він викликає дію.

Сервіси застосовуються для додатків, які працюють протягом тривалого часу, таких як мережевий монітор або перевірка оновлень.

Джерело даних можна уявити як сервер баз даних. Його завдання - управління доступом до даних, що зберігаються, наприклад баз даних SQLite. Якщо додаток зовсім простий, джерело даних створювати не обов'язково. Якщо писати більш складний додаток або додаток, в якому до даних звертається кілька дій або додатків, джерело даних служить засобом організації доступу до інформації.

Android-додаток може запускатися для обробки елемента даних або реагування на події, наприклад, на отримання текстового повідомлення.

Додаток для Android розгортається на пристрої разом з файлом AndroidManifest.xml. Цей файл містить необхідну інформацію про конфігурацію, яка дозволяє правильно встановити додаток на пристрої. Він включає також необхідні імена класів і типи подій, які може обробляти додаток, і дозволу, необхідні для його роботи. Так, якщо з додатком потрібен доступ до

мережі – наприклад, щоб завантажити файл, – відповідний дозвіл має бути явно вказано у файлі маніфесту. Цей конкретний дозвіл можуть мати багато додатків. Такий захист шляхом декларування допомагає зменшити ймовірність пошкодження пристрою з вини некоректно написаного додатки.[6]

2.1 Бібліотеки для розробки Android-застосунків

Для роботи з проектами для системи Android існують різні бібліотеки, які представляють специфічні можливості.

Бібліотека «EventBus» – розсилає і реагує на події. Дана бібліотека служить для організації комунікацій (обміну даними і подіями) між не пов'язаними частинами програми. Тобто, вона дозволяє найпростішим чином відправити довільні дані з однієї частини програми (наприклад активність), в іншу (наприклад фрагмент).

«joda-time-android» – популярна бібліотека Joda-Time для Android, що дозволяє працювати з датою і часом.

Бібліотека «daimajia/AndroidImageSlider» дозволяє використовувати слайдер картинок з анімацією.

«AndroidSlidingUpPanel» – бібліотека, яка відповідає за висування панелі зверху чи знизу.

«Rebound» – бібліотека була розроблена програмістами із «Facebook». Сама вона написана на Java і дозволяє створювати реалістичні анімаційні ефекти у компонентів.

«GSON» – бібліотека для роботи з JSON була розроблена програмістами Google і дозволяє конвертувати об'єкти JSON в Java-об'єкти і навпаки.

Бібліотека «Android-WScratchView» – призначена для створення візуального ефекту стирання захисного шару з скретч-карт.

«CircleImageView» – компонент для створення круглих аватарів. Інша бібліотека зі схожою назвою «Pkmmte / CircularImageView». І ще одна бібліотека «vinc3m1 / RoundedImageView», що дозволяє створювати не тільки кру-

глі зображення, а й овальні і прямокутні із закругленими кутами. А також спрощений компонент «pavlospt / CircleView», який дозволяє створювати цілий елемент з заголовком і підзаголовком.

«Card Library» – бібліотека для створення карток.

Бібліотека «svg-android» дозволяє працювати з векторними зображеннями SVG.

Бібліотека «jsoup» призначена для парсингу HTML-текстів.

Бібліотека «opencsv» призначена для парсингу CSV-файлів.

Бібліотека «Android-Query» (AQuery) – простий спосіб використання асинхронних завдань і управління елементами користувальницького інтерфейсу.

Бібліотека «Android Asynchronous Http Client» – потужна бібліотека для асинхронних запитів.

Бібліотека «OkHttp» – HTTP-клієнт. Бібліотека «Retrofit» для REST.

«Picasso» – бібліотека для завантаження зображень з різноманітним функціоналом.

«Universal Image Loader» – бібліотека для завантаження зображень з мережі або локальних носіїв

Бібліотека «Glide» схожа по синтаксису і функціоналу на Picasso, підтримує анімовані GIF-файли.

«koush / ion» – і ще одна популярна бібліотека для асинхронних з'єднань і завантажень зображень.

Бібліотека «ProgressPieView» дозволяє працювати з індикатор прогресу у вигляді пирога. Візуальне представлення її компонентів зображено на рис. 2.3.

«Fresco | An image management library» – бібліотека для завантаження зображень від Facebook.

«AChartEngine» – бібліотека для малювання графіків

Бібліотека «aFreeChart» малює графіки і діаграми

«Androidplot» – ще одна бібліотека для малювання графіків

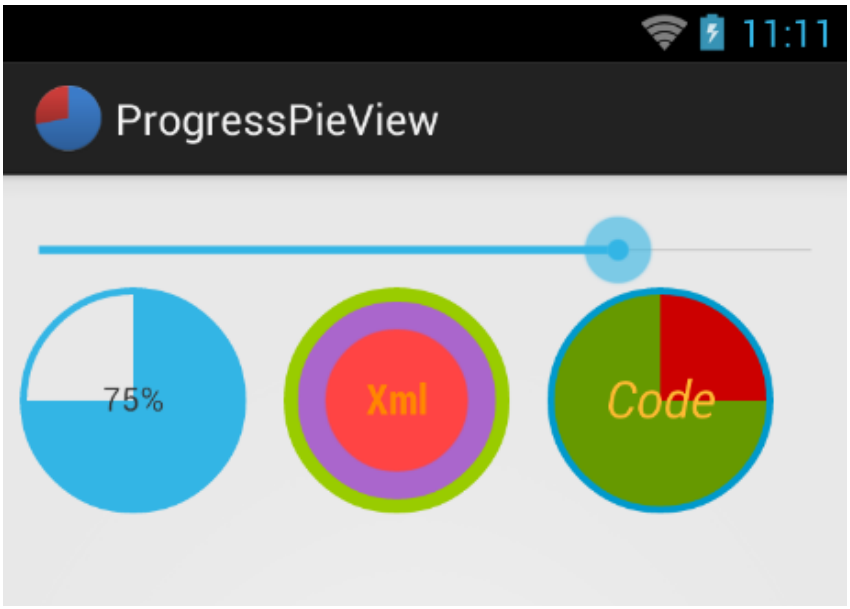

Рисунок 2.3 – Візуальне представлення компонентів бібліотеки «ProgressPieView»

Бібліотека «frakbot / JumpingBeans» – призначена для створення стрибаючого тексту.

«SlidingMenu» – бібліотека з відкритим кодом для створення меню, яке викликається з краю екрану.

«DashClock» – бібліотека для заміни годинників на екрані блокування для Android 4.2+. Можна додати не тільки годинник, але і пошту, календар, прогноз погоди та ін. Можна писати власні розширення для бібліотеки. Є документація для розширень.

Бібліотека «Android Holo Color Picker» дозволяє вибрати колір за допомогою колірного колеса.

Більш свіжа бібліотека «LarsWerkman / Lobsterpicker» написана в стилі Material Design. «android-flip» – бібліотека для створення анімації перегортання.

«ParallaxPagerTransformer» – бібліотека-паралакс для компоненту ViewPager.

«Leonids» – бібліотека для створення візуальних ефектів з частками [7].

2.2 Програмні оболонки

Android SDK (Software Development Kit – комплект розробки програмного забезпечення) є набором інструментів, що дозволяє розробляти API платформи Android. Він містить бібліотеки, утиліти для створення та збірання за стосунку, менеджменту образів віртуальних пристроїв і реалізації інших різлічних команд. Android SDK можна отримати безкоштовно для всіх основних платформ і може бути завантажений з сайту Google.[8]

Android SDK не включає компілятор мови програмування Java. У звязку з цим, для здійснення компіляції Android-застосунку потрібен також набір інструментів Java Development Kit (JDK) [9].

Для спрощення розробки може бути використана одне з інтегрованих середовищ (Intergrated Development Environment, IDE). Підтримка розробки під Android наявна у всіх основних середовищах розробки для Java, в тому числі IntelliJ IDEA, NetBeans і Eclipse. Слід відмітити, що наявність IDE не є обов'язковою для розробки, поскольку всі необхідні для збірання та развертивання застосунка операції можуть бути виконані за допомогою командного рядка.

Пакет Android SDK містить бібліотеки та інструменти, необхідні для розробки Android-додатків:

- «SDK Platform» – окрема платформа для кожної версії Android;
- «SDK Tools» – інструменти налагодження і тестування, а також інші корисні утиліти;
- документація – надає автономний доступ до новітньої документації API;
- допоміжний інструментарій Android – додаткові API, відсутні в стандартній платформі;
- приклади додатків – якщо знадобиться переглянути реальні приклади коду, щоб краще зрозуміти, як користуватися API, можна скористатися прикладами додатків;

– «Google Play Billing» – інтеграція сервісу білінгу в додатки.

2.2.1 Менеджер пакетов Android SDK

Android SDK підтримує розробку під всі офіційні версії платформи Android р використанням великої кількості зовнішніх бібліотек, включаючи багато бібліотек сторонніх розробників. Для управління пакетами, що забезпечують розробку з використанням даних бібліотек використовується утиліта Android. Її можна знайти у підкаталозі "tools" усередині каталогу SDK. Утиліта Android підтримує як інтерфейс командного рядка, і графічний інтерфейс користувача.

З метою менеджменту бібліотек, пропонується використовувати графічну оболонку, яку можна відкрити за допомогою утиліти Android без параметрів. Після того, як запуск оболонки зроблено, на екрані з'явиться перелік пакетів, до якого будуть включені пакети вже встановлені в системі, а також пакети, доступні для інсталяції [10].

За бажанням встановити необхідні пакети, їх можна вказати в загальному переліку, після чого скористатися кнопкою "Install packages". Після того, як ліцензія буде підтверджена користувачем, система почне завантажувати зазначені пакети з Інтернету, після чого здійснить їхнє встановлення. Пакети в списку згруповані за версіями платформи Android. Кожна з версій платформи має подвійну нумерацію: «комерційну» і «внутрішню»: наприклад, Android 4.0.3 відповідає API 15. Перший тип нумерації використовується для позначення підтримки можливостей платформи пристроями на ринку, тоді як у процесі розробки додатків повсюдно використовується другий тип нумерації.

Кожна група в переліку включає такі складові як: основний набір API для розробки додатків під платформу (SDK Platform), приклади додатків (Samples for SDK), документація на API (Documentation for Android SDK), вихідні тексти бібліотек платформи (Sources for Android SDK), образи віртуа-

льних пристроїв для різних архітектур (ARM EABI v7a System Image, Intel x86 Atom System Image та інші).

Крім цього, список може містити бібліотеки сторонніх розробників, призначені для певного класу пристроїв.

Для старту програмування необхідно встановити основний набір бібліотек (SDK Platform), образи віртуальних пристроїв (System Images) для конкретної платформи, а також набір інструментів не прив'язаний до версії API (Android SDK Tools, Android SDK Platform-tools у групі Tools).

Інші пакети є необов'язковими.

2.2.2 Збірання проекту

Складання Android-проекту може проводитись різними способами. У найпростішому випадку використовується утиліта «Ant», що входить у JDK і є стандартним засобом складання Java-проектів.

Складальний файл Ant зазвичай називається build.xml і розташовується в кореневому каталозі проекту. Якщо створювати проект командою з п. 1.3, цей файл також буде створено.

У складальному файлі визначаються цілі, кожна з яких відповідає деякій операції (наприклад, компіляція, складання, розгортання, тестування проекту). Операції складаються з команд, виконання яких призводить до досягнення вказаної мети.

Між цілями можуть бути встановлені залежності, при цьому команди необхідні для досягнення деякої мети, виконуються лише після того, як виконані команди залежних цілей.

При цьому підтримується досить гнучкий механізм, що дозволяє не виконувати деякі команди, якщо вони вже були виконані раніше та їх повторне виконання дасть той самий результат (наприклад, компіляція модуля не запускається повторно, якщо цей модуль вже був скомпілюваний раніше його вихідний текст не змінювався). [11]

2.2.3 Компоненти Android-застосунку

Типовий Android-програма складається з компонентів, які можуть ставитись до наступних типів:

− сервіс (service) - фоновий процес;

− слухач широкомовних повідомлень (broadcast receiver) – обробник деякої глобальної події в операційній системі (наприклад, вимкнення екрану, низький заряд батареї тощо).

− активність (activity) - компонент, що здійснює взаємодію з користувачем;

− провайдер контенту (content provider) – компонент, що здійснює надання доступу до даних, що знаходяться у деякому сховищі;

Ці компоненти є досить незалежними один від одного і мають чітко визначені інтерфейси для взаємодії з іншими компонентами.

Особливість Android-додатків полягає в тому, що компоненти різних програм можуть взаємодіяти між собою. Наприклад, у випадку, коли програмі необхідно надіслати повідомлення електронною поштою, воно може викликати стандартну активність з функціоналом поштового клієнта, причому після завершення цієї активності користувач повернеться до роботи з вихідним додатком.

І навпаки: програміст може розробити власний поштовий клієнт та зареєструвати його в системі при установці програми, тим самим дозволивши іншим програмам його використання.

Ця особливість дещо розмиває саме поняття програми та змушує розглядати всю платформу як відкриту систему, компоненти якої здатні до кооперативної поведінки.

Взаємодія компонентів здійснюється за допомогою надсилання асинхронних повідомлень. У програмі кожне з таких повідомлень асоціюється із сутністю, званою «інтент» (intent).

2.2.4 Інтенти

В Android-додатках інтент – це об'єкт, що інкапсулює в собі запит на виконання деякої дії. Інтент може включати наступні компоненти:

− URI, що ідентифікує дані, над якими необхідно виконати дія;

− набір категорій, що дозволяють групувати дії;

− дію, яку необхідно виконати (обов'язковий компонент);

− додаткові параметри (extras), необхідні для виконання дії.

Слід зазначити, що інтент, як правило, не містить у явному вигляді вказівки адресата повідомлення, що надсилається. Натомість вказується дія, яка необхідна виконати. Кожен компонент системи під час встановлення реєструє певний набір інтентів, які він здатний виконувати.

У момент активації відповідного інтенту, система здійснює пошук компонента, який здатний виконати цю дію. Після цього система передає керування такому компоненту.

У випадку, якщо система знайде кілька таких компонентів, користувачеві буде надано вибір.

2.2.4 Життєвий цикл активності

Активності є короткоживучими компонентами в архітектурі Android-програми.

Це означає, що вони мають властивість автоматично створюватися та знищуватися у різних випадках. Наприклад, коли змінюється орієнтація екрана, або нестача пам'яті для інших програм. Крім цього, активність може переходити у фон або на передній план, приймати та втрачати фокусування. Для правильної обробки всіх цих ситуацій запроваджується поняття життєвогоциклу активності та надаються засоби, що дозволяють виконувати ті чи інші дії під час переходу між різними фазами цього життєвого циклу.

Життєвий цикл активності включає наступні основні стану:

− paused – активність знаходиться на передньому плані, але не у фокусі, тобто перекрита спливаючою вікном або вікном діалогу; користувач не може безпосередньо взаємодіяти з такою активністю;

− running/resumed – активність знаходиться на передньому плані та у фокусі; у цьому стані користувач може безпосередньо взаємодіяти з додатком у вигляді графічного інтерфейсу;

− stopped – активність знаходиться у фоні та не відображається на екрані; користувач не може взаємодіяти з такою активністю.

Переходи між перерахованими станами здійснюються за подіями, ініційованими користувачем (наприклад, перемиканням на іншу активність), або системним подіям (наприклад, через брак пам'яті). Кожен перехід супроводжується викликом певних методів у класі Activity, від якого повинні успадковуватися будь-які активності. У свою чергу, у класах-спадкоємцях зазначені методи можуть перевизначатися щоб належним чином відреагувати на перехід між станами життєвого циклу.

Можливі переходи разом із відповідними callback-методами зображені на рис.А.1 додатку А.

Метод onCreate() викликається безпосередньо після створення активності. Тут здійснюється ініціалізація інтерфейсу користувача, прив'язка даних до елементів інтерфейсу створення потоків тощо. Парним до цього методу є метод onDestroy(), в якому необхідно звільнити ресурси, отримані під час виклику onCreate().

Метод onPause() викликається, коли активність втрачає фокус у зв'язку з перекриттям її екрана спливаючою вікном, діалогом чи інший активністю. Після повернення з цього методу активність перейде в стан paused.

Активність, яка має цей стан, може бути знищена операційною системою будь-коли в тому випадку, якщо системі буде недостатньо оперативної пам'яті та є інші більш пріоритетні програми. Виходячи з цього, метод onPause() повинен передбачати збереження стану активності щоб бути в змо-

зі відновити його при перезапуску. Відновлення стану здійснюється в callback-методі на Resume ().

Якщо розробник змінить системну конфігурацію програми, яка містить у собі зміну локалізації або поворот екрану, це призведе до знищення активності та надалі до її повторного створення. При цьому виконуються всі callback-методи, включаючи onDestroy() для знищуваної і onStart() для новоствореної активності. Для того щоб відрізнити розглянуту ситуацію від ситуації, коли програма закрита користувачем, а потім знову відкрито, передбачена пара методів onSaveInstanceState() і назаписіInstanceState(). Реалізація цих методів у класі Activity автоматично зберігає і відновлює стан всіх елементів інтерфейсу користувача. Проте в деяких складних випадках може знадобитися перевизначення і цих методів забезпечення коректного функціонування програми у разі зміни конфігурації.

3 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

3.1 Проектування діаграми прецедентів

Прецеденти – це текстовий документ, що описує послідовність подій, пов'язаних з використанням системи.

Діаграма прецедентів – в UML, діаграма, на якій зображений відношення між акторами і прецедентами в системі.

Актор – це набір ролей, який виконує користувач в ході взаємодії з деякою сутністю (система, підсистема, клас).

На діаграмі прецедентів ілюструється набір прецедентів системи та виконавці, а також взаємозв'язку між ними. Прецеденти визначають, як виконавці взаємодіють з програмною системою. У процесі цієї взаємодії виконавцем генеруються події, які передаються системі, які представляють собою запити на виконання деякої операції. Як правило, окремі кроки або види діяльності у вигляді прецеденту не представляються.

При створенні діаграм використання спочатку визначаються виконавці (ролі, користувачі). Виконавець (actor) є зовнішнім по відношенню до системи поняттям, яке певним чином бере участь у процесі, описуваному прецедентом. Актор є якоюсь 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.

Поведінка, що 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 та кінцевим користувачем для обговорення функціональної складової і поведінки системи.

На основі цього типу діаграм проводиться ітераційний цикл загальної постановки задачі разом з 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 (або використовуються системою) в даному прецеденті.

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, однак, реальних користувачів, які виступають в цій ролі по відношенню до ІС, може бути багато. Актори визначаються на основі складеного завдання або у процесі обговорення з замовником.

Спроектована діаграма прецедентів згідно додатку, що розроблюється, зображена на рис. 3.1.

У даному випадку у системі присутній один рівень доступу для користувачів оскільки недоцільно розмежовувати дану інформацію, адже вона знаходиться у громадському доступі. Кожен користувач зможе скористатися усіма можливими функціями додатку.

3.2 Проектування діаграми класів

Діаграмою класів в термінології UML називається діаграма, на якій зображений набір класів, які не мають явного відношення до проектування бази даних, а також зв'язків між цими класами. Крім того, діаграма класів може включати коментарі обмеження. Обмеження можуть неформально задаватися природною мовою або ж можуть формулюватися мовою об'єктних обмежень OCL (Object Constraints Language). Інакше кажучи, діаграма класів може відображати тільки імена класів або імена і відповідні атрибути класів, або імена, атрибути і операції (методи) класів.

Рисунок 3.1 – Діаграма прецедентів

Клас – це опис набору об'єктів з однаковими атрибутами, операціями, зв'язками і семантикою. Кожен клас повинен володіти ім'ям, який вирізняє його від інших класів. Ім'я - це текстовий рядок. Ім'я класу може складатися з будь-якого числа букв, цифр і розділових знаків (за винятком двокрапки і крапки) і може записуватися в кілька рядків.

Атрибут (властивість) – це іменована властивість класу, яка описує діапазон значень, які може приймати примірник атрибута. Клас може мати будь-яке число атрибутів або не мати жодного. В останньому випадку блок атрибутів залишають порожнім. Атрибут представляє деяку властивість сутності, що моделюється, якою володіють всі об'єкти даного класу. Ім'я атрибута, як і ім'я класу, може являти собою текст. На практиці для іменування атрибута використовуються одне або кілька коротких іменників, що виражають якесь властивість класу, до якого належить атрибут.

Атрибути можуть бути такими, типи значень яких вважаються заздалегідь визначеними в UML, як: розмір, площа, кут, видимість. Останній атрибут може мати наступні значення:

- загальна (public) означає, що операція класу доступна для кожного об’єкту системи з будь-якого місця програми;
- захищена (protected) означає, що операція класу може бути доступна тільки для тих об’єктів системи, які є екземплярами цього класу або екземплярами його спадкоємців;
- приватна (private) означає, що операція класу може бути доступна тільки для тих об’єктів системи, які є екземплярами цього класу – тобто класу, в якому вона визначена.

Одночасно користувач може визначати специфічні для нього атрибути. Операція – це сервіс, який може надавати примірник класу, якщо відповідний виклик. Операція називається і список аргументів. На діаграмі може бути показано не тільки класи, а й окремі їх екземпляри.[5]

Спроектована діаграма класів згідно з додатком, що розроблюється, зображена на рис. 3.2.

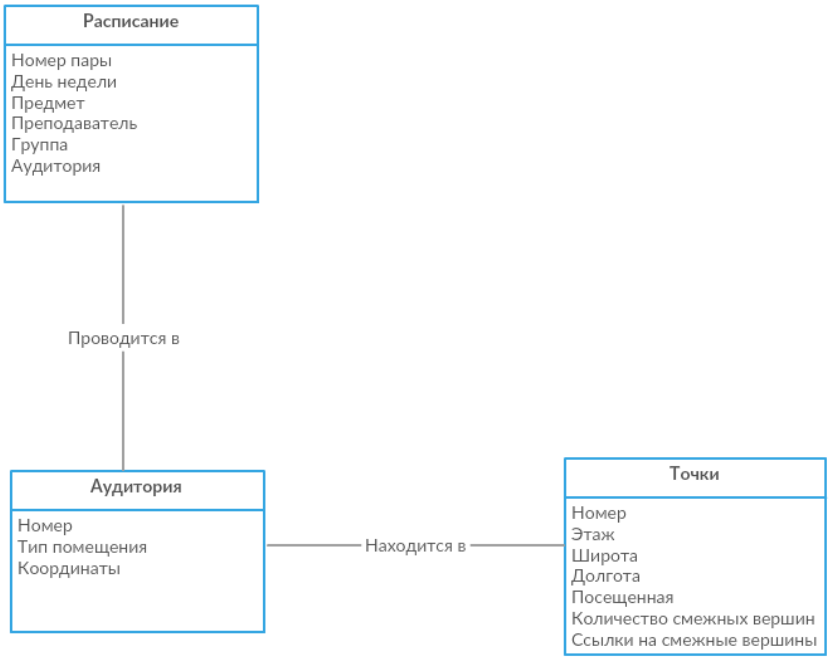

Рисунок 3.2 – Діаграма класів

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Функціональна схема додатку

Компоненти програми можна віднести до одного з чотирьох типів. Компоненти кожного типу призначені для певної мети, вони мають власний життєвий цикл, який визначає спосіб створення і припинення існування компонента.

Існують чотири стандартні блоки додатку Android:

- Activity (активність).
- Intent Receiver.
- Service (служба).
- Content Provider.

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

Коли розробник вирішує які компоненти необхідно створювати для додатку, їх необхідно записати у файлі на ім'я AndroidManifest.xml. Це – файл XML, де оголошуються елементи додатку, а також їхні можливості і вимоги.

Activity (активність) – основний з вище перелічених блоків Андроїд. Активність являє собою один повноцінний екран додатку, зазвичай – єдиний. Кожна активність здійснена як єдиний клас, який розширює базовий клас Activity. Клас відображає призначений для користувача інтерфейс, складений з Views на її макеті, і відповідає на події ініційовані користувачем. Більшість додатків складається з декількох екранів. Переміщення в інший екран досягається стартом нової активності. Коли новий екран відкривається, попередній екран припиняється і поміщається у стек хронології. Користувач може перейти на попередню активність перемістившись назад через раніше відкриті активності в хронології. Андроїд зберігає стеки хронології для кожної програми.

Незважаючи на те що Activity спільно формують чітку взаємодію користувача з додатком, кожна з них не залежить від інших операцій. Будь-які з цих операцій можуть бути запущені іншим додатком

Для організації переходу від одного екрану до іншого Андроїд використовує спеціальний клас, що називається Intent. Він описує те, що додаток збирається зробити. Дві найважливіших частині структури Intent – дія і дані до дії. Intent може містити наступні значення для дії: MAIN (головний екран додатка), VIEW, PICK, EDIT, і т.д. Дані виражені у вигляді Uniform Resource Indicator (URI) – послідовність символів, що ідентифікує логічний чи фізичний ресурс. Приклади ресурсів включають електронні документи, датчики дверей ліфту, простір імен XML, веб-сторінки та ідентифікаційні мікрочипи для домашніх тварин.

Існує пов'язаний клас, під назвою IntentFilter. Якщо Intent являє собою запит на якусь дію, то IntentFilter є описом того, що Intent Activity, здатний обробити. Активність, яка в змозі відобразити інформацію для користувача, видає IntentFilter, який повідомляє, що може зробити VIEW і містить інструкції для обробки. IntentFilters задається в файлі AndroidManifest.xml до відповідної активності.

Переміщення між екранами додатку досягається за допомогою Intent. Щоб переміститися вперед, активність викликає startActivity(myIntent). У такому випадку система звертається до IntentFilter всіх встановлених додатків і вибирає активність, Intent якої відповідає myIntent. Викликаній активності повідомляється про Intent, який змусив її початися. Процес виконання Intent відбувається, в той момент коли викликається метод startActivity.

IntentReceiver використовується, коли є необхідність, щоб код в своєму додатку виконався у відповідь на зовнішню подію, наприклад, у разі надходження телефонного виклику, або коли розрядився акумулятор. Intent Receiver не відображуються у користувальницькому інтерфейсі, однак вони спроможні відобразити повідомлення. Додаток не повинен працювати для його Intent Receiver; система запустить додаток, в разі необхідності, коли

Intent Receiver буде викликаний. Додатки можуть також послати свої власні Intent Receiver з іншим Context.broadcastIntent ().

Service (служба) являє собою компонент, який працює у фоновому режимі і виконує тривалі операції, пов'язані з роботою віддалених процесів. Служба не має інтерфейсу для користувача. Треба відзначити, що можна з'єднатися з Service (і запустити його, якщо він вже не працює) з методом Context.bindService(). В разі підключення до Service, з ним можна взаємодіяти через інтерфейс, виставлений Service.

Додатки можуть зберігати свої дані в файлах, базі даних SQLite, персональних налаштуваннях або будь-якому іншому механізмі, який має сенс. Content Provider, однак, корисний, якщо необхідно, щоб дані застосування були розділені з іншими додатками. Content Provider – клас, який здійснює стандартний набір методів, щоб дозволити іншим додаткам зберігати і відновлювати тип даних, які оброблені іншим (that) Content Provider.

Користувальницькі інтерфейси в Андроїд можуть бути створені двома шляхами, через XML-код або в java-коді. Створення структури графічного інтерфейсу користувача в XML переважно, тому що за принципом зразкового управління засобами перегляду, користувальницький інтерфейс повинен завжди бути відокремлений від логіки програми.

Основний функціональний модуль додатки Android – Activity – об'єкт класу android.app.activity. Activity відповідає за функціональну складову додатку і не має окремої присутності на екрані. Щоб дати Activity присутність на екрані і проектувати його користувальницький інтерфейс, необхідно працювати з Views і Viewgroups – основними одиницями графічного представлення візуальних компонентів користувальницького інтерфейсу на платформі Андроїд.

View – об'єкт, який розширює базовий клас android.view.view. View є структурою даних і його властивості знаходяться у макеті. View безпосередньо відповідає за інформаційне наповнення для певного контейнеру макету. Об'єкт View обробляє вимір, його схему розміщення, малюнок, зміни центру,

прокрутку, і клавіші / знаки для області екрану, яку він представляє. Клас View слугує базовим класом для всіх графічних фрагментів – ряд підкласів, які малюють інтерактивні елементи екрану. Серед дочірніх графічних об’єктів класу View знаходяться такі як: текстові представлення у вигляді статичного тексту TextView та рядку для введення тексту EditText, звичайна кнопка Button, кнопка-перемикач, яка дозволяє користувачеві обрати один з пунктів (опцій) з визначеного набору, RadioButton, кнопка-прапорець Checkbox, ScrollView і т.д.

Viewgroup – об'єкт класу android.view.viewgroup. Viewgroup – спеціальний тип об'єкту View, функція якого – утримувати набор View і Viewgroup і керувати ними. Viewgroups дозволяють додавати структуру до користувальницького інтерфейсу і створювати складні елементи екрану, до яких можна звернутися як до єдиного об'єкту. Клас Viewgroup слугує базовим класом для Layouts – ряду повністю здійснених підкласів, що забезпечує загальні типи Layouts екрану. Layouts дають можливість вбудувати структуру для ряду View.

На платформі Андроїд визначається користувальницький інтерфейс Activity використання дерева View і Viewgroup вузлів, як показано на рис. 4.1. Дерево може бути настільки ж простим або складним, як зробить розробник. Можна його побудувати, використовуючи набори зумовлених графічних фрагментів і Layouts Андроїда, або замовних типів View, які створюються самостійно.

Щоб прикріпити дерево до екрану і візуалізувати його, Activity викликає свій метод setContentView() і передає інформацію на кореневий об'єкт вузла. Як тільки система Андроїд отримує інформацію на кореневий об'єкт вузла, вона починає працювати безпосередньо з вузлом, щоб виміряти, і окреслити дерево. Коли Activity стає активним і отримує пріоритет, система реєструє його і звертається до кореневого вузла задля виміру і окреслення дерева. Тоді кореневий вузол надає запит, щоб його дочірні вершини окреслили себе

– у свою чергу, кожен Viewgroup вузол в дереві відповідальний за окреслення його прямих дочірніх вузлів.

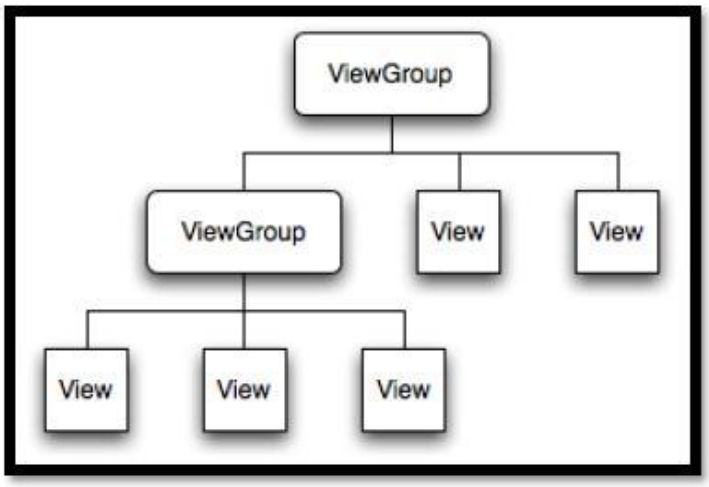

Рисунок 4.1 – Деревоподібна структура користувальницького інтерфейсу операційної системи «Android»

Як згадано раніше, у кожної групи View є відповідальність вимірювання її доступного простору, розташування її дочірніх вузлів, і виклик draw() на кожному дочірньому вузлі, щоб дозволити їм візуалізувати себе. Дочірні вузли можуть запрошувати розмір і місце розташування в батьківського, але у батьківського об'єкта є кінцеве рішення, де і наскільки великий кожен спадкоємець може бути.[12]

Даний додаток використовує 4 активності:

- «MainActivity» – головна активність, що містить карту отриману за допомогою Google Maps API. На ній розташовано план будівлі університету, прив’язаний до конкретних координат. Крім того активність містить елементи управління, такі як: зміна поверху, яка несе за собою зміну накладеного слою плану будівлі на карту; кнопка визначення місця знаходження, кнопку, що веде до другої активно-

сті. Крім того, на даній активності реалізований алгоритм пошуку оптимального маршруту до пункту призначення;

- «InfoActivity» – активність, що містить у собі три вкладки. На першій вкладці розташовано перелік усіх кімнат. На другій міститься розклад за обраною групою. На третій розташовані список з усіма викладачами університету;
- «GroupActivity» – активність, яка дозволяє змінити номер групи для перегляду розкладу;
- «TeacherActivity» – активність, яка містить інформацію про обраного викладача: предмети, які вони викладають, та пари, які вони проводять.

Крім того, додаток містить відповідні 4 макети:

- «activity_main» – макет, що відповідає за формування головної сторінки; містить ключ доступу до «Google Maps API»;
- «activity_info» – макет, що відповідає за формування інформаційної сторінки;
- «activity_group» – макет, що відповідає за формування сторінки «GroupActivity»;
- activity_teacher – макет, що відповідає за формування сторінки «TeacherActivity»;

Також додаток містить 3 фрагменти, які поміщуються до вкладок активності «InfoActivity»:

- «TeacherFragment» – фрагмент, який оброблює список вчителів;
- «ScheduleFragment» – фрагмент, який оброблює список з розкладом;
- «RoomFragment» – фрагмент, який оброблює список приміщень учбового закладу.

4.2 Алгоритм роботи додатку

При запуску додатку, одразу відкривається головна сторінка, що містить карту отриману за допомогою Google Maps API. На ній розташовано план будівлі університету, прив'язаний до конкретних координат, в залежності від обраного поверху – за умовчуванням встановлене значення, яке відповідає першому поверху. Користувач може переміщатися по поверхах будівлі університету, переглядаючи розташування інфраструктурних об'єктів на плані. Сам план будівлі накладений на відповідні координати цифрової картографічної мапи, завдяки чому, додаток може отримувати географічні координати користувачів, які безпосередньо знаходяться на території університету, що забезпечую отримання точних даних. Крім того активність містить елементи управління, такі як: зміна поверху, яка несе за собою зміну слою плану будівлі та кнопку визначення місця знаходження. Також на даній активності реалізований алгоритм пошуку оптимального маршруту до пункту призначення «A*». Користувач має змогу визначити пункт призначення та програма прокладе до нього маршрут від поточного місцязнаходження. Натиснувши на кнопку розташовану у нижньому правому кути, користувач може потрапити з цієї активності на активність з повним переліком усіх приміщень, переглянути їх на мапі, переглянути усіх викладачів університету, їхні часи роботи, переглянути розклад згідно до групи, у разі невизначеної групи обрати необхідну. Натиснувши на будь-яке значення на вкладці «Помещения», користувач тим самим обере кінцеве значення і програма прокладе маршрут. У разі натискання на будь-який пункт у вкладці «Расписание», буде відкрито розклад занять, який відповідає обраному дню тижня та обраній групі. Для того щоб змінити номер групи, користувач може натиснути кнопку у нижньому правому куті, і тоді відкриється активність вибору групи «GroupActivity». Активність містить лише випадаючий список з групами, що навчаються в університеті та обрати необхідну. При зміні групи, інформація на вкладці «Расписание» замінюється на відповідне обраній групі. На остан-

ній вкладці «Преподаватели» користувач має змогу переглянути список викладачів, що викладають у даному навчальному закладі. Натиснувши на будь-який пункт цього списку, користувач потрапить на активність «Teachers», де матиме змогу переглянути предмети, що веде викладач, та пари, які він проводитиме.

Для реалізації програмного забезпечення було проведено об’ємну роботу по взаємодії з картою Google Maps. Для цього було використано інтерфейс прикладного програмування від Google – Google Maps API. За допомогою інструментів, що надаються Google Maps API, було накладено поверхи будівлі з безпосередньою прив’язкою до координат, що дозволяє визначати місце знаходження користувача та точні відстані між вершинами графу. Також за допомогою інструментів Google Maps API було проведено роботу з графом та прорисовкою оптимального маршруту на накладеному поверсі.

4.3 Проектування інфологічної модель бази даних

Інфологія ІС – складене слово від двох коренів слів «інформація» і «логіка», тобто інформаційна логіка (функціонування) інформаційної системи. Відповідно, інфологічне (інформаційно-логічне) моделювання та проектування представляють процеси, пов'язані з поданням семантики у вигляді моделі, якій властиві морфізм і заходи. Основними конструктивними елементами інфологічних моделей є сутності, зв'язки між ними і їх властивості (атрибути). Інфологічна модель бази даних являє собою опис об'єктів (сутностей), з набором атрибутів і зв'язків між ними та призначена для структурного утворення предметної області. Вона повинна бути стабільної та незмінною.

Загальновідомо, що при побудові інфологічних моделей баз даних, зокрема, часто користуються мовою ER-діаграм (від англ. entity-relationship, тобто «сутність-зв'язок»). Її компонентами є сутності і зв'язки. Сутністю може бути будь-який інформаційний об’єкт, що необхідно зберігати у базі да-

них. Це може бути формалізований опис конкретного галузі предметної області інформаційної системи, що розроблюється. Дуже важливо розрізняти поняття сутності та екземпляру сутності, адже екземпляром сутності є лише конкретний прояв дії або об’єкту галузі, що описується. [13]

Атрибутом сутності виступає та чи інша характеристика присутня сутності. Вона може бути як кількісною, так і якісною. Назва атрибута повинна унікальною у межах однієї сутності, але атрибути різних сутностей можуть співпадати. Атрибут вказує на ту інформацію, яку необхідно зібрати про сутність. Атрибути також має екземпляри, але в даному випадку кожному атрибуту привласнюється тільки одне значення до відповідної сутності.

Ключем сутності називають набір атрибутів або один атрибут, що однозначно ідентифікують сутність у межах моделі.

Зв'язок – асоціювання двох або більше сутностей. Якби призначенням бази даних було тільки збереження окремих, не пов'язаних між собою даних, то її структура могла б бути дуже простою. Проте одна з основних вимог до організації бази даних – це забезпечення можливості відшукання одних сутностей за значеннями інших, для чого необхідно встановити між ними певні зв'язки. А так як в реальних базах даних нерідко містяться сотні або навіть тисячі сутностей, то теоретично між ними може бути встановлено більше мільйона зв'язків. Наявність такої безлічі зв'язків і визначає складність інфологічних моделей.

При побудові інфологічних моделей можна використовувати мову ER-діаграм (від англ. Entity-Relationship, тобто сутність-зв'язок). У них сутності зображуються позначеними прямокутниками, асоціації – поміченими ромбами або шестикутниками, атрибути – поміченими овалами, а зв'язки між ними – ненаправленими ребрами, над якими може проставлятися ступінь зв'язку (1 або буква, що заміняє слово "багато") і необхідне пояснення.

Між двома сутностями, можливі чотири види простих зв'язків: «один до одного», «один до багатьох», «багато до одного», «багато до багатьох» і

більш складні зв'язки, наприклад, семантичні зв'язки, що визначають якість функціонування семантичних інформаційних систем (інформаційних порталів, бібліотек, навчальних систем і т.д.).

Перший тип – зв'язок ОДИН-ДО-ОДНОГО (1: 1): в кожен момент часу кожному представнику (екземпляру) сутності А відповідає 1 чи 0 представників сутності В.

Другий тип – зв'язок ОДИН-ДО-БАГАТЬОХ (1: М): одному представнику сутності А відповідають 0, 1 або кілька представників сутності В.

Так як між двома сутностями можливі зв'язки в обох напрямках, то існує ще два типи зв'язку БАГАТО-ДО-ОДНОГО (М: 1) і БАГАТО-ДО-БАГАТЬОХ (М: N).

Крістофер Дейт – один з найбільших фахівців в області баз даних – визначає три основні класи сутностей: стрижневі, асоціативні і характеристичні, а також підклас асоціативних сутностей – позначення.

Стрижнева сутність (стрижень) – це незалежна сутність.

Асоціативна сутність (асоціація) – це зв'язок виду "багато-до-багатьох" між двома або більше сутностями або екземплярами сутності.

Асоціації розглядаються як повноправні суті:

- вони можуть брати участь в інших асоціаціях і позначеннях точно так же, як стрижневі сутності;
- можуть мати властивості, тобто мати не тільки набір ключових атрибутів, необхідних для вказівки зв'язків, а й будь-яке число інших атрибутів, що характеризують зв'язок.

Характеристична сутність (характеристика) – це зв'язок виду "багато-до-одного" або "один-до-одного" між двома сутностями (окремий випадок асоціації). Єдина мета характеристики в рамках розглянутої предметної області складається в описі чи уточнення деякої іншої сутності. Необхідність в них виникає в зв'язку з тим, що сутності реального світу мають іноді багатозначні властивості.

Позначаюча сутність або позначення – це зв'язок виду "багато-до-одного" або "один-до-одного" між двома сутностями і відрізняється від характеристики тим, що не залежить від позначаємої сутності.

Як правило, позначення не розглядаються як повноправні сутності, хоча це не привело б до будь-якої помилки.

Позначення і характеристики не є повністю незалежними сутностями, оскільки вони припускають наявність деякої іншої сутності, яка буде "позначатися" або "характеризуватися". Однак вони все ж є окремі випадки сутності і можуть, звичайно, мати властивості, можуть брати участь в асоціаціях, позначеннях і мати свої власні (нижчого рівня) характеристики. Підкреслимо також, що всі екземпляри характеристики повинні бути обов'язково пов'язані з будь-яким екземпляром характеризуємої сутності. Однак допускається, щоб деякі екземпляри характеризуємої сутності не мали зв'язків.

Ключ або можливий ключ – це мінімальний набір атрибутів, за значеннями яких можна однозначно знайти необхідний екземпляр сутності. Мінімальність означає, що виключення з набору будь-якого атрибута не дозволяє ідентифікувати сутність по тим атрибутам, що залишилися. Кожна сутність має хоча б один можливий ключ. Один з них приймається за первинний ключ. При виборі первинного ключа слід віддавати перевагу простим ключам або ключам, складеним з мінімального числа атрибутів. Недоцільно також використовувати ключі з довгими текстовими значеннями.

Не допускається, щоб первинний ключ стрижневий сутності (будь-який атрибут, який бере участь в первинному ключі) брав невизначений значення. Інакше виникне суперечлива ситуація: з'явиться екземпляр стрижневий сутності, який не володіє індивідуальністю, і, отже не існуючий. З тих же причин необхідно забезпечити унікальність первинного ключа.

Тепер про зовнішні ключі:

- якщо сутність С пов'язує сутності А і В, то вона повинна включати зовнішні ключі, відповідні первинним ключам сутностей А і В

- якщо сутність У позначає сутність А, то вона повинна включати зовнішній ключ, відповідний первинному ключу сутності А

Цілісність (від англ. Integrity – недоторканність, збереженість, цілісність) – розуміється як правильність даних в будь-який момент часу. Але ця мета може бути досягнута лише в певних межах: СУБД не може контролювати правильність кожного окремого значення, що вводиться в базу даних (хоча кожне значення можна перевірити на правдоподібність).

Підтримка цілісності бази даних може розглядатися як захист даних від невірних змін або руйнувань (не плутати з незаконними змінами і руйнуваннями, які є проблемою безпеки). Сучасні СУБД мають ряд засобів для забезпечення підтримки цілісності (так само, як і засобів забезпечення підтримки безпеки).

Виділяють три групи правил цілісності:

- цілісність по сутностей
- цілісність по посиланнях
- цілісність, що визначається користувачем

Не допускається, щоб будь-який атрибут, який бере участь в первинному ключі, приймав невизначене значення.

Значення зовнішнього ключа повинно або:

- бути рівним значенню первинного ключа мети;
- бути повністю невизначеним, тобто кожне значення атрибуту, який бере участь в зовнішньому ключі має бути невизначеним. [14]

Інфологічна модель бази даних цього проекту зображена на рис. 4.2.

4.4 Проектування даталогічної моделі бази даних

Даталогічне проектування – створення схеми бази даних на основі конкретної моделі даних, наприклад, реляційної моделі даних. Для реляційної моделі даних даталогічна модель – набір схем відносин, зазвичай із зазначен-

ням первинних ключів, а також «зв'язків» між відносинами, що представляють собою зовнішні ключі.

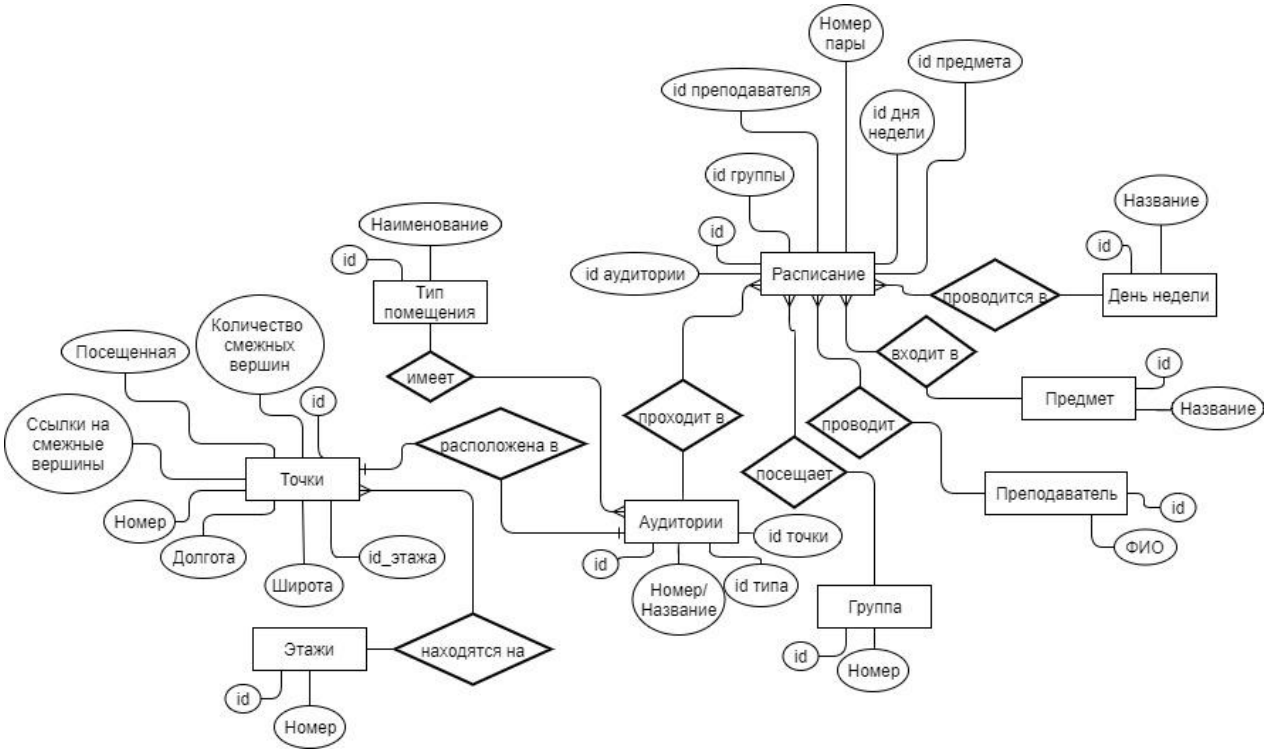

Рисунок 4.2 – Інфологічна модель бази даних (діаграма «сутність-зв’язок»)

Будь-яка СУБД оперує з допустимими для неї логічними одиницями даних, а також допускає використання певних правил композиції логічних структур більш високого рівня зі складових інформаційних одиниць нижчого рівня. Крім того, багато СУБД накладають кількісні та інші обмеження на структуру бази даних. Тому перш ніж приступити до побудови даталогической моделі, необхідно детально вивчити особливості СУБД, визначити чинники, що впливають на вибір проектного рішення, ознайомитися з існуючими методиками проектування, а також провести аналіз наявних засобів автоматизації проектування, можливості і доцільності їх використання.

Хоча даталогічне проектування є проектуванням логічної структури бази даних, на нього впливають можливості фізичної організації даних, що надаються конкретною СУБД. Тому знання особливостей фізичної організації даних є корисним при проектуванні логічної структури.

Логічна структура бази даних, а також сама заповнена даними база даних є відображенням реальної предметної області. Тому на вибір проектних рішень найбезпосередніший вплив надає специфіка відображається предметної області, відображена в інфологічній моделі.

Кінцевим результатом даталогічного проектування є опис логічної структури бази даних на мові опису даних. Однак якщо проектування виконується «вручну», то для більшої наочності спочатку будується схематичне графічне зображення структури бази даних. При цьому повинно бути забезпечено однозначна відповідність між конструкціями мови опису даних і графічними позначеннями інформаційних одиниць і зв'язків між ними. Графічне представлення використовується і при автоматизованому проектуванні структури бази даних як частиною інтерфейсу засіб спілкування з проектувальником, і при документуванні проекту.

Спроектувати логічну структуру бази даних означає визначити всі інформаційні одиниці і зв'язку між ними, задати їх імена; якщо для інформаційних одиниць можливе використання різних типів, то необхідно визначити їх тип. Слід також задати деякі кількісні характеристики, наприклад довжину поля.

Кожному типу моделі даних і кожного різновиду моделі, яку підтримує конкретної СУБД, притаманні свої специфічні особливості. Разом з тим є багато спільного у всіх структурованих моделях даних і принципах проектування БД в їх середовищі. Все це дає можливість використовувати єдиний методологічний підхід до проектування структури бази даних.

В БД відображається певна предметна область. Тому процес проектування БД передбачає попередню класифікацію об'єктів предметної області, систематизоване представлення інформації про об'єкти і зв'язки між ними.

На проектні рішення впливають особливості необхідної обробки даних. Тому відповідна інформація повинна бути певним чином представлена і проаналізована на початкових етапах проектування БД.

Дані про предметну область і особливості обробки інформації в ній фіксуються в інфологічної моделі. У такій моделі відображена вся інформація, що циркулює в інформаційній системі, але це зовсім не означає, що вся вона повинна зберігатися в базі даних. У зв'язку з цим одним з перших кроків проектування є визначення складу БД, тобто переліку тих показників, які доцільно зберігати в БД.

При проектуванні логічної структури БД здійснюються перетворення вихідної інфологічної моделі в модель даних, підтримувану конкретної СУБД, і перевірка адекватності отриманої даталогіченої моделі відображається предметної області.

Для будь-якої предметної області існує безліч варіантів проектних рішень її відображення в даталогіченої моделі. Методика проектування повинна забезпечувати вибір найбільш підходящого проектного рішення.

Мінімальна логічна одиниця даних (незважаючи на їх різні назви) семантично для всіх СУБД однакова і відповідає або ідентифікатором об'єкта, або властивості об'єкта або процесу.

Зв'язки між сутностями предметної області, відображені в інфологічної моделі, можуть відображатися в даталогічній моделі або за допомогою спільного розташування відповідних їм інформаційних елементів, або шляхом оголошення зв'язку між ними.

Не всі види зв'язків, що існують в предметній області, можуть бути безпосередньо відображені в конкретній даталогічній моделі. Так, багато СУБД не підтримують безпосередньо відношення М: М між елементами. У цьому випадку в даталогічну модель вводиться додатковий допоміжний елемент, що відображає цей зв'язок (таким чином, ставлення М: М хіба що розбивається на два відносини 1: М між цим знову введеним елементом і вихідними елементами).

Слід звернути увагу на те, що відносини, які мають місце в предметній області, можуть бути передані не тільки за допомогою структури бази даних, але і програмним шляхом (тобто завжди існує альтернатива між декларатив-

ним і процедурним способом опису явища). Наприклад, при відображенні узагальнених об'єктів можна не виділяти підкласи на рівні логічної структури бази даних. В цьому випадку підкласи будуть виділятися програмним шляхом при обробці даних, що зберігаються.

Рішення про те, який із способів відображення (структурний / декларативний або програмний / процедурний) слід використовувати в кожному конкретному випадку, буде залежати від багатьох факторів, таких, як стабільність сутності, що відображається обсяг номенклатури, особливості СУБД, характер обробки даних та ін. Так, якщо в предметній області використовується класифікація співробітників по статі, в базі даних не слід створювати класифікатор статей, оскільки він буде містити всього дві позиції і ніколи не змінюється. Як правило, не слід виділяти за цією ознакою і відповідні підкласи для об'єктів предметної області, тому що в більшості випадків вони обробляються спільно і в основному мають однаковий набір властивостей, що характеризує їх. Однак в деяких предметних областях поділ на такі підкласи може бути доцільним. Якщо ж відображається сутність не стабільна, то її краще передавати за допомогою даних, так як в противному випадку буде часто турбуватися перетворення програми, що зазвичай забезпечити важче, ніж зміна даних.

При відображенні узагальнених об'єктів в БД можливі різні варіанти: зберігати всю інформацію про всі узагальнені об'єкті в одному файлі / таблиці, кожному підкласу об'єктів нижчого рівня виділяти окремі самостійні файли / таблиці. І перший, і другий варіанти широко використовуються у різноманітних СУБД. У першому випадку підкреслюється спільність об'єктів різних підкласів, що входять в узагальнений об'єкт. У другому випадку, навпаки, узагальнений об'єкт як єдине ціле не відображається в структурі бази даних.

Інші способи відображення пов'язані з явним або неявним виділенням підкласів в логічній структурі БД. Неявне виділення підкласу полягає в тому, що в запису відводяться поля для фіксації значень властивостей, загальних

для об'єктів різних підкласів, і значення ознаки підкласу, а замість полів, наявність яких залежить від підкласу, використовується одне поле зі змінним складом, зміст якого буде залежати від того, до якого підкласу відноситься описуваний об'єкт. Реалізація принципу явного виділення підкласів в структурі БД істотно залежить від специфіки СУБД.

При проектуванні логічної структури БД основне значення має специфіка предметної області, що відображається. Однак, як зазначалося вище, і характер обробки інформації впливає на прийняте проектне рішення. Наприклад, рекомендується зберігати разом інформацію, часто оброблювану спільно, і, навпаки, розділяти по різних файлах інформацію, не що використовується одночасно. Інформацію, яку використовують часто, і інформацію, частота звернення до якої мала, також слід зберігати в різних файлах, причому останню може виявитися вигідним винести в архівні файли, а не підтримувати в складі БД.

Як зазначалося вище, описується не окремий об'єкт, а клас об'єктів. Але в окремих випадках буває, що клас включає в себе тільки один екземпляр. Наприклад, якщо предметною областю є якийсь конкретний інститут, то клас об'єктів буде містити тільки один екземпляр об'єкта. Таким «виродженим» класами об'єктів зазвичай не ставиться у відповідність окремий файл бази даних. [15]

На основі інфологічної моделі з попереднього пункту була спроектована даталогічна модель та сама база даних, що зображена на рис. 4.3.

. У якості системи керування базою даних була обрана SQLite – компактна вбудовувана реляційна база даних, що поставляється з вихідними кодами. Вперше випущена в 2000 році, призначена для надання звичних можливостей реляційних баз даних без властивих їм накладних витрат. За час експлуатації встигла заслужити репутацію як переносима, легка у використанні, продуктивна і надійна база даних.

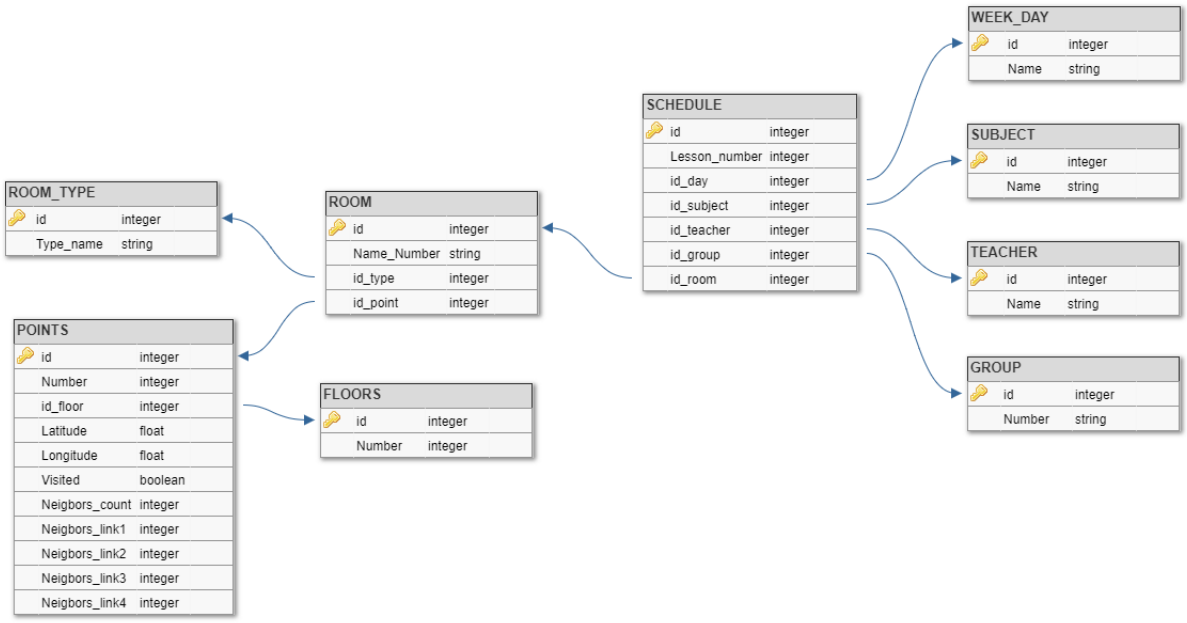

Рисунок 4.3 – Даталогічна модель бази даних (схема бази даних)

Слово «вбудована» (embedded) означає, що база даних існує не як процес, окремий від обслуговується процесу, а є його частиною – частиною деякого прикладного застосування. SQLite не використовує парадигму клієнт-сервер, вона являє собою бібліотеку, з якої програма компонується, і SQLite стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. І клієнт, і сервер працюють в одному процесі – це позбавляє від проблем конфігурації. Все, чого потребує програміст, вже скомпільовано в його додатку. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому пристрої, на якому виповнюється програма. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції запису весь файл, який зберігає базу даних, блокується; ACID-функції досягаються в тому числі за рахунок створення файлу журналу.

Кілька процесів або потоків можуть одночасно без будь-яких проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, якщо ніякі інші запити в даний момент не обслуговуються; в іншому випадку спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу.

У комплекті поставки йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина є кросплатформною утилітою командного рядка.

Завдяки архітектурі SQLite її можливо використовувати як на вбудовуваних системах, так і на виділених машинах з великими масивами даних.

SQLite підтримується динамічна типізація даних. Підтримуються такі типи даних:

- NULL – NULL-значення;
- INTEGER – цілочисельне значення зі знаком;
- REAL: число з плаваючою комою;
- TEXT: текстовий рядок з кодуванням UTF-8, UTF-16BE або UTF-16LE;
- BLOB: тип даних, що зберігається точно в такому ж вигляді, в якому і був отриманий.

На поточному ринку вбудовуваних баз даних представлено багато продуктів від різних виробників, але тільки один з них поставляється з відкритим кодом, не вимагає ліцензійних зборів і спроектований виключно як вбудовувана БД – це SQLite.

SQLite має модульну архітектуру, яка відображає унікальні підходи до управління реляційними базами даних. Вісім окремих модулів згруповані в три головних підсистеми (рис. 4.4). Вони поділяють обробку запиту на окремі завдання, які працюють подібно конвеєру. Верхні модулі компілюють за-

пити, середні виконують їх, а нижні працюють з диском і взаємодіють з операційною системою.

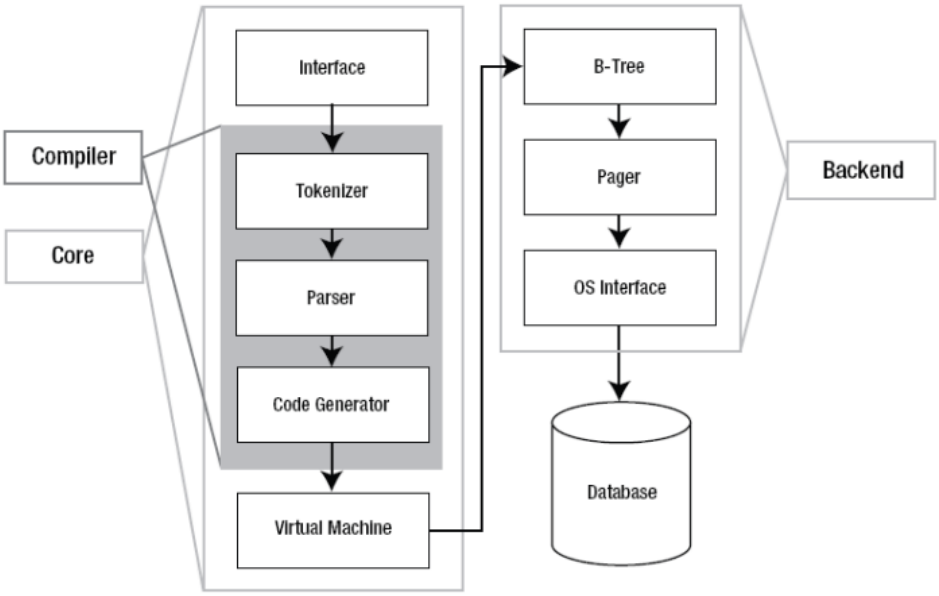

Рисунок 4.4 – Архітектура SQLite

Незважаючи на маленький розмір, SQLite надає великий спектр особливостей і можливостей. Він підтримує вельми повний набір стандарту ANSI SQL92 для особливостей мови SQL, а також такі особливості як тригери, індекси, стовпці з автоінкрементом та LIMIT / OFFSET особливості. Так само підтримуються такі рідкісні властивості, як динамічна типізація і вирішення конфліктів.[16]

Використовуючи цю систему керування базами даних, була створена база даних, яка має таблиці, наведені на рис. 4.5 – 4.13.

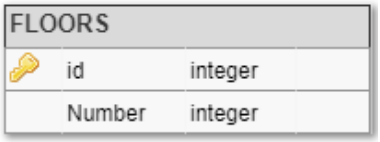

Рисунок 4.5 – Таблиця «FLOORS»

Таблиця «FLOORS» містить у собі інформацію про поверхи будівлі учбового закладу. Структура даної таблиці наведена на рис. 4.5.

Таблиця «POINTS» містить у собі інформацію координатні точки на електронній карті, а саме: порядковий номер координатної точки, номер поверху, на якому розташована дана точка, координата широти, координата довготи, кількість сусідніх вершин, посилання на сусідні вершини та позначка про відвідання даної вершини у ході виконання алгоритму пошуку оптимального шляху. Структура даної таблиці наведена на рис. 4.6.

POINTS	
id	integer
Number	integer
id_floor	integer
Latitude	float
Longitude	float
Visited	boolean
Neigbors_count	integer
Neigbors_link1	integer
Neigbors_link2	integer
Neigbors_link3	integer
Neigbors_link4	integer

Рисунок 4.6 – Таблиця «POINTS»

Таблиця «ROOM» містить у собі інформацію про приміщення учбового закладу, а саме: назва чи номер приміщення, помітка про тип приміщення та посилання на координатну точку, яка відповідає розміщенню даного приміщення. Структура даної таблиці наведена на рис. 4.7.

ROOM	
id	integer
Name_Number	string
id_type	integer
id_point	integer

Рисунок 4.7 – Таблиця «ROOM»

Таблиця «SCHEDULE» містить у собі інформацію про розклад занять в учбовому закладі, а саме: порядковий номер заняття, день неділі, назва предмету, ПІБ вчителя, номер групи та номер аудиторії, в якій проходитиме заняття. Структура даної таблиці наведена на рис. 4.8.

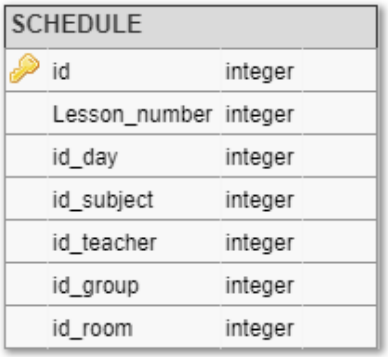

Рисунок 4.8 – Таблиця «SCHEDULE»

Таблиця «WEEK_DAY» містить у собі інформацію про дні неділі. Структура даної таблиці наведена на рис. 4.9.

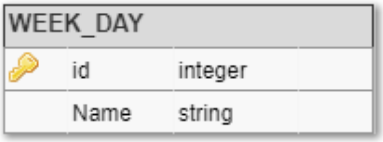

Рисунок 4.9 – Таблиця «WEEK_DAY»

Таблиця «SUBJECT» містить у собі інформацію про предмети. Структура даної таблиці наведена на рис. 4.10.

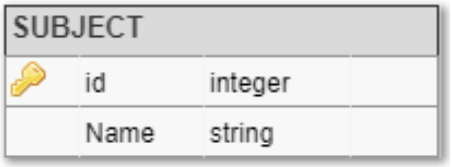

Рисунок 4.10 – Таблиця «SUBJECT»

Таблиця «TEACHER» містить у собі інформацію про викладачів. Структура даної таблиці наведена на рис. 4.11.

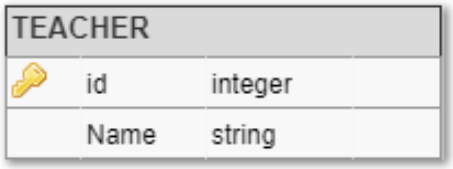

Рисунок 4.11 – Таблиця «TEACHER»

Таблиця «GROUP» містить у собі інформацію про групи студентів, що навчаються у даному навчальному закладі. Структура даної таблиці наведена на рис. 4.12.

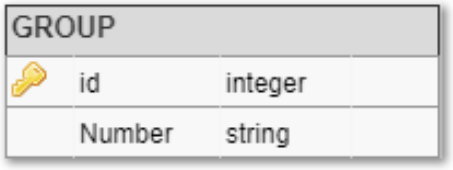

Рисунок 4.12 – Таблиця «GROUP»

Таблиця «ROOM_TYPE» містить у собі інформацію про типи приміщень в учбовому закладі – це адміністративні, учбові чи інші приміщення. Структура даної таблиці наведена на рис. 4.13.

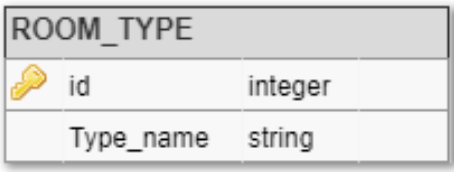

Рисунок 4.13 – Таблиця «ROOM_TYPE»

4.5 Розробка зовнішнього вигляду додатку

Розробка зовнішнього вигляду та дизайну додатку відбувалась виключно дотримуючись стандартів та канонів цієї області. В першу чергу орієнтованість направлена на Material design. Це мова візуальних образів, який створила корпорація Google для уніфікації інтерфейсів її продуктів і сервісів.

На діях користувача сфокусовано основну увагу. Взаємодією з дизайном керує призначений для користувача досвід, а не навпаки. Всі дії відбуваються в одному оточенні, інтерактивні об'єкти без переривання послідовності переходять з однієї середи в іншу.

Крім того додаток розроблювався відповідно вимогам зручності та використовуємості користувальницького інтерфейсу, базуючись на UX-дизайні та естетиці зовнішнього вигляду.

Нижче приведені рис. 4.14 та 4.21, на яких зображена візуальна частина додатку.

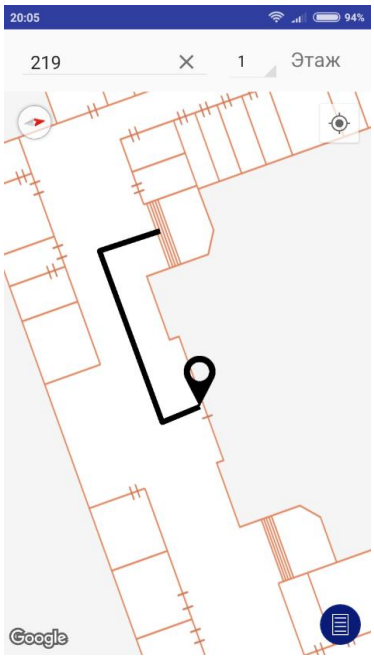

Рисунок 4.14 – Головна активність додатку: прокладення маршруту на першому поверсі (маркером помічена початкова точка)

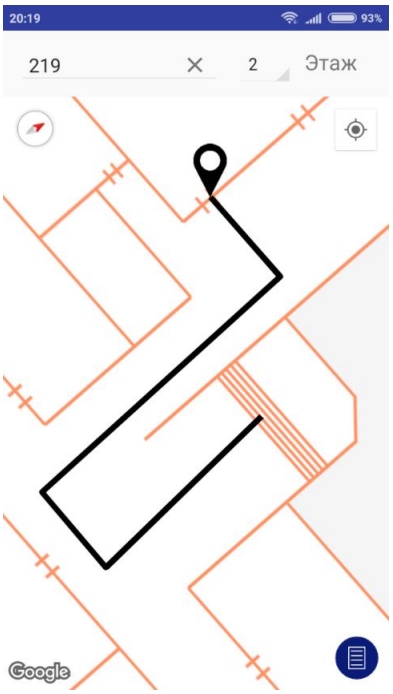

Рисунок 4.15 – Головна активність додатку: прокладення маршруту на другому поверсі (маркером помічена кінцева точка)

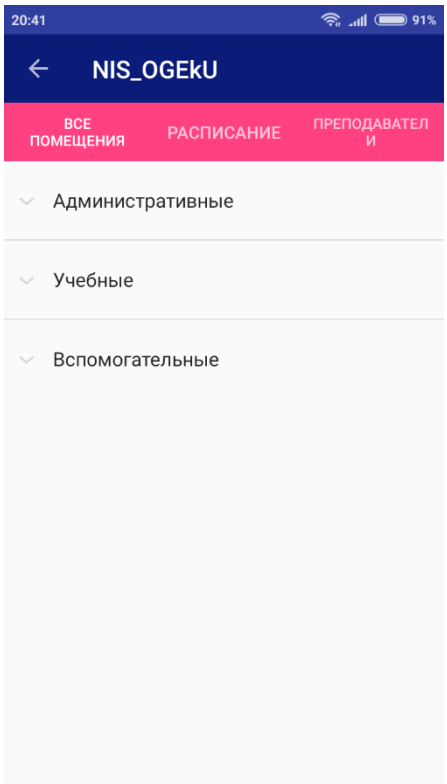

Рисунок 4.16 – Вкладка «Все помещения» на інформаційній активності

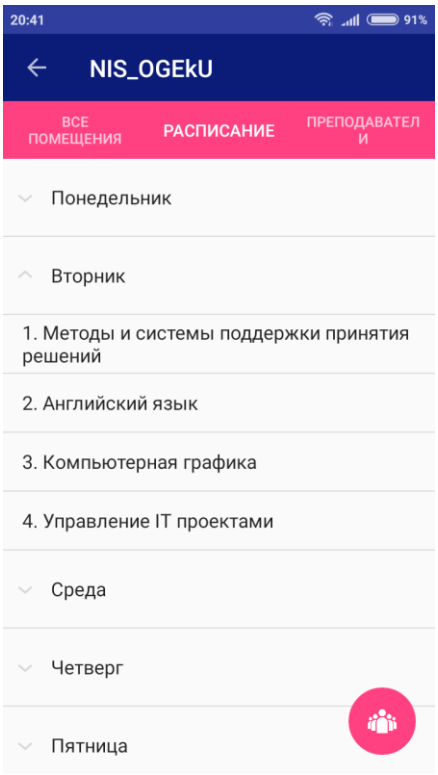

Рисунок 4.17 – Вкладка «Расписание» на інформаційній активності

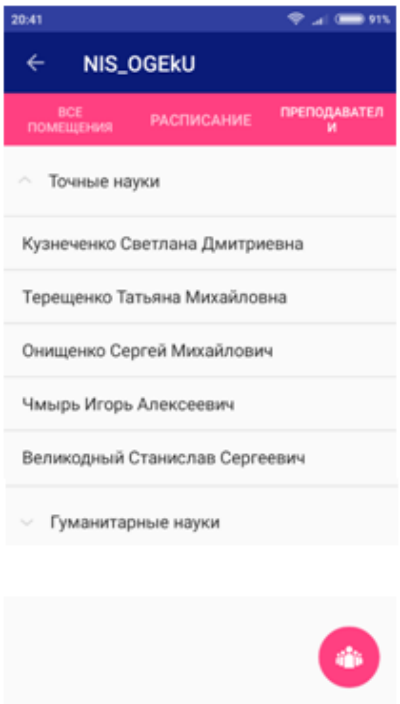

Рисунок 4.18 – Вкладка «Преподаватели» на інформаційній активності

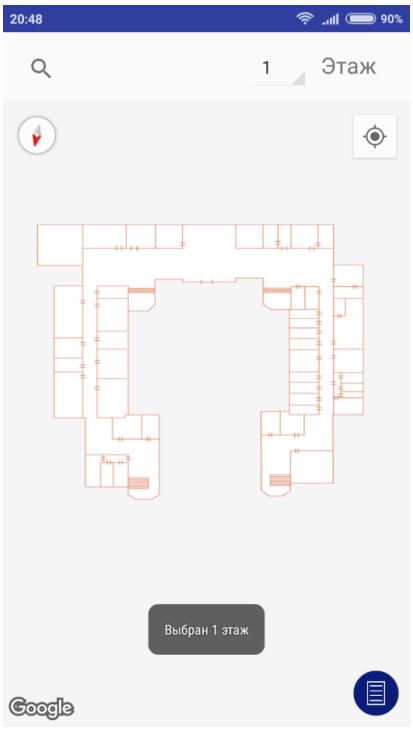

Рисунок 4.19 – Зовнішній вигляд накладеного слою першого поверху на карту

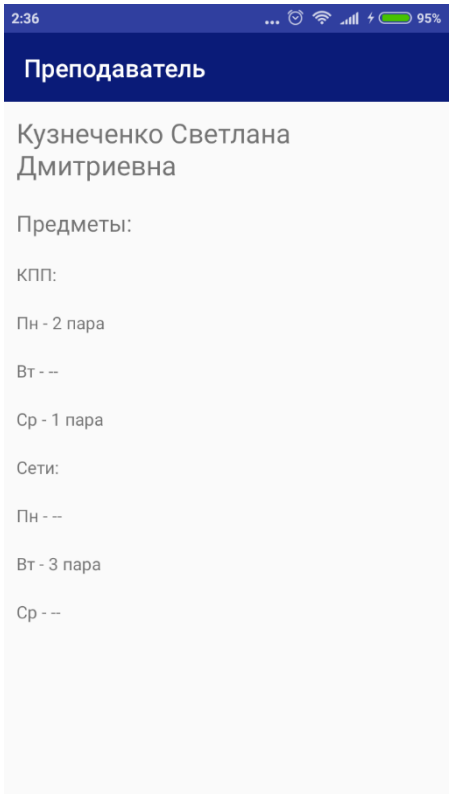

Рисунок 4.21 – Активність перегляду інформації про викладачів «TeacherActivity»

ВИСНОВКИ

У ході створення навігаційно-інформаційної системи Одеського державного екологічного університету були виконані наступні дії:

- проаналізована та описана предметна область;
- розглянуті програмні аналоги, що існують на ринку;
- проведено огляд існуючих алгоритмів пошуку найкоротшого маршруту та на його основі обрано оптимальний алгоритм для даної області;
- підтверджена актуальність додатку;
- встановлено мету;
- змодельована робота системи через UML діаграму прецедентів
- змодельована база системи через діаграму класів;
- розглянуті, проаналізовані та обрані програмні засоби для реалізації додатку;
- розроблена інфологічна модель;
- розроблена даталогічна модель;
- розроблена візуальна частина додатку;
- проведене роботу по накладанню плану будівлі університету на Google Maps;
- розроблений власноруч алгоритм пошуку оптимального маршруту на основі алгоритму A*;
- цілком розроблена функціональна частина додатку;
- протестовано роботу додатку;
- розроблено курівництво користувача.

Для реалізації програмного забезпечення було проведено об’ємну роботу по взаємодії з картою Google Maps. Для цього було використано інтерфейс прикладного програмування від Google – Google Maps API. За допомогою інструментів, що надаються Google Maps API, було накладено поверхи будівлі з безпосередньою прив’язкою до координат, що дозволяє визначати

місце знаходження користувача та точні відстані між вершинами графу. Також за допомогою інструментів Google Maps API було проведено роботу з графом та прорисовкою оптимального маршруту на накладеному поверсі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Програмне забезпечення «ТачИнформ» – Офіційний сайт. URL: http://touchinform.com/ (дата звернення 14.05.2022)
2. Виджет «Этажей» – Офіційний сайт. URL: http://floors-widget.2gis.ru/ (дата звернення 14.05.2022)
3. Мелешко Є.В., Якименко М.С., Поліщук Л.І. Алгоритми та структури даних: Навчальний посібник для студентів технічних спеціальностей денної та заочної форми навчання. Кропивницький: Видавець Лисенко В.Ф., 2019. 156 с.
4. Вікіпедія «Задача про найкоротший шлях» – Офіційний сайт. URL: https://uk.wikipedia.org/wiki/Задача_про_найкоротший_шлях (дата звернення 14.05.2022)
5. Омельчук Л.Л. Об’єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник. Київ: 2021. 265 с.
6. Давидов М.В., Демчук А.Б., Лозинська О.В. Програмне забезпечення мобільних пристроїв: навчальний посібник. Львів: Видавництво «Новий Світ-2000» 2020. 218 с.
7. Поляков А. О., Федорченко В. М., Шматко О. В. Аналіз методів і технологій розроблення мобільних додатків для платформи Android: навчальний посібник. Харків: ХНЕУ ім. С. Кузнеця, 2017. 286 с.
8. Ткаченко О. А. Розробка мобільних додатків під Android : навч. посіб. Київ. нац. ун-т культури і мистецтв. Київ : КНУКіМ, 2017. 274 с. : іл.
9. Проектування інформаційних систем: Загальні питання теорії проектування ІС: навч. посіб. для студ. спеціальності 122 «Комп’ютерні науки» / КПІ ім. Ігоря Сікорського; уклад.: О. С. Коваленко, Л. М. Добровська. Київ : КПІ ім. Ігоря Сікорського, 2020. 192с. URL:

https://ela.kpi.ua/bitstream/123456789/33651/1/PIS_KL.pdf (дата звернення 14.05.2022)

10. Дворецький М. Л., Нездолій Ю. О., Дворецька С. В., Кандиба І. О. Розробка мобільних застосунків для OS Android : навч. посіб. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. 140 с.
11. Реліз вільного середовища розробки Android Studio URL: http://laptop.ucoz.ru/news/reliz_svobodnoj_sredy_razrabotki_android_studio_1_0/2014-12-09-6371 (дата звернення 14.05.2022)
12. Genymotion. URL: http://wikiprograms.org/genymotion/ (дата звернення 14.05.2022)
13. Створення Android застосувань. Структура Android застосувань URL:http://4pda.biz/stati/495-sozdanie-android-prilozhenij-struktura-android-prilozheniya.html (дата звернення 14.05.2022)
14. Бесплатная интернет библиотека: «Информатика и синергетика учебное пособие» URL:http://doc.knigi-x.ru/22tehnicheskie/530914-1-kafedra-geodeziya-geoinformatika-navigaciya-rozenberg-cvetkov-informatika-sinergetika-uchebnoe-posobie-rekomendovano.php (дата звернення 14.05.2022)
15. Формалізація моделі "Сутність-зв'язок". Монографія URL: http://csc.knu.ua/uk/library/books/bui-silveistruk-33.pdf (дата звернення 14.05.2022)
16. Даталогічне проектування бази даних URL: https://studfile.net/preview/9391593/page:5/(дата звернення 14.05.2022)
17. База даних SQLite. URL:https://ua.wikipedia.org/wiki/SQLite (дата звернення 14.05.2022)

ДОДАТОК А
ЖИТТЄВИЙ ЦИКЛ ANDROID ФРАГМЕНТУ

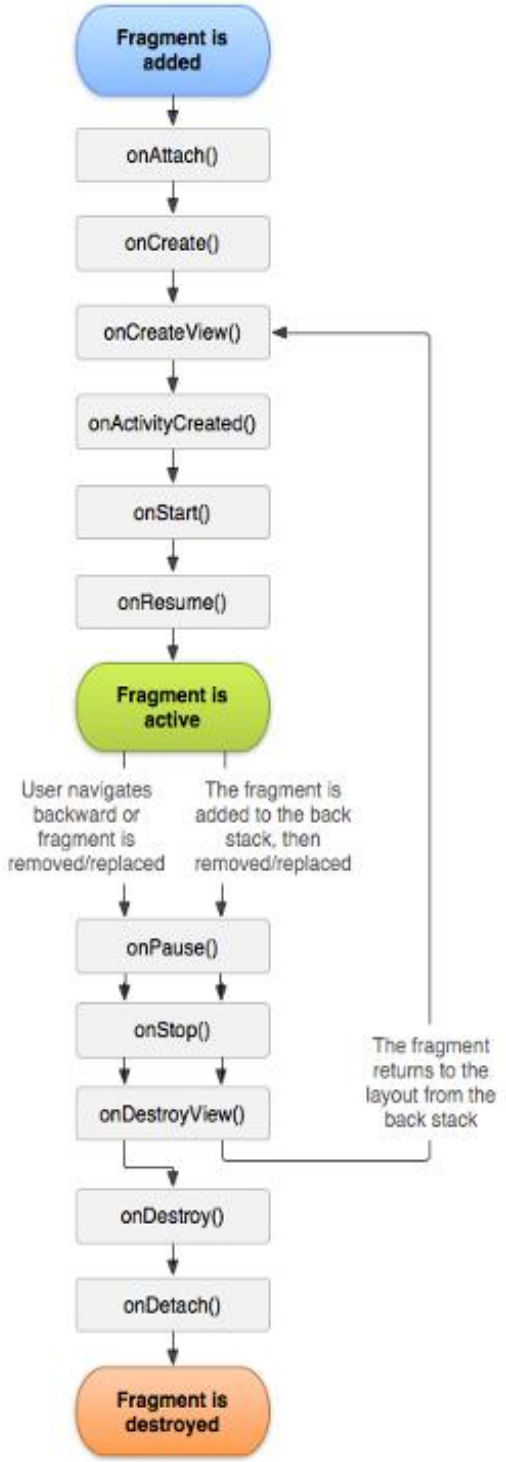

Рисунок А.1 – Життєвий цикл роботи фрагменту у мобільній ОС Android

ДОДАТОК Б
ЖИТТЄВИЙ ЦИКЛ ANDROID ДОДАТКУ

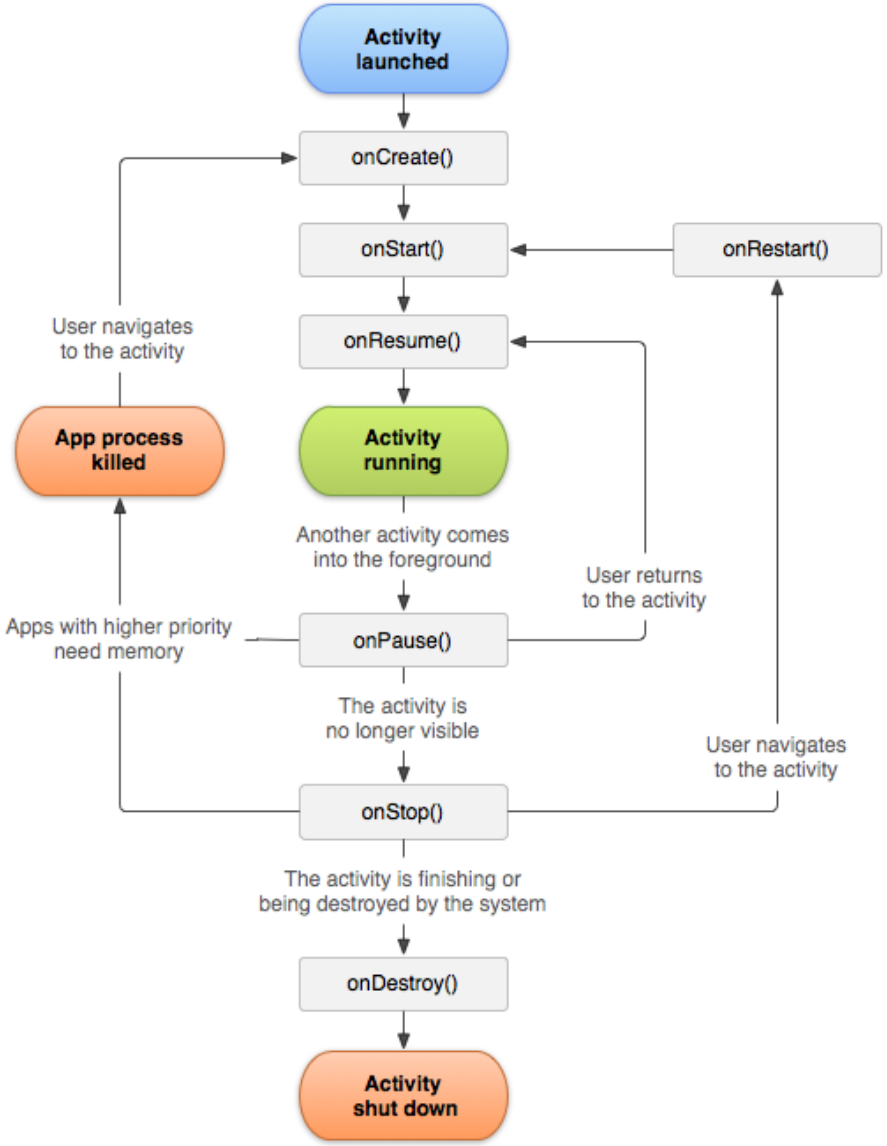

Рисунок Б.1 – Життєвий цикл додатку для мобільної ОС Android

Додаток В

ПРИКЛАД ПРОГРАМНОГО КОДУ ДЛЯ ANDROID ANNOTATIONS

```
package com.aatech.easylearn.fragments;

import android.content.Context;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.StaggeredGridLayoutManager;
import android.util.Log;
import android.view.MenuItem;

import com.aatech.easylearn.R;
import com.aatech.easylearn.adapters.QuestionItemAdapter;
import com.aatech.easylearn.converters.impl.QuestionItemConverter;
import com.aatech.easylearn.fillers.impl.QuestionItemFiller;
import com.aatech.easylearn.listeners.EndlessRecyclerOnScrollListener;
import com.aatech.easylearn.models.QuestionItem;

import org.androidannotations.annotations.AfterViews;
import org.androidannotations.annotations.Background;
import org.androidannotations.annotations.EFragment;
import org.androidannotations.annotations.OptionsItem;
import org.androidannotations.annotations.OptionsMenu;
import org.androidannotations.annotations.OptionsMenuItem;
import org.androidannotations.annotations.UiThread;
import org.androidannotations.annotations.ViewById;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;

import java.io.IOException;
import java.util.Collections;
```

```
import java.util.List;

@EFragment(R.layout.recycler_list)
@OptionsMenu(R.menu.questions_fragment)
public class QuestionsFragment extends Fragment {
    @ViewById
    RecyclerView recyclerContentView;

    @OptionsMenuItem
    MenuItem filtersOption;

    @OptionsItem(R.id.filterGoodOptionItem)
    void filterGoodSelected() {
        filter = "/good";
        getAdapter(recyclerContentView).clearNews();
        getQuestionsItems(recyclerContentView, 0);
    }

    @OptionsItem(R.id.filterAllOptionItem)
    void filterAllSelected() {
        filter = "/all";
        getAdapter(recyclerContentView).clearNews();
        getQuestionsItems(recyclerContentView, 0);
    }

    @OptionsItem(R.id.categoryAllOptionItem)
    void categoryAllSelected() {
        filter = "/all";
        category = "/new/all";
        filtersOption.setVisible(true);
        getAdapter(recyclerContentView).clearNews();
        getQuestionsItems(recyclerContentView, 0);
    }
```

```
    @OptionsItem(R.id.categoryVideoOptionItem)
    void categoryVideoSelected() {
        filter = "/all";
        category = "/video";
        filtersOption.setVisible(true);
        getAdapter(recyclerContentView).clearNews();
        getQuestionsItems(recyclerContentView, 0);
    }

    @OptionsItem(R.id.categoryReviewOptionItem)
    void categoryReviewSelected() {
        filter = "";
        category = "/reviews";
        filtersOption.setVisible(false);
        getAdapter(recyclerContentView).clearNews();
        getQuestionsItems(recyclerContentView, 0);
    }

    public StaggeredGridLayoutManager newsLayoutManager = new
StaggeredGridLayoutManager(1, StaggeredGridLayoutManager.VERTICAL);

    public QuestionsFragment() {
        Bundle bundle = new Bundle();
        this.setArguments(bundle);
    }

    private String filter = "/good";
    private String category = "/new/all";

    @AfterViews
    void calledAfterViewInjection() {
        getQuestionsItems(recyclerContentView, 0);
```

```
        recyclerContentView.addOnScrollListener(new
EndlessRecyclerOnScrollListener(newsLayoutManager) {

            @Override

            public    void    onLoadMore(RecyclerView    recyclerView,    int
current_page) {

                getQuestionsItems(recyclerView, current_page);

            }

        });

    }


    @Background

    public void getQuestionsItems(RecyclerView view, Integer page){

        Document webpage;

        try {

            webpage    =    Jsoup.connect("https://stopgame.ru/questions"    +
category + filter + "/page" + (page + 1)).get();

            List<QuestionItem>          questionItems          =          new
QuestionItemFiller().fill(new
QuestionItemConverter().convert(webpage.select(".question-block")));

            updatePosts(view, questionItems, page);

        } catch (IOException ex) {

            Log.e("AsyncLoader", ex.toString());

        }

    }


    @UiThread

    public    void    updatePosts(RecyclerView    view,    List<QuestionItem>
asyncResult, int page) {

        if(view.getLayoutManager()==null)

            view.setLayoutManager(newsLayoutManager);


        QuestionItemAdapter adapter = getAdapter(view);


        if (page == 0) {

            adapter.clearNews();
```

```
        }


        adapter.addNews(asyncResult);
    }


    private QuestionItemAdapter getAdapter(RecyclerView view) {
        QuestionItemAdapter adapter = (QuestionItemAdapter)
view.getAdapter();
        if(view.getAdapter() == null) {
            adapter = new QuestionItemAdapter(view.getContext(),
Collections.EMPTY_LIST);
            view.setAdapter(adapter);
        }
        return adapter;
    }
}
```