НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення веб-технологій та мобільних пристроїв»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система керування подіями на основі інтерактивної карти»

Виконав:
студент IV курсу, групи ТІ-62
Пахут Сергій Вадимович _____

Керівник:
доцент, к.т.н.
Гагарін Олександр Олександрович _____

Рецензент:
доцент, к.т.н.
Отрох Сергій Іванович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім’я, по батькові)

1. Тема роботи Система керування подіями на основі інтерактивної карти
керівник роботи Гагарін Олександр Олександрович, доцент, к.т.н.
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**
2. Строк подання студентом роботи 11.06.2020
3. Вихідні дані до роботи: Веб-сервер – Apache. Мова програмування серверної частини – PHP. Технології програмування клієнтської частини – Android Java. Середовище програмування – Android Studio IDE.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1) Постановка задач для системи керування подіями
2) Аналіз проблеми організації та пошуку подій на основі розташування користувача
3) Обгрунтування засобів розробки
4) Опис програмної реалізації
5. Перелік ілюстративного матеріалу
1) Структура бази даних – плакат
2) Діаграма прецедентів – плакат

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 01.02.2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	01.02.2020	
2.	Вивчення та аналіз задачі	01.03.2020	
3.	Розробка архітектури та загальної структури системи	01.04.2020	
4.	Розробка структур окремих підсистем	15.04.2020	
5.	Програмна реалізація системи	01.06.2020	
6.	Оформлення пояснювальної записки	10.06.2020	
7.	Захист програмного продукту	11.06.2020	
8.	Передзахист	11.06.2020	
9.	Захист	16.06.2020	

Студент _____ _____С.В. Пахут
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____О.О. Гагарін
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

до бакалаврської дипломної роботи Пахута Сергія Вадимовича

на тему: «Система керування подіями на основі інтерактивної карти»

Дана дипломна робота присвячена розробці системи керування подіями з пошуком найближчих актуальних подій і відображенням їх на інтерактивній карті.

В роботі зроблено аналіз проблеми пошуку за розташуванням, обрано технології, які б задоволняли поставленим вимогам. Систему розділено на функціональні частини, а саме: база даних, серверна частина і клієнтська частина (мобільний додаток).

Серверна частина розроблялася з використанням мови програмування PHP, клієнтська – використовуючи мову Java і Android API; база даних – MySQL.

Загальний об'єм роботи 63 сторінки, 7 рисунків, 2 додатки, 8 бібліографічних найменувань.

Ключові слова: мобільний додаток, клієнт-серверна архітектура, веб сервер, база даних, таблиця, програмний інтерфейс.

ANNOTATION

to the diploma thesis by Pakhut Serhii Vadymovych

on the topic «Event management system based on an interactive map»

This thesis is devoted to the development of an event management system with the search for the nearest current events and display them on an interactive map.

The analysis of the problem of search by location is made in the work, the technologies which would satisfy the set requirements are chosen. The system is divided into functional parts, namely: database, server part and client part (mobile application).

The server part was developed using the PHP programming language, the client part - using the Java language and Android API; database - MySQL.

Total capacity: 63 pages, 7 figures, 2 applications, 8 references.

Tags: mobile application, client-server architecture, web server, database, table, software interface.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУБД – система управління базами даних

API – прикладний програмний інтерфейс

БД – база даних; в контексті даної системи – реляційна база даних MySQL

XML – eXtensible Markup Language (мова розмітки)

HTTP – протокол передачі даних (гіпертексту)

WebSocket – це протокол, що призначений для обміну між браузером та веб-сервером в реальному часі

PHP – скриптова мова програмування для серверу

Web Server – це сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, видає їм HTTP-відповіді, зазвичай разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними. В контексті даної системи – Apache Web Server

ПЗ – програмне забезпечення

IDE – інтегроване середовище розробки — комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду та інших інструментів

ВСТУП

Протягом останнього часу щоденно у сфері програмування та освіти загалом відбуваються явища (відкриті лекції, конференції, тренінги, курси), що можуть зацікавати та принести користь учням, студентам або спеціалістам певної галузі, проте далеко не завжди реклама таких подій вчасно досягає цільову аудиторію.
Актуальність даної теми зумовлена ще й тим, що вже було створено декілька сервісів, що дозволяють переглядати та керувати подіями (наприклад, додаток Meetup), вони використовують різні критерії пошуку та рекомендацій, проте жодна з них не має в основі орієнтації на місцезнаходження користувача, що давало б змогу підписуватися на актуальні заходи в залежності від вподобань користувача а також від його розташування.

Метою даної роботи є впровадження системи, яка дозволяла б керувати заходами, які користувачу пропонується відвідати, з огляду на його місцезнаходження. Такий підхід має свої переваги та недоліки. Основною перевагою є відкидання заходів, які є актуальними, проте знаходяться поза зоною доступу користувача, що унеможливлює їх відвідування. Оскільки основним критерієм є розташування, недоліком може бути нагромадження неактуальних, проте близьких подій у зоні відвідування користувача.

У першому розділі ставляться задачі, що мають вирішуватися системою. Прийнято до уваги як задачі, що стосуються мобільного додатку (в основному отримання та оновлення місцезнаходження), так і серверної частини (обробка запитів, робота з базою даних). Задача про «оновлення і відображення розташування користувача та інформації про отримані події» стосується роботи з інтерфейсом та даними; алгоритму відправлення запитів та обробки результатів вцілому.

У другому розділі аналізується проблема та пропонуються варіанти

вирішення, розкриваються їх переваги і недоліки.

У третьому розділі розповідається про засоби розробки. Для мобільного додатку використовується мова програмування Java (для HTTP запитів бібліотека Volley, для відображення карти компонент GoogleMap). Для побудови серверної частини використовується мова програмування PHP (веб-сервер Apache). У якості системи керування базами даних використовується MySQL. Розробка відбувається в IDE AndroidStudio (додаток) та PHPStorm (код для серверної частини).

В четвертому розділі мова йде про архітектуру та функціонал додатку: розглядається діаграма прецедентів та надається схема навігації по екранам додатку. Також розкривається суть процесів, що відбуваються на сервері, а також йдеться про структуру бази даних.

В останньому розділі розповідається про процес установки додатку, вимоги до програмного забезпечення і до користувача, а також про те, як відбуватиметься робота з інтерфейсом додатку.

1. ПОСТАНОВКА ЗАДАЧ ДЛЯ СИСТЕМИ КЕРУВАННЯ ПОДІЯМИ

Дане програме забезпечення призначене для керування подіями з прив`язкою до місцезнаходження на основі інтерактивної карти.

Потенційними користувачами є в першу чергу постійні учасники різних заходів (відкриті лекції, конференції, курси і т.д.) – студенти, учні, спеціалісти. Проте, в широкому розумінні, додаток дозволяє організовувати події за інтересами в будь-якій галузі, тому може бути корисним широкому колу користувачів, які організовують заходи локального масштабу в будь-якій галузі.

1.1. Задача отримання місцезнаходження користувача

Для отримання актуальних подій в даній місцевості необхідно спочатку встановити місцезнаходження користувача. Для вирішення цієї задачі в додатку використовується готовий компонент GoogleMap Fragment. Цей компонент відображає карту та дозволяє позначати на ній певні об`єкти, в тому числі мітку з положенням користувача. Також є можливість ставити маркери на карті. Таким чином, завантажені події відображатимуться у вигляді маркерів із заголовком, описом події та часом. Даний компонент також дозволяє обробляти натискання на об`єкти (карту, маркер і т.д.), що використовується для побудови навігації між екранами, наприклад після натискання на карту з`являється кнопка переходу в інтерфейс додавання події. Після натискання на маркер є можливість перейти до екрану детального перегляду події, підписатися на неї або відписатися.

За допомогою Maps SDK можна додавати в додатку карти на основі даних сервісу "Google Карти". API автоматизує доступ до серверів Google, завантаження

даних, відображення карти і взаємодію користувачів з нею. За допомогою викликів API можна також додавати на карту маркери, багатокутники і накладення або змінювати уявлення окремих областей. Ці об'єкти містять додаткову географічну інформацію і відповідають за взаємодію користувача з картою. За допомогою API можна додати такі графічні елементи:

- ✓ значки, прив'язані до певних точок на карті (маркери);
- ✓ ламані лінії (полілінії);
- ✓ замкнуті фігури (багатокутники);
- ✓ бітові зображення, прив'язані до певних місць на карті (наземні накладення);
- ✓ набори картинок, що відображаються поверх фрагментів карти (фрагментного накладення).

Хоча б’єкт GoogleMap дозволяє відображати елементи, вказуючи їх координати, для отримання координат пристрою використовується клас FusedLocationProviderClient.

При встановленні відповідного обробника він повертає об’єкт Location, що містить координати пристрою та іншу інформацію. Таким чином, ці дані записуються в пам’яті і надсилаються на сервер для пошуку.

1.2. Задача обміну інформацією з сервером: пошуку та збереження інформації в базі даних

Для обміну інформацією між користувачами необхідно зберігати дані кожного користувача та інформацію про його місцезнаходження. Здійснення запитів на сервер та обробка відповідей має спрощену структуру завдяки бібліотеці Volley для Android. Вона дозволяє відправляти GET та POST HTTP запити, інкапсулюючи в

собі деталі реалізації цих методів, а самі запити можна поміщати у чергу для обробки.

HTTP - широко поширений протокол передачі даних, спочатку призначений для передачі гіпертекстових документів (тобто документів, які можуть містити посилання, що дозволяють організувати перехід до інших документів).

Абревіатура HTTP розшифровується як HyperText Transfer Protocol, «протокол передачі гіпертексту». Відповідно до специфікації OSI, HTTP є протоколом прикладного (верхнього, 7-го) рівня. Актуальна на даний момент версія протоколу, HTTP 1.1, описана в специфікації RFC 2616.

Протокол HTTP припускає використання клієнт-серверної структури передачі даних. Клієнтську програму формує запит і відправляє його на сервер, після чого серверне програмне забезпечення обробляє цей запит, формує відповідь і передає його назад клієнтові. Після цього клієнтську програму може продовжити відправляти інші запити, які будуть оброблені аналогічним чином.

Завдання, яке традиційно вирішується за допомогою протоколу HTTP - обмін даними між призначеним для користувача додатком, що здійснює доступ до веб-ресурсів (зазвичай це веб-браузер) і веб-сервером. На даний момент саме завдяки протоколу HTTP забезпечується робота Всесвітньої павутини.

Також HTTP часто використовується як протокол передачі інформації для інших протоколів прикладного рівня, таких як SOAP, XML-RPC і WebDAV. У такому випадку говорять, що протокол HTTP використовується як «транспорт». API багатьох програмних продуктів також має на увазі використання HTTP для передачі даних - самі дані при цьому можуть мати будь-який формат, наприклад, XML або JSON.

Як правило, передача даних по протоколу HTTP здійснюється через TCP / IP-з'єднання. Серверне програмне забезпечення при цьому зазвичай використовує TCP-порт 80 (і, якщо порт не вказано явно, то зазвичай клієнтське програмне

забезпечення за замовчуванням використовує саме 80-й порт для відкритих HTTP-з'єднань), хоча може використовувати і будь-який інший.

Таким чином, HTTP-запити відправляються на веб-сервер Apache (на якому має бути встановлений PHP) обробляються PHP-скриптами, після чого сервером видається відповідь: в разі успіху – дані, в разі невдачі – код помилки.

Іншою важливою задачею є збереження та пошук інформації в базі даних MySQL. Оскільки в даній системі є такі одиниці даних, як «користувач», «подія», «ділянка» і т.д., і всі вони пов`язані між собою, було обрано систему керування базами даних MySQL, типом збереження даних InnoDB, що підтримує первинні ключі.

Primary Key (Первинний ключ) є полем в таблиці, яке однозначно ідентифікує кожен рядок / запис в таблиці бази даних. Первинні ключі повинні містити унікальні значення. Первинний ключ стовпець не може мати значення NULL. Таблиця може мати тільки один первинний ключ, який може складатися з одного або декількох полів. Коли кілька полів використовуються в якості первинного ключа, їх називають складовим ключем. Якщо таблиця має первинний ключ, визначений на будь-якому полі (ях), то ви не можете мати два записи, які мають однакове значення цього поля (їй).

Саме завдяки первинним ключам є можливість для пошуку по декільком таблицям із використанням відношення Primary Key – Foreign Key.

1.3. Задача оновлення і відображення розташування користувача та інформації про отримані події

Для реалізації цієї задачі використовуються дані, отримані від серверу в попередній задачі. Завантажені події обробляються у циклі та додаються до фрагменту GoogleMap у вигляді маркерів. Паралельно з цим, постійно оновлюється

місцезнаходження користувача, яке відправляється на сервер з метою отримання актуальних подій.

Відповідне використання інформації про місцеположення залежить від цілей додатка. Наприклад, якщо додаток допомагає користувачеві знайти свій шлях під час прогулянки чи їзди на транспорті, або якщо додаток відстежує розташування активів, йому потрібно регулярно отримувати місцезнаходження пристрою. Ця інформація та багато іншого доступна в об’єкті Location, який додаток може отримати у з’єднаного постачальника послуг розташування. У відповідь API періодично оновлює додаток з найкращим доступним місцеположенням на основі доступних на даний момент постачальників послуг, таких як WiFi та GPS (Global Positioning System). Точність розташування визначається провайдерами, дозволами про місцезнаходження, які запитувалися додатком, та параметрами, які були встановлені у запиті про місцезнаходження. Вимагати регулярних оновлень про місцезнаходження пристрою можна за допомогою методу requestLocationUpdates() у постачальника локацій.

Для даного додатку важливою задачею є набуття актуальної інформації, для цього необхідно постійно оновлювати інформацію про місцезнаходження користувача та події поблизу. Оскільки додаток не потребує дуже точних даних про місцезнаходження, використовується лише WiFi (без даних GPS).

Щоб постійно оновлювати інформацію про поточні події, в додатку використовується функція locate(). Вона запитує поточні координати пристрою, відправляє запит на сервер і отримує поточні події з бази даних. Вона повинна виконуватися в циклі, при чому нові актуальні події можуть з’явитися на сервері, а можуть і не з’явитися. Тому, для вирішення проблеми постійного оновлення можливе використання технології WebSocket, яка дозволяла б отримувати оновлення про актуальні події саме для цього користувача у потрібний час. Ця технологія забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет.

Таким чином, ініціатором запиту не обов’язково має бути клієнт. В свою чергу, це дозволило б уникнути великої кількості необов’язкових запитів та знизити навантаження на сервер.

Для відображення отриманих подій було вирішено скористатися готовим рішенням Android Google Maps – маркерами.

Маркери позначають окремі місця розташування на карті. Маркери можна налаштовувати, наприклад змінювати їх кольори або значки. Інформаційні вікна можуть містити додаткову інформацію про зазначені точки.

За замовчуванням використовується стандартний значок, як в Google Картах, проте він був перевизначений в додатку. За допомогою API також можна змінити колір значка, його зображення або точку прив'язки. Маркери (об'єкти Marker) додаються на карту за допомогою методу GoogleMap.addMarker (markerOptions).

Маркери - це інтерактивні елементи. За умовчанням вони приймають події click і часто використовуються з прослуховувачем подій для виведення інформаційних вікон. Установка для властивості маркера draggable значення true дозволяє користувачеві змінювати положення маркера на карті. Можливість переміщення маркера активується довгим натисканням.

За замовчуванням, коли користувач торкається маркера, в нижньому правому куті мапи з’являється панель інструментів, яка надає швидкий доступ до мобільного додатку "Google Карти".

У маркері можна зберегти довільний об'єкт даних, використовуючи метод Marker.setTag (), і витягти цей об'єкт даних за допомогою методу Marker.getTag (). Так, в додатку був використаний цей метод, щоб визначити на яку саме подію натиснув користувач (записується її id) і надати можливість перейти до редагування цієї події.

В додатку відслідковуються події маркера і в них є свої обробники. Для цього необхідно було призначити об'єкту GoogleMap, до якого відносяться маркери,

відповідний прослуховувач. Якщо для одного з маркерів карти виникає подія, відповідний об'єкт Marker, що передається у вигляді параметра, виконує зворотний виклик. Щоб зіставити цей об'єкт Marker з початковим посиланням на Marker, слід використовувати метод equals (), а не оператор ==.

Для розширення функціоналу додатку можна також використовувати зображення подій і при відображенні на карті додавати їх до кожного маркера. Проте, таке рішення вимагає ряду додаткових функцій, що завантажували б зображення на сервер при створенні події і роздавали користувачам при завантаженні подій що, в свою чергу, значно збільшило би навантаження на сервер.

2. АНАЛІЗ ПРОБЛЕМИ ОРГАНІЗАЦІЇ ТА ПОШУКУ ПОДІЙ НА ОСНОВІ РОЗТАШУВАННЯ КОРИСТУВАЧА

Проблема організації керування подіями на основі розташування складається з ряду задач, для деяких з них існують готові рішення (наприклад, бібліотека для HTTP запитів для Android), деякі з них були частково вирішені в аналогічних додатках (наприклад, знаходження та оновлення розташування в GoogleMaps). З іншої сторони, деякі задачі (наприклад, оновлення карти в режимі реального часу та відображення актуальних подій) потребували свого рішення.

Проблема пошуку полягала у тому, щоб вибирати заходи на основі розташування, при цьому здійснюючи фільтрування неактуальних даних.

2.1. Огляд технологій

Фрагмент GoogleMap та Google API для нього, які використовуються в додатку, описані в документації для Android. Maps SDK для Android забезпечує підтримку спеціальних можливостей. По замовчуванню вони задіюються автоматично для будь-якої програми, що використовує API.

Коли користувачі включають функцію озвучування дій TalkBack на своїх мобільних пристроях, кожен окремий жест гортання на екрані виконує перехід від одного елемента користувальницького інтерфейсу до іншого. Альтернатива жесту гортання для вибору елементів інтерфейсу - переміщення пальця по інтерфейсу. Як тільки той чи інший елемент інтерфейсу потрапляє у фокус, функція TalkBack вимовляє його назву. Якщо користувач двічі торкнеться будь-якого місця на екрані, буде виконано дію, що знаходиться у фокусі. [3]

Для побудови серверної частини була використана мова PHP, вказівки про те,

як обробляти GET і POST HTTP-запити, а також про те, як працювати з базою даних, містяться в документації. Також там міститься інформація про особливості підключення і роботу з базою даних. [4]

Оскільки в даній системі скрипти на сервері не видають HTML-сторінок, а використовуються виключно як API, що повертає текст, було вирішено не використовувати сторонніх фреймворків. Проте, при розширенні функціоналу, систему було б доречно перевести на фреймворк (наприклад, Yii2 або Laravel) для спрощення роботи.

Для побудови чіткої структури взаємозв'язків між даними (наприклад, зв'язок між класом «подія» і «користувач») була обрана реляційна база даних MySQL.

Механізм зберігання даних – InnoDB. Причиною такого вибору стала велика кількість переваг цього механізму над іншими, наприклад MyISAM. Так, застосування InnoDB дозволяє використання базою даних таких функцій, як транзакція, зовнішні ключі. Він також сумісний з ACID (набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність).

Для створення таблиць, структри бази даних і реалізації потрібного функціоналу була використана документація MySQL. [5]

2.2. Аналіз аналогічних систем

Хоча прямих аналогів даній програмній системі не існує, можна виділити додатки або функції в додатках, що виконують схожі задачі, зокрема:

- ✓ Функція пошуку місць в Google Places [6]
- ✓ Додаток Meetup для пошуку подій з можливостю підписки [7]

Перевагою пошуку в Google Places є можливість переглянути місця поблизу на карті та шукати певне місце, проте цей додаток має дещо іншу мету, тому

недоліками в даному контексті є відсутність рекомендацій для користувача на основі минулих відвідувань, а також відустність часових рамок для подій. Google Place скоріше надає інформацію про місця, аніж про тимчасові заходи.

Перевагою додатку Meetup є зручний пошук, який навіть враховує розташування, проте немає зручного представлення подій на карті разом з розташуванням користувача і оновленням даних в режимі реального часу.

3. ОБГРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

Програмна система складається з мобільного додатка, написаного на мові Java та серверної частини на мові PHP, що обробляє запити мобільних клієнтів. Дані зберігаються в реляційній базі даних (MySQL).

При взаємодії додатку з сервером йдеться мова про клієнт-серверну архітектуру, в якій завдання або мережеве навантаження розподілені між постачальниками послуг, що називаються серверами, і замовниками послуг, що називаються клієнтами. Фактично клієнт і сервер - це програмне забезпечення. Зазвичай ці програми розташовані на різних обчислювальних машинах і взаємодіють між собою через обчислювальну мережу за допомогою мережевих протоколів, але вони можуть бути розташовані також і на одній машині. Програми-сервери очікують від клієнтських програм запити і надають їм свої ресурси у вигляді даних (наприклад, завантаження файлів за допомогою HTTP, FTP, BitTorrent, потокове мультимедіа або робота з базами даних) або у вигляді сервісних функцій (наприклад, робота з електронною поштою, спілкування за допомогою систем миттєвого обміну повідомленнями або перегляд web-сторінок у всесвітній павутині). Оскільки одна програма-сервер може виконувати запити від безлічі програм-клієнтів, її розміщують на спеціально виділеній обчислювальній машині, налаштованої особливим чином, як правило, спільно з іншими програмами-серверами, тому продуктивність цієї машини повинна бути високою. Через особливу роль такої машини в мережі, специфіки її обладнання та програмного забезпечення, її також називають сервером, а машини, які виконують клієнтські програми, відповідно, клієнтами. [1]

3.1. Мобільний додаток Java

Java – це об'єктно-орієнтована мова, якою можна управляти або через віртуальну машину, або за допомогою браузера. Це дає зручність в разі повторного використання розробленого коду та оновлення програмного забезпечення, що у випадку з ОС для мобільних відбувається досить часто.

Коли обговорюється виробництво додатків для мобільних, найчастіше це має на увазі нативні продукти, тобто ті, які пишуться під певну мобільну ОС. Тут програмування на Java актуально в першу чергу для тих, хто вирішив створювати додатки під Android, а ось для iOS-розробників Java вже є скоріше другорядним засобом, на відміну від комбінації з ObjectiveC і Swift.

Побудова додатку ведеться в Android Studio — інтегрованій середі розробки (IDE) для роботи з платформою Android. Це дозволяє редагувати інтерфейс в «дизайнері інтерфейсів» IDE, збирати проект за допомогою Gradle та встановлювати тестову версію на мобільний через USB.

При запуску установки «Run» автоматично запускається збірка проекту в Gradle. Система збірки Android збирає ресурси програми та вихідний код та пакує їх у APK, які ви можете тестувати, розгортати, підписувати та розповсюджувати. Android Studio використовує Gradle, вдосконалений інструментарій збірки, для автоматизації та управління процесом збирання, дозволяючи визначати гнучкі налаштовані конфігурації збірки. Кожна конфігурація збірки може визначати власний набір коду та ресурсів, використовуючи при цьому деталі, загальні для всіх версій вашої програми. Плагін Android для Gradle працює з набором інструментів збірки, щоб забезпечити процеси та настроювані налаштування, характерні для створення та тестування додатків Android (рисунок 1). [2]

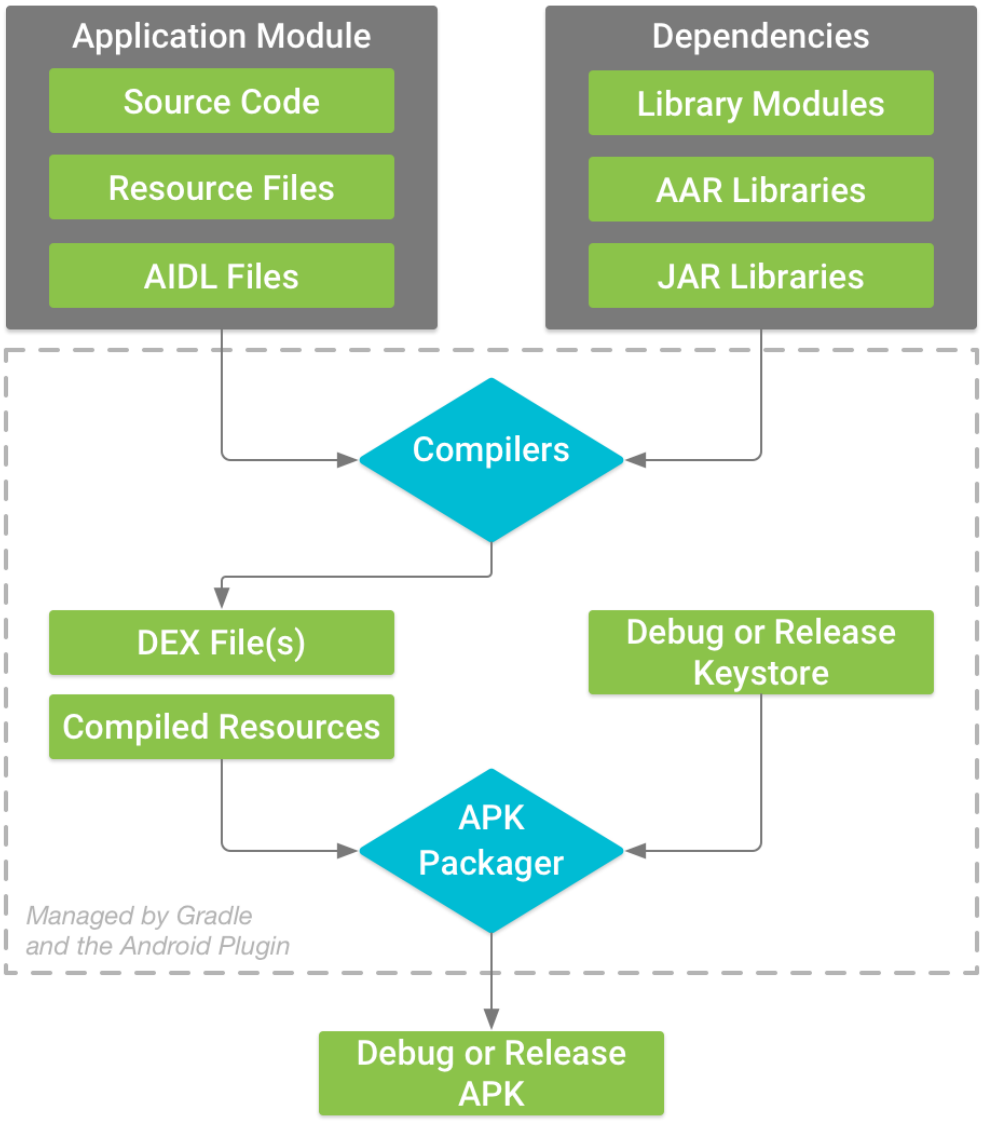


Рисунок 1. Схема налагодження та збірки додатку

Щоб побудувати інтерфейс додатку, необхідно вказати компоненти та їх ієрархію в файлі xml. Базовим елементом при цьому завжди виступатиме макет (layout). Макет визначає візуальну структуру користувальницького інтерфейсу, наприклад, призначеного для користувача інтерфейсу операції або віджета додатка. Існує два способи оголосити макет:

Оголошення елементів призначеного для користувача інтерфейсу в XML. В

Android є зручний довідник XML-елементів для класів View і їх підкласів, наприклад таких, які використовуються для віджетів і макетів.

Створення екземплярів елементів під час виконання. Ваша програма може програмним чином створювати об'єкти View і ViewGroup (а також керувати їх властивостями).

Платформа Android надає гнучкість при використанні будь-якого з цих способів для оголошення призначеного для користувача інтерфейсу додатку і його управління. Наприклад, ви можете оголосити в XML макети за замовчуванням, включаючи елементи екрану, які будуть відображатися в макетах, і їх властивості. Потім ви можете додати в додаток код, який дозволяє змінювати стан об'єктів на екрані (включаючи оголошені в XML) під час виконання.

З огляду на потреби інтерфейсу було обрано ConstraintLayout, який дозволяє створювати великі і складні макети з ієрархією плоского перегляду, але без вкладених груп перегляду. (рисунок 2).

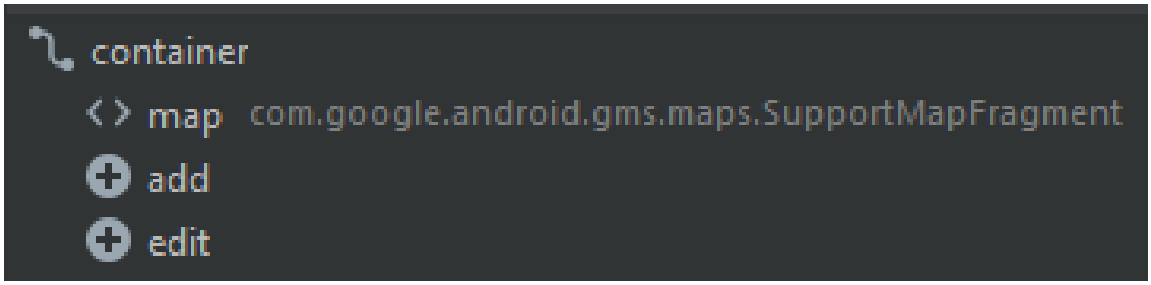


Рисунок 2. Ієрархія компонентів для інтерфейсу карти

Основною частиною інтерфейсу є карта, вона реалізована за допомогою компоненту Map Fragment. Цей фрагмент є найпростішим способом розміщення карти в додатку. Це обгортка навколо перегляду карти, щоб автоматично обробляти необхідні потреби життєвого циклу. Будучи фрагментом, цей компонент може бути доданий до файлу макета діяльності просто за допомогою XML нижче.

```
<fragment
   class = "com.google.android.gms.maps.MapFragment"
   android: layout_width = "match_parent"
   android: layout_height = "match_parent" />
```

GoogleMap має бути отриманий за допомогою getMapAsync (OnMapReadyCallback). Цей клас автоматично ініціалізує систему карт та подання. Перегляд можна видалити, коли викликається метод onDestroyView () MapFragment і встановлено параметр useViewLifecycleInFragment (boolean). Коли це відбувається, MapFragment більше не діє, поки подання не буде відтворено знову пізніше, коли буде викликаний метод onCreateView (LayoutInflater, ViewGroup, Bundle) MapFragment.

Будь-які об’єкти, отримані з GoogleMap, пов’язані з представленням. Важливо не триматися за об'єкти (наприклад, маркер) поза життєвим циклом, інакше це спричинить витік пам'яті.

Для того, щоб дізнатися розміщення користувача, необхідно взяти дозвіл на отримання розташування і звернутися до відповідного класу. Використовуючи API розташування служб Google Play, додаток може запитати останнє відоме місце на пристрої користувача. У більшості випадків нас цікавить поточне місцезнаходження користувача, яке зазвичай еквівалентне останньому відомому розташуванню пристрою. Зокрема, скористаємося провідником розташування, щоб отримати останнє відоме місце на пристрої. Постачальник плавного розташування є одним із API-адрес розташування в службах Google Play. Він керує базовою технологією розташування та надає простий API, щоб ви могли задавати вимоги на високому рівні, як-то висока точність або низька потужність. Це також оптимізує використання пристрою живлення від акумулятора.

Щоб не тільки отримати, але й постійно оновлювати дані про розташування,

звернемося до об’єкту «fused location provider». Відповідне використання інформації про місцеположення може бути корисним для користувачів вашого додатка. Наприклад, якщо ваш додаток допомагає користувачеві знайти свій шлях під час прогулянки чи їзди на транспорті, або якщо ваш додаток відстежує розташування активів, йому потрібно регулярно отримувати місцезнаходження пристрою. Окрім географічного розташування (широта та довгота), ви можете надати користувачеві додаткову інформацію, таку як підшипник (горизонтальний напрямок ходу), висота чи швидкість пристрою. Ця інформація та багато іншого доступна в об’єкті Location, який ваш додаток може отримати у з’єднаного постачальника послуг розташування. У відповідь API періодично оновлює ваш додаток з найкращим доступним місцеположенням на основі доступних на даний момент постачальників послуг, таких як WiFi та GPS (Global Positioning System). Точність розташування визначається провайдерами, дозволами про місцезнаходження, які ви запитували, та параметрами, які ви встановили у запиті про місцезнаходження.
Отримувати регулярні оновлення про місцезнаходження пристрою можна за допомогою методу requestLocationUpdates () у постачальника локацій.

Оскільки користувач переходить у ваш додаток, виходить із нього та повертається назад, діяльності "Activity" переходять через різні стани в їх життєвому циклі. Клас активності пропонує ряд зворотних викликів, які дозволяють діяльності знати, що стан змінився: що система створює, зупиняє або поновлює діяльність або знищує процес, в якому знаходиться діяльність.

У межах методів зворотного виклику життєвого циклу ви можете оголосити, як поводиться ваша діяльність, коли користувач залишає та знову входить у діяльність. Наприклад, якщо ви створюєте програвач потокового відео, ви можете призупинити відео та припинити мережеве з'єднання, коли користувач переключиться на інший додаток. Коли користувач повернеться, ви можете знову підключитися до мережі та дозволити користувачеві відновити відео з того самого

місця. Іншими словами, кожен зворотний виклик дозволяє виконувати конкретну роботу, відповідну певній зміні стану. Правильна робота в потрібний час та належне виконання переходів роблять ваш додаток більш надійним та ефективним. Наприклад, хороша реалізація зворотних викликів життєвого циклу може допомогти забезпечити уникнення таких помилок:

- ✓ Збій, якщо користувач отримує телефонний дзвінок або переходить на інший додаток під час використання вашої програми.
- ✓ Споживання цінних системних ресурсів, коли користувач активно ним не користується.
- ✓ Втрата прогресу користувача, якщо він покине вашу програму і пізніше повернеться до неї.
- ✓ Збій або втрата користувачем прогресу, коли екран змінює орієнтацію між пейзажем та портретом.

Документ починається з опису парадигми життєвого циклу. Далі, він пояснює кожен із зворотних викликів: що відбувається внутрішньо під час їх виконання та що слід реалізувати під час них. Потім коротко вводиться взаємозв'язок між станом активності та вразливістю процесу до вбивства системою. Нарешті, в ньому обговорюються декілька тем, пов'язаних з переходами між станами діяльності.

Для навігації в переходах між етапами життєвого циклу діяльності клас активності надає основний набір із шести зворотних викликів: onCreate (), onStart (), onResume (), onPause (), onStop () та onDestroy (). Система викликає кожен із цих зворотних викликів, коли діяльність переходить у новий стан.

Життєвий цикл додатку має типову структуру (рисунок 3).

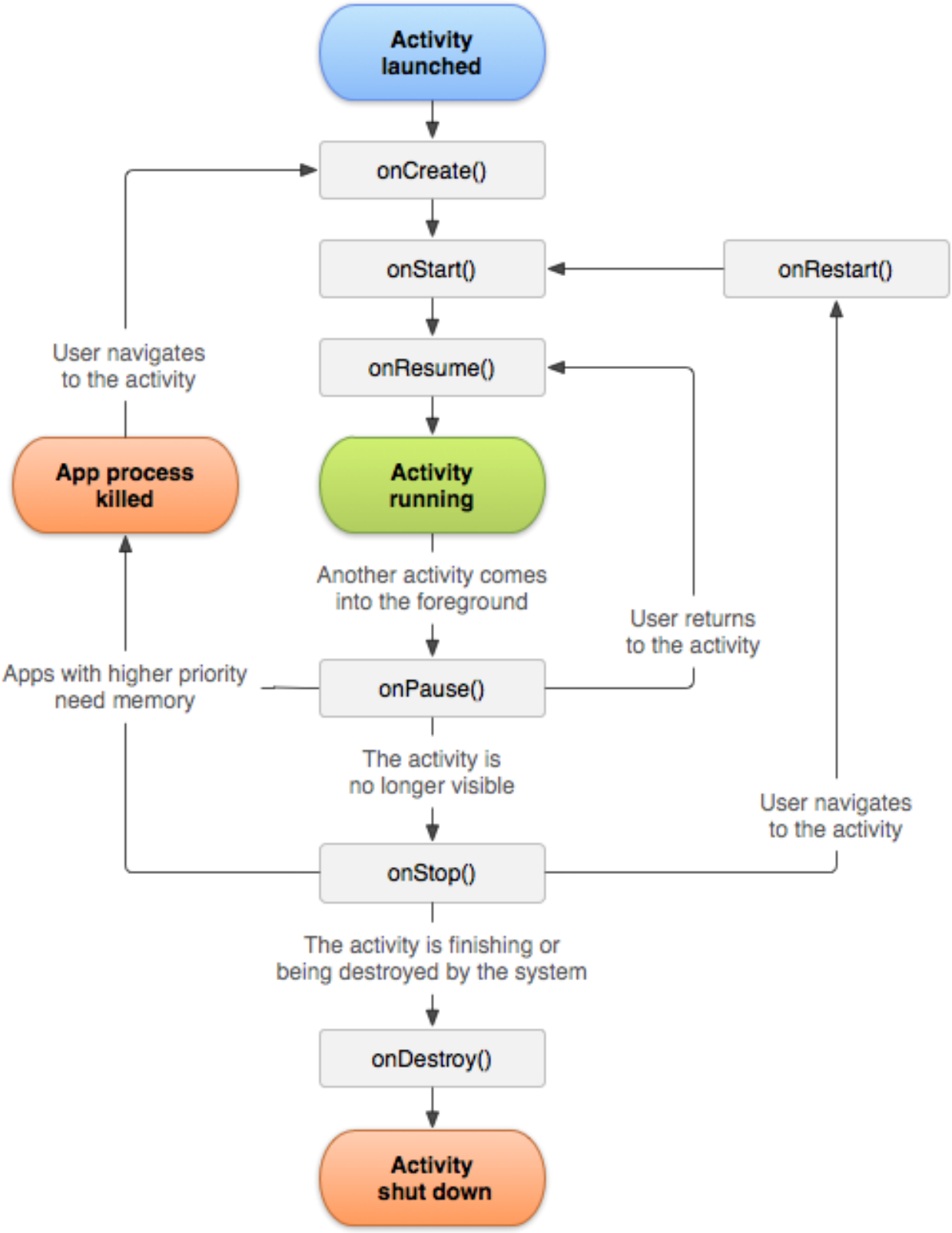


Рисунок 3. Схема життєвого циклу додатка

Всі налаштування інтерфейсу та початкова обробка даних відбуваються в методі onCreate(). Цей зворотний виклик спрацьовує, коли система вперше створює активність. При створенні діяльності активність переходить у новий стан. У методі onCreate () виконується основна логіка запуску програми, яка повинна відбуватися лише один раз протягом усього життєвого циклу. Наприклад, ваша реалізація onCreate () може прив'язувати дані до списків, пов'язувати діяльність із ViewModel та інстанціювати деякі змінні області класу. Цей метод отримує параметр saveInstanceState, який є об'єктом Bundle, що містить раніше збережений стан діяльності. Якщо активність раніше ніколи не існувала, значення об'єкта Bundle є нульовим.

3.2. Серверна частина Apache Web Server + PHP

Серверну частину складають скрипти, написані на мові програмування PHP, однак щоб запускати скрипти, при цьому обмеживши доступ до інших файлів і папок, необхідно використовувати веб-сервер.

Веб-сервер - сервер, що приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, і видає їм HTTP-відповіді, як правило, разом з HTML-сторінкою, зображенням, файлом, медіа-потоком або іншими даними. Веб-сервером називають як програмне забезпечення, яке виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює.

Клієнт, яким зазвичай є веб-браузер, передає веб-серверу запити на отримання ресурсів, позначених URL-адресами. Ресурси - це HTML-сторінки, зображення, файли, медіа-потоки або інші дані, які необхідні клієнту. У відповідь веб-сервер передає клієнту запитані дані. Цей обмін відбувається по протоколу HTTP.
Ядро Apache включає в себе основні функціональні можливості, такі як обробка конфігураційних файлів, протокол HTTP та система завантаження модулів. Система

конфігурації Apache заснована на текстових конфігураційних файлах. Має три умовних рівня конфігурації:

- ✓ Конфігурація сервера (httpd.conf). Директиви конфігурації згруповані в три основні розділи: директиви, що керують процесом Apache в цілому (глобальний оточення); директиви, що визначають параметри «головного» сервера, або сервера «за замовчуванням», який відповідає на запити, які не обробляються віртуальними хостами (визначають також установки за замовчуванням для всіх інших віртуальних хостів); установки для віртуальних хостів, що дозволяють обробляти запити Web одним-єдиним сервером Apache, але направляти по роздільним адресами IP або іменам хостов.
- ✓ Конфігурація віртуального хоста (httpd.conf с версії 2.2, extra / httpd-vhosts.conf).
- ✓ Конфігурація рівня каталогу (.htaccess).

Має власну мову конфігураційних файлів, заснований на блоках директив. Практично всі параметри ядра можуть бути змінені через конфігураційні файли, аж до управління MPM. Велика частина модулів має власні параметри.

Частина модулів використовує в своїй роботі конфігураційні файли операційної системи (наприклад / etc / passwd і / etc / hosts). Крім цього, параметри можуть бути задані через ключі командного рядка.

Apache HTTP Server підтримує модульність. Існує більше 500 модулів, що виконують різні функції. Частина з них розробляється командою Apache Software Foundation, але основна кількість - окремими open source-розробниками.

Модулі можуть бути як включені до складу сервера в момент компіляції, так і завантажені динамічно, через директиви конфігураційного файлу. У модулях реалізуються такі речі, як:

- ✓ Підтримка мов програмування.

- ✓ Додавання функцій.
- ✓ Виправлення помилок або модифікація основних функцій.
- ✓ Посилення безпеки.
- ✓ Частина веб-додатків, наприклад панелі керування ISPmanager і VDSmanager реалізовані у вигляді модуля Apache.

Apache має вбудований механізм віртуальних хостів. Він дозволяє повноцінно обслуговувати на одному IP-адресу безліч сайтів (доменних імен), відображаючи для кожного з них власне вміст.

Для кожного віртуального хоста можна вказати власні настройки ядра і модулів, обмежити доступ до всього сайту або окремих файлів. Деякі MPM, наприклад Apache-ITK, дозволяють запускати процес httpd для кожного віртуального хоста з окремими ідентифікаторами uid і guid.

Також існують модулі, що дозволяють враховувати і обмежувати ресурси сервера (CPU, RAM, трафік) для кожного віртуального хоста.

Директиви Directory відповідають за налаштування прав доступу до тієї чи іншої директорії в файловій системі. Синтаксис тут такий:
<Directory /адреса/в/файловій/системі/>
Параметр значення
</Directory>
Тут доступні такі основні опції:

- ✓ AllowOverride - вказує чи потрібно читати .htaccess файли з цієї папки, це такі ж файли налаштувань і таким же синтаксисом. All - вирішувати все, None - не читати ці файли.
- ✓ DocumentRoot - встановлює з якої папки потрібно брати документи для відображення користувачеві.

- ✓ Options - вказує які особливості веб-сервера потрібно дозволити в цій папці. Наприклад, All - дозволити все, FollowSymLinks - переходити по символічним посиланням, Indexes - відображати вміст каталогу якщо немає файлу індексу.
- ✓ Require - встановлює, які користувачі мають доступ до цього каталогу. Require all denied - всім заборонити, Require all granted - всім дозволити. можна використовувати замість all директиву user або group щоб явно вказати користувача.
- ✓ Order - дозволяє управляти доступом до директорії. Приймає два значення Allow, Deny - дозволити для всіх, крім зазначених або Deny, Allow - заборонити для всіх, крім зазначених.

Тепер перейдемо безпосередньо до обробки запитів в PHP. Усередині PHP-скрипта є кілька способів отримання доступу до даних, переданих клієнтом по протоколу HTTP. До версії PHP 4.1.0 доступ до таких даних здійснювався по іменах переданих змінних (нагадаємо, що дані передаються у вигляді пар «ім'я змінної, символ« = », значення змінної»). Таким чином, якщо, наприклад, було передано first_name = Sergiy, то всередині скрипта з'являлася змінна $ first_name зі значенням Sergiy. Якщо потрібно розрізняти, яким методом були передані дані, то використовувалися асоціативні масиви $HTTP_POST_VARS і $ HTTP_GET_VARS, ключами яких були імена переданих змінних, а значеннями - відповідно значення цих змінних. Таким чином, якщо пара first_name = Sergiy передана методом GET, то $HTTP_GET_VARS ["first_name"] = " Sergiy".

Використовувати в програмі імена переданих змінних безпосередньо небезпечно. Тому було вирішено починаючи з PHP 4.1.0 задіяти для звернення до змінних, переданим за допомогою HTTP-запитів, спеціальний масив - $_REQUEST. Цей масив містить дані, передані методами POST і GET, а також за допомогою HTTP cookies. Це суперглобальний асоціативний масив, тобто його значення можна отримати в будь-якому місці програми, використовуючи як ключ ім'я відповідної

змінної (елементу форми).

Після введення масиву $ _REQUEST масиви $ HTTP_POST_VARS і $HTTP_GET_VARS для однорідності були перейменовані в $ _POST і $ _GET відповідно, але самі вони з ужитку не зникли з міркувань сумісності з попередніми версіями PHP. На відміну від своїх попередників, масиви $_POST і $ _GET стали суперглобальними, тобто доступними безпосередньо і всередині функцій і методів. Наведемо приклад використання цих масивів. Припустимо, нам потрібно обробити форму, що містить елементи введення з іменами first_name, last_name, kurs (наприклад, форму form.html, наведену вище). Дані були передані методом POST, і дані, передані іншими методами, ми обробляти не хочемо. Це можна зробити наступним чином (рисунок 4).

```
<?php
$str = "Здравствуйте,
".$_POST ["first_name"]."
".$_POST ["last_name"] ."! <br>";
$str .= "Вы выбрали для изучения курс по ".
$_POST["kurs"];
echo $str;
?>
```

Рисунок 4. Робота з суперглобальними масивами

Дуже важливим аспектом серверної частини є зберігання даних та їх обробка. Для цього необхідно із скрипта PHP підключитися до бази даних. Для підключення

до MySQL з PHP нам треба вказати настройки підключення: адреса сервера, логін, пароль, назва бази даних і т.д. Так як зазвичай підключення до БД використовуються безліччю скриптів, то нерідко настройки підключення виносяться в окремий файл, завдяки чому легше їх оперативно змінювати. Якщо ми будемо підключатися до сервера на локальній машині, то адресою сервера буде localhost. За замовчуванням на локальному сервері MySQL вже є користувач root, під яким ми і будемо підключатися. І також нам необхідний пароль, який ми вказали при установці MySQL.

Насамперед необхідно підключити скрипт з налаштуваннями за допомогою інструкції require_once. Щоб з'єднатися застосовуємо функцію mysqli_connect(). Вона приймає всі конфігураційні налаштування і підключається до сервера. У разі помилки підключення спрацьовує оператор die(), який виводить повідомлення про помилку і завершує роботу скрипта. А в разі успішного підключення функція mysqli_connect() повертає об'єкт підключення у вигляді змінної $ link.

Після закінчення роботи підключення потрібно закрити. Для цього застосовується функція mysqli_close(), яка в якості параметра приймає об'єкт підключення.

Щоб здійснити запит до бази даних, нам треба використовувати функцію mysqli_query(), яка приймає два параметри: об'єкт підключення і рядок запиту на мові SQL. Наприклад:

```
<?php
require_once 'connection.php'; // підключаємо скрипт
// підключаємося до сервера
$link = mysqli_connect($host, $user, $password, $database)
    or die("Помилка " . mysqli_error($link));
// виконуємо операції з базою даних
$query ="SELECT * FROM phones";
```

```
$result = mysqli_query($link, $query) or die("Помилка " . mysqli_error($link));
if($result)
{
   echo "Виконання запиту пройшло успішно";
}
// закриваємо підключення
mysqli_close($link);
?>
```

Більшість баз даних підтримують концепцію підготовлених запитів. Для формування підготовлених запитів в додатку використовується PDO. Такі запити є більш оптимізовані і надають захист проти SQL-ін`єкцій.

Це можна описати, як якийсь вид скомпільованої шаблону SQL запиту, який буде запускатися додатком і налаштовуватися за допомогою вхідних параметрів. У підготовлених запитів є дві незаперечних переваги:

✓ Запит необхідно один раз підготувати і потім його можна запускати стільки раз, скільки потрібно, причому як з тими ж, так і з відмінними параметрами. Коли запит підготовлений, СУБД аналізує його, компілює і оптимізує план його виконання. У разі складних запитів цей процес може займати відчутний час і помітно сповільнити роботу програми, якщо буде потрібно багато разів виконувати запит з різними параметрами. При використанні підготовленого запиту СУБД аналізує / компілює / оптимізує запит будь-якої складності тільки один раз, а додаток запускає на виконання вже підготовлений шаблон. Таким чином підготовлені запити споживають менше ресурсів і працюють швидше.

✓ Параметри підготовленого запиту не потрібно екранувати лапками; драйвер це робить автоматично. Якщо в додатку використовуються виключно підготовлені запити, розробник може бути впевнений, що ніяких SQL-ін'єкцій трапитися не може

(однак, якщо інші частини тексту запиту створюються з неекранованим введенням, то SQL ін'єкція як і раніше можлива).

Підготовлені запити також корисні тим, що PDO може емулювати їх, якщо драйвер бази даних не має подібного функціоналу. Це означає, що додаток може користуватися однією і тією ж методикою доступу до даних незалежно від можливостей СУБД.

Якщо СУБД підтримує вихідні параметри, додаток може користуватися ними також як і вхідними. Вихідні параметри зазвичай використовують для отримання даних зі збережених процедур. Користуватися вихідними параметрами трохи складніше, так як розробнику необхідно знати максимальний розмір одержуваних значень ще на етапі завдання цих параметрів. Якщо витягають значення виявиться більше, ніж передбачалося, буде викликана помилка. Можна задати параметр одночасно вхідним і вихідним; синтаксис при цьому той же, що і для вихідних параметрів. У наступному прикладі рядок 'привіт' передається в збережену процедуру, а потім цей рядок буде замінена повертається значенням.
Якщо є які-небудь помилки підключення, то спрацює механізм винятків - PDOException. Потрібно завжди обертати операції PDO в блок try / catch. Можна зловити виняток, якщо ви хочете обробляти помилки, або залишити його для глобального обробника винятків (exception), створеного за допомогою set_exception_handler(). У PDO є спеціальні функції для помилок:

- ✓ errorCode() - поверне номер помилки,
- ✓ errorInfo() - поверне масив з номером помилки і описом.

Вони потрібні так як за замовчуванням режим роботи з помилками ERRMODE_SILENT.

Багато веб-додатків виграють від створення постійних підключень до серверів баз даних. Постійні з'єднання не зачиняються в кінці скрипта, а кешуються і

повторно використовуються, коли інший сценарій запитує з'єднання, використовуючи ті ж облікові дані. Постійне з'єднання дозволяє зменшити ресурси на створення нового з'єднання кожного разу, коли сценарій повинен звернутися до бази даних, що призводить до більш швидкої роботи веб-додатків.

Для вибірки даних використовуються методи fetch () або fetchAll (). Перед викликом функції потрібно вказати PDO як дані будуть діставатися з бази. PDO :: FETCH_ASSOC поверне рядки у вигляді асоціативного масиву з іменами полів в якості ключів. PDO :: FETCH_NUM поверне рядки у вигляді числового масиву. За замовчуванням вибірка відбувається з PDO :: FETCH_BOTH, який дублює дані як з чисельними так і з асоціативними ключами, тому рекомендується вказати один спосіб, щоб не мати дублюючих масивів.

При виникненні помилок на сервері буде отримане виключення Exception, яке буде записане в лог. Лог - це файл в якому зберігаються записи про події на сервері. Список необхідно вести для відстеження інформації про відвідувачів і їх діях на сервері.

Якщо повідомлення в логах супроводжуються достатнім обсягом контексту, то це значно спрощує налагодження, тому що вам буде доступно більше даних про ситуацію, в якій сталася подія. Тому, в додатку в лог-файл записується текст помилки, дата і користувач, у якого вона виникла.

Статична функція getLogger () поверне логер, далі функція log () запише наші дані. Але перед цим необхідно задати шлях, де будуть зберігатися логи. Наприклад, поточна папка:

Logger :: $ PATH = dirname (__ FILE__);

Ця функція log() є обгорткою для стандартної функції логування в PHP - error_log(), яка відправляє повідомлення про помилку заданому обробнику помилок.

Для ефективної побудови типових запитів до бази даних було створено і використано клас Builder, що містить методи, які дозволяють покроково будувати

MySQL-запит. Для реалізації цього класу було використано шаблон проектування Fluent interface. Завдяки цьому прийому читабельність коду стала близькою до звичайного тексту.

З іншої сторони, при цьому клас Builder використовує підготовлені запити, що дозволяє повністю уникнути SQL-ін’єкцій. Іншими словами, в системі використовується власна реалізація класу QueryBuilder, який реалізований у більшості фреймворків (Laravel, Yii2 та інші).

3.3. СКБД MySQL

Для збереження даних було обрану реляційну базу даних MySQL. Реляційна база даних - це набір даних з зумовленими зв'язками між ними. Ці дані організовані у вигляді набору таблиць, що складаються із стовпців і рядків. У таблицях зберігається інформація про об'єкти, представлених в базі даних. У кожному стовпчику таблиці зберігається певний тип даних, в кожному осередку - значення атрибута. Кожна стоку таблиці являє собою набір пов'язаних значень, що відносяться до одного об'єкту або сутності. Кожен рядок в таблиці може бути позначена унікальним ідентифікатором, званим первинним ключем, а рядки з декількох таблиць можуть бути пов'язані з допомогою зовнішніх ключів. До цих даних можна отримати доступ багатьма способами, і при цьому реорганізовувати таблиці БД не потрібно.

Нормальна форма - якість зв'язку в реляційної моделі даних, що характеризує його з точки зору надмірності, потенційно призводить до логічно помилкових результатів вибірки або зміни даних. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення.

Процес перетворення відносин бази даних до виду, який відповідає нормальним формам, називається нормалізацією. Нормалізація призначена для

приведення структури БД до виду, що забезпечує мінімальну логічну надмірність, і не має на меті зменшення або збільшення продуктивності роботи або ж зменшення або збільшення фізичного обсягу бази даних. Кінцевою метою нормалізації є зменшення потенційної суперечливості збереженої в базі даних інформації. Як зазначає К. Дейт, загальне призначення процесу нормалізації полягає в наступному:

- ✓ виключення деяких типів надмірності;
- ✓ усунення деяких аномалій оновлення;
- ✓ розробка проекту бази даних, який є досить «якісним» поданням реального світу, інтуїтивно зрозумілий і може служити хорошою основою для подальшого розширення;
- ✓ спрощення процедури застосування необхідних обмежень цілісності.

Усунення надмірності проводиться, як правило, за рахунок декомпозиції відносин таким чином, щоб в кожному відношенні зберігалися тільки первинні факти (тобто факти, що не виводяться з інших збережених фактів). [8]

Для роботи з таблицями звернемося до синтаксису MySQL. CREATE TABLE створює таблицю із заданим іменем. Ви повинні мати CREATE привілей для таблиці. За замовчуванням таблиці створюються в базі даних за замовчуванням за допомогою механізму зберігання InnoDB. Помилка виникає, якщо існує таблиця, якщо немає бази даних за замовчуванням або якщо база даних не існує.
MySQL не обмежує кількість таблиць. Базова файлова система може мати обмеження на кількість файлів, що представляють таблиці. Окремі двигуни зберігання можуть встановлювати особливі обмеження для двигуна. InnoDB дозволяє до 4 мільярдів таблиць.

При створенні таблиці CREATE TABLE є можливість вказати наступні характеристики:

- ✓ Назва таблиці

- ✓ Тимчасові таблиці
- ✓ Клонування та копіювання таблиці
- ✓ Типи даних та атрибути стовпців
- ✓ Індекси, foreign-ключі та обмеження перевірки
- ✓ Параметри таблиці
- ✓ Розбиття таблиці

При створенні колонок, необхідно вказувати їх типи. Таблиця бази даних містить декілька стовпців із конкретними типами даних, такими як числові або рядкові. MySQL надає більше типів даних, крім простого числового чи рядкового. Кожен тип даних в MySQL може бути визначений за такими характеристиками:

- ✓ Тип значення, які він представляє.
- ✓ Простір, який займає, і чи є значення фіксованої або змінної довжини.
- ✓ Значення типу даних можуть бути індексовані чи ні.
- ✓ Як MySQL порівнює значення певного типу даних.

Для формування більш складних запитів, в мобільному додатку не тільки створені таблиці, але й встановлені зв`язки між ними за допомогою первинних ключів (primary keys).

Первинний ключ таблиці являє собою стовпчик або набір стовпців, які ви використовуєте у своїх найважливіших запитах. Він має асоційований індекс для швидкого виконання запитів. Ефективність запиту виграє від оптимізації NOT NULL, оскільки вона не може включати будь-які значення NULL. За допомогою системи зберігання InnoDB дані таблиці фізично організовані для здійснення надшвидких пошуків і сортування на основі стовпчика або стовпців первинного ключа.

Якщо ваша таблиця велика і важлива, але в ній немає очевидного стовпчика або набору стовпців, який би використовувався в якості основного ключа, ви можете створити окремий стовпець зі значеннями автоматичного збільшення, який слід використовувати як первинний ключ. Ці унікальні ідентифікатори можуть слугувати вказівниками на відповідні рядки в інших таблицях, коли ви приєднуєте таблиці за допомогою зовнішніх ключів.

Деякі таблиці в базі даних MySQL пов'язані між собою. Найчастіше рядок в одній таблиці пов'язаний з декількома рядками в іншій таблиці. Вам потрібен стовпець для з'єднання відповідних рядків у різних таблицях. У багатьох випадках ви включаєте стовпчик в одну таблицю для зберігання даних, які відповідають даним у стовпчику первинного ключа іншої таблиці.

Поширена програма, для якої потрібна база даних з двома пов'язаними таблицями, - це програма замовлення клієнта. Наприклад, одна таблиця містить інформацію про клієнта, таку як ім'я, адреса та номер телефону. Кожен клієнт може мати від нуля до багатьох замовлень.

Таким чином, в базі даних серверної частини мобільного додатку визначені допоміжні таблиці, які дозволяють вирішувати проблему при відношеннях N до N. Так, існує N користувачів додатку, кожен з яких може підписатися на N подій. Тоді таке відношення з допоміжною таблицею виглядатиме так, як показано на рисунку (рисунок 5).

Завдяки таким зв’язкам можна виконувати більш складні запити і робити вибірки з кількох таблиць. Так, можна зробити вибірку, що покаже кількість підписників на кожну подію:

```
SELECT event.id, COUNT(*) AS count
FROM `event`
INNER JOIN participant
```

on event.id = participant.event_id

GROUP BY participant.event_id

ORDER BY event.id

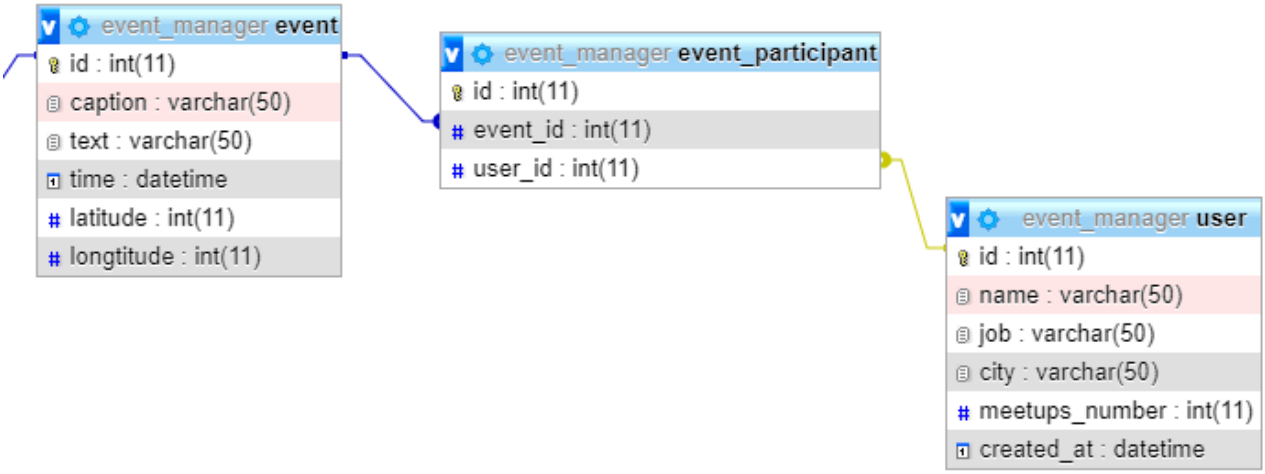


Рисунок 5. Введення допоміжної таблиці «учасник» для вирішення зв`язку N до N між «подією» та «користувачем»

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1. Діаграма прецедентів

Діаграма варіантів використання (англ. Use case diagram) в UML - діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні.
Прецедент - можливість модельованої системи, завдяки якій користувач може отримати конкретний, вимірний і потрібний йому результат. Прецедент відповідає окремому сервісу системи, визначає один з варіантів її використання і описує типовий спосіб взаємодії користувача з системою.

Діаграма класів (англ. class diagram) — статичне представлення структури моделі. Відображає декларативні елементи, такі як: типи даних, класи, їх зміст та відношення. Діаграма класів може містити позначення для пакетів (в тому числі вкладених пакетів). Також, діаграма класів може містити позначення деяких елементів поведінки, проте їх динаміка розкривається в інших типах діаграм. Серед взаємозв’язків виділяють асоціацію, агрегацію та композицію. На діаграмі класів показують інтерфейси, класи, об'єкти й кооперації, а також їх відносини.

В той час, як діаграма класів характеризує форми, які приймають дані, та їх взаємозв’язки, діаграма Use case безпосередньо показує функціонал системи та способи, якими кінцевий користувач взаємодіє з нею.

Для даної програмної системи (додаток + серверна частина) діаграма Use case приймає наступний вигляд (рисунок 6).

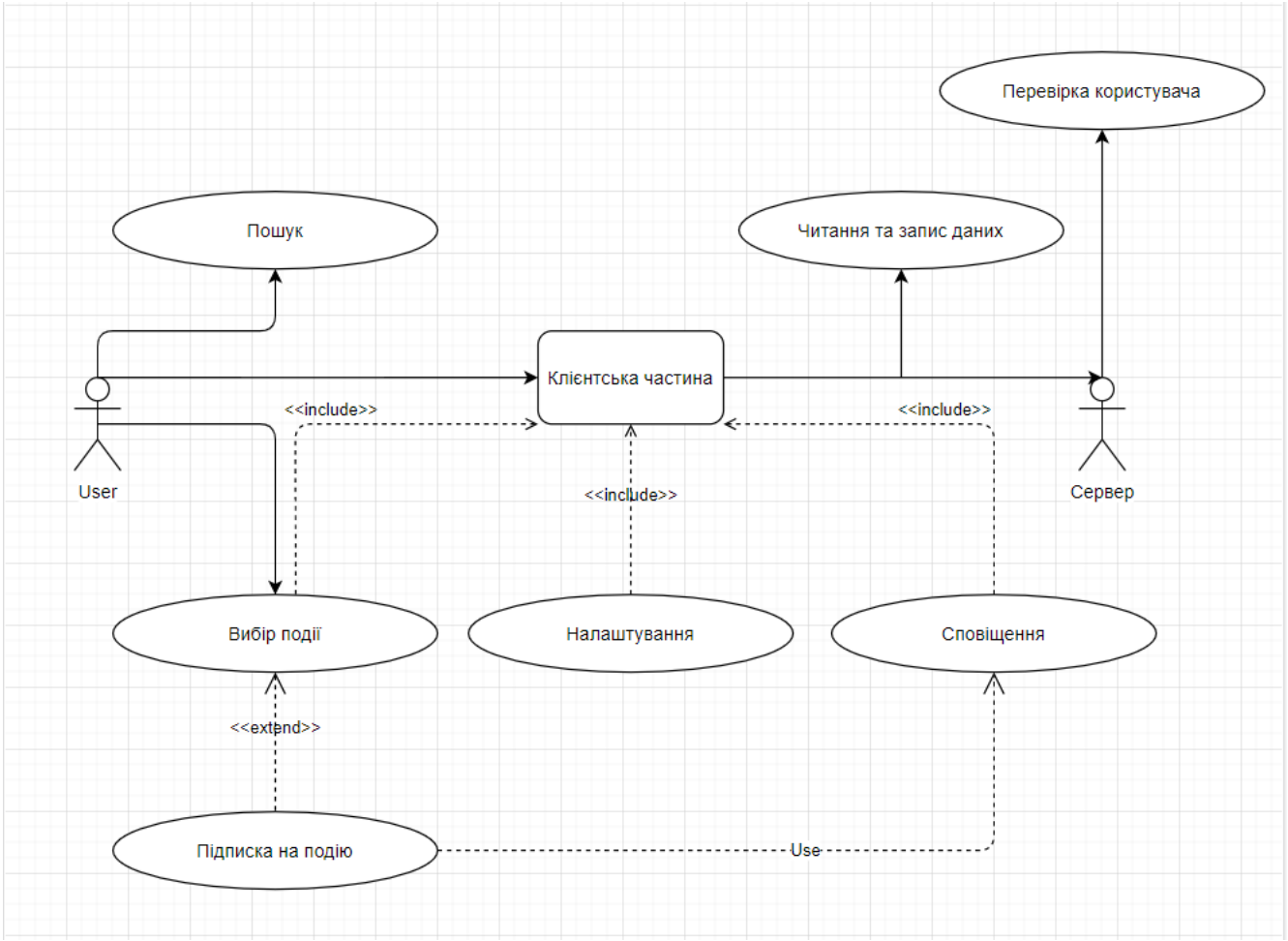


Рисунок 6. Діаграма прецедентів

4.2. Таблиці БД і їх зв`язки

В додатку А представлене схематичне зображення таблиць бази даних. Основною одиницею даних в системі є «подія», що має поля, необхідні для заповнення, а також час і координати. Час також необхідний для встановлення нагадування при підписці у клієнтській частині. При фільтрації актуальних подій, до уваги беруться координати: якщо різниця між координатами користувача і події більша певного критичного значення, подія вважається неактуальною. Запит до БД, який враховуватиме цю різницю, виглядає так:

```
SELECT * FROM events
WHERE longtitude BETWEEN :x_min AND :x_max
AND latitude BETWEEN :y_min AND :y_max
```

Для отримання крайніх значень x та y на сервері, беруться до уваги координати користувача: xk, yk, а також максимально допустиме відхилення Δ:

$x\_min = xk - \Delta$
$x\_max = xk + \Delta$
$y\_min = yk - \Delta$
$y\_max = yk + \Delta$

Для отримання підписок, що зберігаються в окремій таблиці, нам знадобиться звязок по первинному ключу id між таблицями event та event_participant. При цьому нас цікавлять підписки тільки цього користувача:

```
SELECT * FROM event_participant WHERE event_id = :event AND user_id = :user
```

4.3. Схеми навігації по екранам додатку

Навігація між екранами відбувається починаючи з головного екрану і закінчуючи екраном додавання події (рисунок 7).

Також було створено прототип авторизації, що зобов’язує користувача зареєструватися або увійти в систему. Дані, введені в формах, перевіряються на сервері; у разі введення невірного паролю, видається відповідна помилка.

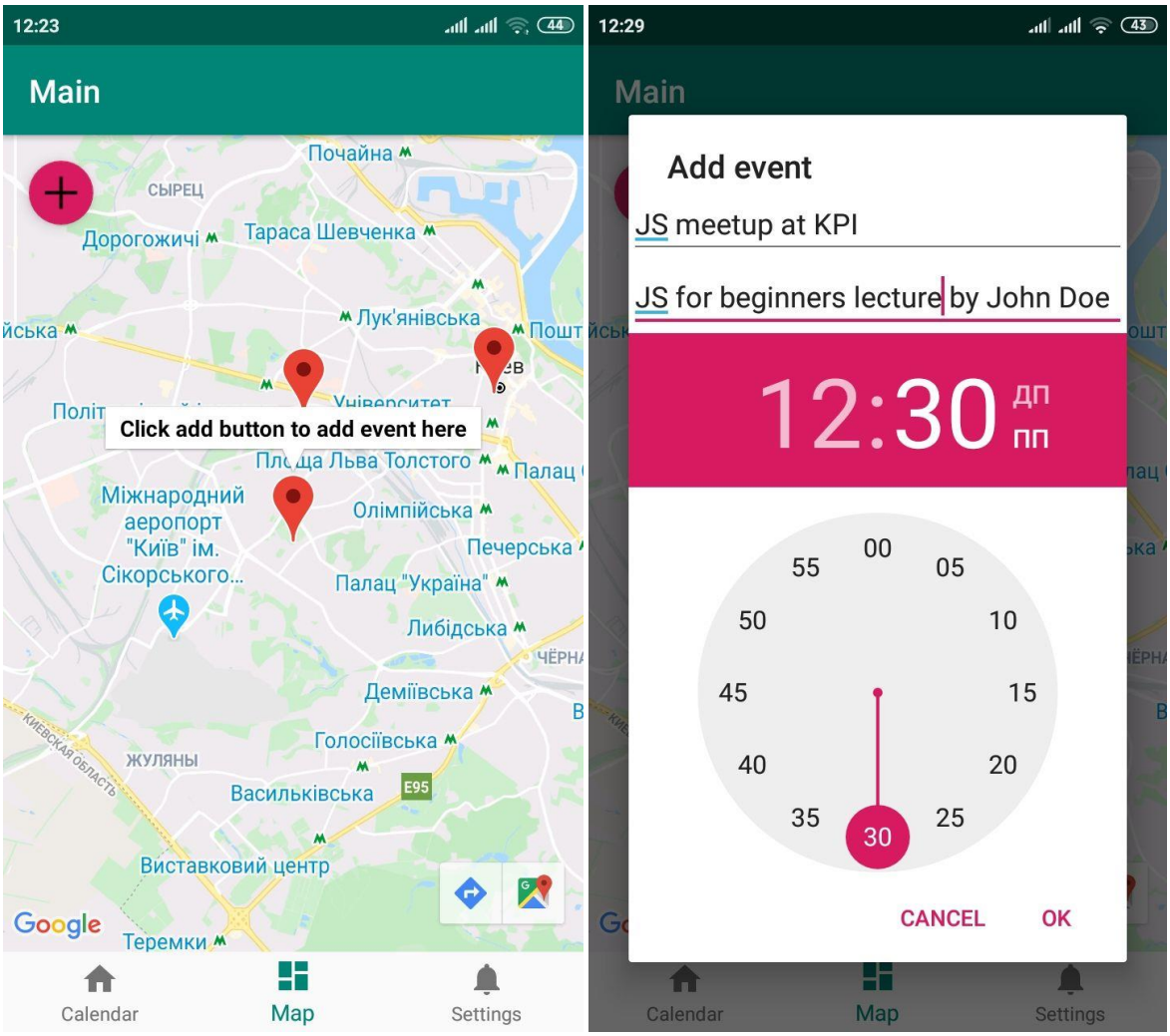


Рисунок 7. Екран перегляду подій та екран додавання події

4.5 Інсталяція, вимоги до ПЗ, робота з інтерфейсом

Архітектура «клієнт-сервер» зазначає базисні концепції організації взаємодії в мережі, яку складають сервери – вузли-постачальники сервісів і клієнти – споживачі цих функцій.

Мінімальні вимоги до програмного забезпечення сервера:

Операційна система Windows або Ubuntu;
Встановлені Apache Web Server з PHP (наприклад, у складі XAMPP);
Значний об`єм вільної пам`яті (кількість необхідної пам`яті напряму залежить від кількості записів у базі даних).

Мінімальні вимоги до апаратного забезпечення клієнта:

смартфон, розмір екрану не менше 3,7 дюймів;
доступ до місцезнаходження пристрою (Wi-Fi, GPS);
об’єм вільної пам’яті 20Мб.

Мінімальні вимоги до програмного забезпечення:

операційна система Android ;
рівень API 21 (Android 5.0);
Google Play Services 12.0.

Інсталяція в тестовому режимі може проводитися через USB, однак в разі пблікації додатку та придбання необхідних дозволів стане можливою бездротова інсталяція через Google Play.

5. ВИСНОВКИ

Задачею для даної роботи було поставлено побудову системи керування подіями, що базувалася б на розташування користувача та дозволяла здійснювати пошук актуальних заходів. Для реалізації даної системи було обґрунтовано вибір засобів розробки, розроблено мобільний доаток для керування подіями і взаємодії користувача із сервером, розроблено скрипти для серверної частини і таблиці для бази даних. Також було проведено тестування та оцінку вихідного програмного продукту.

Для реалізації даної системи були сформульовані та вирішені наступні проблеми:

- ✓ Проблема встановлення місцезнаходження користувача: були створені відповідні функції, що запитують дозвіл на отримання координат у системи та збереження інформації про останнє місцезнаходження
- ✓ Проблема обміну інформацією клієнта з сервером, а також редагування сервером інформації в базі даних: були написані скрипти для API, що перевіряють вхідні дані, роблять необхідні запити до БД і повертають оброблену інформацію у вигляді тексту
- ✓ Проблема постійного оновлення і відображення отриманих даних на інтерактивній карті: було створено функції, що з певним інтервалом роблять запити і отримують інформацію про поточні події; коли дані отримані, обробник оновлює карту і встановлює на ній нові події

Для виконання поставлених цілей було обрано і обгрунтовано відповідні технології для клієнтської та серверної частини, а також для бази даних.

Було проведено передпроектне обстеження предметної області програмних продуктів, що існують на ринку. Після цього було встановлено їх переваги та недоліки, а також було проведене порівняння з функціоналом даної системи. Виходячи з цього, було прийнято рішення розробити та впровадити додаток з використання серверних технологій.

Було обрано клієнт-серверну архітектуру, а саме: поділ проекту на дві функціональні частини – клієнтська частина, під якою виступає програмний додаток для ОС Android на мові Java та серверна частина, яка базується на мові PHP із використанням бази даних MySQL. Було проаналізовано інструменти проектування, розробки програмного продукту, зберігання та обробки даних.

Архітектуру програмного продукту спроектовано у спеціалізованому середовищі Draw.io. Функціональні модулі програмного продукту розроблено у IDE Android Studio.

Проведене у два етапи тестування (розробником та кінцевим користувачем) показало відсутність критичних дефектів функціонування, продукт функціонує коректно.

Оцінка отриманого продукту за допомогою опитувань кінцевих користувачів підтвердила, що усі функціональні вимоги та поставлені задачі до розробки платформи було виконано.

В результаті опитування користувачів та аналізу системи було встановлено наступні напрямки для її вдосконалення:

- Впровадження категорій подій (наукові лекції, культурні заходи, спортивні збори і т.д.) та можливість шукати події за категоріями
- Впровадження критерію пошуку за словом або фразою, що буде суміщатися з існуючими критеріями
- Надання користувачу можливості самому обирати координати і радіус, за якими будуть шукатися підходящі заходи

Також слід зазначити основні проблеми з поточною реалізацією системи і способи їх вирішення:

- Проблема отримання даних з серверу у режимі реального часу, а не з інтервалами. Вирішується за допомогою переходу від звичайних HTTP-запитів до асинхронних і симетричних запитів за допомогою WebSockets
- Проблема зберігання і передачі медіа (зображень і відео події), а не тільки тексту. Вирішується отриманням серверу без великих обмежень на трафік і впровадженням ряду функцій для обробки медіа в коді
- Проблема взаємодії користувачів вцілому і підписників конкретного заходу: можливість відправляти приватні повідомлення, коментувати захід, відповідати на коментарі. Щоб вирішити цю проблему, необхідно виділити найбільш корисні функції для комунікації, спланувати архітектуру коду і структуру таблиць бази даних, потім додати відповідні функції для клієнта і сервера

Додаток може зайняти свою нішу у сфері організації подій для широкого кола користувачів, та зокрема для діячів науково-освітньої сфери.

6. ПЕРЕЛІК ПОСИЛАНЬ

1 Архитектура Клиент — сервер [Електронний ресурс] — Режим доступу: https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80

2 Configure your build [Електронний ресурс] — Режим доступу: https://developer.android.com/studio/build

3 Google Maps Platform Documentation [Електронний ресурс] — Режим доступу: https://developers.google.com/maps/documentation

4 Постоянные соединения с базами данных [Електронний ресурс] — Режим доступу: https://www.php.net/manual/ru/features.persistent-connections.php

5 Creating Tables [Електронний ресурс] — Режим доступу: https://dev.mysql.com/doc/internals/en/creating-tables.html

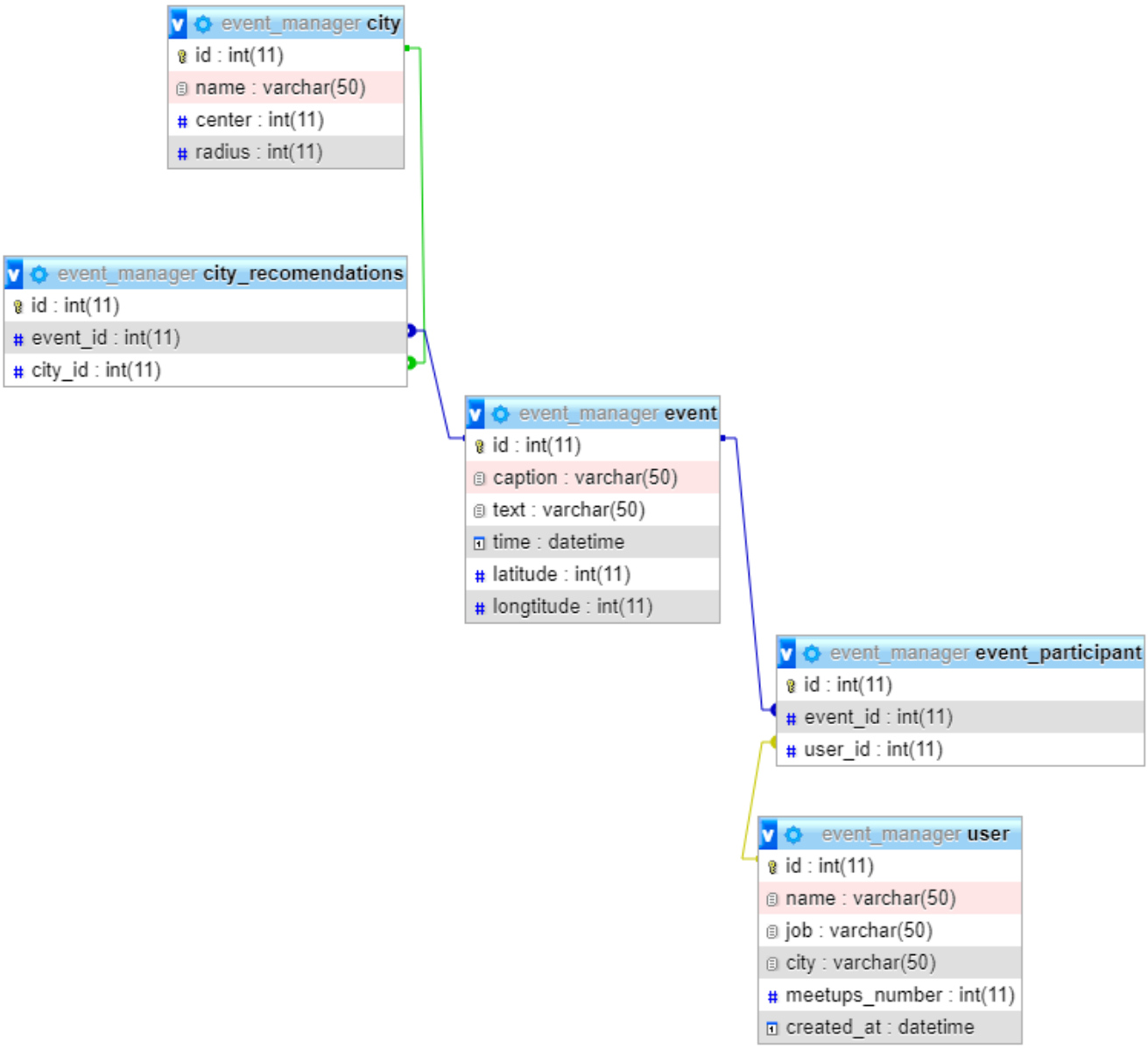
6 Benefits of using Google Places [Електронний ресурс] — Режим доступу: https://support.google.com/google-ads/answer/143059

7 Meetup search [Електронний ресурс] — Режим доступу: https://www.meetup.com/ru-RU/find/events/

8 Нормальная форма [Електронний ресурс] — Режим доступу: https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0

Додаток А

Структура бази даних системи керування подіями



Додаток Б

Клас «подія», що використовується в клієнтській частині і допоміжний клас «планувальник»

```
package com.example.scheduler.event;


import android.content.Context;

import com.example.scheduler.notifications.Scheduler;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Locale;


public class MapEvent implements Serializable {


    private int id;

    private String author;

    private String caption;

    private String text;
```

```
    private long time;
    private float x; // Lat [-90, 90]
    private float y; // Lng [-180, 180]
    private byte notify; // 0 => no, 1 => mute, 2 => 0.5, 3 => 5


    public static int NOTIFY_MAX = 3;


    public MapEvent(int id, String author, String caption, String text, long time, double x, double y, int notify){
        this.id = id;
        this.author = author;
        this.caption = caption;
        this.text = text;
        this.time = time;
        this.x = (float)x;
        this.y = (float)y;
        this.notify = (byte)notify;
    }
```

```
    public static MapEvent[] fromText(String eventsText) {
        ArrayList<MapEvent> events = new ArrayList<>();
        for (String eventText : eventsText.split("¶")) {
            String[] p = eventText.split("□");
            try {
                events.add(new MapEvent(Integer.parseInt(p[0]), p[1], p[2], p[3], Long.parseLong(p[4]), Double.parseDouble(p[5]), Double.parseDouble(p[6]), Byte.parseByte(p[7])));
            } catch(Exception ignored) {}
        }
        return events.toArray(new MapEvent[0]);
    }
    public String toText() {
        return String.format(Locale.US, "%d□%s□%s□%s□%s□%f□%f□%d", id, author, caption, text, time, x, y, notify);
    }

    public int getId() {
```

```
        return this.id;
    }
    public String getAuthor() {
        return author;
    }
    public String getCaption() {
        return caption;
    }
    public String getText() {
        return text;
    }
    public String getTime() {
        Calendar c = Calendar.getInstance();
        c.setTimeInMillis(time);
        return String.format(Locale.US, "%02d.%02d.%d %02d:%02d",
c.get(Calendar.DAY_OF_MONTH), c.get(Calendar.MONTH), c.get(Calendar.YEAR),
c.get(Calendar.HOUR_OF_DAY), c.get(Calendar.MINUTE));
    }
    public double getX() { return x; }
```

```
public double getY() { return y; }

public byte getNotify() {

    return notify;

}



public void setAuthor(String author) {

    this.author = author;

}

public void setCaption(String caption) {

    this.caption = caption;

}

public void setText(String text) {

    this.text = text;

}

public void setTime(int year, int month, int day, int hour, int minute) {

    Calendar c = Calendar.getInstance();

    c.set(Calendar.YEAR, year);

    c.set(Calendar.MONTH, month);
```

```
        c.set(Calendar.DAY_OF_MONTH, day);
        c.set(Calendar.HOUR, hour);
        c.set(Calendar.MINUTE, minute);
        this.time = c.getTimeInMillis();
    }
    public void setNotify(int notify) {
        this.notify = (byte)notify;
    }

    public void updateNotification(Context context) {
        if(notify != 0) {
            Scheduler.schedule(context, (int)(time / 1000), author + ":" + caption, text, notify,
null, time);
        }
        else {
            Scheduler.cancel(context, (int)(time / 1000));
        }
    }
```

```
}

package com.example.scheduler.notifications;


import android.app.AlarmManager;

import android.app.Notification;

import android.app.NotificationChannel;

import android.app.NotificationManager;

import android.app.PendingIntent;

import android.content.Context;

import android.content.Intent;

import android.graphics.Bitmap;

import android.net.Uri;

import android.os.Build;

import androidx.core.app.NotificationCompat;

import com.example.scheduler.R;


public class Scheduler {
```

```
    private static String getChannelId(Context context) {

        final String CHANNEL_ID = "schedulerChannel";


        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

            NotificationManager notificationManager = context.getSystemService(NotificationManager.class);


            if(notificationManager.getNotificationChannel(CHANNEL_ID) == null) {

                NotificationChannel channel = new NotificationChannel(CHANNEL_ID, "scheduler", NotificationManager.IMPORTANCE_HIGH);

                channel.setDescription("description");

                notificationManager.createNotificationChannel(channel);

            }

        }

        return CHANNEL_ID;

    }

    private static Notification buildNotification(Context context, String caption, String text, byte notify, Bitmap icon) {
```

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(context, getChannelId(context))
        .setSmallIcon(R.drawable.bell)
        .setContentTitle(caption)
        .setContentText(text)
        .setPriority(NotificationCompat.PRIORITY_MAX)
        .setVisibility(NotificationCompat.VISIBILITY_PRIVATE);


    switch (notify) {
        case 1:
            break;
        case 2:
            builder.setSound(Uri.parse("android.resource://" + context.getPackageName() + "/" + R.raw.notification));
            builder.setVibrate(new long[]{0, 240, 240, 1220, 1220}); // { delay, vibrate, sleep, vibrate, sleep }
            break;
        case 3:
```

```
            builder.setSound(Uri.parse("android.resource://" + context.getPackageName() + "/" + R.raw.long_alarm));
            builder.setVibrate(new long[]{
                0,
                401, 401, 553, 553, 401, 401, 935, 935,
                401, 401, 553, 553, 401, 401, 935, 935,
            });
            break;
        }

        if(icon != null) builder.setLargeIcon(icon);

        return builder.build();
    }

    public static void schedule(Context context, int id, String caption, String text, byte notify, Bitmap icon, long timestamp) {
        if(notify == 0) return;
```

```
        Intent intent = new Intent(context, NotificationReceiver.class);

        intent.putExtra(NotificationReceiver.ID, id);

        intent.putExtra(NotificationReceiver.NOTIFICATION, buildNotification(context, caption, text, notify, icon));

        PendingIntent pendingIntent = PendingIntent.getBroadcast(context, id, intent, PendingIntent.FLAG_UPDATE_CURRENT);


        AlarmManager alarmManager = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);


        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {

            alarmManager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP, timestamp, pendingIntent);

        }

        else {

            alarmManager.setExact(AlarmManager.RTC_WAKEUP, timestamp, pendingIntent);

        }

    }

    public static void cancel(Context context, int id) {
```

```
        Intent intent = new Intent(context, NotificationReceiver.class);

        PendingIntent pendingIntent = PendingIntent.getBroadcast(context, id, intent, 0);


        AlarmManager alarmManager = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);

        alarmManager.cancel(pendingIntent);

    }

}
```