МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Бакалаврська кваліфікаційна робота

на тему: Розробка мобільного додатку для пошуку роботи

Виконав студент 4 курсу групи К-25
спеціальність 122«Комп'ютерні науки»
Лаврека Данило Михайлович

Керівник к.геогр.н., доцент
Коваленко Людмила Борисівна

Консультант _____

Рецензент д.ф.-м. н., професор
Ковальчук Володимир Володимирович

Одеса 2020

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

ІС – інформаційна система

ОС – операційна система

ПЗ – програмне забезпечення

ПП – програмний продукт

ПС – програмна система

РСКБД – реляційна система керування базами даних

СКБД – система керування базами даних

IDE – Integrated Development Environment – інтегроване середовище розробки

ВСТУП

На даний час у молодих людей, зайнятих навчанням, все частіше виникає проблема пошуку підробітку, тобто короткострокової роботи на кілька днів. У зв'язку з цим актуальним завданням є створення доступного сервісу для пошуку одноразової роботи або працівника. Такого роду додаток особливо буде корисно у випадках, коли роботодавець має знайти працівника за короткий інтервал часу (1-2 години) і пошук за газетними оголошеннями не може забезпечити зазначені терміни.

Подібну функціональність може забезпечити використання мобільних технології, так як практично у кожної сучасної людини є смартфон, який в будь-який момент часу може оповістити користувача про зміну активності в застосунку.

Аналіз ринку мобільних застосунків показав, що для жителів міста Одеси відсутні програмні системи, що задовольняють необхідним вимогам: зручного пошуку, зрозумілого інтерфейсу, наявності карт. Тому розробка подібного програмного забезпечення є досить актуальним завданням.

Метою кваліфікаційної роботи є розробка мобільного застосунку для допомоги у пошуку короткострокової роботи.

Для досягнення поставленої мети були сформульовані наступні завдання:

- провести аналіз предметної області та аналіз вимог до програмної системи, що розроблюється;
- обґрунтувати вибір програмних засобів розробки та технологій;
- провести проектування мобільного застосунку з використанням мови UML;
- виконати реалізацію клієнтської та серверної частин застосунку;
- підготувати інструкцію користувача;
- виконати тестування програмної системи.

Структура дипломної роботи складається з вступу, трьох розділів, висновків, переліку посилань на 16 найменувань, додатків. Повний обсяг проекту становить 72 сторінки, містить 10 рисунків.

1 ВИЗНАЧЕННЯ ТА АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

1.1 Призначення програмної системи

Передбачається, що програмна система буде містити анкети по шукачів роботи, за допомогою яких замовник може знайти робітника. Також в системі буде зберігатися список доступних вакансій, за допомогою яких робітник може знайти роботу. Мобільний застосунок призначений для пошуку короткострокової роботи в межах району міста, наприклад, недалеко від свого будинку. Це потребує обов'язкової наявності вбудованої карти. Також програма повинна мати систему хештегів, за допомогою яких можна визначити тип роботи і спростити пошук.

1.2 Керування правами доступу

Формування ролей покликане визначити чіткі і зрозумілі для користувачів програмної системи правила розмежування доступу. Рольове розмежування доступу дозволяє реалізувати гнучкі та динамічно змінні в процесі функціонування програмної системи правила розмежування доступу.

Таке розмежування доступу є складовою багатьох сучасних комп'ютерних систем. Рольовий підхід часто використовується в системах, для користувачів яких чітко визначено коло їх посадових повноважень і обов'язків.

Так як привілеї не призначаються користувачам безпосередньо і отримуються ними тільки через свою роль (або ролі), управління індивідуальними правами користувача по суті зводиться до призначення йому ролей. Це спрощує такі операції, як додавання користувача або зміна підрозділу користувачем [1][1)].

В програмній системі, що розроблюється призначимо наступні ролі:

[1)] [1] David., Ferraiolo, Richard., Kuhn, D. Role-based access control (вид. 2nd ed). Boston: Artech House, 2007. p. 456.

– адміністратор – здійснює підтримку програмної системи;
– гість – може зареєструватися і переглядати список доступних робіт, та робітників;
– користувач – це зареєстрований гість;
– анкета робітника – профіль користувача, який пропонує свої послуги, як робітник.

1.3 Створення історій користувача

Історія користувача – це одне чи більше речень, звичайною мовою предметної області, які описують чого користувач хоче досягти. Історії користувача використовуються в гнучких методологіях для з'ясування базових функцій, що будуть реалізовуватись. Кожна історія користувача достатньо коротка. Історії користувача пишуться споживачами програмного забезпечення і є основним інструментом їх впливу на розробку програми. Вона висвітлює "Хто", "Що", "Чому" вимог у простий й точний спосіб.

Історії користувача – швидкий спосіб оперування вимогами користувача, без необхідності застосування занадто формалізованих документів, та виконання адміністративних задач пов'язаних з опрацюванням цих документів. Наміром з яким використовують історії користувача є швидше та менш накладне реагування на швидко змінювані вимоги реального світу.

Історії користувача – це неформальний опис вимог до тих пір, поки відсутні відповідні тести прийнятності. Перед тим як реалізовувати історію користувача відповідна процедура прийняття має бути написана користувачем, що тестуванням чи іншим чином визначає чи задоволені вимоги історії користувача. Деяка формалізація відбувається коли розробник приймає історію користувача, та відповідну процедуру прийнятності [2][1)].

Коли приходить час створення історій користувача, один з розробників зустрічається з представником замовника. Замовник відповідальний за фор-

[1)] [2] Розповідь користувача – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Розповідь_користувача (дата звернення 04.01.2020).

мулювання історій користувача. Розробник може використати серії питань, щоб допомогти споживачу, наприклад питати чи не потрібна йому певна функція, але має бути обережним щоб не керувати процесом створення ідей.

Підготуємо історію користувача для програмної системи, що розроблюється:

a) як гість я можу подивитися список робіт і анкет робочих біля свого будинку, щоб ознайомитися з роботою застосунку;
b) як гість я можу зареєструватися, щоб отримати призначений для користувача обліковий запис (для реєстрації потрібен перевірений e-mail, пароль і ім'я);
    1) тест 1: користувач не може ввести пароль менше 6 символів і більше 30;
    2) тест 2: ім'я користувача не може бути менше 2 і більше 30 символів;
    3) тест 3: e-mail користувача повинен бути унікальним в системі;
c) як гість я можу увійти в систему під раніше створеним обліковим записом для подальшої роботи (для входу використовується e-mail і пароль);
d) як користувач я можу доповнити свій профіль, щоб мати можливість виставити оголошення або анкету робітника;
    1) доповнити профіль користувач може: номером телефону (з підтвердженням), прізвищем, країною і містом проживання, датою народження;
    2) номер телефону потрібен щоб викласти оголошення про роботу та анкету робочого;
    3) користувач може управляти яка інформація буде видна іншим користувачам, а яка ні.

e) як користувач я можу змінити дані свого облікового запису, щоб не створювати новий профіль (для зміни пароля потрібно ввести старий пароль);

f) як користувач я можу викласти оголошення про роботу, щоб на неї відгукнувся інший користувач (щоб викласти роботу треба вказати інформацію: адреса (необов'язково), оплата, час роботи, щоб встановити час, хештеги, опис; інші дані беруться з профілю користувача – ім'я, номер телефону);

g) як користувач я можу викласти свою анкету робочого, щоб інший користувач міг запропонувати мені роботу (анкета робочого формується з профілю користувача, також користувач додає хештеги, адресу і опис);

h) як гість я можу відновити пароль, щоб відновити доступ до свого облікового запису (для відновлення пароля потрібен e-mail користувача або номер телефону);

i) як користувач я можу подивитися повний опис роботи і анкету користувача, який поставив її такою, щоб ознайомитися з роботою (повний опис роботи складається з: адреси, номера телефону користувача, який її виставив, оплати, час, коли цю роботу потрібно виконати і за скільки, хештеги, її опис);

j) як користувач я можу подивитися повну анкету робочого, щоб ознайомитися з ним (повна анкета робочого складається з: ім'я і прізвища, номера телефону, хештегів, опис);

k) як користувач я можу редагувати своє оголошення про роботу та анкету робочого, щоб не створювати нові;

l) як користувач я можу використовувати пошук з фільтрами, щоб знайти задовольняє мене роботу або робітника.

    1) пошук може виконуватися, як по карті, так списком;
    2) фільтри для пошуку оголошення про роботу: оплата, дата початку, хештеги;

3) фільтри для пошуку анкети робочого: хештеги.

m) як користувач я можу додавати роботи в обране, щоб не втратити їх;

n) як користувач я можу поскаржитися на іншого користувача, роботу або анкету робочого, щоб повідомити адміністратора про порушення порядку;

o) як адміністратор я можу видаляти замовлення користувача, щоб контент був легальним;

p) як адміністратор я можу перевіряти скарги користувачів, щоб стежити за легальністю контенту в застосунку;

q) як адміністратор я можу заблокувати користувача, щоб стежити за порядком в системі;

1) адміністратор може заблокувати користувача на час або на завжди;

2) адміністратор може зняти блокування в будь-який час, якщо вона була досконала помилково.

r) як користувач я можу видалити обліковий запис, якщо у мене немає в ній потреби (для видалення профілю потрібно ввести свій пароль).

2 ВИБІР ТА ОБГРУНТУВАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір архітектури програмної системи

Мікросервісна архітектура – принципова організація розподіленої системи на основі мікросервісів і їх взаємодії один з одним і з середовищем по мережі, а також принципів, що направляють проектування архітектури, її створення та еволюцію. Варіант сервіс-орієнтованої архітектури програмного забезпечення, орієнтований на взаємодію наскільки це можливо невеликих, слабо пов'язаних і легко змінюваних модулів – мікросервісів.

Мікросервіси – архітектурний стиль, за яким єдиний застосунок будується як сукупність невеличких сервісів, кожен з яких працює у своєму власному процесі і комунікує з рештою, використовуючи легковагові механізми, зазвичай HTTP. Ці сервіси будуються навколо бізнес-потреб і розгортаються незалежно з використанням зазвичай повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі по собі вони можуть бути написані з використанням різних мов і технологій зберігання даних [3][1)].

Мікросервісна архітектура добре надається для процесу безперервної поставки, на відміну від сервіс-орієнтовної архітектури, мікросервісна спрямована на створення одного застосунка, в той час як сервісно-орієнтована система являє собою множину застосунків, які взаємодіють між собою.

Основні властивості мікросервісної архітектури:

- високий рівень незалежності: незалежна розробка, незалежне розгортання;
- незалежне масштабування;
- невелика кодова база зменшує кількість конфліктів та дозволяє швидко залучувати нових розробників;
- простота заміни однієї реалізації сервісу іншою;

[1)] [3] Мікросервіси – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Мікросервіси (дата звернення 04.01.2020).

- простота додавання нового функціоналу в систему;
- ефективне використання ресурсів;
- еластичність: вихід з ладу одного сервісу зазвичай не призводить до виходу з ладу всієї системи;
- сервіси організовані відносно бізнес логіки яку вони виконують;
- кожен сервіс незалежно від інших може бути реалізований за допомогою будь-якої мови програмування, СУБД, та ін.;
- архітектурно побудовані за симетричним принципом (виробник-споживач).

До позитивних сторін мікросервісної архітектури слід віднести чіткий розподіл по модулях, висока доступність та відносна простота розгортання. Кожен сервіс піднімається самостійно, що робить процес розгортання і налагодження більш чистим.

Система, що розробляється, являє собою триланкову клієнт-серверну архітектуру, в ній присутній клієнт, сервер додатків (до якого підключено клієнтську програму) і сервер баз даних (з яким працює сервер додатків). Схема такої архітектури представлена на рис.2.1.

Рисунок 2.1 – Схема роботи три ланкової архітектури

2.2 Вибір технологій та засобів розробки серверної частини

2.2.1 Вибір мови програмування та середовища розробки

Для серверної частини було обрано мову програмування Java та середовище розробки Intellij IDEA.

Java – об'єктно-орієнтована мова програмування, що була розроблена компанією Sun Microsystems в 1991 році і офіційно випущена 23 травня 1995 року. Відмінною особливістю Java в порівнянні з іншими мовами програмування загального призначення є забезпечення високої продуктивності програмування та ефективність використання пам'яті [4][1)].

В Java використовуються практично ідентичні угоди для оголошення змінних, передачі параметрів, операторів і для управління потоком виконанням коду. В Java додані всі хороші риси C ++. Мова Java вже багато років є сандартом у розробці корпотивних систем та має величезну кодову базу та багато готових рішень.

Java надає програмісту багатий набір класів об'єктів для ясного абстрагування багатьох системних функцій, використовуваних при роботі з вікнами, мережею і для введення-виведення. Ключова риса цих класів полягає в тому, що вони забезпечують створення незалежних від використовуваної платформи абстракцій для широкого спектра системних інтерфейсів.

Величезна перевага Java полягає в тому, що на цій мові можна створювати застосунки, здатні працювати на різних платформах.

Середовище розробки Intellij IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala) від компанії JetBrains. Дана IDE дає можливість роботи з системами контролю версій (Git), різноманітними базами даних, а також підключати різноманітні плагіни

[1)] [4] Герберт Шилдт. Java. Полное руководство (Java SE 7, 8-е издание). М.: Изд. Дом «Вильямс», 2012. 1104с.

для комфортної розробки ПЗ. Також за допомогою Intellij IDEA можна проводити тестування готового продукту [5][1)].

Програма містить повний набір необхідних для створення повноцінних додатків компонент: редактор, середа компіляції і виконання, а також відладчик.

Природно, IntelliJ IDEA – не єдина середовище створення додатків для Java, можна також використовувати Eclipse або NetBeans, але IntelliJ IDEA все ж дуже популярна серед розробників. Програму можна розглядати як інтелектуальне середовище розробки, яка розуміє код. У процесі його написання програмістом вона займається побудовою синтаксичного дерева, визначенням особливостей розміщених посилань, аналізом можливих шляхів виконання операторів і передачі даних. Ґрунтуючись на отриманих результатах, програма звертає увагу фахівця на існуючі помилки і самостійно усуває їх, надає варіанти автоматичного доповнення коду. Завдяки зазначеним особливостям вона позбавляє користувача від повсякденної рутини і дозволяє йому сконцентруватися на більш важливих завданнях.

Дана програма допомагає фахівцеві економити час внаслідок глибокого аналізу контексту і видалення невідповідних варіантів. Ця та інші деталі забезпечують підвищення рівня продуктивності користувача, одночасно дозволяючи йому отримувати більше задоволення від діяльності. Враховуючи все вище сказане та суб'єктивний досвід будемо використовувати в роботі саме IDE Intellij IDEA.

2.2.2 Опис технології Spring Framework

Розробка серверної частини велася за допомогою ряду підпроектів Spring Framework.

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

[1)] [5] Intellij IDEA – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/IntelliJ_IDEA (дата звернення 04.01.2020)

Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB) [6][1)].

Spring Framework складається з кількох модулів, які надають широкий спектр послуг [7][2)]:

- контейнер інверсії управління: конфігурація компонентів додатків і управління життєвим циклом об'єктів Java, здійснюється головним чином через інверсію управління;
- аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні процедури;
- доступ до даних: робота з реляційною системою управління базами даних на платформі Java з використанням JDBC і об'єктно-реляційні відображення та інструментів з NoSQL баз даних;
- управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів;
- модель-вигляд-управління (Model-View-Controller): програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful;
- аутентифікація і авторизація: налаштовувані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою підпроекту Spring Security (колишня система безпеки AcerI для Spring).

[1)] [6] Spring Framework – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Spring_Framework (дата звернення 04.01.2020).

[2)] [7] Кларенс Хо, Роб Харроп. Spring 3 для профессионалов. М. : «Вильямс», 2012. 880 с.

- віддалене керування: конфігураційний вплив і управління Java-об'єктами для місцевої (локальної) або віддаленої конфігурації через JMX;
- тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів.

Для швидкого, якісного та зручного написання бізнес логіки проекту використовувався Spring Boot. Він дозволяє забути про довготривалі налаштування та конфігурації сервера додатку та бази даних [8][1)]

Spring Security використовувався для реалізації серверної безпеки, а також реєстрації та авторизації.

Spring Security – це Java/Java EE фреймворк, що надає механізми побудови систем аутентифікації та авторизації, а також інші можливості забезпечення безпеки для промислових додатків, створених за допомогою Spring Framework. Проект був розпочатий Беном Алексом (Ben Alex) в кінці 2003 року під ім'ям «Acegi Security» і був публічно представлений під ліцензією Apache License в березні 2004. Згодом був включений в Spring як офіційний дочірній проект.

2.2.3 Опис хмарної платформи Heroku

В розробці були використані хмарні технології: сервер і база даних розгорнуті на сервісі Heroku. Це було зроблено для того щоб хост не перебував на локальному комп'ютері. Даний сервер є безкоштовним для використання. Також Heroku полегшує управління обраної бази даних PostgreSQL.

Heroku - це хмарна платформа, заснована на керованій контейнерної системі, з інтегрованими службами передачі даних і потужної екосистемою для розгортання і запуску сучасних додатків. Розробка з Heroku – це орієнтований на додатки підхід до доставки програмного забезпечення, інтегрова-

[1)] [8] Крейг Уоллс. Spring в действии. М.: «Manning», 2014. 624 с.

ний з найпопулярнішими інструментами і робочими процесами для сьогоднішнього дня [9][1)].

Heroku - це класичний PaaS хостинг застосунків. На відміну від звичайних vps хостингів – він надає не прямий доступ до віртуальної машини з рутовий шеллом, а тулку для публікації застосунка в передналаштованому середовищі. Тобто прозоро вже налаштований якийсь веб-сервер, балансувальник, від користувача вимагається вказати тип і версію середовища (node.js, python, тощо) та залити свій застосунок через git. Один web dyno (так називається екземпляр застосунку) надається безкоштовно. Переваги PaaS в тому, що низкорівневою конфігурацією займається провайдер – користувач займаєтеся тільки застосунком. На heroku є безліч так званих аддонів – memcached, mongodb, mysql, postgres, rabbitmq, sphinx і інші засобів розробника.

Heroku запускає застосунки всередині діносів – смарт-контейнерів в надійному, повністю керованому середовищі виконання. Розробники розгортають свій код, написаний в Node, Ruby, Java, PHP, Python, Go, Scala або Clojure, в систему збирання, яка створює застосунок, готовий до виконання. Системні і мовні стеки контролюються, виправляються і оновлюються, тому вони завжди готові і оновлені. Середовище виконання підтримує запуск додатків без ручного втручання.

Робота з Heroku – це орієнтований на застосунки підхід до постачання програмного забезпечення, тому розробники можуть зосередитися на безперервному створенні і наданні застосунків, не відволікаючись на сервери або інфраструктуру. Розробники запускають додатки безпосередньо з популярних інструментів, таких як Git, GitHub або системи безперервної інтеграції (CI). Інтуїтивно зрозуміла веб-панель Heroku Dashboard спрощує управління додатком і підвищує видимість продуктивності.

Heroku Elements дозволяє розробникам розширювати свої застосунки за допомогою надбудов, налаштовувати їх стек застосунків за допомогою

[1)] [9] Heroku – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Heroku (дата звернення 04.01.2020).

Buildpacks і запускати свої проекти за допомогою кнопок. Доповнення – це сторонні хмарні сервіси, які розробники можуть використовувати для негайного розширення своїх застосунків за допомогою ряду функцій, таких як сховища даних, протоколювання, моніторинг та багато іншого. Heroku надає три повністю керованих доповнення служби даних: Heroku Postgres, Heroku Redis і Apache Kafka на Heroku.

Управління Heroku є ключовим компонентом платформи. Це допомагає розробникам усувати неполадки і загальні проблеми і "навчати" застосунки швидко виявляти й усувати негативні тенденції в їх працездатності. Heroku надає набір інструментів, який попереджає, якщо щось піде не так, або автоматично масштабує веб-діноси, якщо час відгуку для веб-запитів перевищує заданий розробником поріг. Метрики застосунків, порогове оповіщення і автомасштабування – це лише деякі з функцій, до яких розробник отримує доступ, без додаткових витрат.

Розробники з усього світу довіряють конфіденційні дані Heroku, і немає нічого важливішого, ніж виконання зобов'язань по зберіганню даних. Heroku регулярно проводить аудити і підтримує відповідність PCI, HIPAA, ISO і SOC, щоб ще більше зміцнити довіру клієнтів.

Розробники застосунків покладаються на програмні абстракції, щоб спростити розробку і підвищити продуктивність. Коли справа доходить до запуску застосунків, контейнеризація абстрагує навантаження на управління обладнанням або віртуальними машинами. Замість апаратного управління застосунки розгортаються в Heroku, яке упаковує код програми та його залежності в контейнери – в легені ізольовані середовища, які забезпечують обчислення, пам'ять, ОС і ефемерну файлову систему. Контейнери зазвичай запускаються на загальному хості, але повністю ізольовані один від одного.

Платформа Heroku використовує модель контейнера для запуску і масштабування всіх дзастосунків Heroku. Контейнери, які використовуються в Heroku, називаються «Дінос». Дінос – це ізольовані віртуальні контейнери Linux, призначені для виконання коду на основі заданої користувачем коман-

ди. Застосунок може масштабироваться до будь-якої кількості дінос на основі вимог до ресурсів. Можливості управління контейнерами Heroku надають простий спосіб масштабування і управління кількістю, розміром і типом дінос, які можуть знадобитися за стосунку розробника в будь-який момент часу. Дінос – це будівельні блоки, які керують будь-яким застосунком Heroku, від простого до складного. Розгортання на дінос з опорою на динамічне управління Heroku дозволяє створювати і запускати гнучкі масштабовані застосунки, звільняючи від управління інфраструктурою, тому є можливість зосередитися на створенні та запуску застосунків.

Для масштабування, необхідно зрозуміти взаємозв'язок між типами процесів і дінос. Тип процесу – це прототип, з якого створюється один або кілька дінос, за аналогією з об'єктно-орієнтованим програмуванням, в якому клас є прототипом, з якого створюються один чи кілька об'єктів. На рис.2.1 показано взаємозв'язок між дінос (по вертикальній осі) і типами процесів (по горизонтальній осі).

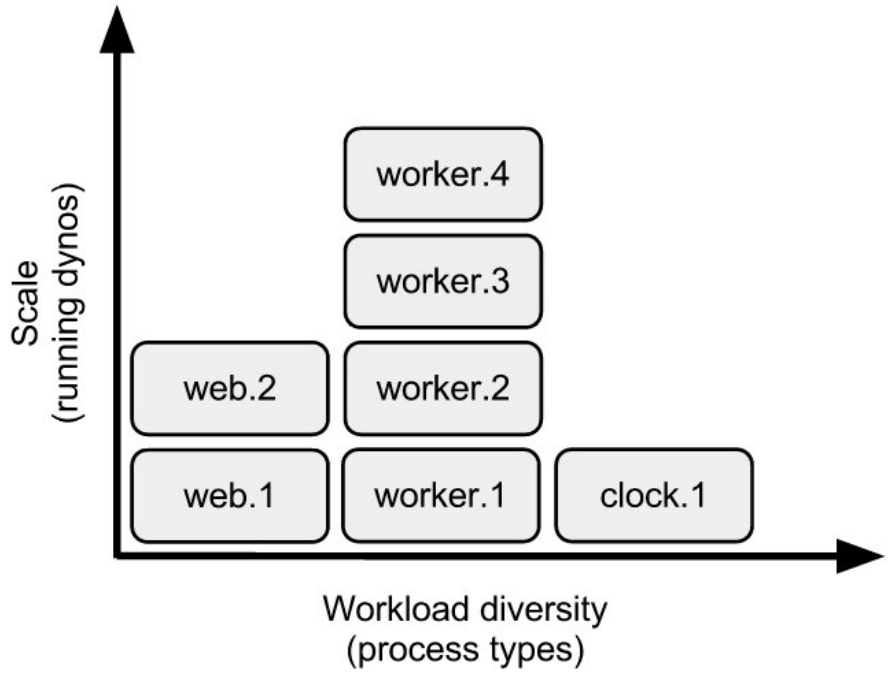


Рисунок 2.1 – Взаємозв’язок між доносами та типами процесів

Дінос, по вертикальній осі, це масштаб. Масштабування в цьому напрямку відбувається, коли необхідно збільшити паралелізм для типу роботи, оброблюваної цим типом процесу. На Heroku це робиться за допомогою команди масштабу:

```
$ Heroku ps: scale web = 2 worker = 4 clock = 1
Scaling web processes ... done, now running 2
Scaling worker processes ... done, now running 4
Scaling clock processes ... done, now running 1
```

Типи процесів на горизонтальній осі – це рознесення робочого навантаження. Кожен тип процесу спеціалізується на певному типі роботи. Наприклад, в деяких застосунках є два типи виконавців: один для термінових завдань, інший – для тривалих завдань. Поділяючи більш спеціалізованих виконавців, можна краще реагувати на свої термінові завдання і більш детально контролювати, як витрачати обчислювальні ресурси. Система розподілу черг може використовуватися для розподілу завдань робочим діносам.

Нижче перелічені продукти Heroku.

Heroku Postgres – хмарна база даних (DBaaS) від Heroku заснована на PostgreSQL. Heroku Postgres надає тривалий захист, відновлення даних, високу доступність.

Heroku Redis – модифікований Redis від Heroku, що надає найкращі умови розробки, він повністю управляється і надається як сервіс Heroku.

Heroku Teams – пристрій тімменеджменту, що надає кошти для об'єднання розробників, процесів і пристроїв, для розробки більш якісного ПЗ. За допомогою Heroku Teams, команди можуть самоорганізовуватися, додавати учасників і керувати ними.

Heroku Enterprise є сервісом для великих організацій, що допомагають їм поліпшити взаємодію між різними командами.

Heroku Connect дозволяє застосункам Heroku взаємодіяти з розробками Salesforce. Це робиться за допомогою безшовної синхронізації даних між базами даних Heroku Postgres і Salesforce.

Heroku Elements надає сервіси для розробки, розширення і оперування додатком; сервіси для автоматизації побудови програми на улюбленому мовою і фреймворку та інше.

Застосунки, для яких потрібне постійне сховище, мають доступ до Heroku Postgres. Бази даних Postgres в Heroku доступні з будь-якого клієнта Postgres, будь то в хмарі або на локальному пристрої. Heroku також надає зручну утиліту з'єднання для створення конкретних команд підключення до базі даних користувача. Пара особливо корисних функцій – це команди fork і follow. Команда fork клонує базу даних і дозволяє розробникам тестувати зміни в коді БД, не ризикуючи внести небажані зміни в дані, які використовуються іншими. Follow більш корисна у виробництві, оскільки вона створює репліки тільки для читання БД. Це особливо корисно для аналізу даних і виконання великих пакетних запитів без погіршення продуктивності частин обробки транзакцій в застосунку. У середовищі Postgres є журнали повтору, які містять інформацію про всі транзакції. Журнали повтору копіюються в кілька центрів обробки даних, так що в разі збою обладнання стан бази даних може бути відновлено.

2.3 Вибір технологій та засобів розробки клієнтської частини

2.3.1 Опис операційної системи Android

Існує кілька найпоширеніших операційних систем для смартфонів, такі як, IOS, Android, WindowsPhone, BlackBerry, Simbian.

Для створення клієнтської частини мобільного додатку була обрана операційна система Android, тому що:

– Android – операційна система з відкритим вихідним кодом;
– поширеність ОС Android (рис. 2.2);
– доступ до розробки будь-якому користувачеві;
– абсолютно безкоштовна для розробки.

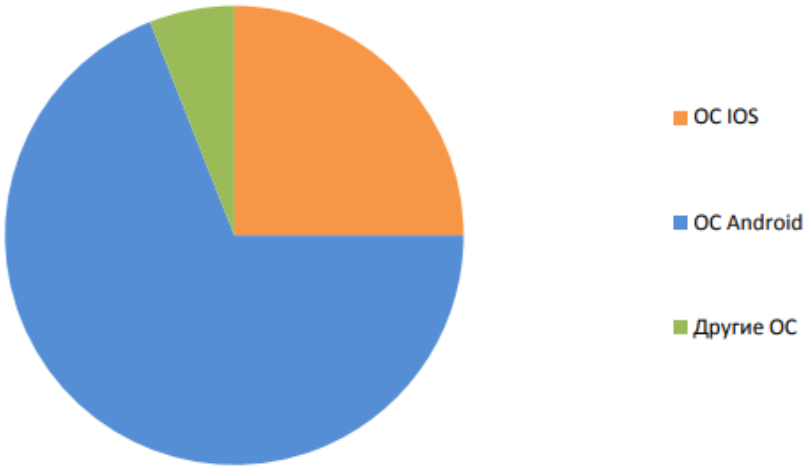


Рисунок 2.2 – Частка пристроїв на різних ОС

Android – операційна система для смартфонів, планшетних комп'ютерів, електронних книг, цифрових програвачів, "розумних" наручних годинників, ігрових приставок, нетбуків, смартбуків, окулярів Google, телевізорів, систем автоматичного керування автомобілем та інших пристроїв. ОС заснована на ядрі Linux і власної реалізації віртуальної машини Java від Google. Спочатку розроблялася компанією AndroidInc., Яку в 2005 році купила Google. Згодом Google ініціювала створення альянсу OpenHandsetAlliance (OHA), який зараз займається підтримкою і подальшим розвитком платформи. Android дозволяє створювати Java-додатка, що керують пристроєм через розроблені Google бібліотеки. AndroidNativeDevelopmentKit дозволяє перенести (але не налагоджувати) бібліотеки і компоненти додатків, написані на Сі та інших мовах. ОС Android встановлена на 86% смартфонів [10][1)].

Android є найпоширенішою операційною системою (ОС) для мобільних пристроїв – телефонів і планшетів. Дана система має безліч характерних рис, які роблять її популярною і привабливою для великої кількості користувачів по всьому світу.

[1)] [10] Коматинени С. Android 4 для профессионалов. Создание приложений для планшетних компьютеров и смартфонов: Научно-популярное издание. Санкт-Петербург: ТОВ «И.Д. Вильямс », 2012. 880 с.

Операційна система Android є невимогливою і здатна працювати на різних конфігураціях. Саме тому більшість світових виробників оснащують свої пристрої даною ОС, оскільки інші програмні продукти призначені для окремих апаратів, що відповідають певній специфікації. Така гнучкість Android пов'язана з тим, що система побудована на ядрі Linux, що має відкритий програмний код, що дає необмежені можливості розробникам. Android може бути запущений на пристроях, що мають об'єм оперативної пам'яті менше 256 Мб. Найбільш нові версії системи вимагають 512 Мб оперативної пам'яті, що також є невеликим значенням для сучасних апаратів.

Система не вимагає наявності високопродуктивного процесора і може працювати на пристроях, оснащених ядром з частотою 600 МГц. Операційна система дає можливість установки додатків з офіційного репозиторію Google, який надає найбільшу в світі базу програм. Це пов'язано з тим, що кожен розробник може самостійно написати будь-яку програму для апарату і розмістити її в магазині. Можливість також реалізована завдяки відкритості операційної системи. Варто відзначити, що додатки на пристрої під управлінням Android можуть бути встановлені як безпосередньо з телефону або планшета, так і через комп'ютер шляхом завантаження файлу .apk і його подальшої установки на апараті [11,12][1)].

Відмінною особливістю Android є його інтегрованість з сервісами Google – Gmail, Hangouts, Voice Search тошо.

На Android офіційно реалізована підтримка Chrome, що дозволяє синхронізувати вкладки, що відкриваються в браузері на смартфоні з комп'ютерним браузером.

Наприклад, ви можете почати перегляд сторінок з вашого телефону і при бажанні продовжити вивчати інформацію, відкривши цю ж вкладку на комп'ютері, не вдаючись до допомоги повторного пошуку.

[1)] [11] Дон Гриффитс, Девид Гриффитс Программирование для Android. СПб.: Питер, 2018. 912 с.

[12] Билл Филлипс, К. Стюарт, Кристин Марсикано Android. Программирование для профессмоналов. СПб.: Питер, 2017. 688 с.

«Андроїд» має досить простий і інтуїтивно зрозумілий інтерфейс. Всі потрібні програми розміщуються одночасно на головному екрані і в меню апарату, яке викликається натисканням на центральну сенсорну клавішу або відповідну кнопку на екрані. Всі настройки розташовуються в секції «Налаштування», а кожна дія користувача пояснюється коментарями і підказками при першому запуску апарату.

Операційна система швидко реагує на натискання користувача і виробляє установку і скачування потрібних програм і файлів зі швидкістю, яка не програє іншим сучасним мобільним ОС.

2.3.2 Загальна схема роботи програми Android

Додатки для Android в своїй роботі використовує вікна (аналогічно Windows), проте в даній системі вищевказані вікна носять іншу назву – Activity. Як і в Windows, кожне вікно має свій життєвий цикл і свої особливості (рис. 2.3).

При створенні нового вікна викликається метод onCreate(), при розробці даний метод перевизначається і в ньому відбувається ініціалізація програми та його компонентів.

Далі викликаються методи onStart() і onResume(). Обидва методи викликаються перед відображенням вікна при його створенні, або відновленні (при перемиканні з іншої програми, при розгортанні згорнутого додатку тощо).

При згортанні викликаються методи onPause() і onStop().

При закритті програми і вікна викликається onDestory(), в даному методі можна зберегти призначені для користувача дані і параметри. Повний опис та послідовність виклику методів можна знайти на офіційному сайті [13][1)].

[1)] [13] Ян Клифтон Проектирование пользовательского интерфейса на Android.Санкт-Петербург: ДМК Прес, 2017. 452 с.

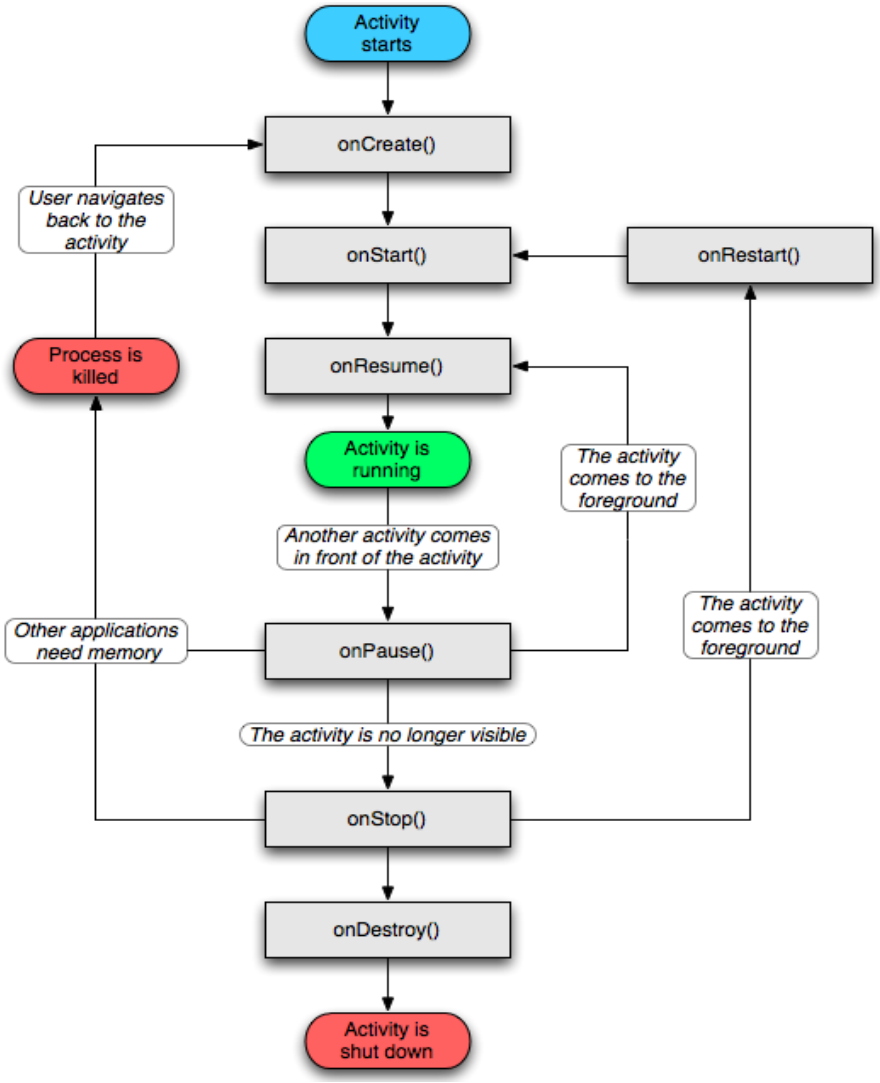


Рисунок 2.3 – Життєвий цикл додатки для системи під управлінням Android

2.3.3 Опис системи автоматичного збирання Gradle

В роботі була використана система автоматичного збирання Gradle, яка розвиває принципи, закладені в Apache Ant та Apache Maven і використовує предметно-орієнтовану мову (DSL) на основі мови Groovy замість традиційної XML-подібної форми представлення конфігурації проекту. Для визначення порядку виконання завдань Gradle використовує орієнтований ациклічний граф ("DAG") [14][1)].

[1)] [14] Gradle – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Gradle (дата звернення 04.01.2020)

На відміну від Apache Maven, заснованого на концепції життєвого циклу проекту, і Apache Ant, в якому порядок виконання задач (targets) визначається відносинами залежності (depends-on), Gradle використовує спрямований ациклічний граф для визначення порядку виконання завдань.

Gradle було розроблено для побудови мультипроектів, які можуть розростатися, і підтримує інкрементальне збирання. Вона визначає, які частини було змінено, і виконує тільки ті задачі, які залежать від цих частин.

Основні плагіни призначені для розробки і розгортання Java, Groovy і Scala додатків, але готуються плагіни і для інших мов програмування.

Розглянемо приклад створення альтернативної папки з ресурсами та підключення її за допомогою build.gradle до проєкту.

```
android {
    compileSdkVersion 20
    buildToolsVersion "20.0.0"

    defaultConfig {
        applicationId "ru.alexanderklimov.hellokitty"
        minSdkVersion 16
        targetSdkVersion 20
        versionCode 1
        versionName "1.0"
    }
    sourceSets {
        main {
            res {
                srcDirs = [
                    'src/main/res',
                    'src/main/presentations/animations',
                    'src/main/presentations/layouts']
            }
        }
    }
}
```

В цьому прикладі вказано, що існує нова папка presentations в папці /src/main/ разом з існуючими папками java и res. Всередині створеної папки є ще дві папки layout та animations, які містять файли ресурсів. Значення sourceSets вказує Gradle, які папки слід використовувати.

Номер версії додатка і вимоги до версії Android прописані в секції defaultConfig.

```
defaultConfig {
    minSdkVersion 8
    targetSdkVersion 19
    versionCode 1
    versionName "1.0"
}
```

Підключення бібліотеки відбувається одним рядком. Наприклад, потрібно додати бібліотеку Picasso:

```
dependencies {
    compile 'com.squareup.picasso: picasso: 2.3.2'
}
```

В Android Studio 3.0 використовується нова версія Gradle, в якій compile вважається застарілою. Замість нього слід використовувати нове слово implementation.

```
implementation 'com.android.support:recyclerview-v7:27.0.0'
```

Є схожа команда, яка підключає бібліотеку, яка буде використовуватися тільки для налагодження програми і в релізной версії вона не потрібна.

```
debugCompile 'junit: junit: 4.12' // старий варіант
testImplementation 'junit: junit: 4.12' // новий варіант для Android Studio 3.0
```

Далі треба включити синхронізацію і через кілька секунд в папці з'являється потрібна бібліотека, готова до використання. Сама бібліотека скачується з спеціального сховища-сховища JCenter. Даний репозиторій використовується за умовчанням і прописаний в buil.gradle проекту.

```
repositories {
    jcenter ()
}
```

Можна вказати інший репозиторій, наприклад, Maven Central.

```
repositories {
    mavenCentral ()
}
```

Також в файлі Gradle використовується інструмент ProGuard. Він може обрізати, оптимізувати і обфусціровать (вдіяти не читаним) код користувача. Для його включення потрібно виставити властивість minifyEnabled в значення true.

```
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile ( 'proguard-
android.txt'), 'proguard-rules.pro'
        }
}
```

У вказаному файлі потрібно прописати правила для роботи ProGuard.

2.3.4 Опис системи керування базами даних PostgreSQL

Для безпечного зберігання даних користувача використовувалась PostgreSQL.

PostgreSQL – об'єктно-реляційна система керування базами даних. Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite) [15][1)].

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

PostgreSQL – це сама просунута реляційна СКБД, що орієнтується в першу чергу на повну відповідність стандартам і розширюваність.

[1)] [15] Лузанов П., Рогов Е., Левшин И. PostgreSQL для начинающих. М., 2017. 147 с.

PostgreSQL, або Postgres, намагається повністю відповідати SQL-стандартам ANSI / ISO і відрізняється від інших СКБД тим, що володіє об'єктно-орієнтованим функціоналом, в тому числі повною підтримкою концепту ACID (Atomicity, Consistency, Isolation, Durability).

Будучи заснованим на потужної технології Postgres відмінно справляється з одночасною обробкою декількох завдань. Підтримка конкурентності реалізована з використанням MVCC (Multiversion Concurrency Control), що також забезпечує сумісність з ACID. Хоча ця СКБД не так популярна, як MySQL, існує багато сторонніх інструментів і бібліотек для полегшення роботи з PostgreSQL.

Сервер PostgreSQL написаний на мові C. Зазвичай розповсюджується у вигляді набору текстових файлів із сирцевим кодом. Для інсталяції необхідно відкомпілювати файли на своєму комп'ютері і скопіювати в деякий каталог.

Дана БД надає величезну кількість інструментів та можливостей для зберігання та обробки даних різних типів, від чисел до даних в форматах JSON та LOB. Для легких маніпуляцій з базою даних була використана технологія об’єктно-реляційного мапінгу. Найпопулярнішим представником, який неофіційно став стандартом, є ORM Hibernate.

До переваг PostgreSQL слід віднести:

– повну сумісність з SQL;
– PostgreSQL підтримується досвідченою спільнотою 24/7;
– підтримка сторонніми організаціями: незважаючи на дуже просунуті функції, PostgreSQL використовується в багатьох інструментах, пов'язаних з СКБД;
– можливість розширення: PostgreSQL можна програмно розширити за рахунок збережених процедур;
– об'єктно-орієнтованість: PostgreSQL – не тільки реляційна, а й об'єктно-орієнтована СКБД.

Недоліки PostgreSQL:

– продуктивність: в простих операціях читання PostgreSQL може поступатися своїм суперникам;
– популярність: через свою складність розробки інструмент не дуже популярний;
– хостинг: через перераховані вище фактори проблематично знайти підходящого провайдера.

СКБД PostgreSQL варто використовувати якщо:

– у пріоритеті надійність і цілісність даних
– БД повинна виконувати складні процедури;
– в майбутньому доведеться замінювати всю БД на інше рішення.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Створення діаграми прецедентів

UML (Unified Modeling Language) – мова графічного опису створення моделей. UML створювався для використання в процесі розробки програмного забезпечення. Головною його метою було досягнення єдиного бачення розробників і користувачів при створювані програми [16][1)].

Створення моделей дозволяє більш наочно документувати рішення. До реалізації ідей в коді зрозуміти і пояснити іншим учасникам проекту, як буде працювати програма. А користувачам надання моделей дозволяє зрозуміти, чи відповідає заявлена робота тому, що їм дійсно потрібно.

Створити модель можна в сотні і тисячі разів швидше, ніж створити реальний прототип програми. Модель набагато легше і швидше допрацювати і змінити, якщо обговорення покаже прийняті рішення не вірними. В результаті створення моделей скорочується необхідність переробок в програмах, що робить розробку дешевше і швидше. Використання моделей при створенні великих систем, дозволяє охопити систему одним поглядом і досягти кращого його розуміння всіма зацікавленими особами.

Таким чином, цілями аналізу і моделювання є:

- досягнення угоди між розробниками, замовниками і користувачами про те, що повинне робити ПП;
- досягнення кращого розуміння розробниками поведінки ПП;
- обмеження системної функціональності;
- створення базису для планування розробки проекту;
- визначення призначеного для користувача інтерфейсу.

Діаграма прецедентів чи варіантів використання (Use Case Diagram) визначає функціональне призначення модельованої системи або предметної об-

---

[1)] [16] Грейди Буч. Язык UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. СПб.: Питер, 2004. 432 с.

ласті. Дана діаграма відображає безліч акторів, що взаємодіють з проектованої системою (програмним засобом) за допомогою варіантів використання. Таким чином, основними елементами діаграми варіантів використання є актор і варіант використання.

Актор – це зовнішня по відношенню до модельованої системі сутність, що взаємодіє з системою для вирішення деяких завдань. Як актор може використовуватися людина, інша система, пристрій або програмний засіб.

Варіант використання визначає деякий набір дій (операцій), які повинні бути виконані моделюється системою або програмним засобом при взаємодії з актором.

Для елементів діаграми прецедентів задаються відповідні специфікації. Специфікації акторів: ім'я актора, кількість екземплярів, стереотип. Специфікації прецедентів: ім'я прецеденту, опис, стереотип.

Мова UML передбачає використання декількох типів зв'язків між елементами діаграми прецедентів, а саме:

- зв'язок комунікації, тобто зв'язок між актором і прецедентом, напрямок стрілки визначає елемент, що є ініціатором комунікації;
- зв'язок використання, зв'язок що показує як один прецедент завжди використовує інший;
- зв'язок розширення, зв'язок що показує як один прецедент періодично використовує інший;
- зв'язок узагальнення актора.

В роботі була створена діаграма прецедентів для застосунку, що розроблюється (рис. 3.1). На діаграмі зображено три актора: адміністратор, користувач та гість.

Актор гість може зареєструватись, переглядати список робіт та анкет робочих, відновити пароль та увійти в систему.

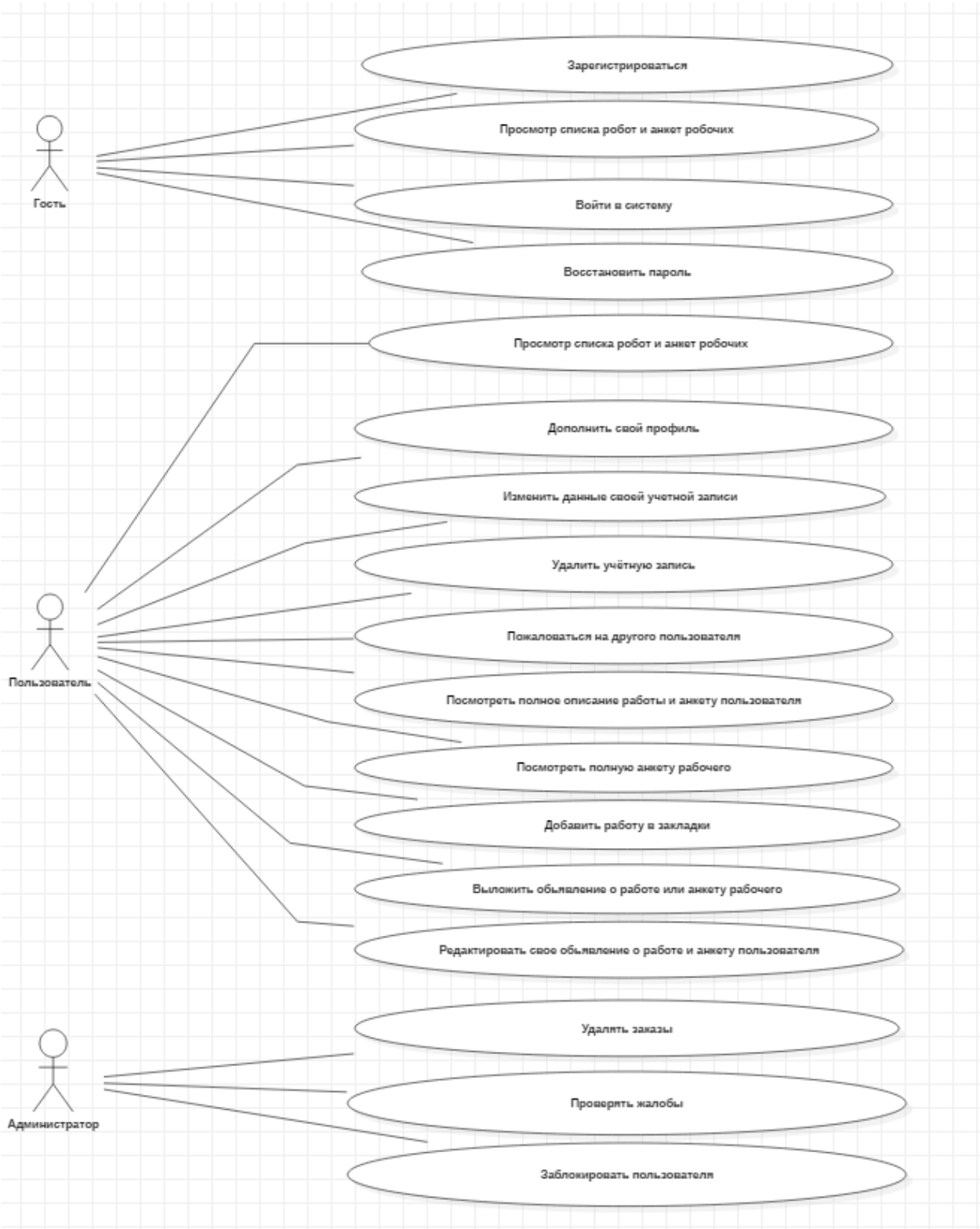


Рисунок 3.1 – Діаграма прецедентів програмної системи

Після входу в систему актор гість стає актором користувач. Актор користувач також може переглядати список робіт та анкет робітників, додавати їх в «Вибрані» та створювати їх, редагувати та видаляти. Користувач може

доповнити та редагувати свій профіль, подати скаргу на іншого користувача, якщо він порушив правила.

Актор адміністратор займається підтримкою системи. Він переглядає скарги, може видалити або заблокувати користувача за порушення правил.

3.2 Підключення до Google Maps API

До мобільного застосунку, що розроблюється, необхідно додати карти, які дозволять відстежувати користувачам місця розташування точок з пропозиціями короткострокової роботи.

На даний час існують комерційні і Open Source-ні постачальники геоданих. До першої групи можна віднести такі сервіси, як Google Maps, Yahoo Maps, Yandex карти та інші. До другої категорії відноситься проект Open Street Map.

Було вирішено використовувати в роботі сервіс Google Maps, що є популярним онлайн сервісом яким користуються по всьому світі(рис.3.2).

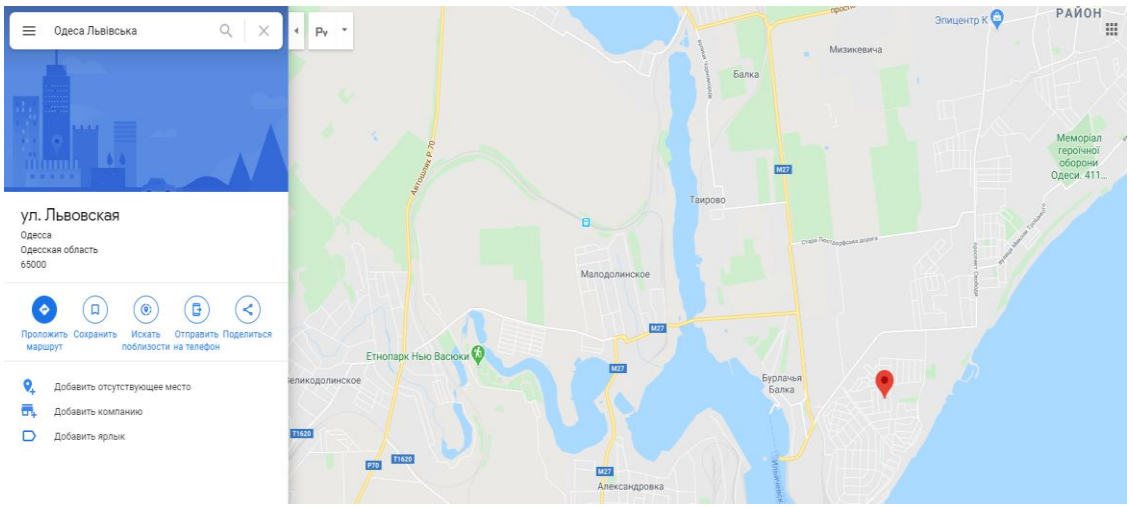


Рисунок 3.2 – Сервіс Google Maps

Карти від Google дозволяють дивитися геодані будь якого місця на планеті у схематичному або супутниковому вигляді з великої кількістю інфо-

рмації. Також сервіс дозволяє будувати маршрути між точками використовуючи різні види транспорту та віртуально мандрувати вулицями в крупних містах.

Для створення мобільного або веб-застосунку, що працює з картами, розробнику спочатку необхідно отримати спеціальний ключ (API Key). Завдяки такому ключу вендор може контролювати використання свого сервісу, і при необхідності, зв'язуватися з розробником. Існують певні обмеження на кількість з'єднань з сервісом. Так, компанія Google встановила ліміт в 25000 підключень до Google Maps в день. Використання Google Maps API платне, але з деякими умовами, тепер Google видає безкоштовно 200$ щомісяця на використання лімітів для карт, і якщо цю квоти не перевищувати, сервіс буде безкоштовним. Але незважаючи на безкоштовні ліміти, для всіх послуг потрібно підв'язати платіжний аккаунт у вигляді кредитної картки.

Щоб отримати Google Maps API Key необхідно мати обліковий запис Google. Далі у діалоговому вікні вибрати «Maps». Якщо ключ API створюється вперше, то спочатку необхідно створити проект вказавши його назву. Далі підключити систему оплати за використання API, створивши платіжний аккаунт. Ввести усі необхідні дані та натиснути «Почати безкоштовний пробний період», Google перевірить картку: зніме з неї $ і поверне його назад через деякий час. Коли підключений платіжний аккаунт, то на нього щомісяця нараховуються ліміти безкоштовного використання карт. Далі можна отримати API ключ, який необхідно скопіювати і зберегти собі для подальшого використання. Далі необхідно для API ключа включити обмеження, щоб ключ потрапивши в руки до третіх осіб, був недійсним. Це робиться в розділі «Захистити облікові дані».

Отриманий API ключ треба використовувати в програмному коді, при необхідності підключення карти. Якщо проект буде вимагати використання більш великих лімітів, то буде сенс перейти до використання альтернативних сервісів: OpenStreetMap, MapBox, 2GIS або інших.

3.3 Створення діаграми класів

Діаграма класів використовуються при моделюванні програмних систем (ПС) найбільш часто. Вона є однією з форм статичного опису системи з точки зору її проектування, показуючи її структуру. Діаграма класів не відображує динамічну поведінку об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси і відносини між ними.

Клас – це основний будівельний блок ПС. Кожен клас має назву, атрибути і операції. Клас на діаграмі показується у вигляді прямокутника, розділеного на 3 області. У верхній міститься назва класу, в середній – опис атрибутів (властивостей), в нижній – назви операцій – послуг, що надаються об'єктами цього класу.

Атрибути класу визначають склад і структуру даних, що зберігаються в об'єктах цього класу. Кожен атрибут має ім'я і тип, який визначає, які дані він представляє. При реалізації об'єкта в програмному коді для атрибутів буде виділена пам'ять, необхідна для зберігання всіх атрибутів, і кожен атрибут матиме конкретне значення в будь-який момент часу роботи програми. Об'єктів одного класу в програмі може бути як завгодно багато, всі вони мають однаковий набір атрибутів, описаний в класі, але значення атрибутів у кожного об'єкта свої і можуть змінюватися в ході виконання програми.

На діаграмах класів зазвичай показуються асоціації та узагальнення. Кожна асоціація несе інформацію про зв'язки між об'єктами всередині ПС. Найбільш часто використовуються бінарні асоціації, що зв'язують два класи. Узагальнення на діаграмах класів використовується, щоб показати зв'язок між класом-батьком і класом-нащадком. Воно вводиться на діаграму, коли виникає різновид будь-якого класу, а також в тих випадках, коли в системі виявляються кілька класів, що володіють схожим поведінкою (в цьому випадку загальні елементи поведінки виносяться на більш високий рівень, утворюючи клас-батько.

Діаграма класів створюються при логічному моделюванні ПС і служать для наступних цілей:

- для моделювання даних;.
- для уявлення архітектури ПС;
- для моделювання навігації екранів;
- для моделювання логіки програмних компонент;
- для моделювання логіки обробки даних.

Після розробки діаграми прецедентів в роботі була створена діаграма класів, яка представлена у Додатку А. На цій діаграмі можна побачити основні та допоміжні класи.

3.4 Реалізація серверної частини

Для реалізації серверу був вибран Spring Framework, який створений для Java-платформи. Spring Boot дозволяє створювати повноційний, продуктивний додаток. Spring Boot дозволяє за короткий термін створити робочий додаток у вигляді серверу. В роботі були використані деякі додаткові бібліотеки такі як JPA (Java persistant api) для роботи з базою данних, spring security для авторизації запитів.

Створено декілька суб'єктів за допомогою анотації @Entity, яка стає доступна завдяки JPA, і на основі цих суб'єктів створена базу даних.

Вирішено використовувати СКБД Postgres, тому що переваги БД, які були описані в розділі 2, роблять її найбільш підходящою для застосунку, що розроблюється, а також вона дозволяє зберігати великі об'єкти такі як картинки. Структура бази даних для мобільного застосунку наведена на рис. 3.3.

База даних складається з дев'яти таблиць, зв'язаних між собою. На БД Postgres можна побачити такі суб'єкти як user – втілює в собі користувача і всі його особисті дані, work – втілює в собі опис роботи з даними роботодавця, work_application – анкета робітника.

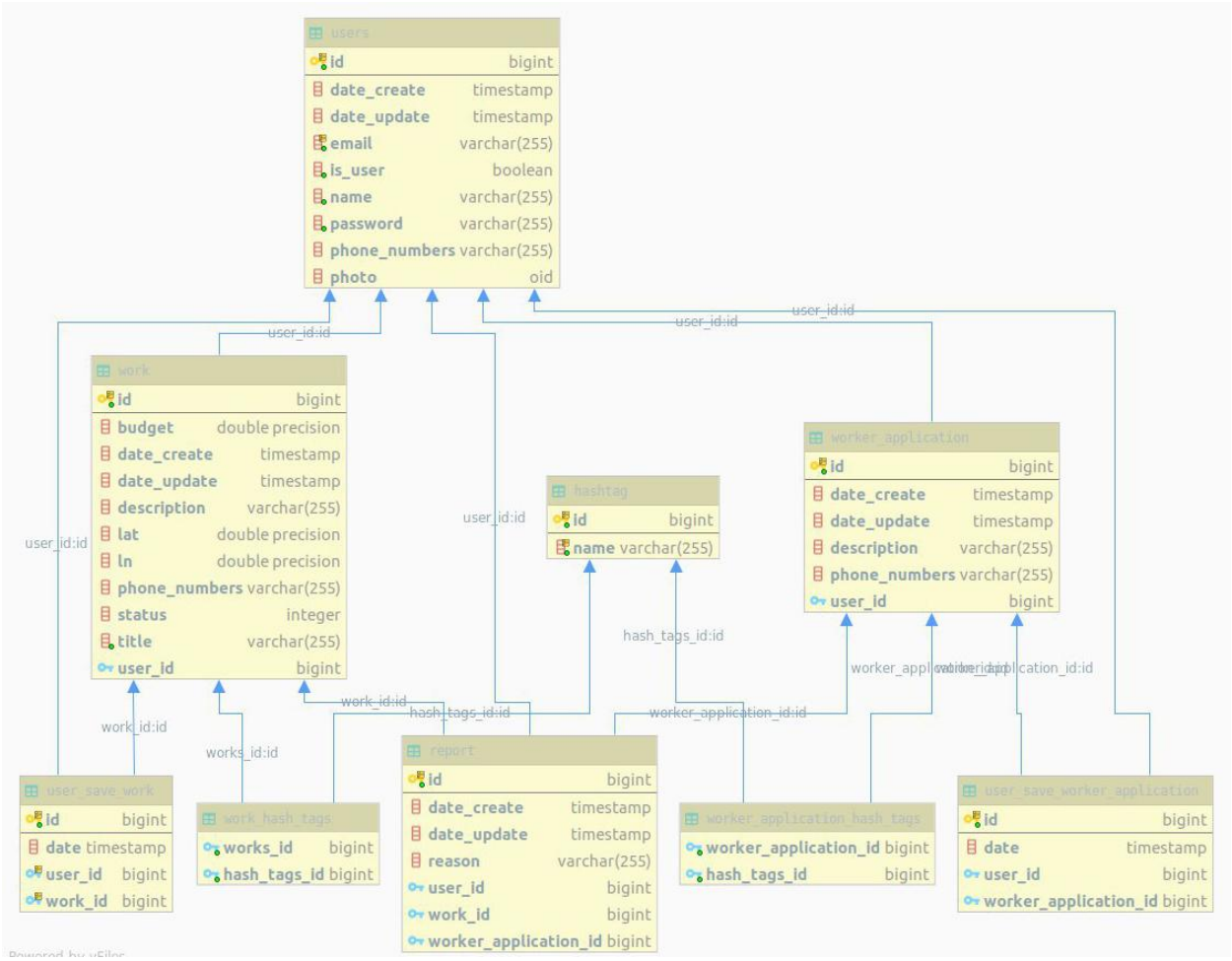


Рисунок 3.3 – Структура бази даних мобільного застосунку

Нижче наведено приклад суб’єкта, створеного за допомогою JPA.

```
@Entity(name="users")
public class User implements UserDetails {
        @Id
        @GeneratedValue
        private Long id;
        @Email
        @NotNull
        @Column(unique = true)
        private String email;
        @NotNull
        private String password;
        @NotNull
        private String name;
```

```
@OneToMany(mappedBy = "user")
@JsonManagedReference("user_work")
private List<Work> works;
@OneToMany(mappedBy = "user")
private List<UserSaveWork>  userSavedWorks;
@ManyToMany(targetEntity =
WorkerApplication.class)
private List<WorkerApplication>
savedWorkerApplication;
@Pattern(regexp = "([+]38[0-9]{10};)+")
private String phoneNumbers;
@Lob
private byte[] photo;
```

Для кожного суб'єкта були побудовані сервіс та репозиторій. Завдяки репозиторіям можна створювати запити до бази данних не використовуючи SQL-мови, окрім певних випадків коли потрібно створити супер специфічний запит. В сервісах зроблена обробка даних для об'єктів, наприклад, перед тим як відправити нового користувача, перевіряється правильність введених даних та шифрується його пароль оскільки зберігання пароля у відкритому вигляді вважається bad practice.

Приклад репозиторію для користувача:

```
@Repository
public interface UserRepository extends JpaRepository<User,
Long> {
   Optional<User> findByEmail(@Email String email);
@Transactional
void deleteByEmail(@Email String email);
@Transactional
@Modifying
@Query(value = “update users u set u.isUser = :is_user where
u.email = :email”)
void updateIsUser(@Param(“is_user”) Boolean isUser,
@Param(“email”) String email);
}
```

Приклад репозиторію для роботи:

```
@Repository
public interface WorkRepository extends JpaRepository<Work,
```

```
Long> {
void deleteByIdAndUser(Long id, User user);
```

Створено декілька контролерів, які позначені анотацією @RestController, оскільки сервер створюється на основі Rest архітектури. В кожному с контролерів створено методи, які відповідають за певний запит до цього серверу. Кожен з методів позначено @RequestMapping завдяки їм можна задати певні параметри запиту такі як тип запиту, та його адрессу. Також існують анотації для атрибутів цього методу. Такі як @RequestBody – тіло запиту, @RequestParam – параметр запиту.

Створено окремий клас Spring Security, який сконфігуровано так, щоб він перевіряв тільки запити, де потрібна авторизація. В певних методах контролерів існує перевірка на авторизацію і якщо користувач не авторизований, то він не отримує всього контенту. Для того щоб Spring Security перевіряв пароль та логін користувача, для суб'єкту user включено реалізацію інтерфейсу UserDetails, а також передано UserService, щоб він міг отримувати користувачів та перевіряти їх дані.

Нижче наведено приклад програмного коду для контролера.

```
@RestController
@RequestMapping(value = "/user")
   public class UserController {
      private UserService userService;
@RequestMapping(value = "/register",
                          method = RequestMethod.POST)
   public void setUser(@RequestBody User newUser) throws
       DataFormatException { userService.saveUser(newUser);}
@RequestMapping(value = "/get")
   public User getUser(Principal principal){
    User user =
            userService.getUserByEmail(principal.getName());
         user.setPassword(null);
         return user; }
@RequestMapping(value="/edit", method = RequestMethod.POST)
   public void editUser(Authentication authentication,
@RequestBody User user){
        user.setIsUser(null);
```

```
        userService.editUser(user,
       ((User) authentication.getPrincipal()).getId());}
@RequestMapping(value = "/delete")
   public void delete(Principal principal){
        userService.removeUser(principal.getName());}
@RequestMapping(value = "/addadmin")
   public void setPermissionToUser(@RequestParam
                       (name = "email") String email){
        userService.changeIsUser(false, email);}
@Autowired
   public void setUserService(UserService userService) {
        this.userService = userService; }
    }
```

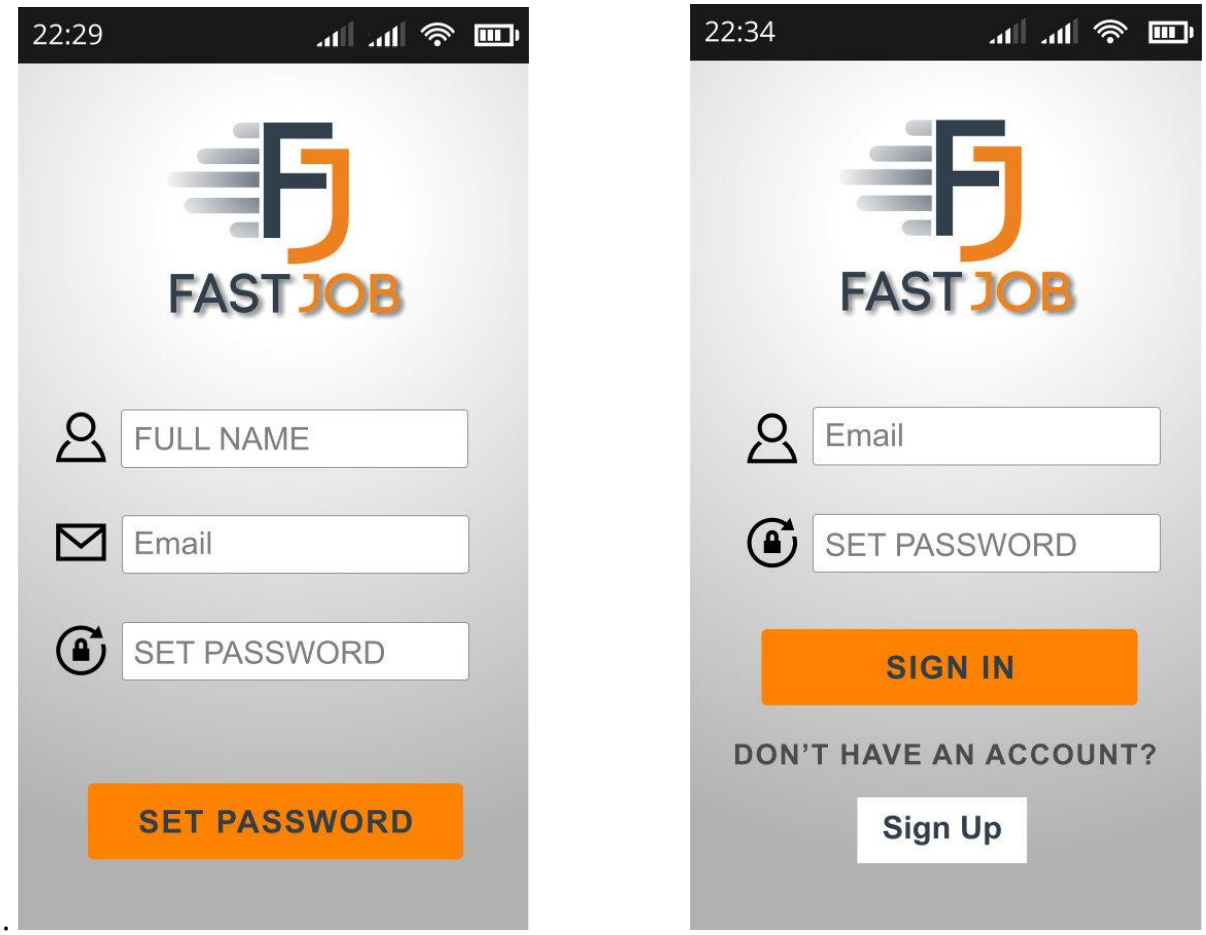
Для хостингу серверу вирішено було використовувати Heroku, оскільки він безкоштовний і в ньому можна підключити БД, а ще Heroku дає можливість використовувати протокол HTTPS для більш безпечного з'єднання з сервером. Недоліком є те, що якщо більше 30 хвилин сервер не отримує запитів, то Heroku його вимикає, проте як тільки хтось відправляє запит, він знову піднімає сервер. Spring boot являє собою потужний інструмент для розробки, оскільки розробник більше турбується за функціонал, чим за написання коду для отримування запитів і подібних речей.

3.5 Реалізація клієнтської частини

У якості клієнта була обрана ОС Android, оскільки ця операційна система найбільш росповсюджена. Для програми створено лояути регістрації, авторизації, показу списку та додавання вакансій. Розглянемо їх детальніше.

На рис.3.4а зображено макет реєстрації. Для регістрації використовується поля: ім'я , електронна пошта та пароль. Ми відмовилися від вказання додаткової інформації, такої як стать, вік і т.д., оскільки користувачам не довподоби довгий процес реєстрації. Додаткову інформацію можно згодом доповнити при необхідності. Щоб завершити регістрацію, порібно натиснути "Sing Up".

На рис.3.4б зображено макет авторизації. Для авторизації користувач використовує електронну пошту та пароль. Також, якщо користувач не зареєструвався, він може пройти дану процедуру натиснувши “Sing Up”. Для входу в систему треба натиснути “Sing In”.

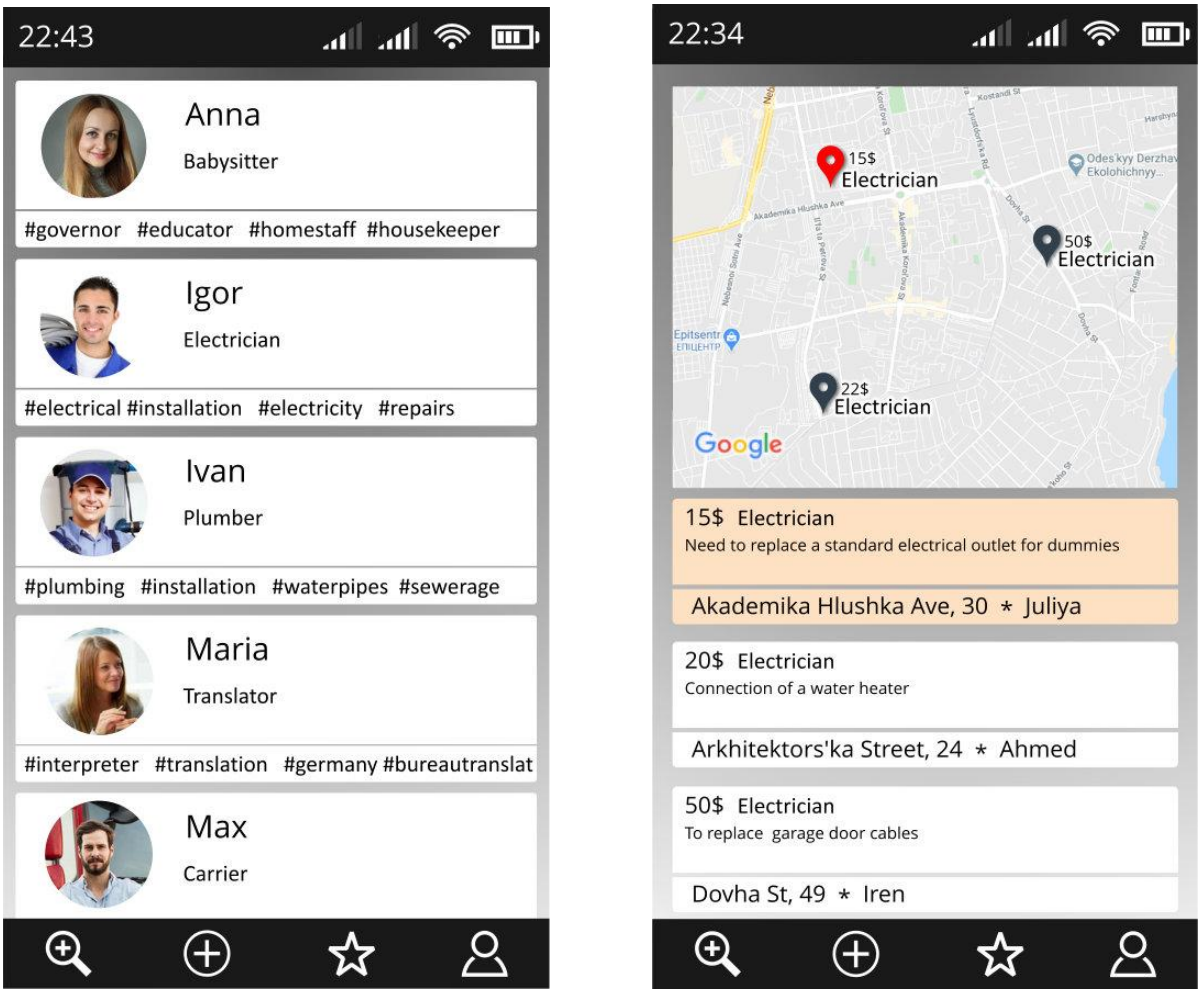


а) б)

Рисунок 3.4 – Вікна входу до застосунку: а – вікно реєстрації користувача; б – вікно авторизації користувача

На рис.3.5 а та рис.3.5 б зображено списки для анкет робітників та заявок користувачів відповідно. Заявки користувачів відображаються на карті, це зроблено для зручного пошуку роботи біля користувача. Для підключення карти був використаний Google Map API, який надає обмежений ліміт використання. Якщо проект буде вимагати використання більш великих лімітів,

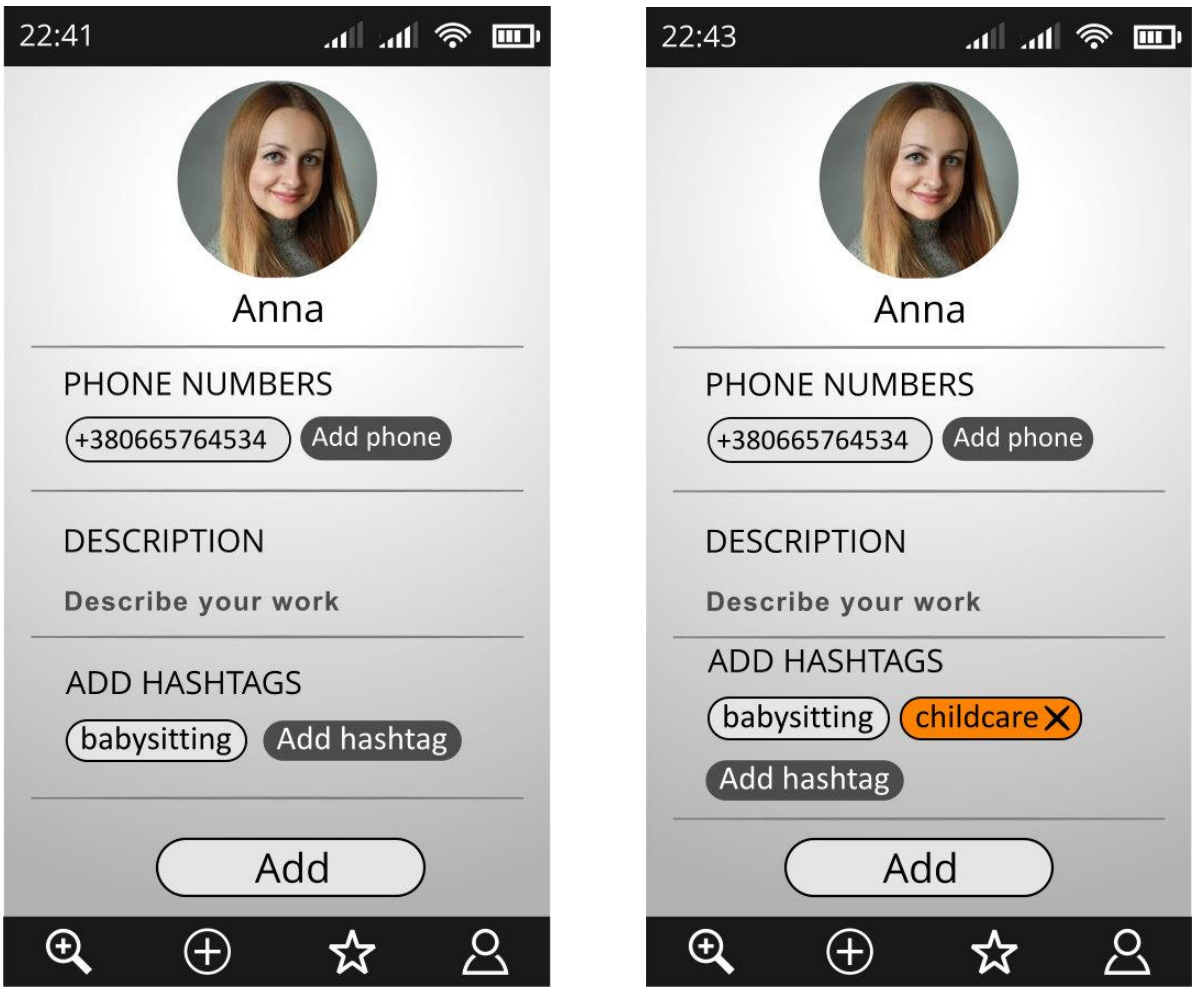
то буде сенс перейти до використання альтернативних сервісів: OpenStreetMap, MapBox, 2GIS або інших.



а) б)

Рисунок 3.5 – Вікна з даними робітників та користувачів: а – список анкет робітників; б – список заявок користувачів

На рис.3.6 а та рис.3.6 б зображено макет для додавання анкет робітників в систему. Щоб додати анкету, користувач має вказати опис своїх послуг, додати хештеги, завдяки яким буде проводитись пошук, а також при необхідності вказати додаткові номери телефонів. Інші дані такі, як і’мя та основний номер телефону, система сама підтягує з профілю користувача. Дана функція реалізована для спрощення користування програмою. Щоб додати анкету, користувач має натиснути на кнопку “Add”.



а) б)

Рисунок 3.5 – Вікна додавання анкети робітників: а – початок анкети; б – кінець анкети

Таким чином, в дипломній роботі створено мобільний застосунок для ОС Android для швидкого пошуку короткострокової роботи, який має зручний та зрозумілий інтерфейс користувача. Програмна реалізація здійснена на мові програмування Java з використанням ряду підпроектів Spring Framework. Для швидкого, якісного та зручного написання бізнес логіки проекту використовувався Spring Boot. Spring Security використовувався для реалізації серверної безпеки, а також реєстрації та авторизації. В якості засобу автоматизації управління та складання проекту використовувався Gradle. При розробці програмної реалізації була використана система контолю вер-

сій Git. Також було здійснено розгортання серверу на Heroku та завантаження програми на GooglePlay.

Інструкція користувача, що наведена у даному розділі демонструє головні можливості інтерфейсу користувача, та послідовність його використання. Окремо було проведено тестування програмної системи, яке не виявило значних проблем при її використанні.

ВИСНОВКИ

В результаті виконання бакалаврської роботи було проведено аналіз вимог до програмної системи: керування правами доступу та історія користувача. Здійснено обґрунтований вибір програмних засобів розробки. На етапі проектування програмної системи було створено діаграми прецедентів та класів. За допомогою них мета створення мобільного застосунку стала більш прозора, та його розробка стала легшою.

Програмна реалізація застосунку здійснена на мові програмування Java з використанням ряду підпроектів Spring Framework. Для швидкого, якісного та зручного написання бізнес логіки проекту використовувався Spring Boot. Spring Security використовувався для реалізації серверної безпеки, а також реєстрації та авторизації. В якості засобу автоматизації управління та складання проекту використовувався Gradle. Також був розроблений графічний інтерфейс програми для ОС Android. Для підключення карт використано API Google Map.

В якості БД була використана об’єктно-реляційна СКБД PostgreSQL. Вона надає величезну кількість інструментів та можливостей для зберігання та обробки даних різних типів. Для легких маніпуляцій з базою даних була використана технологія об’єктно-реляційного мапінгу.

При розробці програмної реалізації була використана система контолю версій Git. Також було здійснено розгортання серверу на Heroku та завантаження програми на GooglePlay.

В роботі наведено інструкцію користувача. Показано головні можливості інтерфейсу користувача, та послідовність його використання. Проведено тестування програмної системи, яке не виявило значних проблем при її використанні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. David., Ferraiolo, Richard., Kuhn, D. Role-based access control (вид. 2nd ed). Boston: Artech House, 2007. p. 456.
2. Розповідь користувача – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Розповідь_користувача (дата звернення 04.01.2020)
3. Мікросервіси – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/ Мікросервіси (дата звернення 04.01.2020)
4. Герберт Шилдт. Java. Полное руководство (Java SE 7, 8-е издание). М.: Изд. Дом «Вильямс», 2012. 1104с.
5. Intellij IDEA – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/ IntelliJ_IDEA (дата звернення 04.01.2020)
6. Spring Framework – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/ wiki/Spring_Framework (дата звернення 04.01.2020).
7. Кларенс Хо, Роб Харроп. Spring 3 для профессионалов. М.: «Вильямс», 2012. 880 с.
8. Крейг Уоллс. Spring в действии. М.: «Manning», 2014. 624 с.
9. Heroku – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Heroku (дата звернення 04.01.2020)
10. Коматинени С. Android 4 для профессионалов. Создание приложений для планшетних компьютеров и смартфонов: Научно-популярное издание. Санкт-Петербург: ТОВ «И.Д. Вильямс », 2012. 880 с.
11. Дон Гриффитс, Девид Гриффитс Программирование для Android. СПб.: Питер, 2018. 912 с.

12. Билл Филлипс, К. Стюарт, Кристин Марсикано Android. Программирование для профессмоналов. СПб.: Питер, 2017. 688 с.
13. Ян Клифтон Проектирование пользовательского интерфейса на Android.Санкт-Петербург: ДМК Прес, 2017. 452 с.
14. Gradle – Вікіпедія. (загол. з екрана). URL: https://uk.wikipedia.org/wiki/Gradle (дата звернення 04.01.2020)
15. Лузанов П., Рогов Е., Левшин И. PostgreSQL для начинающих. М., 2017. 147 с.
16. Грейди Буч. Язык UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. СПб.: Питер, 2004. 432 с.