Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: Розробка сервісу для пошуку місць проведення дозвілля
з використанням інтелектуального підбору рекомендацій
на основі аналізу взаємодії користувача із додатком
Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
Олексій ВЛАСЕНКО
(розшифровка підпису)(підпис)(дата)

Науковий керівник
доцент, к. фіз.-мат. наук
Сергій ПОЛЯКОВ
(посада)(розшифровка підпису)(підпис)(дата)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри

Олексій БИЧКОВ
(розшифровка підпису)(підпис) (дата)

2021
(рік)

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента
Олексія ВЛАСЕНКО
захищена з оцінкою

_____

Голова екзаменаційної комісії
професор, д.т.н. Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій _____
(Олексій БИЧКОВ)
“___”_____ 2021 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
Власенко Олексію Владиславовичу
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи “Розробка сервісу для пошуку місць проведення дозвілля з використанням інтелектуального підбору рекомендацій на основі аналізу взаємодії користувача із додатком”
затверджена наказом вищого навчального закладу від “____”_____2021 р. №_____

2. Строк здачі студентом закінченої роботи _____
3. Вихідні дані до роботи

1. Кросплатформний мобільний додаток
2. Сервер
3. Вебдодаток для адміністратора

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналіз задачі створення мобільного додатку
2. Проектування мобільного додатку та серверу
3. Тестування та розгортання мобільного додатку та серверу
4. Оцінка вартості розгортання та реалізації розробленого програмного продукту

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

Методи пошуку закладів, функціонал додатку, засоби реалізації, інтерфейс та робота додатку, додаток адміністратора, інтелектуальний підбір рекомендацій

Консультанти розділів проекту

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Сергій ПОЛЯКОВ	15.03.2021	12.04.2021
2	Сергій ПОЛЯКОВ	14.04.2021	21.05.2021
3, 4	Сергій ПОЛЯКОВ	21.05.2021	02.06.2021

7. **Дата видачі завдання** _____

Керівник _____ (Сергій ПОЛЯКОВ)
(підпис)

Завдання прийняв до виконання _____ (Олексій ВЛАСЕНКО)
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	25.01.2021-12.02.2021	виконано
2	Аналіз концепцій та алгоритмів	12.02.2021-10.03.2021	виконано
3	Розробка серверу та REST-Api для додатку	10.03.2021-24.03.2021	виконано
4	Розробка алгоритмічної моделі рекомендацій на основі колаборативної фільтрації	24.03.2021-04.04.2021	виконано
5	Розробка мобільного додатку	04.04.2021-29.04.2021	виконано
6	Розгортання серверу, адмінпанелі та додатку	29.04.2021-13.05.2021	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	16.06..2021	виконано

Студент – бакалавр _____(Олексій ВЛАСЕНКО)
(підпис)

Керівник роботи _____(Сергій ПОЛЯКОВ)
(підпис)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 68 с., 14 рис., 3 додат., 11 джерела.

Тема: Розробка сервісу для пошуку місць проведення дозвілля з використанням інтелектуального підбору рекомендацій на основі аналізу взаємодії користувача із додатком

Об'єкт дослідження: метод пошуку місць для проведення дозвілля.

Мета роботи: розробка мобільного додатку та серверу, за допомогою яких буде здійснюватись пошук закладів.

Предмет дослідження: сервіс підбору місць для проведення дозвілля на основі фільтрів, інтересів та взаємодії користувача із додатком.

Результати дослідження:

Досліджено методи інтелектуального підбору рекомендацій на основі фільтрації вмісту та колаборативної фільтрації. Створено повноцінний мобільний додаток, готовий до релізу.

Висновок

В результаті роботи над дипломним проєктом було розроблено та розгорнуто повноцінну систему пошуку та підбору рекомендацій.

ДОДАТОК, СЕРВЕР, JAVASCRIPT, NODE.JS, REACT NATIVE, REST API, REDUX, DOCKER, MONGODB, РЕКОМЕНДАЦІЙНА СИСТЕМА ІНТЕЛЕКТУАЛЬНИЙ ПІДБІР.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 68 с., 14 рис., 2 доп., 11 источников.

Тема: Разработка сервиса для поиска мест проведения досуга с использованием интеллектуального подбора рекомендаций на основе анализа взаимодействия пользователя с приложением

Объект исследования: Метод поиска мест для проведения досуга.

Цель работы: Разработка мобильного приложения и сервера, с помощью которых будет осуществляться поиск заведений.

Предмет исследования: Сервис подбора мест для проведения досуга на основе фильтров, интересов и взаимодействия пользователя с приложением.

Результаты исследования:

Исследованы методы интеллектуального подбора рекомендаций на основе фильтрации содержимого и колаборативних фильтрации. Создано полноценное мобильное приложение, готовое к релизу.

Вывод:

В результате работы над дипломным проектом была разработана и развернута полноценная система поиска и подбора рекомендаций.

ПРИЛОЖЕНИЕ, СЕРВЕР, JAVASCRIPT, NODE.JS, REACT NATIVE, REST API, REDUX, DOCKER, MONGODB, РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА, ИНТЕЛЛЕКТУАЛЬНЫЙ ПОДБОР.

SUMMARY

Final qualifying bachelor's thesis: 68 pages, 14 figures, 2 appendices, 11 sources.

Topic: Mobile application for searching points of interest, using an intelligent selection of recommendations based on the analysis of user interaction with the application

Object of research: a method of finding points of interest.

Purpose: development of a service for finding points of interest. Service will includes mobile application and a server.

Results of the research:

Methods of intelligent selection of recommendations based on content filtering and collaborative filtering are studied. A service (mobile application and server) is ready for release.

Conclusion

As a result of work on the diploma project, a full-fledged service for finding points of interest was developed

APP, SERVER, JAVASCRIPT, NODE.JS, REACT NATIVE, REST API, REDUX, DOCKER, MONGO DB, RECOMMENDATION SYSTEMS INTELLECTUAL SELECTION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІС - інформаційна система

ІТ - інформаційні технології

ОС - операційна система

ПЗ - програмне забезпечення

БД - база даних

СУБД - система управління базами даних

API - Application Programming Interface

REST - Representational State Transfer

IDE - Integrated development environment

JS - Java Script

JSC - Java Script Core

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

HTTP - HyperText Transfer Protocol

TCP - Transmission Control Protocol

ВСТУП

Мета і задачі дослідження

Метою бакалаврської роботи є розробка мобільного додатку пошуку закладів для проведення дозвілля використовуючи інтелектуальний підбір рекомендацій.

Проєкт, що розроблюється повинен включати в себе клієнтський додаток, серверну частину з базою даних. Клієнтський додаток в свою чергу буде поділятися на сторону клієнта та адміністратора. Особливо бажаними вимогами являються швидка робота додатку та серверу та релевантність пошуку.

Досягнення мети включало розв'язання таких **задач**:

1) огляд існуючих концепцій пошуку місць та закладів;
2) вивчення можливостей застосування інтелектуального підбору;
3) вибір релевантного алгоритму та обгрунтування доцільності його використання;
4) реалізація додатку;
5) розробка та розгортання серверу.

Об'єктом дослідження є метод пошуку місць для проведення дозвілля.

Предметом дослідження сервіс підбору місць для проведення дозвілля на основі фільтрів, інтересів та взаємодії користувача із додатком.

Методи дослідження

Для розробки сервісу досліджено існуючі рішення для пошуку закладів, проаналізовано їхні позитивні та негативні сторони. Також було досліджено рішення відомих компаній, що використовують алгоритми підбору рекомендацій. Під час розробки було враховано підкреслені переваги та недоліки з досвіду відомих компаній.

Наукова новизна отриманих результатів

Досліджено можливості застосування рекомендаційного підбору для поставленої задачі по розробці проєкту. Запропоновано рішення, яке може стати зручним інструментом для пошуку місць та закладів для проведення дозвілля.

Практичне значення одержаних результатів

Одержано мобільний додаток, що дозволяє виконувати пошук закладів по обраним фільтрам та інтересам враховуючи інтелектуальний підбір. Такий проєкт має перспективи для розвитку та монетизації.

Особистий внесок студента

Основним результатом є:

1. запропонований автором підхід до вирішення поставленої задачі, застосування інтелектуального підбору даних;
2. повноцінний розроблений мобільний додаток, що виконує пошук місць для проведення дозвілля на основі інтелектуального підбору рекомендацій

Структура та обсяг роботи

Робота викладена на 69 сторінках друкованого тексту, який складається із вступу, чотирьох розділів, висновків, списку використаних джерел 11 найменувань). Робота містить 14 рисунків та 2 додатки, обсягом 13 стор.

1. АНАЛІЗ ЗАДАЧІ СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

1.1 Характеристика предметної області

Ідея створення даного проєкту виникла у 2020 році під час локдауну. В цей період люди були змушені дотримуватись жорстких правил карантину та були позбавлені можливості відвідувати розважальні заклади та заклади харчування. Окрім того деякі кав'ярні, кафе, ресторани були закриті в минулому році у зв'язку із фінансовими труднощами. За даними звіту аналітичного центру "Ресторани України" 3850 закладів було закрито під час карантину (рис. 1.1). Саме це наштовхнуло на думку про створення додатку, який допоможе користувачам знаходити цікаві та нові для себе місця для проведення дозвілля, що також допоможе перезавантажитись після важкого та затяжного періоду карантину. Для власників різного роду закладів це стане додатковою можливістю прорекламувати себе або нагадати про себе своїм відвідувачам.

Рис. 1.1. Розвиток ресторанного господарства України

Крім того, варто підкреслити, що у додатку буде застосований інтелектуальний підбір рекомендацій для підвищення ефективності досягнення поставлених цілей. Тут мається на увазі вірогідність того, що користувач відвідає запропонований йому заклад.

Отже, у даній курсовій роботі розробляється мобільний додаток для операційних систем Android та iOS, що буде використовуватись для пошуку місць

проведення дозвілля з використанням інтелектуального підбору рекомендацій на основі аналізу взаємодії користувача із додатком.

Пропонується забезпечити наступні функції в рамках дипломної роботи:

- рекомендації закладів для користувача;
- додавання закладів у розділ “Обране”;
- пошук закладів по фільтрам;
- отримання інформації про запропонований заклад;
- прокладання маршруту до закладу через Google Maps;
- додавання та редагування інформації про заклад власниками.

Даний проект має перспективи для розвитку. До функціоналу, описаного вище можна додати наступні можливості:

- ставити оцінку закладу;
- залишати відгуки;
- завчасно бронювати місце або стіл;
- переглянути меню або прайс на послуги;
- програми бонусів та знижок для відвідувачів, тощо;

Керуючись концепцією бережного виробництва Lean Startup було вирішено виконати додавання цих функцій після запуску проєкту. Lean Startup - бестселер Еріка Рікса, що рекомендується для читання підприємцям та усіх, хто прагне запускати власні інноваційні продукти.

Всупереч обґрунтованим розрахункам, серйозним бізнес-планам, продуманим бізнес-моделям, великим інвестиціям, більшість стартапів зазнають краху. Ерік Рис, автор методики «ощадливого стартапу» впевнений в тому, що традиційний підхід до розвитку бізнесу не варто застосовувати до стартапів.

Стартап діє в умовах надзвичайної невизначеності, і це потрібно враховувати при його запуску. На початковому етапі розвитку стартапу необхідно залишатися гнучким, щоб вчитися на помилках і максимально швидко перевіряти гіпотези

засновників, а значить, необхідно уникати великих вливань і витрат. Саме такий підхід лежить в основі методу Lean, мета якого допомогти підприємцям підвищити шанси стартапу на успіх.

Для стартапу необхідно якомога швидше приступити до створення мінімально робочого продукту (MVP). MVP - це версія продукту, що дозволяє запустити цикл" створити, оцінити та навчитися з мінімальними зусиллями, витративши якнайменше часу на розробку.

Такий продукт може бути позбавлений опцій, які в майбутньому будуть найбільше цінувати клієнти, але в той самий час йому досить бути таким, щоб можна було оцінити його успіх чи невдачу - він повинен бути зрозумілий першим користувачам.

Оцінити - це означає визначити, чи приводять зусилля по створенню продукту до потрібних результатів. І це ключова відмінність оцінки за методом ощадливого стартапу від стандартних методів, коли оцінюється виконання термінів і освоєння бюджету, але може не враховуватися, що стартап створив те, що нікому не потрібно. Основний метод оцінки в Lean startup - це облік інновацій.

Навчитися означає зрозуміти, чи рухатися далі тим же шляхом або потрібно зробити віраж - кардинальний перегляд бізнес-моделі. Коли підприємець бачить, що обраний шлях не веде до успіху, він повинен бути готовий знайти нову стратегічну гіпотезу і перестати витрачати кошти на проходження непотрібного напрямку.

Таким чином при першому релізі додатку буде забезпечений мінімальний набір функцій. Увага буде зосереджена на просуванні додатку. Далі буде проведено оцінку успішності проєкту, аналіз поведінки користувача у додатку, зважено плюси та мінуси. Після отримання результатів та висновків про те, що реліз був успішним, з’явиться необхідність у реалізації додаткових функцій.

1.2 Інтелектуальний підбір рекомендацій

Рекомендаційна система — технологія підбору релевантних, відносно інтересів користувача, результатів пошуку. Дана технологія стала досить розповсюдженою в останні роки. Велика кількість відомих компаній застосовує інтелектуальний підбір рекомендацій з метою зацікавити та повертати користувача до використання додатку. Нижче наведені приклади таких компаній:

- TikTok - сервіс коротких відео;
- Netflix - онлайн-кінотеатр;
- Instagram - соціальна медіа-мережа;
- YouTube - відеохостинг;
- Google - пошукова система;
- Spotify - потоковий аудіо-сервіс.

Розділяють два методи для створення систем рекомендацій: фільтрація вмісту, колаборативна фільтрація.

Для використання фільтрації вмісту необхідно створити профілі для користувачів та об’єктів пошуку.

Профілі користувачів можуть містити таку інформацію:

- стать
- вік
- країна
- сімейний стан
- інтереси
- відповіді на певні питання.

Профілі об’єктів включають загальну інформацію в залежності від різновиду об’єкту. Наприклад:

- для музики, фільмів (автор, жанр, настрій, тощо);

- відео YouTube (канал, тип контенту, тривалість тощо.

Суть колаборативної фільтрації полягає в аналізі поведінки користувачів у минулому - інформація про купівлю того чи іншого товару, відгуки, оцінки, час перегляду контенту, поширення, тощо. В цьому методі не має значення те, які об’єкти використовуються, але при цьому можна брати до уваги неявні характеристики, які складно було б врахувати при створенні профілю. Проте у цього методу є одна суттєва проблема - відсутність даних про поведінку користувача, які входять у систему вперше.

На практиці системи рекомендацій збирають дані про користувачів застосовуючи явні та неявні методи спільно.

Далі буде наведено приклади застосування явного збору даних:

- оцінювання конкретного об’єкта за запропонованою шкалою;
- ранжування об’єктів від найкращого до найгіршого на думку користувача;
- вибір одного із двох об’єктів;
- створення списку улюблених об’єктів.

Неявний збір даних:

- відслідковування того, який контент користувач переглядає;
- поведінка користувача у рамках сервісу;
- аналіз вмісту пристрою користувача (ПК, телефон, тощо).

Рекомендаційні системи є затребуваним інструментом у сьогоденні. Він допомагає компаніям залучати юзерів до користування додатком, повернення та продовжувати час взаємодії із сервісом.

1.3 Огляд існуючих засобів пошуку закладів проведення дозвілля

Основним способом пошуку інформації в сучасності є пошукова стрічка браузера. Пошук закладів для проведення дозвілля не є виключенням. Зараз майже у кожного закладу є свій сайт, на якому можна переглянути фотографії, дізнатись ціни, знайти номер телефону для бронювання та адресу. Проте для того, щоб зайти на сайт певного закладу потрібно знати його назву.

Назву закладу можна дізнатись по рекомендаціям знайомих, із реклами у соціальних мережах, з пошуку в Google Maps або з сайтів, на яких представлені групи закладів різного призначення.

Говорячи про Google Maps можна сказати, що це досить зручний спосіб пошуку місць - можна виконати пошук ресторанів, що знаходяться поблизу (рис. 1.2), отримати багато важливої інформації (переглянути фотографії, меню, графік роботи, відгуки, способи надання безпечного обслуговування, в період пандемії, частота відвідування у різні години знайти сайт закладу тощо). Проте недоліком такого методу може стати відсутність деякої інформації, її неактуальність, особоливо, якщо мова йде про меню або вартість послуг. Крім того для бронювання або перегляду меню потрібно буде виходити за рамки додатку Google Maps - дзвонити та переходити на сайт.

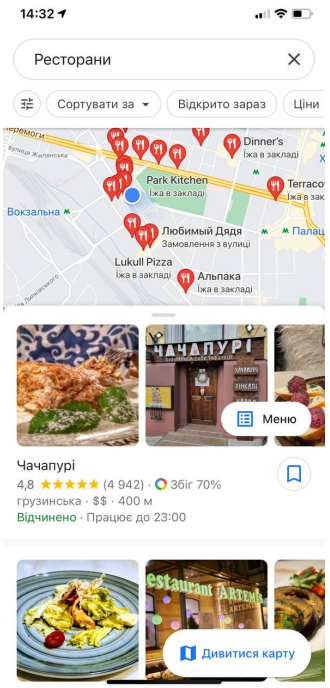

Рис. 1.2. Пошук у Google Maps

Прикладом сайту для вибору закладів є Zoon.com.ua (рис. 1.3). Сайт наповнений безліччю закладів із різних сфер: їжа, краса, медицина, розваги, покупки, курси, фітнес, тощо. Пошук можна виконувати по назві та по категоріям. Користувач може побачити заклад на мапі, переглянути меню, фотографії. Сервіс працює не тільки в Києві, а й у інших великих містах світу - міжнародний сервіс має назву Nicelocal.

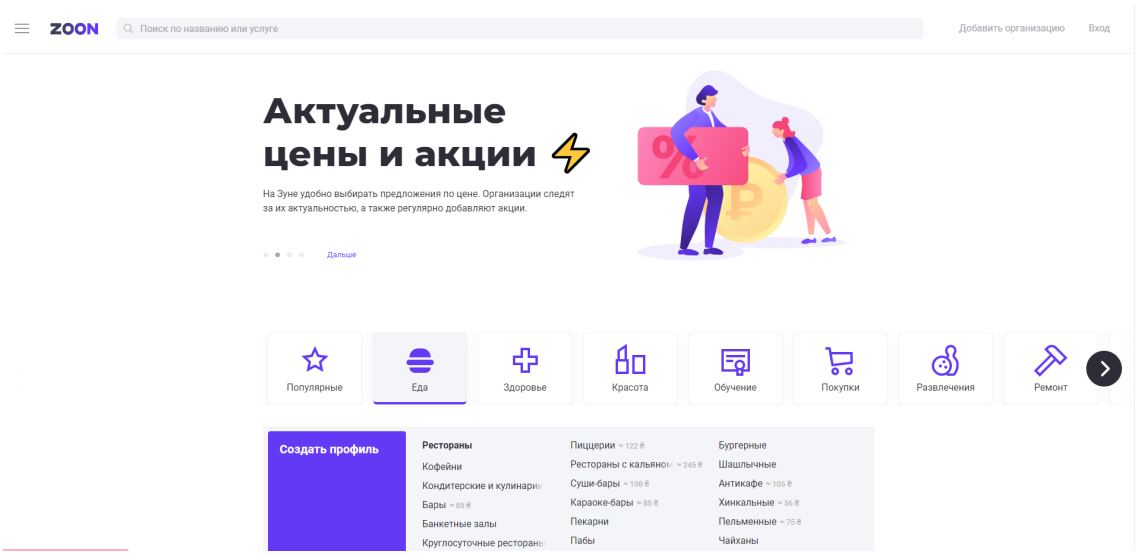

Рис. 1.3. Сайт сервісу Zoon

Проєкт “Shake&Go” буде у форматі мобільного додатку, адже станом на 2021 рік було визначено, що мобільні додатки подобаються користувачам значно більше аніж сайти, навіть якщо мова йде про сайти у мобільному браузері. Нижче, на рисунку 1.4 представлено статистичні дані щодо використання мобільного браузеру та додатків у США.

Рис. 1.4. Середній показник щоденного використання мобільних додатків і браузерів (2018-2022, США)

У Zoon є мобільний додаток, що також стає плюсом для сервісу. Додаток має хороший інтерфейс, він зручний у використанні та забезпечує майже усі потреби відвідувачів.

Недоліком Zoon можна назвати те, що користувач отримує список закладів, наприклад, по обраній категорії, за одним із двох варіантів: за рейтингом або у зв’язку із програмами просування закладів в рамках цього сервісу. Тобто на перших позиціях у пошуку найчастіше знаходитимуться ті самі заклади. Для ілюстрації та кращого розуміння даного моменту пропонується переглянути рисунок 1.5. На рисунку представлений пошук за різними категоріями при якому на перших місцях знаходяться одні і ті ж заклади.

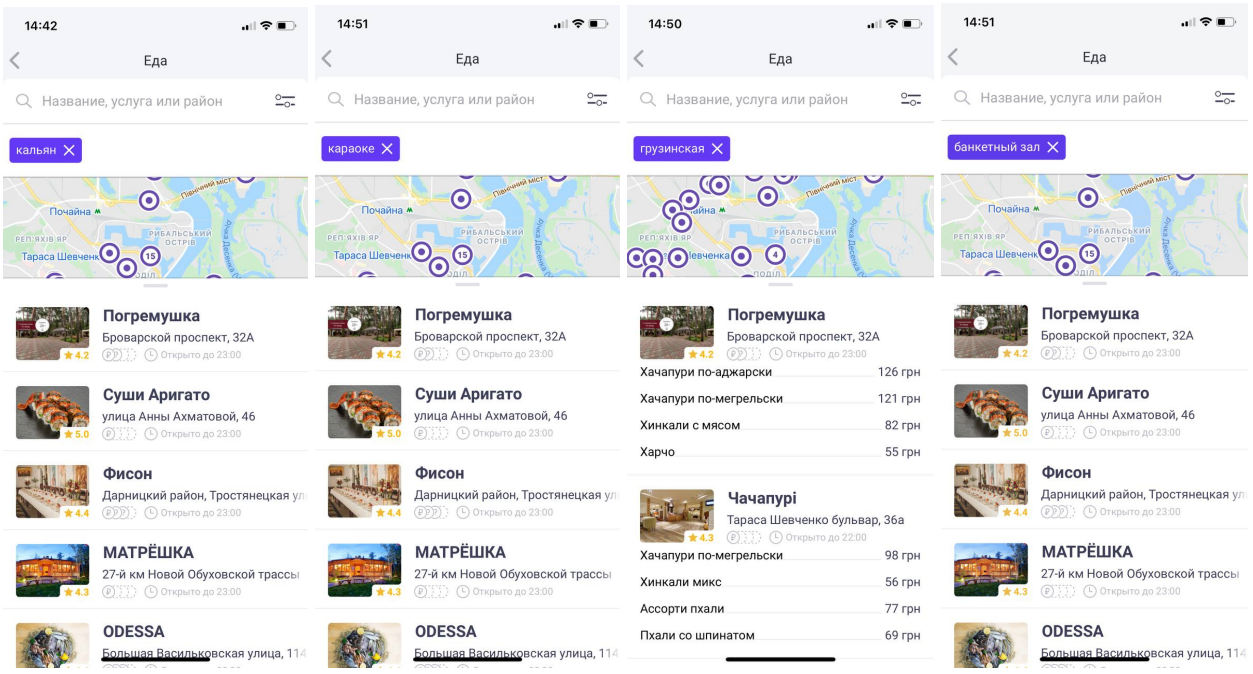

Рис. 1.5. Пошук за різними категоріями у Zoon

Основна ідея “Shake&Go” - показувати користувачам нові місця, про які вони раніше, можливо, навіть і не чули. Звісно, для монетизації проєкту будуть створені програми для просування закладів, але не дивлячись на це список буде підбиратись за налаштованими фільтрами, та інтелектуальним аналізом і будуть показані в рандомному порядку.

Також варто зазначити, що цільовою аудиторією стануть користувачі 16-25 років, адже саме ця аудиторія є найбільш активною в мобільних додатках. Саме тому метод використання додатку повинен бути незвичайним та цікавим для них. Додаток має назву “Shake&Go”, що перекладається з англійської як “Труси та йди”. Ця назва виходить із основної механіки додатку - щоб отримати список рекомендованих закладів потрібно потрусити мобільний телефон у руках.

Отже, проєкт, що розробляється, об'єднує в собі усі найкращі сторони описаних вище засобів пошуку місць для дозвілля і додасть певні покращення до процесу пошуку.

1.4 Аналіз вимог до функціонування додатку

Для того, щоб мобільний додаток “Shake&Go” був доступний якомога більшій кількості студентів та викладачів, він повинен працювати на Android та iOS починаючи з Android 4.4 та iOS 9 та бути готовим працювати на останніх версіях.

За функціоналом на початкових стадіях додаток буде досить простим, але достатнім для досягнення основної мети користувача. Основна мета користувача - знайти місце для відвідування із категорії, що його зацікавить.

Для втілення цієї мети користувачеві буде запропоновано налаштувати фільтри. Фільтри, за якими користувач зможе здійснити пошук:

- категорії (їжа, розваги, тощо);
- підкатегорії (грузинська кухня, італійська кухня);
- рівень цін (низький, середній, високий);
- графік роботи;
- радіус територіального пошуку.

Після налаштування фільтрів буде отримано список карток з інформацією про заклади. Також у користувача будуть “Обрані” заклади, які він сам додаватиме у цей список. Доступ до цих функцій користувач зможе отримати на головному екрані (рис. 1.2). Додавання до обраних також можна зробити свайпом (змахуванням) картки закладу вправо.

Рис. 1.6. Кнопки “Обране” та “Фільтри”

Якщо користувача зацікавило запропоноване місце у нього буде можливість отримати більше деталей. Для цього потрібно натиснути на картку. Після цього буде відкрито картку окремого закладу, де користувач зможе прочитати опис закладу, переглянути фотографії, меню тощо.

Важливою функцією буде можливість поділитися знайденим закладом, наприклад, із друзями, адже найчастіше заклади відвідують у компанії друзів. Наступна можливість - прокладення маршруту за допомогою Google Maps.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі було проаналізовано задачу створення програмного додатку для пошуку місць та закладів для проведення дозвілля на основі інтелектуального підбору рекомендацій:

- окреслено задачі, які повинен виконувати додаток;
- сплановано додавання функцій у майбутньому;
- проаналізовано предметну область розроблюваного проекту;
- досліджено алгоритми інтелектуального підбору рекомендацій;
- підтверджено доцільність використання рекомендаційного підбору у даному проєкті;
- здійснено огляд та порівняння існуючих засобів пошуку місць;
- розроблено основні правила та вимоги до додатку.

2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ТА СЕРВЕРУ

2.1 Вибір засобів реалізації

2.1.1 Вибір мови програмування

Для написання програми додатку була обрана мова JavaScript.

JavaScript — прототип-орієнтована динамічна мова, що має декілька парадигм та підтримує об'єктно-орієнтований, імперативний та декларативний (тобто функціональне програмування) стилі. [1]

JavaScript (JS) — це невибаглива до ресурсів мова програмування з функціями першого класу, код якої інтерпретується та компілюється під час виконання. Хоча JavaScript насамперед відома як скриптова мова для веб-сторінок, вона також використовується у багатьох небраузерних середовищах на кшталт Node.js, Apache CouchDB та Adobe Acrobat.

Обрана мова програмування є однією із найпопулярніших серед розробників. По цій причині у мережі є безліч безкоштовних навчальних матеріалів, корисних форумів та інших сайтів пов’язаних із JavaScript, що буде корисним для написання даного проекту. Також, завдяки своїй поширеності, розробку на JavaScript підтримує велика кількість різноманітних IDE та редакторів коду. Тому написання та розробка додатку за допомогою цієї мови не стане проблемою.

Стандартом для JavaScript є ECMAScript. Станом на 2012 рік усі сучасні браузери вже мали повну підтримку ECMAScript 5.1. Застарілі браузери підтримують щонайменше ECMAScript 3. 17 червня 2015 року ECMA International випустила шосту базову версію ECMAScript з офіційною назвою ECMAScript 2015, яка у попередніх обговореннях іменувалася ECMAScript 6 або ES6. Відтоді стандарти ECMAScript оновлюються раз на рік. Ця документація посилається на найсвіжішу версію чернетки, тобто ECMAScript 2019 на цей час.

JavaScript складається з трьох частин: [1]

- Ядро (ECMAScript) - базова функціональність JavaScript.
- Об'єктна модель документа (Document Object Model, DOM) - засоби для роботи з вмістом веб-сторінок.
- Об'єктна модель браузера (Browser Object Model, BOM) - засоби для взаємодії з браузером.

Ядро мови JavaScript складається з певної кількості звичайних функцій, які дозволяють робити наступне:

- Зберігати дані всередині змінних.
- Операції над фрагментами текстів (в програмуванні - "рядки").
- Запускати код відповідно до певних подій, що відбуваються на web сторінці.
- Додавати різноманітні ефекти анімації.
- Здійснювати перевірку введення даних в поля форми до відправки на сервер, що знімає навантаження з самого сервера.
- Створювати і зчитувати cookie, витягувати дані про комп'ютер користувача.
- Змінювати вміст HTML-елементів, додавати нові теги, змінювати стилі.
- та багато іншого.

Ще більш цікавим є функціональність, створена поверх основної мови JavaScript. Так звані інтерфейси прикладного програмування (далі API) надають вам додаткові можливості для використання в вашому коді JavaScript.

API - це готові набори блоків коду, які дозволяють розробнику реалізовувати програми, які в іншому випадку було б важко або неможливо реалізувати.

Вони зазвичай поділяються на дві категорії: 3rd party APIs, browser APIs.

Browser APIs вбудовані в веб-браузер і можуть відображати дані з навколишнього комп'ютерного оточення або робити корисні складні речі. Наприклад, API-інтерфейс DOM (Document Object Model) дозволяє маніпулювати HTML і CSS,

створювати, видаляти і змінювати HTML, динамічно застосовувати нові стилі до вашої сторінці і т.д.

Існують також і обмеження:

- JavaScript не може закривати вікна і вкладки, які не були відкриті з її допомогою.
- Не може захистити вихідний код сторінки і заборонити копіювання тексту або зображень зі сторінки.
- Не може здійснювати кроссдоменні запити, отримувати доступ до веб-сторінок, розташованих на іншому домені. Навіть коли сторінки з різних доменів відображаються одночасно в різних вкладках браузера, то код JavaScript, що належить одному домену не матиме доступу до інформації про веб-сторінку з іншого домену. Це гарантує безпеку приватної інформації, яка може бути відома власнику домену, сторінка якого відкрита в сусідній вкладці.
- Не має доступу до файлів, розташованих на комп'ютері користувача, і доступу за межі самої веб-сторінки, єдиним винятком є файли cookie, які JavaScript може записувати і зчитувати.

З цього можна зробити висновок, що JavaScript розроблена таким чином, щоб утруднити виконання шкідливого коду. [1]

Ця мова була обрана для написання проекту через ряд своїх переваг. Серед основних переваг використання мови програмування JavaScript можна виділити наступне:

- Кожен сучасний браузер підтримує JavaScript.
- Широкий вибір корисних функціональних налаштувань.
- Мова весь час покращується та розвивається.
- Висока швидкість роботи.
- Пряме підключення скриптів до HTML-коду.
- Мова проста для вивчення, навіть для новачків у сфері програмування.

- Доступна офіційна документація та велика кількість навчальних матеріалів.

Окрім очевидних плюсів використання, ця мова також має ряд недоліків.

JavaScript не є строго типізованою мовою. Тобто, основна частина перевірок типів виконується під час виконання програми. Це може призвести до неочікуваних помилок, ускладнить їх пошук та вирішення.

Через те, що JavaScript компілюється під час виконання будь-хто може отримати доступ до вихідного коду програми. Проте, частково вирішити проблему безпеки можна використовуючи обфускацію - процес заплутування коду, при якому код зберігає свою функціональність, але стає незрозумілим людині. Для того, щоб розшифрувати такий “заплутаний” код, використовують SourceMaps - спеціальні файли, в яких записана відповідність між оригінальним та обфускованим кодом.

Навіть беручи до уваги названі недоліки, для написання проекту вибрана мова JavaScript, адже у порівнянні із усіма перевагами вони незначні.

2.1.2 Вибір фреймворку для реалізації додатку

React Native - це JS-фреймворк для створення нативно відображаючих iOS- і Android-додатків. В його основі лежить розроблена в Facebook JS-бібліотека React, призначена для створення користувацьких інтерфейсів.

React - це JS-бібліотека для створення користувацьких інтерфейсів, зазвичай для веб-додатків. Вона розроблена в Facebook і поширюється під ліцензією open source з 2013 року. React широко поширена, і на відміну від більш великих MVC-фреймворків вирішує більш вузьке завдання: рендеринг інтерфейсу.

Популярність бібліотеки React має ряд причин. Вона компактна і має високу продуктивність, особливо при роботі з швидкоплинними даними. Завдяки компонентній структурі, React заохочує до написання модульного коду.

React Native - це та ж React, але для мобільних платформ. У неї є ряд відмінностей. Наприклад, замість тега div використовується компонент View, а замість тега img - Image.

React Native був обраний для розробки проекту завдяки багатьом перевагам.

Додатки на React Native пишуться на мові JavaScript - одній з найпопулярніших мов програмування. Розробник пише основну частину коду на Javascript - спільною мовою для всіх платформ, а цей код взаємодіє з нативними компонентами операційних систем. В результаті ми отримуємо мобільні додатки, що працюють на всіх існуючих платформах (iOS, Android, Universal Windows Platform).

Цей фреймворк достатньо простий та зрозумілий для роботи. При створенні програми під Android можна працювати і в Linux, і в Windows. Але при розробці програми під iOS передбачається, що розробник працює в MacOS X.

Кросплатформеність і простота розробки зменшують час, необхідний для реалізації проекту (якщо порівнювати з нативною розробкою). Додатково на терміни позитивно впливає підтримка з боку розвиненої спільноти розробників React: у відкритому доступі є велика кількість плагінів (модулів), які можна використовувати в React Native. Їх застосування також спрощує роботу розробника.

Додатки, розроблені на React Native, схожі із нативними судячи з поведінки і зовнішнього вигляду. Це реальні мобільні додатки, і вони відповідають очікуванням користувача, який звик використовувати Android або iOS. У певному сенсі React Native займає свою нішу в сфері мобільної розробки: він ідеально підходить для тих випадків, коли потрібна швидкість нативних додатків, але не потрібна їх складність.

React Native забезпечує ще одну серйозну перевагу: оновлення встановлюються в додатках автоматично, і не потрібно виконувати стандартну ручну процедуру установки через App Store (для iOS) або Play Store (для Android). Можливість автоматичного оновлення всіх додатків виключає ймовірність проблем з більш ранніми версіями.

Є також і ряд недоліків. Наприклад, React Native не створений для 3D-ігор, погано працює із складними анімаціями, але це не має значення для проекту “Shake&Go”.

Виходячи з названих переваг фреймворку, можна зробити висновок, що він чудово підходить для розробки мобільного додатку та економить час. Саме тому вибір зупинився на ньому.

2.1.3 Вибір бази даних

База даних - це організована структура, яка призначена для зберігання, зміни та обробки взаємозалежної інформації, переважно великих обсягів.

Об'єднання великої кількості даних в єдину базу дає змогу для формування безлічі варіації групування інформації — особисті дані клієнта, історія замовлень, каталог товарів та будь-що інше.

Розрізняють бази даних реляційні та нереляційні.

Реляційна база даних - це сукупність взаємопов'язаних таблиць, кожна з яких містить інформацію про об'єкти певного типу. Рядок таблиці містить дані про один об'єкт (наприклад, товар, клієнт), а стовпці таблиці описують різні характеристики цих об'єктів - атрибутів (наприклад, найменування, код товару, відомості про клієнта). Записи, тобто рядки таблиці, мають однакову структуру - вони складаються з полів, що зберігають атрибути об'єкта. Кожне поле, тобто стовпець, описує тільки одну характеристику об'єкта і має строго певний тип даних. Всі записи мають одні і ті ж поля, тільки в них відображаються різні інформаційні властивості об'єкта.

У реляційній базі даних кожна таблиця повинна мати первинний ключ - поле або комбінацію полів, які єдиним чином ідентифікують кожен рядок таблиці. Якщо ключ складається з кількох полів, він називається складеним. Ключ повинен бути унікальним і однозначно визначати запис. За значенням ключа можна відшукати єдиний запис. Ключі служать також для впорядкування інформації в БД.

Таблиці реляційної БД повинні відповідати вимогам нормалізації відносин. Нормалізація відносин - це формальний апарат обмежень на формування таблиць, який дозволяє усунути дублювання, забезпечує несуперечність збережених в базі даних, зменшує трудовитрати на ведення бази даних.

Над реляційними таблицями можливі наступні операції:

- Об'єднання таблиць з однаковою структурою.
- Перетин таблиць з однаковою структурою.
- Віднімання таблиць з однаковою структурою.
- Вибірка (горизонтальне підмножина).
- Проекція (вертикальне підмножина).

Реляційні таблиці можуть бути пов'язані одна з одною, отже, дані можуть вилучатись одночасно з декількох таблиць. Таблиці зв'язуються між собою для того, щоб в кінцевому рахунку зменшити обсяг БД. Зв'язок кожної пари таблиць забезпечується при наявності в них однакових стовпців.

Існують наступні типи інформаційних зв'язків:

- один до одного;
- один до багатьох;
- багато до банатьох.

Зв'язок один-до-одного передбачає, що одному атрибуту першої таблиці відповідає тільки один атрибут другий таблиці і навпаки.

Зв'язок один-до-багатьох передбачає, що одному атрибуту першої таблиці відповідає декілька атрибутів другої таблиці.

Зв'язок багато-до-багатьох передбачає, що одному атрибуту першої таблиці відповідає декілька атрибутів другої таблиці і навпаки.

Нереляційна база даних - це база даних, що не включає модель таблиць і ключів. Для таких баз даних потрібні методи і процеси маніпулювання даними, призначені для вирішення проблем великих даних, з якими стикаються великі

компанії. Найпопулярніша нова нереляційна база даних називається NoSQL (не тільки SQL).

Більшість нереляційних баз даних включені в такі веб-сайти, як Google, Yahoo!, Amazon і Facebook. Ці веб-сайти представляють безліч нових додатків кожен день з мільйонами користувачів, тому вони не зможуть впоратися з великими стрибками трафіку за допомогою існуючих рішень. Оскільки СУБД не можуть впоратися з цією проблемою, вони переключилися на новий тип СУБД, який здатний обробляти дані веб-масштабу нереляційним способом.

Цікавим аспектом нереляційної бази даних, такий як NoSQL, є масштабованість. NoSQL використовує систему BASE. Нереляційні бази даних відмовляються від табличної форми рядків і стовпців, які реляційні бази даних використовують на користь спеціалізованих середовищ для зберігання даних, до яких можуть звертатися спеціальні API-інтерфейси запитів. Сталість є важливим елементом в цих базах даних. Щоб забезпечити високу пропускну здатність величезних обсягів даних, найкращим варіантом для продуктивності є «в пам'яті», а не читання і запис з дисків.

Для дипломного проєкту було обрано базу даних нереляційного типу - MongoDB. На відміну від реляційних баз даних MongoDB пропонує документо-орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати. Це допоможе зекономити час на розробку архітектури бази даних, що дозволить приділити більше уваги розробці самого додатку.

2.1.3 Вибір фреймворку для реалізації сервера

Для написання серверу на мові JavaScript необхідно встановити Node.js.

Node.js представляє середу виконання коду на JavaScript, яка побудована на основі движка JavaScript Chrome V8, який дозволяє транслювати виклики на мові

JavaScript в машинний код. Node.js перш за все призначена для створення серверних додатків на мові JavaScript.

Для створення API у проєкті використовується Express.js. Це фреймворк, який дозволяє з легкістю описувати контролери та валідацію не витрачаючи часу на реалізацію всього з нуля.

Для підключення до бази даних MongoDB було використано Mongoose - ORM бібліотека для JS.

2.1.4 Вибір засобів написання, компіляції та розгортання

Для написання, компіляції та розгортання додатку будуть використані наступні засоби:

- IDE WebStorm;
- Metro Bundler;
- App Store, Play Store;
- Docker (для розгортання серверу).

WebStorm - середовище розробки для JavaScript, використовується як для frontend'a, так і для створення додатків на Node.js. Цей інструмент розроблено компанією JetBrains і він є платним.

Його головною перевагою є зручний і розумний редактор JavaScript, HTML і CSS, який підтримує також і інші мови, наприклад TypeScript, CoffeeScript, Dart, Less, Sass і Stylus і фреймворки, наприклад, Angular, React і Meteor.

WebStorm робить розробку проекту простою і зручною, забезпечуючи підсвічування і автодоповнення коду, його аналіз по ходу редагування, швидку навігацію і рефакторинг. Він має потужні інструменти налагодження та інтеграції з системами управління версіями (Git, GitHub, Subversion, Perforce, Mercurial, CVS), розуміє структуру проекту і код, відстежує помилки за допомогою систем ESLint, JSHint, JSLint, TSLint, Stylelint і пропонує їх рішення. Вбудовані в IDE інструменти

для тестування і роботи з проектом допомагають в розробці і роблять її зручніше і продуктивніше.

Додаток React Native - це компільований додаток, який запускає Javascript-код. Кожного разу, коли будується та запускається проект React Native, запускається програма яка компілює цей код. Ця програма має назву Metro Bundler.

Найважливішою особливістю Metro є компіляція Javascript-коду у режимі реального часу. Це досягається завдяки іншій розробці Facebook - Watchman. Watchman спостерігає реальні зміни всередині файлів та змушує Metro скомпілювати Javascript-код тільки тоді, коли це дійсно необхідно, це зменшує навантаження на пристрій, на якому розробляється програмне забезпечення та дозволяє швидко відобразити зміни у коді. Подібна функція є інтегрованою у бібліотеку Flutter, яка також є інструментом крос-платформи для створення мобільних додатків.

Поєднує весь код Javascript в один файл і переводить будь-який код Javascript, який пристрій не зрозуміє (наприклад, JSX або дещо новіший синтаксис JS), у зрозумілий.

Перетворює медіа-файли (наприклад, файли PNG) в об'єкти, які можуть відображатися компонентом Image.

Ця програма-пакувальник використовується бібліотекою React, і виконує усі свої обов’язки за лічені секунди. Саме тому для компіляцію обрано саме Metro Bundler.

Розгортання додатків - це механізм їх розподілу для поширення та встановлення на інших пристроях. Додаток “Shake&Go” буде розгортатися у App Store для iOS та у Play Store для Android.

2.2 Проектування архітектури додатку

Архітектура програмного забезпечення - сукупність найважливіших рішень про організацію програмної системи. Архітектура включає:

- вибір структурних елементів і їх інтерфейсів, за допомогою яких складена система, а також їх поведінки в рамках співпраці структурних елементів;
- з'єднання обраних елементів структури і поведінки у більш крупні системи;
- архітектурний стиль, який направляє всю організацію - всі елементи, їх інтерфейси, їх співпраця і їх з'єднання.

Додаток буде розроблено на основі архітектури «клієнт-сервер» (див. рис. 2.1), котра є одним із архітектурних шаблонів програмного забезпечення та має найпоширенішу концепцію у створенні розподілених мережевих застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти[2]:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

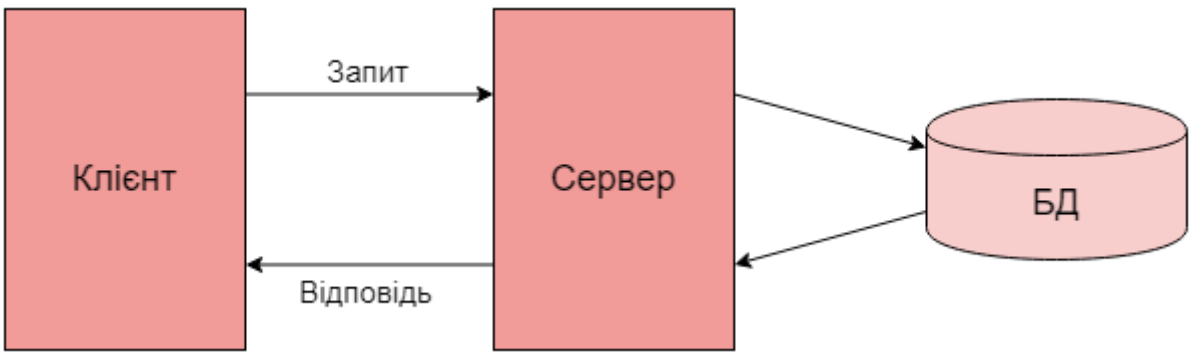

Рис. 2.1. Архітектура “клієнт-сервер”

Як сервери так і клієнти функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого

боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів. [2]

Додаток “Shake&Go” буде взаємодіяти із сервером на основі REST API принципу.

REST (RESTful) - це загальні принципи організації взаємодії додатка або сайту з сервером за допомогою протоколу HTTP. Особливість REST в тому, що сервер не запам'ятовує стан користувача між запитами - в кожному запиті передається інформація, що ідентифікує користувача (наприклад, token, отриманий через OAuth-авторизацію) і всі параметри, необхідні для виконання операції.

Вся взаємодія з сервером зводиться до 4 операцій:

1. отримання даних з сервера (зазвичай у форматі JSON, або XML);
2. додавання нових даних на сервер;
3. модифікація існуючих даних на сервері;
4. видалення даних на сервері.

Операція отримання даних не може приводити до зміни стану сервера. Для кожного типу операції використовується свій метод HTTP-запиту:

1. отримання - GET
2. додавання - POST
3. модифікація - PUT
4. видалення - DELETE

За основу додатку береться фреймворк React Native, який було описано у попередніх пунктах, версії 0.62.3 (остання на момент розробки додатку), при чому з можливістю оновлення до останньої актуальної версії у будь-який момент. Оновлення React Native дозволяють виправити помилки в роботі фреймворку, оптимізувати функціонування додатку та створює нові можливості для розробників.

На операційній системі iOS використовується JavaScript-движок JSC (JavaScript Core), який вбудований в операційну систему завдяки Safari WebKit. Завдяки цьому JSC на iOS працює набагато швидше і стабільніше ніж на Android. При чому розмір додатку менше приблизно на 8 Мб.

У Android немає вбудованого движка JavaScript, тому усі додатки, які написані на фреймворку React Native мають в собі заархівований JSC-движок. Через це додатки на Android працюють повільніше та займають більше оперативної та фізичної пам’яті.

Для того, щоб виправити цей недолік, Facebook розробили власний JavaScript-движок, який має назву Hermes. Він потребує значно менше оперативної пам’яті, збільшує швидкість роботи та прискорює запуск додатку.

На рисунку 2.2 зображено роботу додатку на JSC-движку. Babel інтерпретує нову версію JavaScript-коду у стару та мініфікує її. Далі, інсталюється та запускається додаток. Парсинг та компіляція JavaScript-коду відбувається під час роботи додатку.

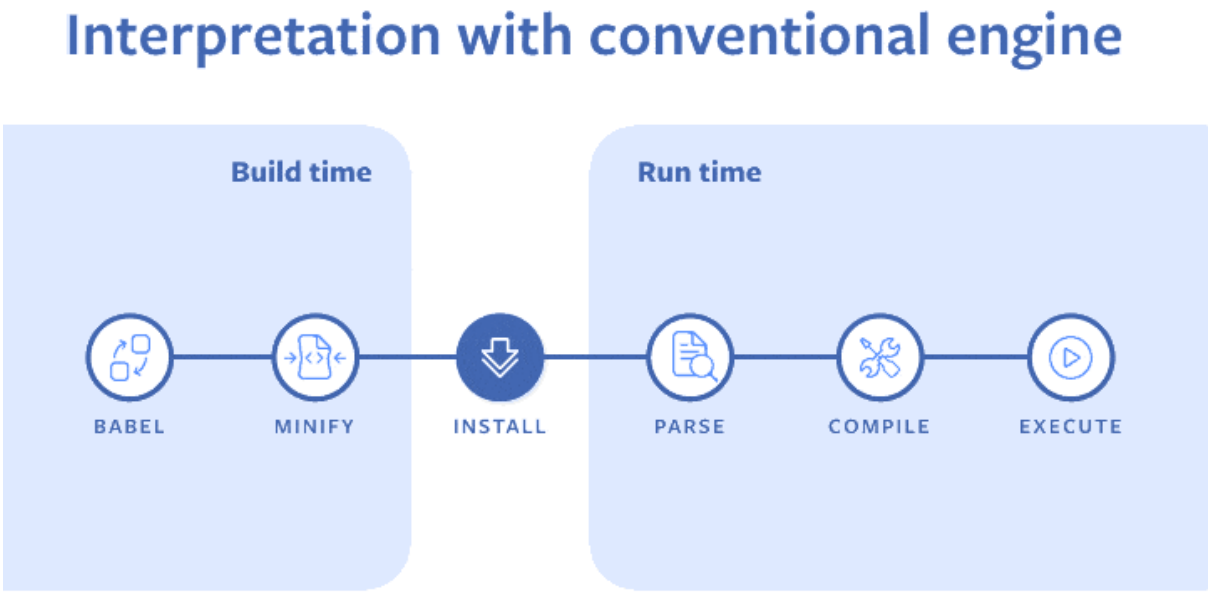

Рис. 2.2. Робота додатку на JSC-движку [3]

У випадку із Hermes-движком парсинг та компіляція JavaScript-коду відбувається під час зборки додатку, а виконується лише під час роботи (див. рис 2.3).

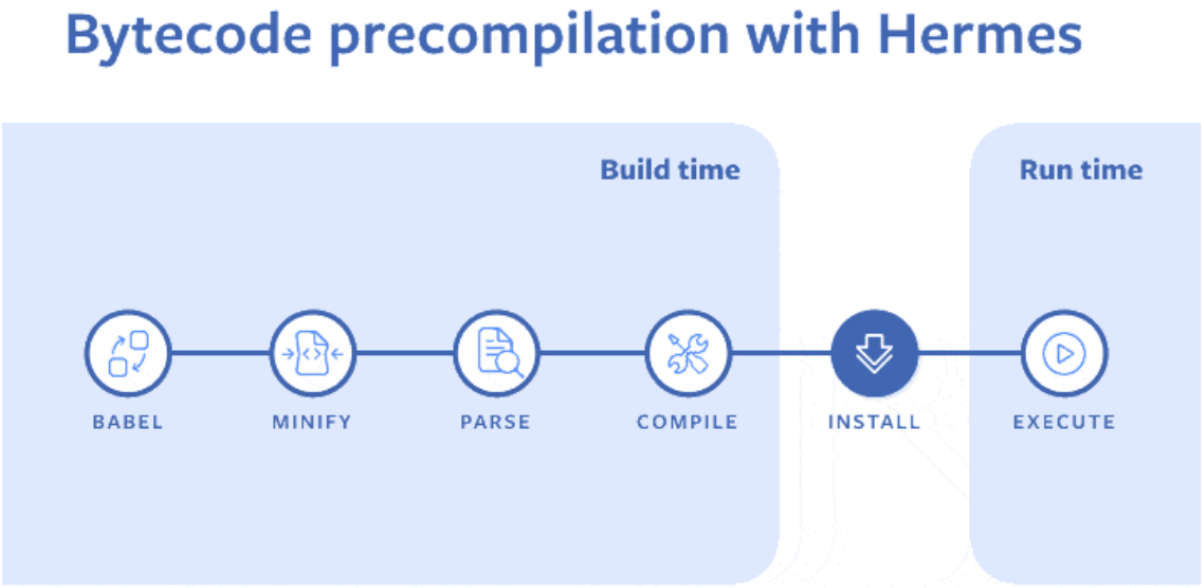

Рис. 2.3. Робота додатку на Hermes-движку [3]

Велику частину розробки додатку на React Native складають open-source бібліотеки, які створюються та підтримуються незалежними розробниками. Для того, щоб знайти необхідну для проекту бібліотеку, можна скористатися такими ресурсами:

- https://reactnative.directory/
- https://js.coach/
- https://github.com/jondot/awesome-react-native

Побудова React Native додатку починається з вибору необхідної бібліотеки навігації. Під час розробки додатку “DigitalSheva” розглядалось дві найбільш популярні бібліотеки: React Navigation 5 та React Native Navigation. Проаналізувавши роботу цих бібліотек, для проекту була обрана React Navigation 5 через ряд своїх переваг.

React Navigation не включає в себе будь-який нативний код у самій бібліотеці, але використовується багато нативних бібліотек для реалізації анімації та жестів, таких як Reanimated, Gesture Handler, Screens тощо. Залежно від типу навігатора, багато компонентів інтерфейсу записуються в JavaScript поверх React Native primitive. Це має багато переваг:

- Прості оновлення OTA (over-the-air updates);
- Спрощує пошук помилок;
- Легко адаптується під вимоги розробника.

Більшість додатків потребують багато додаткових налаштувань навігації, щоб зробити це за допомогою нативного API, вам потрібно буде написати багато нативного коду. Бібліотека React Navigation включає навігатори, написані повністю з JavaScript (наприклад, Stack Navigator) та навігатори, реалізовані поверх стандартних навігацій платформи (наприклад, Native Stack Navigator). Це дозволяє вибирати навігатори, що підходять для вашого додатку, залежно від того, чи хочете ви мати оригінальну поведінку платформи або налаштувати її самостійно.

React Navigation 5 має також і недоліки, які варто описати, проте вони незначні у порівнянні із явними перевагами, описаними вище.

Оновлення бібліотеки можуть вимагати значних змін у коді уже створених додатків.

Багато навігаторів не використовують безпосередньо нативні навігаційні API на iOS і Android; вони використовують частини найнижчого рівня, а потім повторно створюють деяку підмножину API зверху. Це було створено для того, щоб розробники могли легко змінювати навігацію під свої потреби.

Якщо потрібна точна поведінка платформи, можна вибрати навігатори, які використовують власні примітиви платформи (наприклад, Native Stack Navigator), або використовувати іншу бібліотеку, яка написана поверх API платформи. [5]

Після вибору бібліотеки навігації слід обрати бібліотеку, яка буде контролювати потік даних. Наразі створено багато таких бібліотек, але для розробки даного проекту було обрано Redux.

Redux - бібліотека для JavaScript з відкритим вихідним кодом, призначена для управління станом програми. Вона допомагає писати програми, які ведуть себе послідовно, працюють в різних середовищах (клієнтських, серверних і нативних), і які легко тестувати. Крім того, вона надає відмінні можливості для розробників, такі як редагування коду в поєднанні з тимчасовим відладчиком коду.

Розглянемо принцип роботи Redux (див. рис. 2.2).

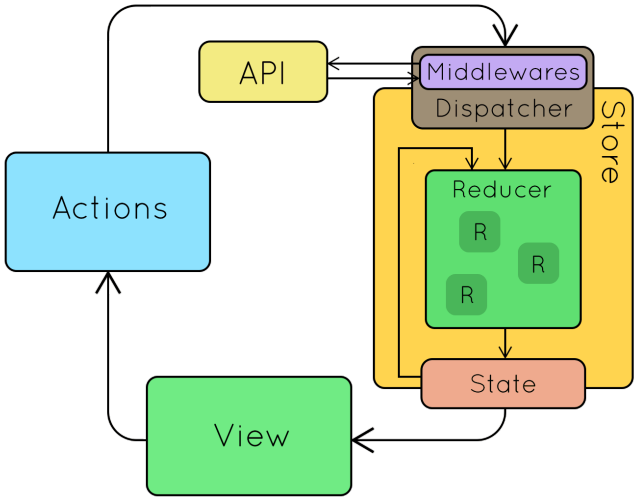

Рис. 2.2. Алгоритм потоку даних, що забезпечується бібліотекою Redux [6]

Початковий стан - вигляд сховища одразу після запуску додатку.

З Redux кожна дія може змінити початковий стан - це означає, що все, що робить користувач, може змінити вигляд додатку.

Кожен раз, коли відбувається дія (action), викликається відправка (dispatch).

Reducer приймає аргументи: стан, встановлений як початковий стан за замовчуванням (Initial state), і дія (Action). Ця дія допомагає редюсеру визначити, що необхідно зробити зі станом. Далі, за необхідністю, викликається Middleware - виклик API.

Маючи список дій (Actions) і редюсер (Reducers), який може обробляти наші дії, потрібно мати сховище (Store). Це місце, де зберігається стан. Для того, щоб створити його, потрібен редюсер і початковий стан.

Для реалізації функціоналу повідомлень буде використано технологію WebSocket.

WebSocket - протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між клієнтом і сервером в режимі реального часу.

2.3 Проектування інтерфейсу

Завдяки вбудованій у фреймворк React Native системи для кросплатформної верстки Yoga Layout Engine розробка інтерфейсу під Android та iOS не відрізняється і необхідності писати код двічі не буде.

Інтерфейс додатку повинен бути інтуїтивно зрозумілий та зручний, відповідати усім необхідним функціям.

При першому відкриванні додатку користувач бачить екран авторизації. Авторизація відбувається із використанням мобільного номеру та коду. Передбачається, що в подальшому буде додано функцію бронювання, тому авторизація за номером телефону є необхідністю.

Після успішної авторизації користувач одразу потрапляє на сторінку із пропозиціями (див. рис 2.3). На ньому зображена картка з однією, або декількома фотографіями, що знаходяться в каруселі та основною інформацією про заклад. До основної інформації відносяться назва, адреса, відстань від поточної локації, години роботи, рівень цін (низький, середній, високий). Також на цьому екрані є дві кнопки: обране та фільтр.

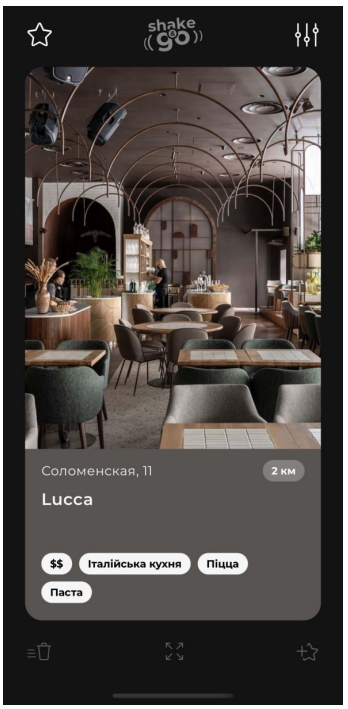

Рис. 2.3. Екран рекомендацій

У розділі “Обране” будуть показані ті заклади, які користувач самостійно додав. Вони будуть мати вигляд карток меншого розміру, ніж на головному екрані. (див. рис 2.4) Для додавання закладу до “обраних” є два варіанти:

- Зробити свайп вправо (виконавши свайп вліво користувач дає зрозуміти, що цей заклад його не цікавить в поточний момент);
- Натиснути на кнопку “Додати до обраних” у відкритій картці закладу.

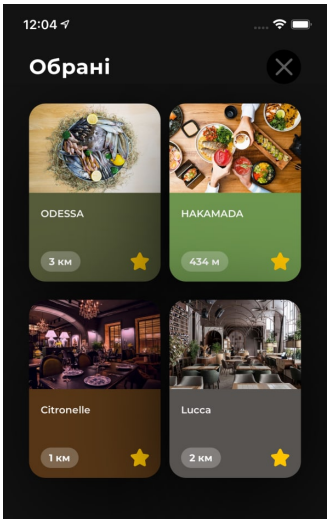

Рис. 2.4. Обрані заклади

Розділ “Фільтр” дозволить користувачу вказати свої вподобання та побажання, щодо списку запропонованих закладів. (див. рис 2.5) Таким чином вірогідність того, що користувач знайде цікавий для нього заклад і відвідає його, буде більшою.

На основі фільтрів та обраних закладів буде оснований інтелектуальний підбір рекомендацій.

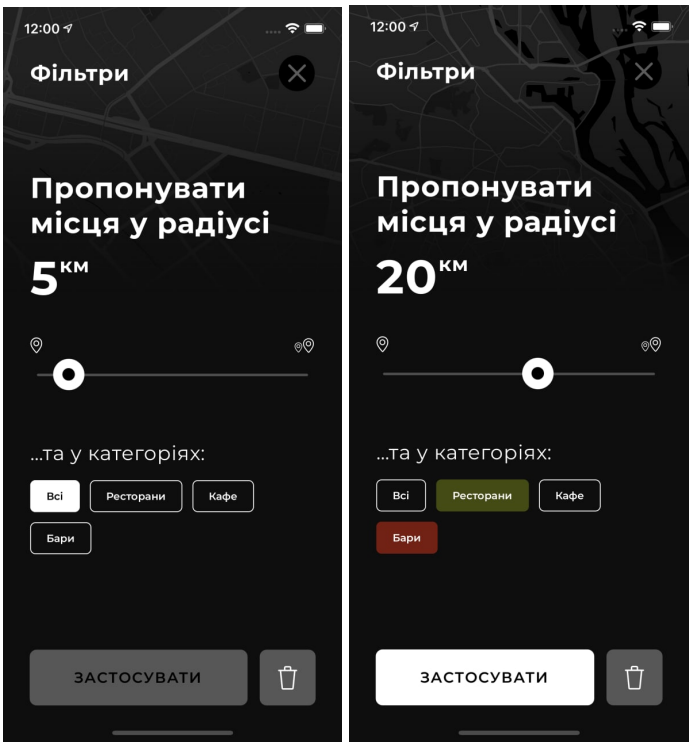

Рис. 2.5. Налаштування фільтрів

Як було зазначено раніше, на головному екрані користувач може змахнути картку вліво або вправо щоб відкинути картку або додати її до обраних, відповідно. Якщо на головному екрані користувач натисне на картку закладу він потрапить до більш детальної інформації (див. рис 2.6). Тут буде короткий опис, кнопки з можливістю перейти на сайт закладу та поділитися. Також, натиснувши на адресу користувач переміститься у додаток Google Maps і отримає маршрут до закладу.

Рис. 2.6. Відкрита картка закладу

Слід зазначити, що власники або представники закладів матимуть окремий веб додаток для додавання або змінення своєї інформації і виступатимуть адміністраторами своєї сторінки. Цей додаток також матиме простий та зрозумілий інтерфейс. Основною задачею адміністраторів буде заповнення полів необхідною інформацією.

Адміністратори будуть додавати наступні дані (рис. 2.7):

- назва закладу;
- більш детальна інформація (опис);
- адреса;
- рівень цін;
- посилання на сайт закладу;
- фотографії;
- основний колір картки;
- категорія закладу;
- теги.

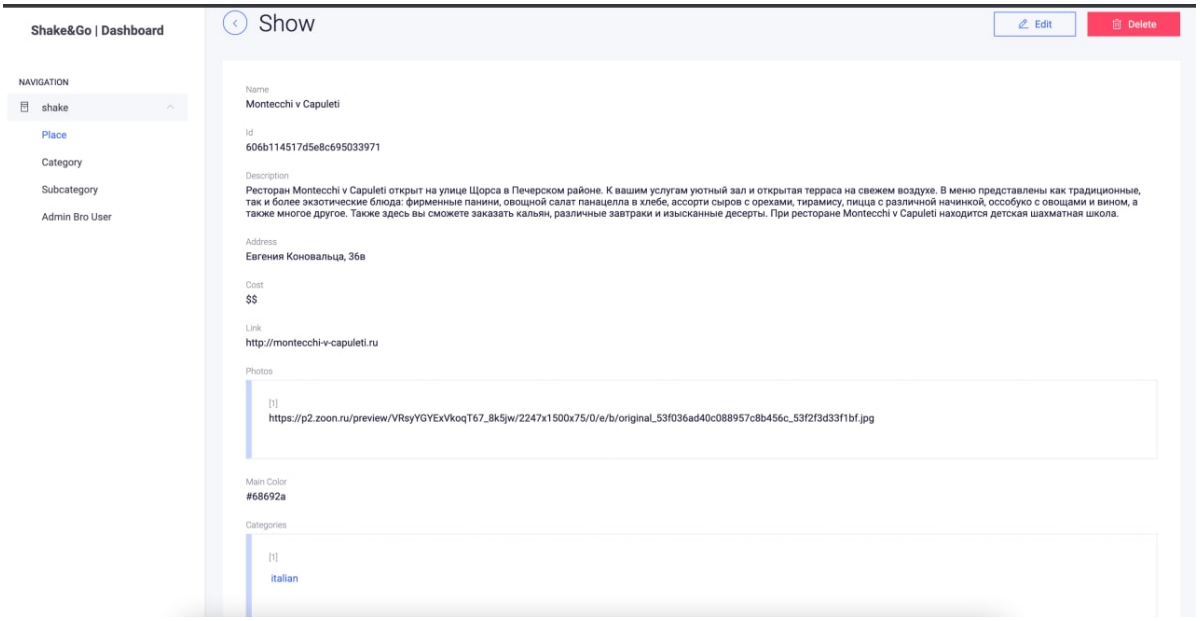

Рис. 2.7. Панель адміністратора

ВИСНОВКИ ПО РОЗДІЛУ 2

Під час роботи над даним розділом було проаналізовано та обрано технології, які будуть використані для реалізації програмного додатку. Усі обрані засоби дозволяють створити додаток таким, яким його заплановано.

Для написання додатку та сервера буде використана мова JavaScript із фреймворками React Native (для додатку) та Node.js (для сервера).

Для запису, обробки, зміни даних обрано нереляційну базу даних MongoDB.

Для написання, компіляції та розгортання додатку будуть використані наступні засоби:

- IDE WebStorm;
- Metro Bundler;
- App Store, Play Store;
- Docker (для розгортання серверу).

3. ТЕСТУВАННЯ ТА РОЗГОРТАННЯ МОБІЛЬНОГО ДОДАТКУ ТА СЕРВЕРУ

3.1 Тестування мобільного додатку

Невід’ємною частиною будь-якої реалізації програмного продукту є процес тестування або так званої перевірки роботи програми згідно вимог, які були зазначенні при початку реалізації продукту. Під час перевірки програмного продукту зазвичай виявляються дефекти системи, при яких система може не працювати, блокуватися або працювати не вірно. В разі виявлення дефекту або помилки, розробник має виправити її.

Тестування за своєю специфікою поділяється на два види: ручне тестування та автоматизоване. Під час ручного тестування всі операції виконуються вручну, тобто розробник самостійно перевіряє і оцінює, як працює та чи інша функція. Автоматизоване тестування дає змогу перевірити код реалізації функцій, за допомогою якого існує можливість протестувати фунціонал, і перевірити, чи повертається потрібний результат. Кожен з цих видів містить в собі вже свої види тестування.

Для додатку “Shake&Go” було проведено тестування методами, описаними нижче.

1. Інсталяційне тестування. Під час проведення даного тестування додаток було успішно встановлено на смартфони з такими операційними системами:

- Android 5.0;
- Android 9.0;
- iOS 9;
- iOS 13.

2. Тестування продуктивності. Під час проведення даного тестування смартфон продовжував успішно працювати з додатком при таких умовах:

- низькому рівні заряду акумулятора;
- поганому покриттю мережею;
- низькій кількості доступної пам'яті;
- одночасному доступі до сервера кількома користувачами.

3. Навантажувальне тестування:

- на смартфонах запускалося декілька сторонніх додатків, які використовували підключення до мережі інтернет;

4. Тестування перериванням:

- Отримання повідомлень під час роботи з додатком;
- Відповідь на дзвінок під час роботи з додатком;
- Підключення та відключення USB кабелю.

5. Також проведено тестування при різних типах підключення до мережі інтернет (2g, 3g, 4g, WiFi), яке успішно завершилося у всіх випадках.

6. End-to-End тестування. Було написано декілька тест-кейсів, які задовільняють вимоги додатку. Усі тест-кейси запускалися на усіх етапах розробки додатку.

Всі тести додаток пройшов успішно, додаток готовий для використання та експлуатації, фатальних багів виявлено не було.

3.2 Тестування серверу

Для серверу додатку “Shake&Go” було проведено тестування методами, описаними нижче.

1. Навантажувальне тестування: сервер було завантажено великою кількістю HTTP-запитів за короткий час.

2. Unit тестування. Було написано багато тест-кейсів, які задовільняють вимоги серверу. Усі тест-кейси запускалися на усіх етапах розробки додатку.

Всі тести додаток пройшов успішно, додаток готовий для використання та експлуатації, фатальних багів виявлено не було.

3.3 Розгортання мобільного додатку

Додаток розгортається стандартними методами, які залежать від операційної системи.

Для розгортання у AppStore (iOS) спершу необхідно придбати акаунт розробника, який коштує $100/рік. У macOS є системний додаток для розробки під будь-яку операційну систему Apple, він має назву XCode. Далі, за допомогою XCode додаток архівується та відправляється до серверів Apple, де, перш ніж продукт стане доступним для завантаження користувачами, відпбувається модерація. Модерація може тривати від трьох робочих днів та більше.

Розгортання додатку для Android відбувається подібним способом. Купується акаунт розробника у PlayStore, проте він коштує $25 і покупка здійснюється одноразово. Архівування додатку відбувається за допомогою Android Studio, але потрібно завантажувати додаток на сайт вручну. Перед публікуванням додатку відбувається перевірка.

Для того, щоб уникнути очікування модерації і зробити процес виправлення помилок та додавання нового функціоналу швидшим додаток “Shake&Go” буде використовувати OTA оновлення за допомогою сервісу Microsoft Code Push.

Code Push - це хмарний сервіс Microsoft App Center, який дозволяє розробникам React Native розгортати оновлення мобільних додатків безпосередньо на пристроях своїх користувачів. Він працює як центральний репозиторій, в якому розробники можуть публікувати певні оновлення (наприклад, JS, HTML, CSS і зміни зображень), а додатки можуть запитувати оновлення (використовуючи надані клієнтські SDK). Це дозволяє мати більш детерміновану і пряму модель взаємодії з кінцевими користувачами, усуваючи помилки і додаючи невеликі функції, які не вимагають перебудови двійкового файлу і поширення його через будь-які загальнодоступні магазини додатків.

3.4 Розгортання серверу

Для розгортання серверу було обрано технологію контейнеризації додатку. Засобом реалізації технології було обрано Docker - ПЗ для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації, контейнеризатор додатків.

Для зберігання створеного контейнеру використовується Github Packages. Завдяки обраним технологіям розгортання додатку буде можливим на будь-якому хмарному сервері у найкоротший термін.

На хмарному сервері повинна бути встановлена ОС Ubuntu Server останньої версії та Docker Engine.

ВИСНОВКИ ПО РОЗДІЛУ 3

У даному розділі було виконано та описано методи тестування та розгортання додатку і сервера.

Сервіс виконує наступні функції:

- рекомендації закладів для користувача;
- додавання закладів у розділ “Обране”;
- пошук закладів по фільтрам;
- отримання інформації про запропонований заклад;
- прокладання маршруту до закладу через Google Maps;
- додавання та редагування інформації про заклад власниками.

4. ОЦІНКА ВАРТОСТІ РОЗГОРТАННЯ ТА ПІДТРИМКИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

Для того, щоб зробити оцінку вартості розгортання та впровадження додатку, необхідно оцінити вартість усіх процесів розробки та підтримки його функціонування, а саме:

- розробка додатку;
- розробка серверу;
- вартість хостингу;
- реклама.

Враховуючи те, що розробка додатку та серверу відбувалися в рамках написання дипломного проєкту, необхідності враховувати вартість цієї роботи немає. Подальша підтримка та виправлення помилок при розробці також не враховуватимуться.

Для розгортання серверу та бази даних було обрано провайдера хмарних послуг - Hetzner. Серед усіх можливих конфігурацій серверу було обрано сервер CX41 (рис. 4.1). Він повністю задовольняє стартові потреби для підтримки безперебійної роботи серверу додатку.

Отже вартість використання серверу буде складати 18,92€ в місяць, що становить близько 634 грн.

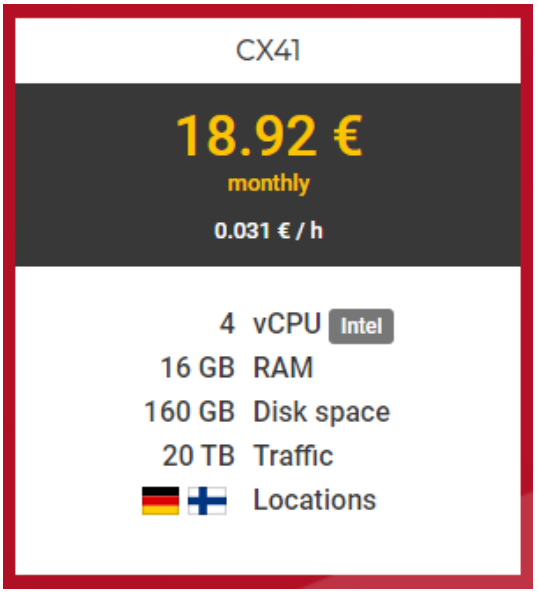

Рис. 4.1. Вартість послуг використання хмарного серверу

Вартість вартість реклами додатку оцінені не були, оскільки залежать від кон'юнктури ринку на момент розгортання і повинні бути оцінені додатково у момент прийняття рішення про розгортання.

ВИСНОВОК

Під час створення мобільного додатку “Shake&Go” у дипломній роботі було пройдено усі етапи, а саме:

1. На етапі аналізу предметної області було досліджено предметну область – пошук місць та закладів для проведення дозвілля. Проведено огляд та аналіз кількох систем аналогів, що реалізують схожі функції предметної області.

2. На етапі проектування додатку розроблено його архітектуру, проаналізовано та обрано методи і засоби реалізації. Вивчено API серверної частини та способи інтеграції. Створено дизайн інтерфейсу користувача.

3. На етапі програмної розробки, з використанням фреймворку React Native реалізовано функції системи, які написані на мові програмування JavaScript та допоміжних бібліотек з імплементацією запитів до серверу. Розроблено інтерфейс додатку. За допомогою Node.js фреймворку було розроблено сервер.

4. На етапі тестування було проведено тестування безпеки та функціональне тестування. Додаток дає можливість швидкого та зручного доступу до рекомендацій закладів, пошук за фільтрами, інтересами та з використанням інтелектуального підбору. У дипломній роботі проект створювався на основі закладів міста Києва. Після успішного релізу є перспективи для розвитку, масштабування та додавання нових функцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація JavaScript [Електронний ресурс] – Режим доступу: https://developer.mozilla.org

2. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: учебник / Вендров А.М. – М.: Финансы и статистика, 2006. – 544 с.

3. Офіційна документація Hermes JavaScript Engine [Електронний ресурс] – Режим доступу: https://engineering.fb.com/android/hermes/

4. Офіційна документація React-Navigation [Електронний ресурс] – Режим доступу: https://reactnavigation.com

5. Офіційна документація Redux [Електронний ресурс] – Режим доступу: https://redux.js.org

6. Спільнота IT-спеціалістів [Електронний ресурс] – Режим доступу: https://habr.com/

7. Офіційний сайт Hetzner [Електронний ресурс] – Режим доступу: https://www.hetzner.com/

8. Сайт про програмування [Електронний ресурс] – Режим доступу: https://metanit.com/

9. Програмне забезпечення - Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: http://en.wikipedia.org/wiki/ Програмне_забезпечення

10. Бондарчук Д. В. алгоритмы интеллектуального поиска на основе метода категориальных векторов

11. Д.Е. Королева, М.В. Филиппов Анализ алгоритмов обучения коллаборативных рекомендательных систем - Москва. // 2013 р. – [Електронний ресурс]. – Режим доступу: http://engjournal.ru/articles/816/816.pdf

ДОДАТОК А

Код програми сервера

Створення схеми місця проведення дозвілля:

```
import mongoose from "mongoose";
import "mongoose-type-url";

const PlaceSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    description: {
      type: String,
      required: false,
    },
    address: {
      type: String,
      required: false,
    },
    cost: {
      type: String,
      required: false,
    },
    link: {
      type: mongoose.SchemaTypes.Url,
      required: false,
    },
    photos: [
      {
        type: mongoose.SchemaTypes.Url,
      },
    ],
    mainColor: String,
    categories: [
      {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: "Subcategory",
      },
    ],
    location: {
      type: {
        type: String,
        enum: ["Point"],
        default: "Point",
```

```
        required: true,
      },
      coordinates: {
        type: [Number],
        required: true,
        default: [0, 0],
      },
    },
    tags: [
      {
        type: String,
        required: false,
      },
    ],
    ownerId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "AdminBroUser",
    },
  },
  { versionKey: false }
);

PlaceSchema.index({ location: "2dsphere" });

export default mongoose.model("Place", PlaceSchema);
```

Створення схеми обраних місць:

```
import mongoose from "mongoose";

const LikeSchema = new mongoose.Schema(
  {
    user: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "User",
    },
    placeId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "Place",
    },
  },
  { versionKey: false }
);

export default mongoose.model("Like", LikeSchema);
```

Створення схеми категорій місця:

```
import mongoose from "mongoose";

const CategorySchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    ua: {
      type: String,
      required: false,
    },
    priority: {
      type: Number,
      required: false,
    },
    color: {
      type: String,
      required: false,
    },
  },
  { versionKey: false }
);

CategorySchema.index({ name: 1 }, { unique: true });

export default mongoose.model("Category", CategorySchema);
```

Створення схеми підкатегорії місця:

```
import mongoose from "mongoose";

const SubcategorySchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    belongToCategory: {
      type: mongoose.Schema.Types.ObjectId,
      required: false,
      ref: "Category",
    },
  },
  { versionKey: false }
);

SubcategorySchema.index({ name: 1 }, { unique: true });
```

```
export default mongoose.model("Subcategory", SubcategorySchema);
```

Створення схеми юзера додатку:

```
import mongoose from "mongoose";

const mongoosePaginate = require("mongoose-paginate-v2");

const schema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
    },
    name: {
      type: String,
      required: true,
    },
    blocked: {
      type: Boolean,
      default: false,
    },
  },
  { versionKey: false }
);

schema.set("timestamps", true);

schema.plugin(mongoosePaginate);

export default mongoose.model("User", schema);
```

Контролер отримання рандомних місць з фільтрами та рекомендаційною системою:

```
import { Place } from "@models";
import utils from "@utils";
import { matchedData } from "express-validator";

const generatePipeline = ({
  count,
  lat,
  long,
  maxDistance = 3000,
  mostPopular,
  categories,
```

```
}) => {
  let pipeline = [];

  if (lat && long) {
    pipeline.push({
      $geoNear: {
        includeLocs: "location",
        distanceField: "distance",
        near: { type: "Point", coordinates: [lat, long] },
        maxDistance,
        spherical: true,
      },
    });
  }

  pipeline.push(
    {
      $lookup: {
        from: "subcategories",
        localField: "categories",
        foreignField: "_id",
        as: "subcategories",
      },
    },
    {
      $lookup: {
        from: "categories",
        localField: "subcategories.belongToCategory",
        foreignField: "_id",
        as: "categories",
      },
    },
    {
      $addFields: {
        subcategories: "$subcategories.name",
        categories: "$categories.name",
      },
    }
  );

  if (mostPopular) {
    pipeline.push(
      {
        $lookup: {
          from: "likes",
          localField: "_id",
          foreignField: "placeId",
          as: "likes",
        },
      },
      {
```

```
        $set: {
          likes: {
            $size: "$likes",
          },
        },
      }
    );
  }

  if (categories) {
    // @TOD Needs to be rechecked
    const categoriesQuery = categories.split(",");

    pipeline.push({
      $match: {
        categories: { $in: categoriesQuery },
      },
    });
  }

  pipeline.push(
    {
      $sample: { size: count },
    },
    {
      $sort: {
        likes: -1,
      },
    }
  );

  return pipeline;
};

exports.randomize = async (req, res) => {
  try {
    const matchedReq = matchedData(req);

    const places = await Place.aggregate(
      generatePipeline(matchedReq)
    ).collation({ locale: "en", strength: 2 });

    if (places.length) {
      res.json(places);
    } else {
      res.send(utils.buildErrObject(404, "NO_PLACES_AVAILABLE"));
    }
  } catch (error) {
    utils.handleError(res, error);
  }
};
```

Контролер оцінки місця (додавання або прибирання з обраних):

```
import utils from "@middleware/utils";
import { Like } from "@models";
import { matchedData } from "express-validator";

const likeExists = async (user, placeId) => {
  return new Promise((resolve, reject) => {
    Like.findOne(
      {
        user,
        placeId,
      },
      (err, item) => {
        utils.itemAlreadyExists(err, item, reject, "LIKE_ALREADY_EXISTS");
        resolve(false);
      }
    );
  });
};

const removeLike = async (user, placeId) => {
  return new Promise((resolve, reject) => {
    Like.findOneAndRemove(
      {
        user,
        placeId,
      },
      (err, item) => {
        utils.itemNotFound(err, item, reject, "NOT_FOUND");
        resolve(utils.buildSuccObject("DELETED"));
      }
    );
  });
};

const getLikes = async (user) => {
  return new Promise((resolve, reject) => {
    Like.find({ user }, "-_id -user")
      .populate("placeId")
      .exec((err, item) => {
        utils.itemNotFound(err, item.reverse(), reject, "NOT_FOUND");
        resolve(item.map((i) => i.placeId));
      });
  });
};

//

exports.createLike = async (req, res) => {
  try {
```

```
    const matchedReq = matchedData(req);

    await likeExists(req.user._id, matchedReq.placeId);

    res.status(200).json(
      await Like.create({
        user: req.user._id,
        placeId: matchedReq.placeId,
      })
    );
  } catch (error) {
    utils.handleError(res, error);
  }
};

exports.deleteLike = async (req, res) => {
  try {
    const matchedReq = matchedData(req);

    res.status(200).json(await removeLike(req.user._id, matchedReq.placeId));
  } catch (error) {
    utils.handleError(res, error);
  }
};

exports.getLikes = async (req, res) => {
  try {
    res.status(200).json(await getLikes(req.user._id));
  } catch (error) {
    utils.handleError(res, error);
  }
};
```

ДОДАТОК Б

Код програми додатку

Створення сервісу управління станом додатку:

```
import {types} from 'mobx-state-tree';
import {createContext, useContext} from 'react';
import Filter, {FilterState} from '@lib/store/FilterStore';
import Likes, {LikesState} from '@lib/store/LikesStore';
import ListPlace, {ListPlaceState} from '@lib/store/ListPlaceStore';
import Place, {PlaceState} from '@lib/store/PlaceStore';
import User, {UserState} from '@lib/store/UserStore';

const RootStore = types
  .model({
    user: types.maybe(User),
    filter: types.maybe(Filter),
    listPlace: types.maybe(ListPlace),
    likes: types.maybe(Likes),
    place: types.maybe(Place),
  })
  .actions((store) => ({}));

export const rootStore = RootStore.create({
  user: UserState,
  filter: FilterState,
  listPlace: ListPlaceState,
  likes: LikesState,
  place: PlaceState,
});

const RootStoreContext = createContext(null);

export const MSTProvider = RootStoreContext.Provider;

export function useStore() {
  const mstStore = useContext(RootStoreContext);
  if (mstStore === null) {
    throw new Error('Store cannot be null, please add a context provider');
  }
  return mstStore;
}
```

Управління фільтрами:

```
import {getParent, types} from 'mobx-state-tree';
import {apiCall, Endpoints, Methods} from '@api/index';
import asyncAction from '@lib/store/helpers/asyncAction';
import {initialCamera, MAX_RADIUS} from '@config/mapConfig';

export const FilterState = {
  filters: {
    maxDistance: MAX_RADIUS * 1000,
    latitude: initialCamera.center.latitude,
    longitude: initialCamera.center.longitude,
    categories: '',
  },
  categoriesList: [],
};

function getFilterCategories() {
  return async function getFilterCategoriesFlow(store) {
    const {data} = await apiCall({
      endpoint: Endpoints.CATEGORIES,
      method: Methods.GET,
    });

    store.setCategoriesList(data);
  };
}

const Filter = types
  .model({
    filters: types.optional(types.frozen()),
    categoriesList: types.array(types.frozen()),
    getFilterCategories: asyncAction(getFilterCategories),
  })
  .actions((store) => {
    return {
      setCategoriesList(list) {
        store.categoriesList = list;
      },
      setFilter(filters) {
        getParent(store).listPlace.reset();

        store.filters = filters;
      },
      resetFilter() {
        store.filters = [];
      },
    };
  });

export default Filter;
```

Управління обраними місцями:

```
import {types} from 'mobx-state-tree';
import {getUniqueId} from 'react-native-device-info';
import {apiCall, Endpoints, Methods} from '@api/index';
import asyncAction from '@lib/store/helpers/asyncAction';

export const LikesState = {
  list: [],
};

function getLikes() {
  return async function getLikesFlow(store) {
    const {data} = await apiCall({
      endpoint: Endpoints.LIKES,
      method: Methods.GET,
      params: {
        user: getUniqueId(),
      },
    });

    store.setLikesList(data);
  };
}

function likePlace({placeId}) {
  return async function likePlaceFlow(store) {
    await apiCall({
      endpoint: Endpoints.LIKES,
      method: Methods.POST,
      data: {
        user: getUniqueId(),
        placeId,
      },
    });
  };
}

function dislikePlace({placeId}) {
  return async function dislikePlaceFlow(store) {
    await apiCall({
      endpoint: Endpoints.LIKES,
      method: Methods.DEL,
      data: {
        user: getUniqueId(),
        placeId,
      },
    });
  };
```

```
}

const Likes = types
  .model({
    list: types.array(types.frozen()),
    getLikes: asyncAction(getLikes),
    likePlace: asyncAction(likePlace),
    dislikePlace: asyncAction(dislikePlace),
  })
  .actions((store) => {
    return {
      setLikesList(data) {
        store.list = data;
      },
    };
  });

export default Likes;
```

Отримання списку місць:

```
import {flow, getParent, types} from 'mobx-state-tree';
import {apiCall, Endpoints, Methods} from '@api/index';
import LastCard from '@lib/models/cards/LastCard';
import asyncAction from '@lib/store/helpers/asyncAction';

export const ListPlaceState = {
  cards: [],
  forceSearch: false,
};

function getPlaces() {
  return async function getPlacesFlow(store) {
    try {
      const {data, error} = await apiCall({
        endpoint: Endpoints.RANDOMIZE,
        method: Methods.GET,
        params: {
          count: 10,
          ...getParent(store).filter.filters,
        },
      });

      store.setCards(data);
    } catch (e) {}
  };
}

const ListPlace = types
```

```
  .model({
    cards: types.array(types.frozen()),
    forceSearch: types.boolean,
    getPlaces: asyncAction(getPlaces),
  })
  .actions((store) => ({
    reset() {
      store.cards = ListPlaceState.cards;
      store.forceSearch = ListPlaceState.forceSearch;
    },

    setCards(data) {
      store.cards = [...data, new LastCard()];
      store.forceSearch = false;
    },

    forceReset() {
      store.cards = ListPlaceState.cards;
      store.forceSearch = true;
    },
  }));

export default ListPlace;
```

Отримання деталей місця:

```
import {types} from 'mobx-state-tree';
import {apiCall, Endpoints, Methods} from '@api/index';
import asyncAction from '@lib/store/helpers/asyncAction';

export const PlaceState = {
  data: null,
};

function getPlace({placeId, onSuccess}) {
  return async function getPlaceFlow() {
    const {data} = await apiCall({
      endpoint: `${Endpoints.PLACE}/${placeId}`,
      method: Methods.GET,
    });

    onSuccess(data);
  };
}
const Place = types.model({
  data: types.maybe(types.frozen()),
  getPlace: asyncAction(getPlace),
});
export default Place;
```

Перелік наявних REST-API ендпоінтів:

```
export default {
  RANDOMIZE: '/randomize',
  LIKES: '/likes',
  CATEGORIES: '/categories',
  PLACE: '/place',
};
```