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

УДК 004.42

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Розробка веб-додатку для сумісного планування і збереження туристичних маршрутів”
Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент

_____/Аліна КОНТОРЕР
(підпис) (розшифровка підпису) (дата)

Науковий керівник

к.ф.-м.н._____/Сергій ПОЛЯКОВ
(посада) (підпис) (розшифровка підпису) (дата)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри

д.т.н.,проф._____/Олексій БИЧКОВ
(підпис) (розшифровка підпису) (дата)

2021

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)
„___”_____ 20___р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Аліні КОНТОРЕР
_____
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи: “Розробка веб-додатку для сумісного планування і збереження туристичних маршрутів” Затверджена наказом вищого навчального закладу від „__”_____20__р. №_____

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до роботи: інтернет-ресурси з питань туризму, використання електронних карт та побудови соціальних веб-застосунків. Технології розробки клієнт-серверних додатків та інструменти побудови UML-діаграм.

4. Зміст пояснювальної записки: Дослідження чинників, які впливають на зручність у плануванні маршрутів, аналіз існуючих систем з подібним функціоналом, можливість користувачів взаємодіяти один з одним в

рамках такої системи, обґрунтування доцільності обраної теми дослідження, виявлення невирішених проблем. Визначення особливостей архітектурного рішення для системи сумісного планування туристичних маршрутів, розробка прототипу та реалізація системи з необхідним функціоналом.

5. Перелік графічного матеріалу:

1. Статистика популярності фреймворків JavaScript (рис. 2.1., с. 29)
2. Діаграма прецедентів (рис. 3.1., с. 43)
3. Структура бази даних додатку (рис. 3.2., с. 49)
4. Схема інкрементної моделі життєвого циклу (рис. 3.3., с. 52)
5. UML-діаграма компонентів (дод. А, с. 59)

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Сергій ПОЛЯКОВ	01.03.2021	05.03.2021
2	Сергій ПОЛЯКОВ	06.03.2021	15.03.2021
3	Сергій ПОЛЯКОВ	16.03.2021	25.05.2021

7. Дата видачі завдання _____

Керівник _____
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____
(підпис) розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1. Дослідження предметної області	01.03.2021	виконано
2. Визначення і аналіз функціональних вимог	16.03.2021	виконано
3. Визначення і аналіз нефункціональних вимог	22.03.2021	виконано
4. Проектування рішення	28.03.2021	виконано
5. Програмна реалізація	12.04.2021	виконано
6. Тестування	16.05.2021	виконано
7. Оформлення і друк пояснювальної записки	.06.2021	виконано

Студент – бакалавр _____

(підпис) (розшифровка підпису)

Керівник роботи

_____

(підпис) (розшифровка підпису)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 65 с., 4 рис., 2 додат., 9 джерел.

Тема: Розробка веб-додатку для сумісного планування і збереження туристичних маршрутів.

Об’єкт дослідження: Об’єктом дослідження є система створення і сумісного редагування туристичних маршрутів.

Мета роботи: Підвищення комфорту туристів при сумісному плануванні маршрутів подорожей, шляхом створення соціальної системи, яка надаватиме можливість будувати та оцінювати туристичні маршрути.

Предмет дослідження: Засоби, методи та технології розробки веб-додатків із соціальною складовою та залученням сторонніх сервісів для візуалізації.

Результати дослідження:

Досліджено способи і методи розробки веб-додатків, проаналізовано сучасні технології і кращі практики. Складено план роботи та реалізовано веб-додаток для сумісного планування і збереження туристичних маршрутів, з використанням обраних і описаних технологій розробки.

Висновок:

В результаті роботи було отримано та впроваджено програму, що забезпечує функціонал сумісного створення, збереження, редагування туристичних маршрутів із використанням візуальної та соціальної складової та сучасних засобів розробки.

АНОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 65 с., 4 рис., 2 доп., 9 источников.

Тема: Разработка веб-приложения для совместного планирования и сохранения туристических маршрутов.

Объект исследования: Объектом исследования является система создания и совместного редактирования туристических маршрутов.

Цель работы: Повышение комфорта туристов при совместном планировании маршрутов путешествий, путем создания социальной системы, которая будет предоставлять возможность строить и оценивать туристические маршруты.

Предмет исследования: Средства, методы и технологии разработки веб-приложений с социальной составляющей и привлечением сторонних сервисов для визуализации.

Результаты исследования: Исследованы способы и методы разработки веб-приложений, проанализированы современные технологии и лучшие практики. Составлен план работы и реализовано веб-приложение для совместного планирования и сохранения туристических маршрутов с использованием избранных и описанных технологий разработки.

Вывод:

В результате работы была разработана и внедрена программа, обеспечивающая функционал совместного создания, хранения, редактирования туристических маршрутов с использованием визуальной и социальной составляющей и современных средств разработки.

ANNOTATION

Graduation Bachelor Work: 65 pp., 4 figs., 2 appendices., 9 sources.

Topic: Development of a web application for collective planning and storing of tourist routes.

Object of research: The object of research is a system of joint creation and editing of tourist routes.

Purpose: To increase the comfort of tourists in the planning of travel routes, by creating a social system that will provide an opportunity to build and evaluate tourist routes.

Subject of research: Tools, methods and technologies for developing web applications with a social component and the involvement of third-party services for visualization.

Results of the research:

The methods and techniques of web application development are researched, modern technologies and best practices are analyzed. A work plan has been drawn up and a web application has been implemented for joint planning and preservation of tourist routes, using selected and described development technologies.

Conclusion:

As a result, a program was developed and implemented that provides the functionality of joint creation, preservation, editing of tourist routes using the visual and social components and modern development technologies.

ЗМІСТ

ВСТУП

В сучасному світі туризм є надзвичайно популярним серед людей будь якого віку, особливо завдяки розвитку всесвітньої павутини. Маючи доступ до Інтернету, людина може дізнатись безліч нової інформації, відкрити для себе нові місця та континенти, надихнутись фотографіями, унікальною культурою і традиціями кожної окремої локації. Все це сприяє бурхливому розвитку туризму і пов’язаних із ним послуг.

Запорукою будь якої успішної подорожі є її планування. Це і визначення місця призначення, покупка білетів, бронювання готелю, хостелу або квартири, складення переліку бажаних до відвідування місць і пам’яток. Всі заплановані переміщення, врешті, складають собою маршрут запланованої подорожі, від першої відвіданої локації, до місця очікування на транспорт, що доставить туриста додому.

Зважаючи на це, швидкими темпами почала розвиватись сфера туристичних послуг в Інтернеті, зросла кількість веб-додатків, які допомагають туристам полегшити той чи інший аспект планування їх подорожей. Такі великі компанії, як TripAdvisor, відвідують різні міста і роблять ревізію закладів, тож турист знає, яке місце точно потрібно відвідати, а яке викреслити зі свого маршруту. Також користувачі самі можуть залишати відгуки і ділитись емоціями від перебування в тому чи іншому закладі.

Airbnb - ще одна компанія, що надає комплексні туристичні та інформаційні послуги, серед яких не лише зручне і безпечне бронювання квартир, а й замовлення так званих “вражень” - заходів, організованих самими місцевими жителями, від дегустації вина і екскурсій околицями до дайвінгу поруч з акулами.

Отже, після аналізу можливостей подібних продуктів можна виявити,

що зв’язок між “туристичним” та “соціальним” дуже тісний, адже спілкування між користувачами, в таку випадку є дуже важливою частиною процесу планування подорожі, тому реалізований застосунок можна назвати “соціальною мережею для туристів”.

Наявність такого додатку, який поєднає соціальну систему туристів та побудову маршрутів і передачу їх між користувачами знаходиться під питанням, адже під час аналізу відомих додатків, які мають солідну базу користувачів, не було виявлено нічого подібного. Такий спосіб поєднання соціальної частини та туризму надасть змогу людям отримати більше емоцій, ніж якби вони будували маршрут своєї подорожі без допомоги однодумців.

Актуальність дослідження:

Актуальність дослідження полягає у збільшенні комфорту для туристів при плануванні їх подорожей самостійно або разом із друзями, а також уникнення неприємних ситуацій під час них за рахунок комунікації між користувачами системи, адже при виїзді за межі своєї країни людина стає більш вразливою та ризик виникнення надзвичайних випадків підвищується.

Об’єкт дослідження:

Об’єктом дослідження є система створення, сумісного редагування і оцінки туристичних маршрутів.

Метод дослідження:

Методом дослідження є аналіз особливостей і методів розробки веб-застосунків.

РОЗДІЛ 1

ОСНОВИ ПОБУДОВИ ВЕБ ДОДАТКІВ

1.1. Клієнт-серверна архітектура

Якісна архітектура веб-додатку робить процес розробки більш ефективним і простим. Веб-додаток з продуманою архітектурою легше масштабувати, змінювати, тестувати і налагоджувати.

Архітектура клієнт-сервер є обчислювальною моделлю, в якій сервер розміщує, розподіляє та контролює більшість ресурсів, а також послуг, що використовуються клієнтом, описує взаємодію між програмами, базами даних та системами проміжного програмного забезпечення в Інтернеті. Такі структурні конструкції складаються з однієї або декількох клієнтських систем, підключених до центральних або основних серверів через мережу, яку ми зазвичай називаємо підключенням до Інтернету. Всі подібні системи, пов'язані з нею, діляться обчислювальними ресурсами.

Архітектура клієнт-сервер також називається структурою обчислень мережі, оскільки кожен запит та пов'язані з ним послуги розподіляються по мережі. В архітектурі клієнт-сервер, коли клієнтський комп'ютер надсилає запит на дані з серверу через Інтернет, сервер приймає запит, обробляє його і доставляє клієнтові необхідну інформацію. Особливістю є те, що серверний комп'ютер може одночасно спілкуватись з безліччю клієнтів. Крім того, один клієнт може підключатися до численних серверів за одиницю часу, і кожен сервер надає різний набір послуг для цього

конкретного клієнта.

Тому у будь-якому типовому веб-додатку існує два різні модулі коду, які працюють поруч. Це:

1. Клієнтський код - код, який знаходиться у браузері та відповідає на певні дії користувача;
2. Серверний код - код, який знаходиться на сервері та відповідає на запити HTTP.

Будь-який код, здатний відповідати на запити HTTP, може запускатися на сервері. Код на стороні сервера відповідає за створення сторінки, яку запитував користувач, а також зберігання різних типів даних. Кінцевий користувач його ніколи не бачить.

Отже, клієнти та сервери - це два різні комп’ютери в різних частинах світу, які пов’язані через Інтернет. Однак клієнт не є обов'язковим, а сервер може знаходитись на іншому континенті, або ж залишатися в тій самій будівлі.

1.2. Соціальні веб-застосунки

Веб додаток розробляється для роботи із клієнтом, тому для комфортної взаємодії необхідно передбачити певні функції, притаманні соціальним веб-застосункам.

1.2.1. Автентифікація

Основна мета автентифікації - це упевнитися в тому, що той, хто намагається виконати будь-яку дію, має на це право. Стандартний спосіб

управління цим процесом полягає в тому, що користувач авторизується через форму входу. Як тільки користувач увійшов у систему, вона відстежує його подальші дії за допомогою сесій, поки він не вийде.

Авторизація - це інша концепція, пов'язана з автентифікацією. Користувач може бути автентифікованим у системі, але чи є у нього права на дії, які він хочете здійснити (переглянути сторінку, змінити або видалити елемент)? Найпростіший приклад це різниця між звичайним користувачем та адміністратором сайту. Вони обидва автентифіковані, але тільки у адміністратора є права на зміну певних речей.

Автентифікація і авторизація йдуть рука об руку - спершу необхідно автентифікувати когось, щоб зрозуміти, хто це, і далі можна перевірити, чи є у нього права на перегляд сторінки або виконання дії.

Найбільш поширений спосіб автентифікації в мережі полягає у введенні електронної пошти або логіну, що є унікальним користувацьким ідентифікатором, і пароля - деяких конфіденційних відомостей. Достовірна (еталонна) пара логін-пароль зберігається в спеціальній базі даних.

Проста автентифікація має наступний загальний алгоритм:

1. Користувач запитує доступ до системи і вводить особистий ідентифікатор і пароль.
2. Введені унікальні дані надходять на сервер автентифікації, де порівнюються з еталонним набором даних для даного ідентифікатора.
3. При збігу даних з еталонними автентифікація визнається успішною, при розходженні - користувач повертається до першого кроку.

Введений пароль може передаватися в мережі двома способами:

1. Незашифрованим, у відкритому вигляді, на основі протоколу парольної автентифікації.

2. З використанням шифрування SSL або TLS. В цьому випадку неповторні дані, введені користувачем, передаються по мережі захищено.

Файл cookie - це невеликий фрагмент даних, створений сервером і переданий у браузер під час відвідування веб-сайту. Браузерам часто потрібно зберігати та відправляти його назад на сервер, щоб повідомити, що запит надходить з того самого пристрою, щоб зберегти аутентичність користувача. Файли cookie браузера зберігаються як пари «ключ-значення».

Автентифікація на основі файлів cookie використовує файли cookie HTTP для автентифікації запитів клієнта та підтримки інформації про сеанси на сервері за протоколом HTTP без статусу.

Ось логічний потік процесу автентифікації на основі файлів cookie:

1. Клієнт надсилає запит на вхід з обліковими даними на сервер сервера.
2. Потім сервер перевіряє облікові дані. Якщо вхід буде успішним, веб-сервер створить сеанс у базі даних та включить заголовок Set-Cookie у відповідь, що містить унікальний ідентифікатор в об'єкті cookie.
3. Браузер зберігає файли cookie локально. Поки користувач залишається авторизованим, клієнт повинен надсилати файл cookie у всіх наступних запитах на сервер. Потім сервер порівнює ідентифікатор сеансу, що зберігається у файлі cookie, з ідентифікатором сеансу в базі даних для перевірки дійсності.
4. Під час операції виходу сервер припинить термін дії файлу cookie, видаливши його з бази даних.

1.2.2. Оновлення даних

Веб-сокети (WebSockets) - це передова технологія, яка дозволяє створювати інтерактивне з'єднання між клієнтом (браузером) та сервером для обміну повідомленнями в режимі реального часу. Веб-сокети, на відміну від HTTP, дозволяють працювати з двонаправленим потоком даних, що робить цю технологію абсолютно унікальною.

Протокол WebSocket має лише дві функції: відкриття рукостискання (handshake) та сприяння передачі даних. Після того, як сервер і клієнт підготують з’єднання, вони можуть надсилати дані один одному з меншими витратами часу та ресурсів у будь який момент часу.

Зв'язок WebSocket відбувається через один сокет TCP, використовуючи протокол WS або WSS. Майже кожен браузер, крім Opera Mini, забезпечує чудову підтримку вебсокетів на теперішній час.

1.3. Бази даних

Застосунок має зберігати користувацькі дані, оновлювати їх та отримувати необхідну для роботи програми інформацію за запитом. Такі дані не можуть зберігатися у коді програми. Тому програма під’єднується до бази даних і активно обмінюється із нею даними протягом усього циклу взаємодії.

База даних - це впорядкований набір структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді в комп'ютерній системі. База даних зазвичай управляється системою управління базами даних (СУБД). Дані разом з СУБД, а також додатки, які з ними пов'язані, називаються системою баз даних, або, для стислості, просто базою даних.

Дані в найбільш поширених типах сучасних баз даних зазвичай зберігаються у вигляді рядків і стовпців формують таблицю. Цими даними можна легко керувати, змінювати, оновлювати, контролювати та організовувати. У більшості баз даних для запису і запитів даних використовується мова структурованих запитів SQL.

1.4. Клієнтське відображення

Односторінковий додаток (SPA) - це одна сторінка, де велика кількість інформації залишається незмінною, і лише деякі елементи потрібно оновлювати у відповідь на користувацькі дії. Це веб-додаток або веб-сайт, який взаємодіє з користувачем шляхом динамічного переписування поточної сторінки, а не завантаження цілих нових сторінок із сервера. Такий формат відрізняється від традиційного завантаження сторінки, коли сервер повторно рендерить повну сторінку при кожному оновленні та відправляє її у браузер.

Цей метод на стороні клієнта пришвидшує час завантаження для користувачів і робить обсяг інформації, яку повинен надсилати сервер, значно меншим та набагато економічнішим.

Рішення SPA мають багато переваг, таких як покращення продуктивності та послідовності програм, а також скорочення часу на розробку та витрат на інфраструктуру.

Відділяючи презентацію від вмісту та даних, команди розробників можуть працювати з різною швидкістю, в той же час інтегруючись для загального рішення. SPA добре підходить для адаптивного дизайну для

мобільних пристроїв, настільних ПК та планшетів.

Від’єднуючи цю внутрішню логіку та дані від того, як вони представлені, розробники перетворюють їх на «послугу», і можуть створити безліч різних інтерфейсних способів показати та використовувати цю послугу.

1.5. Відображення графіки

У HTML5 представлено два елементи для роботи з web графікою: Canvas і SVG. Дві ці технології досить сильно відрізняються одна від одної. Важливо знати про їх переваги та недоліки, щоб вибрати найбільш доцільне для конкретного завдання рішення. Елемент SVG дозволяє створювати векторну графіку, а елемент Canvas призначений для створення растрових зображень. Елемент Canvas також використовується технологією WebGL для апаратного прискорення 3D-графіки.

1.5.1. SVG

Масштабована векторна графіка (Scalable Vector Graphics - SVG) є мовою розмітки, розширеною з XML для опису двомірної векторної графіки.

Для створення зображення в векторній графіці використовуються геометричні примітиви (точки, лінії, криві, багатокутники). З їх допомогою можна створювати зображення, які не втрачають в якості при масштабуванні.

SVG - технологія малювання зі зберіганням об'єктів в пам'яті. Як і

HTML, SVG має об'єктну модель документа (DOM). DOM в SVG, як і в HTML, є моделлю подій. Це означає, що при використанні цієї технології для реалізації інтерактивних дій (таких як управління мишею і т.п.) з боку розробника потрібно менше зусиль, оскільки події прив'язуються безпосередньо до елементів DOM.

SVG має як звичайні атрибути, так і атрибути уявлення. Ключовим моментом є те, що до атрибутів уявлення можна застосовувати стилі відповідно до правил використання стилів CSS. Наприклад, для зміни кольору фігури можна застосовувати властивість fill.

1.5.2. Canvas

Canvas пропонує досить потужне API для дій над двовимірними графічними об'єктами. В тому числі, це вивід і перетворення тексту довільного шрифту, відображення двовимірних об'єктів, робота з векторною графікою, перетворення над фігурами (прозорість, переміщення, поворот і навіть матриця перетворення), робота з окремими лініями і тінями. Також можлива робота з зображеннями в піксельному форматі (що дозволяє зробити практично повноцінний Photoshop всередині окремого браузера) і корекція кольору.

<Canvas> - це HTML елемент, який використовується для створення растрової графіки за допомогою JavaScript. Елемент <canvas> надає зручний API для малювання 2D графіки за допомогою JavaScript.

На відміну від svg, canvas працює з растровою графікою. Це технологія миттєвого малювання, вона не зберігає свої елементи в дереві DOM, отже немає ніякого способу змінити існуючий малюнок або реагувати на події. Це означає, що, за необхідності створити новий кадр, вся сцена має бути перемальована заново.

Елемент <canvas> має тільки два атрибути - ширину і висоту. Якщо атрибути висоти і ширини не встановлені, то згідно специфікації HTML5 ширина елемента canvas буде дорівнювати 300 пікселям, а висота - 150. При зміні цих атрибутів canvas оновлюється.

Для малювання в першу чергу необхідно отримати доступ до контексту, який надає API для створення графіки. Контекст можна отримати за допомогою методу getContext() елемента canvas. В якості першого параметра необхідно вказати тип контексту, який ми хочемо використовувати. На даний момент більшість сучасних браузерів підтримує два типи контексту: «2d» (дозволяє створювати двовимірну графіку) і «webgl» (дозволяє використовувати технологію WebGL для створення тривимірної графіки). Якщо зазначений тип контексту не підтримується браузером, метод getContext() повертає null.

1.5.3. WebGL

WebGL - це ще одна технологія, яка використовує елемент canvas для створення графіки. WebGL дозволяє веб-контенту використовувати API, заснований на OpenGL ES 2.0, для візуалізації тривимірної графіки, але можливо працювати і з двомірної графікою.

Для початку малювання так само, як і в canvas, необхідно отримати доступ до контексту. Це робиться за допомогою методу getContext. В якості типу контексту необхідно вказати webgl або experimental-webgl. Контекст, іменований як «experimental-webgl» - це тимчасове ім'я для контексту, що використовується на час процесу розробки специфікації.

WebGL працює з растровою графікою, відповідно, особливості роботи із canvas застосовні і до цієї технології. Але для WebGL характерна більш висока продуктивність (порівняно з продуктивністю нативних додатків),

оскільки WebGL використовує засоби апаратного прискорення графіки.

1.6. Google maps

Для складання маршруту необхідно мати перед собою мапу місцевості, що надає можливість інтерактивної взаємодії -- переглядати і створювати маршрути, змінювати і додавати проміжні пункти між точками призначення. Аналіз технологій, які надають таку можливість привів до компанії Google та її системи Google Maps.

Google Maps - це веб-платформа для картографування та програма для споживачів, яку пропонує Google. Вона надає супутникові знімки, аерофотозйомку, карти вулиць, інтерактивні панорамні види вулиць на 360°, умови руху в режимі реального часу та планування маршруту для пішохідних, автомобільних, повітряних маршрутів та громадського транспорту. У 2020 році Google Maps використовували понад 1 мільярд людей щомісяця по всьому світу.

Карти Google почали свою роботу як настільна програма C++, наразі ж в інтерфейсі служби, що стала популярним веб-додатком, використовуються JavaScript, XML та Ajax. Карти Google пропонують API, який дозволяє вставляти карти на сторонні веб-сайти, а також локатор для підприємств та інших організацій у багатьох країнах світу.

1.6.1. API

API Google Maps, який тепер називається Google Maps Platform, містить близько 17 різних API, які тематизуються за такими категоріями: Карти, місця та маршрути.

Google запустив API Карт Google, щоб дозволити розробникам інтегрувати Карти Google на свої веб-сайти. Це була безкоштовна послуга, яка не потребувала ключа API до червня 2018 року, коли було оголошено, що для доступу до API потрібен ключ API, пов’язаний з обліковим записом Google Cloud з увімкненою оплатою.

Використовуючи API Карт Google, можна вбудувати Карти Google у зовнішній веб-сайт, на який можуть бути накладені дані для певного сайту. Незважаючи на те, що спочатку існував лише API для карт на JavaScript, API Maps був розширений, включивши послугу для отримання статичних зображень карти та веб-служби для виконання геокодування, генерації вказівок та отримання висоти профілі.

API Google Maps є безкоштовним для комерційного використання за умови, що веб-сайт, на якому він використовується, є загальнодоступним і не стягує плату за доступ, а також не генерує більше 25 000 запитів до карти на день. Сайти, які не відповідають цим вимогам, можуть придбати API Карт Google для бізнесу.

1.6.2. Побудова маршрутів засобами Google Maps

Система побудови маршрутів Google Maps надає змогу не тільки врахувати кількість часу, яка буде витрачена для досягнення кінцевої точки, а й розставляти зупинки, обрати тип транспорту, в зв’язку з чим маршрут може бути перебудований для зручності користувача, а також показати альтернативні маршрути на випадок непередбачуваних ситуацій.

Для побудови маршрутів Google Maps використовує алгоритм стиснення ієрархій. Алгоритм стиснення ієрархій (Contraction Hierarchies) був запропонований в 2008 р, а результати експериментального порівняння його ефективності з іншими алгоритмами пошуку найкоротшого шляху

показали, що він є одним із найбільш ефективних серед алгоритмів пошуку найкоротшого шляху.

На відміну від більшості алгоритмів пошуку найкоротшого шляху велика частина часу роботи алгоритму потрібна не на обчислення самого шляху, а на обробку та спрощення самого графа. У зв'язку з цим можливими є більш ефективний розподіл пам'яті і зменшення часу роботи самого алгоритму.

Алгоритм стиснення ієрархій базується на зменшенні розмірності графа за один крок на одну вершину. Для цього визначається найкоротший шлях між досліджуваної вершиною і всіма суміжними їй вершинами, а досліджувана вершина позначається як пройдена.

1.7. Висновок до розділу 1

У даному розділі було розглянуто основи побудови веб додатків, детально описано стандарти, технології та способи розробки клієнтоорієнтованих веб систем із соціальною та візуальною складовими. Дані теоретичні відомості допоможуть у подальшому створенні прототипу веб-додатку для сумісного планування і збереження туристичних маршрутів та спростять його розробку.

РОЗДІЛ 2

ТЕХНОЛОГІЇ РОЗРОБКИ ВЕБ ДОДАТКІВ

2.1. Клієнтська частина

Фронтендом називають клієнтську сторону призначеного для користувача інтерфейсу.

Розробка фронтенду включає створення функціональності і призначеного для користувача інтерфейсу, що працюють на стороні клієнта додатку або веб-сайту. Сюди відноситься все, що користувач бачить, відкриваючи веб-сторінку.

2.1.1. HTML, CSS та JS

Фронтальний стек складається з багатьох різних мов та бібліотек. Хоча вони варіюються від програми до програми, існує лише кілька загальних мов, які розуміють усі веб-браузери. Цими трьома основними інтерфейсними мовами кодування є HTML, CSS та JavaScript.

Разом вони створюють основний набір інструкцій, які веб-браузери використовують для відображення веб-сторінок, з якими користувачі взаємодіють щодня. Усі інші бібліотеки та інтерфейсна інженерія побудовані на цих трьох основних мовах, що робить їх необхідними навичками для будь-якого розробника інтерфейсу.

HTML використовується для декларативного визначення макета

інтерфейсу користувача, тоді як JavaScript використовується для реалізації взаємодій із користувачем. Завантажений HTML-документ перекладається браузером в елементи дерева об'єктної моделі документа (DOM). DOM - це, по суті, API для доступу до веб-сторінки та маніпулювання нею, і тому вимагає остаточного результату завантаження сторінки. Код JavaScript, що працює у браузері, може використовувати API DOM для зміни сторінки та, отже, надає можливість створення динамічних веб-сторінок. Нарешті, стилі CSS застосовуються, щоб створити бажаний зовнішній вигляд програми.

HTML

HTML є першим шаром будь-якого веб-сайту і створює кодову версію каркасу на веб-сторінці. Ці каркасні рамки існують для стилів у CSS та всіх подій у JavaScript.

HTML5 є останньою специфікацією мови HTML і є суттєвим розривом з попередньою практикою розмітки. Метою глибоких змін у мові було стандартизувати багато нових способів, якими розробники її використовують, а також затвердити єдиний набір найкращих практик щодо веб-розробки.

Більшість індивідуальних змін є результатом масштабних цілей у вдосконаленні мови. Ці цілі в першу чергу включають:

- Заохочення семантичної (змістовної) розмітки;
- Відокремлення дизайну від змісту;
- Підвищення доступності та реагування на дизайн;
- Зменшення накладання між HTML, CSS та JavaScript;
- Підтримка мультимедійного досвіду, усуваючи при цьому необхідність у таких плагінах, як Flash або Java.

CSS

Каскадні таблиці стилів (CSS) - це те, що надає HTML візуальну привабливість та залучає користувача. Простіше кажучи, таблиці стилів диктують презентацію елементів HTML на сторінці.

Веб-сайт може працювати без CSS, але це не буде відповідати найкращому користувацькому досвіду. CSS робить інтерфейс веб-сайту більш привабливим, і це створює чудові зручності для користувачів. Без CSS веб-сайти були б менш приємними для ока і, ймовірно, набагато складнішими для навігації. Окрім верстки та формату, CSS відповідає за колір шрифту та багато іншого.

Є ряд переваг CSS, серед яких:

1) Більша швидкість сторінки

Більше коду означає меншу швидкість сторінки. А CSS дозволяє використовувати менше коду, за рахунок використання одного правила CSS, що застосовується до всіх елементів, позначених певним тегом в документі HTML.

2) Кращий досвід користування

CSS не тільки покращує вигляд веб-сторінки, але і дозволяє створити більш зручне форматування. Коли кнопки та текст знаходяться в логічних місцях і добре організовані, покращується взаємодія з користувачами.

3) Швидший час розробки

За допомогою CSS можна застосовувати певні правила форматування та стилі до кількох сторінок з одним рядком коду. Одну каскадну таблицю стилів можна відтворити на декількох сторінках веб-сайту. Таким чином, загальні для багатьох блоків і розділів сайту правила будуть автоматично застосовуватись, будучи прописаними один раз у загальному файлі стилів.

4) Адаптивний дизайн між пристроями

Адаптивний веб-дизайн має значення. Веб-сторінки повинні бути повністю видимими та легко підлаштовуватись під будь-які пристрої. Незалежно від того, смартфон чи планшет, настільний комп'ютер чи навіть розумний телевізор, CSS поєднується з HTML, щоб зробити адаптивний дизайн можливим.

Нова тенденція стилю полягає у використанні того, що називається препроцесорами CSS. Сюди входять Less, Sass та низка менш популярних технологій. Препроцесори - це мови сценаріїв, які компілюються в CSS для браузера і станом на 2021 рік дуже популярні, оскільки прискорюють процес розробки. Вони також вміщують деяку логіку програмування.

JavaScript

Оскільки все більше людей використовували браузери для повсякденного використання та користування Інтернетом, виникла потреба в мові програмування, яка зробила б веб сайти інтерактивними. Тоді і з'явився JavaScript, щоб оживити браузер. Це мова, яка здатна зробити все те, що роблять інші мови програмування, але вона також має особливі стосунки з браузером.

JavaScript є мовою виконання веб-браузерів. Це означає, що коли користувач відкриває веб-сторінку, сторінка завантажує як основний JavaScript, який є стандартним для сторінки і визначається браузером, так і будь-який новий JavaScript, доданий на сторінку розробником. Новий JavaScript завантажується паралельно з основним і може виконувати дії та приймати рішення.

JavaScript відрізняється від фонових мов, таких як Java або Ruby, тим, що він видимий користувачеві і не компілюється заздалегідь, щоб таємно

працювати за завісою. Це справжня мова програмування інтерфейсного проектування та основна мова, яка пов’язує всі технології веб-розробки разом.

Однією з основних особливостей JavaScript є те, що він включає програмування на основі подій. Він має вбудовані події, такі як "onClick" та "onHover", які чекають взаємодії користувача на веб-сайті перед виконанням певного сегмента коду. Наприклад, коли користувач натискає кнопку нічного режиму сайту, це подія, яка запускає сегмент коду JavaScript, який змінює CSS на всьому веб-сайті від світлих до темних кольорів.

JavaScript також можна використовувати для створення динамічного вмісту на веб-сайті. На основі вводу користувача можна створювати різні теги HTML.

ECMAScript 6, також відомий як ECMAScript 2015, був першою версією нової ери стандарту ECMAScript. ES6 є значним оновленням мови та першим оновленням мови з моменту стандартизації ES5 у 2009 році.

2.1.2 JavaScript фреймворки

Фреймворки JS - це засоби програмування JavaScript, в яких є попередньо написаний код для використання в стандартних функціях і завданнях програмування. Це основа для створення веб-сайтів або веб-додатків навколо.

Створення сайтів цілком можливо без їх використання, але правильно підібране середовище може значно полегшити роботу. Більш того, вони є безкоштовними і мають відкритий вихідним код.

В першу чергу вони підвищують продуктивність, адже вже містять заздалегідь написані і готові до використання функції і шаблони. Деякі компоненти веб-сайту не повинні бути виготовлені унікальними, тому є можливість створювати і розширювати попередньо створені компоненти. Фреймворки більш адаптовані для дизайну веб-сайтів, і більшість розробників масштабних сайтів обирають їх.

Основними фреймворками JavaScript вже довго залишаються три найбільш поширені технології -- React.js, Angular і Vue.js. Порівняльний графік популярності цих засобів наведено на наступному рисунку.

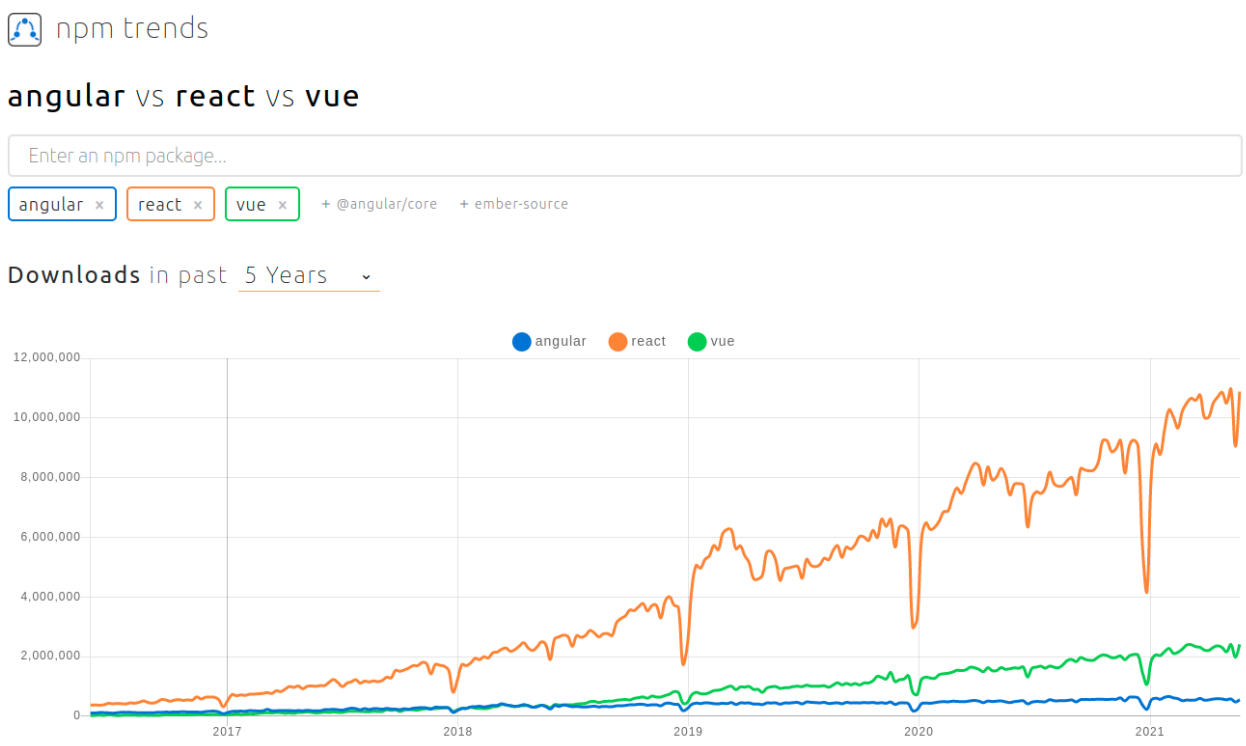


Рис. 2.1. Графік завантажень пакетів фреймворків за останні 5 років.

На графіку чітко видно розподіл завантажень і тенденції серед frontend розробників. Розглянемо детальніше кожен із фреймворків для того, щоб

обрати найбільш доцільну технологію.

1. React

В даний час лідером в області інфраструктури JavaScript UI є React. Спочатку розробники Facebook почали працювати над ним, щоб спростити свою роботу, адже React заснований на компонентах багаторазового використання. Простіше кажучи, це блоки коду, які можна класифікувати як класи або функції. Кожен компонент представляє певну частину сторінки, таку як логотип, кнопка або поле вводу. Використовувані ними параметри визначають властивості компонента.

React використовує JSX, синтаксис XML, який поєднує в собі JavaScript і HTML тому всі його шаблони -- це повноцінний JavaScript. Не дивлячись на переваги JSX, таке рішення зменшує привабливість для обрання React як засобу розробки, адже окрім вже вивчених технологій, розробнику знадобиться опанувати хоча і схожу, та все ж відмінну від звичних нову для себе мову програмування.

2. Angular

Angular - одне з найпотужніших середовищ JavaScript. Це середовище розробки відоме перш за все тому, що воно надає розробникам найкращі умови для об'єднання JavaScript з HTML і CSS.

Angular має компонентну структуру, як і React. Ви можете маніпулювати, комбінувати і використовувати їх у міру необхідності. У програмах на Angular використовується TypeScript - це розширений набір JavaScript, який використовує той же синтаксис, але також підтримує статичну типізацію і класи. У TypeScript розробник отримує модифікатори доступу, перерахування, узагальнення, гібридні типи і багато іншого.

Хоча Angular був досить популярним фреймворком і до

сьогоднішнього дня використовується на певних проектах, він має ряд недоліків, які заважають йому достойно конкурувати з більш новими і сучасними рішеннями.

До найбільш вагомих мінусів варто віднести:

- Погану продуктивність із коробки;
- Взаємодію із деревом сайту DOM напряму, що суперечить сучасним стандартам і найкращим практикам фронтенд розробки;
- Низький рівень абстракції, коли розробнику доводиться зав’язувати всю логіку розробки на HTML;
- Високий поріг входу і складність в опануванні всього фреймворку через надзвичайно велику кількість функцій, змін та правил;
- Відсутність підтримки синтаксису попередніх версій, що створює необхідність міграції цілого проекту після виходу оновлення.

3. Vue.js

Vue - це JavaScript-фреймворк з відкритим вихідним кодом для створення інтерфейсу. Інтеграція з Vue в проектах, що використовують інші бібліотеки JavaScript спрощена, оскільки вона розроблена із націленням на максимальну адаптацію.

Vue.js також досить простий в освоєнні: все що потрібно, це JavaScript і HTML. Іншою сильною стороною Vue.js є його інтерфейс командного рядка (CLI). Це базовий інструмент, який прискорює розробку, пропонуючи масу плагінів, пресетів, миттєвого прототипування і інтерактивного інструменту розробки проектів. Деякі з його функцій включають компоненти, шаблони, переходи і двостороннє зв'язування даних, а також фокус реактивності. Реактивність виникає при зміні або оновленні будь-якого з об'єктів JavaScript в Vue. Vue.js використовує те, що називається Shadow DOM, що робить рендеринг сторінки швидким.

2.1.3. Обґрунтування вибору технологій розробки

Для розробки веб додатку було обрано Vue.js фреймворк, виходячи із перелічених нижче переваг, наявності вже існуючого досвіду роботи із технологією, та її загальної перспективності серед інших популярних JavaScript фреймворків.

Повторно використовувані компоненти

Інкапсульовані компоненти є окремими повноцінними фрагментами коду, які розробники можуть повторно використовувати в якості шаблонів для аналогічних елементів.

Відмінне модульне тестування

Процес тестування налаштований за модульною системою, яка перевіряє, як найменші частини працюють в додатку самі по собі. Компонентний підхід значно спрощує цей процес.

Продуктивність

Vue.js використовує віртуальну модель DOM, копію вихідної моделі DOM веб-сайту, яку фреймворк використовує для визначення оновлень елементів без необхідності візуалізації всієї моделі DOM, тим самим підвищуючи продуктивність програми і роблячи рендеринг сторінок досить швидким.

Масштабованість

Vue.js допомагає розробляти дійсно великі і комплексні шаблони для багаторазового використання, що, завдяки простій структурі, не потребує великої кількості зусиль.

Відмінна документація

Vue.js має повну, детальну і чітко структуровану документацію, яка може збільшити швидкість навчання розробників і заощадити багато часу на розробку програми з використанням базових знань HTML і JavaScript.

CLI

Автоматична генерація каркасу проекту спрощує початок роботи і не забирає час на визначення правильної організації початкової структури проекту, що завжди є ідентичною. На цьому функціонал cli не закінчується, вона дозволяє:

- Генерувати базові файли компонентів;
- Завантажувати додаткові пакети;
- Змінювати версії завантажених пакетів, при виникненні проблем;
- Запускати додаток з потрібною конфігурацією;
- Налаштовувати та запускати процеси розгортання програми в інтернеті.

2.2. Серверна частина

Для вирішення поточної задачі завдяки сучасним веб технологіям можна розробити повноцінний веб додаток не використовуючи backend мов програмування. Vue.js надає достатньо інструментів для успішної взаємодії із зовнішніми API та PaaS (програмне забезпечення моделі “платформа як послуга”).

2.2.1. Firebase & Firestore

Завдяки технологіям і сервісам, що надаються Google Cloud

Платформою, розробники можуть користуватися сервісами автентифікації Firebase та базою даних із того самого середовища Firestore.

Firebase - це платформа для розробки мобільних та веб-додатків. Вона складається з кількох різних модулей і сервісів, які доповнюють одна одну. Первинним продуктом Firebase була база даних у режимі реального часу, яка надає API, що дозволяє зберігати та синхронізувати дані між кількома клієнтами. Однак зараз він пропонує ще більше можливостей, включаючи нову базу даних Firestore, хмарні функції для розширення та написання власного API, а також інші корисні функції. Крім того, Firebase має план ціноутворення, який включає безкоштовний пакет, який підходить для малих та середніх проектів, але, якщо проект вже виходить на ринок і залучає більше відвідувачів, об’єм послуг Firebase можна легко масштабувати, щоб задовольнити всі поточні вимоги.

Автентифікація.

Firebase є зручним рішенням для забезпечення автентифікації без необхідності створювати та власноруч впроваджувати складні алгоритми і безпекові складові роботи з персональними даними користувачів. Він зберігає дані та ідентифікує користувача, який увійшов у систему та вийшов із програми. Без Firebase було б важко здійснити процес безпечно та безперешкодно.

Автентифікація Firebase використовує не лише електронні листи, паролі та телефонні номери для проведення цього процесу, але також підтримує постачальників об’єднаних ідентифікаційних даних. Дійсно, користувачі можуть входити у свої програми за допомогою Google, Twitter, GitHub Facebook тощо.

Автентифікація SDK Firebase, яка включає такі перевірки ідентичності, підтримує налаштування iOS, C ++, Android, Web та Unity:

- Адреса електронної пошти та автентифікація на основі пароля;
- Підтвердження ідентифікації номера телефону;
- Тимчасова анонімна автентифікація облікового запису;
- Спеціальна авторизація.

З іншого боку, FirebaseUI Auth для ідентифікації користувачів працює лише для Android, iOS та веб-налаштувань. Але це все ще простий і рекомендований спосіб заповнити систему входу та покращити взаємодію з користувачем у розробленій програмі.

Автентифікація Firebase використовує процедури перевірки входу OpenID Connect та OAuth 2.0, коли йдеться про ідентифікацію на стороні сервера. Ця функція автентифікації також поєднується з іншими службами Firebase для покращення входу та доступу до даних.

База даних у реальному часі та Firestore.

Cloud Firestore є надзвичайно масштабованою для мобільних, веб- та серверних розробок. SDK Firebase, доданий у мобільний додаток, дозволяє отримувати безпосередній доступ до даних без потреби проміжного компонента. Він підтримує синхронізацію даних протягом усієї програми, незалежно від підключення до Інтернету.

Служба баз даних Firestore відповідає за оновлення даних у режимі реального часу. Існує можливість налаштувати клієнтський SDK і спостерігати за змінами, внесеними в дані. Ця функція допомагає у підтримці відстеження даних, як тільки вони змінюються, та оперативно вносити зміни до програми, залежно від оновлених даних.

2.2.2. API

Інтерфейс прикладного програмування (Application Programming Interfaces, APIs) - це готові конструкції мови програмування, що дозволяють

розробнику будувати складну функціональність з меншими зусиллями. Вони "приховують" складніший код від програміста, забезпечуючи простоту використання.

API - це, по суті, набір правил, які визначають, як дві машини спілкуються між собою. Деякі приклади взаємодії на основі API включають хмарну програму, що взаємодіє із сервером, сервери, що пінгують один одного, або програми, що взаємодіють з операційною системою. Майже всі компанії, які використовують будь-які сучасні технології, використовують API на певному рівні для отримання даних або взаємодії з базою даних для використання клієнтами.

Більшість веб-API розташовуються між додатком та веб-сервером. Користувач ініціює виклик API, який повідомляє програмі щось робити, тоді програма буде використовувати API, щоб попросити веб-сервер щось зробити. API є посередником між програмою та веб-сервером, а виклик API - це запит. І кожного разу, коли ви використовуєте програмне забезпечення для спілкування з іншим програмним забезпеченням або веб-серверами в Інтернеті, ви використовуєте API для запиту потрібної вам інформації.

Для розробки надаються платні або безкоштовні API. Для їх використання необхідно отримати ключ API і налаштувати середовище взаємодії, відповідно до правил ініціалізації окремо взятої платформи.

2.3. Додаткові технології розробки

Окрім основних технологій розробки, сучасний інженер ПЗ має можливість залучити інструменти, що спростять певні етапи створення

програми або пришвидшать роботу над незначними її компонентами.

2.3.1. Інструменти роботи зі стилями

Візуальна складова є одним із найважливіших аспектів взаємодії веб додатку із користувачем. Однак, не завжди вона виправдовує кількість витраченого часу на прописування усіх стилів CSS.

SCSS - це препроцесор, який дозволяє використовувати функції, які ще не є частиною більш широкого стандарту CSS, і забезпечує кращі робочі процеси для ведення таблиць стилів. За допомогою препроцесора SCSS ви можете зменшити кількість повторень і забезпечити написання чистого, відстежуваного і масштабованого коду на майбутнє.

SCSS повністю сумісний з синтаксисом CSS, в той же час підтримуючи всю потужність Sass.

Сухий (не повторюваний) код є однією з основних найкращих практик у сучасній розробці будь якого ПЗ.

Було також використано Vue Material - це масштабована бібліотека, розроблена з урахуванням останніх специфікацій дизайну Google. Бібліотека містить корисні компоненти для полегшення процесу створення складних оболонок додатків.

Маючи понад 1,7 мільйона завантажень, він пропонує легку бібліотеку для створення додатків, які вміщуються на кожному екрані. Vue Material сумісна з сучасними веб-браузерами, а її API зручний для початківців, що робить її ідеальною для нашого списку 2020 року.

2.3.2. Допоміжні інструменти екосистеми Vue.js

Фреймворк Vue також надає інструменти покращення коду і структури програми, спрощення процесу побудови SPA, а саме цей формат і був

обраний у даній роботі як основа створюваного додатку.

Vue Router

Vue.js реалізує шаблон SPA через свій основний плагін, vue-router. Для vue-router кожна URL-адреса маршруту відповідає компоненту. Це означає, що розробник задає для vue-router інструкції, як поводитися, коли користувач переходить до певної URL-адреси з точки зору її компонента. Іншими словами, кожен компонент оновленого відображення є елементом старого. Vue Router, додає маршрутизацію на стороні клієнта для обробки навігації між посиланнями, не викликаючи перезавантаження сторінки.

Коли програма використовує маршрутизацію на стороні клієнта, вона все одно робить початковий запит на сторінку, що міститься на стороні сервера. Різниця із серверною маршрутизацією полягає в тому, що сервер повертає однаковий вміст, незалежно від того, який шлях визначено у адресному рядку. Справжня робота з маршрутизацією виконується на клієнті. Коли браузер завантажує JavaScript, бібліотека маршрутизації на стороні клієнта, а саме Vue Router, використовує шлях в URL-адресі для візуалізації правильного вмісту.

Коли користувач натискає посилання у меню навігації, клієнтський маршрутизатор заважає браузеру робити новий запит за оновленою адресою адресного рядка. Натомість маршрутизатор змінює URL-адресу, не викликаючи перезавантаження сторінки, а вміст сторінки відображається разом із новим завантаженим компонентом, що був визначений розробником як відповідний до поточної адреси. Справа в тому, що вміст сторінки оновлюється, але сторінка зберігає той самий стан.

Vuex

Обробка стану в односторінкових програмах може бути складним

процесом. У міру того як додаток стає більшим і складнішим, все більше виникає ситуацій, коли певний фрагмент стану потрібно використовувати в декількох компонентах.

Vuex - це шаблон управління додатком і бібліотека для програм Vue.js. Він служить централізованим сховищем усіх компонентів програми, з правилами, що гарантують, що стан програми може змінюватись лише передбачувано.

Vuex - офіційне рішення для управління Vue. Він працює, маючи центральне сховище для загального стану та забезпечуючи методи, що дозволяють будь-якому компоненту у вашій програмі отримувати доступ до цього стану.

Vuex - реалізація шаблону Flux, тому використовується низка наступних концепцій:

- State: це дані програми, з якими взаємодіють компоненти. Об'єкт стану - це єдиний об'єкт, який містить усі стани додатку і служить єдиним джерелом істини. Основне місце розташування -- Vuex Store.
- Getters: Vuex дозволяє визначати “геттери”, за допомогою яких програма отримує відфільтровані за будь якими правилами дані зі сховища. Результат цієї фільтрації кешуєтсья сховищем і переобчислюється лише тоді, коли вхідні дані змінюються.
- Mutations: єдиний спосіб змінити стан програми - це здійснення мутацій. Вони викликаються при необхідності внести синхронні зміни і певним чином впливають на стан програми.
- Actions: працюють аналогічно мутаціям, за винятком того, що можуть виконувати асинхронні дії.

2.4. Висновок до розділу 2

У даному розділі було розглянуто основні технології розробки клієнтської і серверної частин веб додатків. Після ґрунтовного аналізу переваг і недоліків кожної з них, було обрано перелік засобів розробки веб-додатку для сумісного планування і збереження туристичних маршрутів, а саме Vue.js, та сервіси Google Cloud -- Firebase і Firestore.

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ СУМІСНОГО ПЛАНУВАННЯ І ЗБЕРЕЖЕННЯ ТУРИСТИЧНИХ МАРШРУТІВ

Для того, щоб правильно впровадити будь-яке програмне забезпечення, спочатку потрібно визначити вимоги та розробити архітектуру майбутнього додатку. У цьому розділі розглядаються основні типи вимог до програмного забезпечення та конкретні вимоги до програмного забезпечення на цю тему, архітектура клієнта, серверної частини та бази даних, а також виконується конкретна реалізація додатку.

3.1. Визначення вимог до програмного забезпечення

Бізнес-вимоги визначають контекст і дозволяють вимірювати переваги, які замовник очікує отримати від реалізації проекту.

Бізнес-вимоги містять високорівневі цілі організації або замовників системи. Як правило, їх висловлюють ті, хто фінансує проект, покупці системи, менеджер реальних користувачів, відділ маркетингу. У цьому документі пояснюється, чому замовнику потрібна така система, тобто описані цілі, які мають бути досягнуті з її допомогою. Визначення меж проекту являє собою перший етап управління загальними проблемами збільшення обсягу робіт.

Для веб додатку для сумісного планування і збереження туристичних

маршрутів було визначено такі бізнес вимоги:

1. Збільшення обсягів туризму і кількості задоволених туристів;
2. Спрощення процесу планування сумісних подорожей;
3. Розвиток туристичної спільноти за рахунок розробки мережі для однодумців.

Вимоги користувачів описують цілі і функції, які користувачам дасть система. До способів уявлення цього виду вимог відносяться варіанти використання, сценарії і таблиці «подія - відгук». Таким чином, в цьому переліку зазначено, що клієнти зможуть робити за допомогою системи.

Було визначено такі вимоги користувачів:

1. Реєстрація та авторизація користувачів;
2. Створення і збереження власних маршрутів;
3. Поширення створених маршрутів за допомогою статичних посилань;
4. Збереження, оцінка, доповнення, поширення маршрутів інших користувачів;
5. Перегляд рейтингу кращих маршрутів за кількістю вподобань.

Взаємодію користувачів із системою можна графічно виразити через діаграму прецедентів. Діаграма прецедентів для даної системи наведена на рис. 3.1.

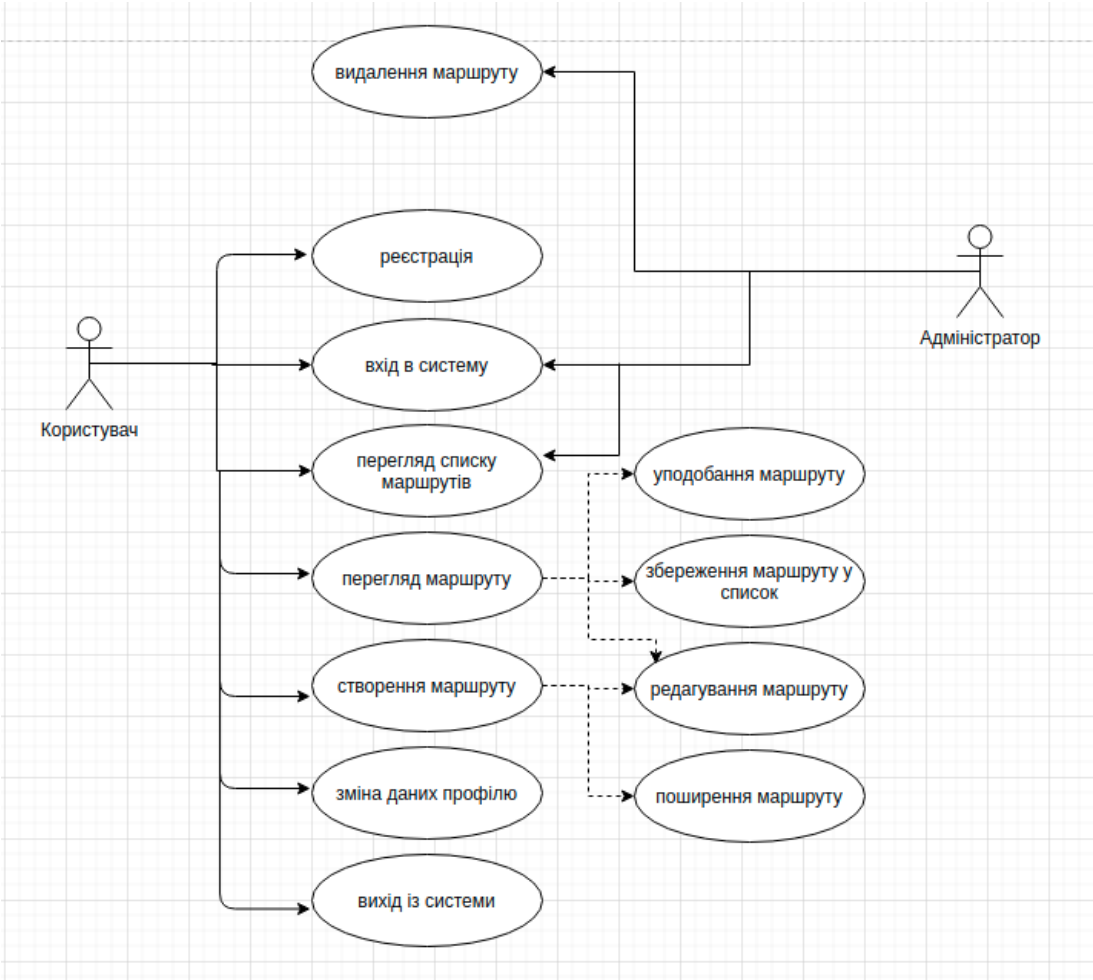


Рис. 3.1. Діаграма прецедентів

Функціональні вимоги визначають функціональність ПЗ, яку розробники повинні побудувати, щоб користувачі змогли виконати свої завдання в рамках бізнес-вимог. Іноді вони називаються вимогами поведінки, вони містять положення з традиційним «повинен» або «повинна»: «Система повинна виконувати певну функцію за певної передумови».

Функціональні вимоги документуються в специфікації вимог до ПЗ, де максимально повно описується очікувана поведінка системи.

Для веб додатку для сумісного планування і збереження туристичних

маршрутів було визначено такі функціональні вимоги:

1. Забезпечення функціоналу інтерактивного створення, збереження та поширення туристичних маршрутів;
2. Забезпечення безпечної автентифікації;
3. Забезпечення перегляду та взаємодії зі збереженими маршрутами інших користувачів;
4. Забезпечення збереження і обміну даними із базою даних;
5. Забезпечення доступу до програмних можливостей Google Maps API.

Системні вимоги - це високорівневі вимоги до продукту, які містять багато підсистем. Говорячи про систему, ми маємо на увазі програмне забезпечення або підсистеми ПО і устаткування. Люди - частина системи, тому певні функції системи можуть поширюватися і на людей.

Системні вимоги, що були визначені для додатку:

1. Наявність інтернет з’єднання на пристрої;
2. Наявність сучасного веб браузеру;
3. Доступ до програмного інтерфейсу Google Maps API.

При подальшому виході ПЗ на ринок до системних вимог також необхідно віднести розміщення додатку на надійному хостингу, що забезпечить безперебійний доступ до сервісу користувачам з будь якої точки Земної кулі.

3.2. Проектування програмного забезпечення

Проектування майбутнього ПЗ є одним із найважливіших етапів його створення. Необхідно розробити якісну масштабовану архітектуру,

задокументувати план подальшої розробки і переконатись, що створювана система відповідає всім попередньо описаним вимогам. Якісно пройдений етап проектування ПЗ є запорукою швидкої і коректної розробки і гарантує досягнення цілей, поставлених замовником перед командою.

3.2.1. Побудова архітектури клієнту

Компонентний підхід дозволяє уникнути мішанини коду і чітко вибудовувати архітектуру програми. Будь-яку складну сторінку завжди можна розбити на менші складові. Кожну з таких частин при виділенні в компонент простіше підтримувати, а за необхідності - повторювати розбиття всередині компонента на ще менші частини.

Першим помітним плюсом подібного розбиття на компоненти буде зручність в підтримці - більше не потрібно тримати в голові логіку всієї програми, можна зосередитися на конкретній її частині. Вносити зміни або доопрацьовувати елемент потрібно буде тільки в одному місці. Ізольованість же компонентів позбавить від появи конфліктів з іншими частинами програми.

Тому застосування компонентного підходу тепер широко використовується в багатьох фреймворками. Vue не залишився в стороні і надає прекрасні можливості по роботі з компонентами.

У програмній реалізації було легко виділено наступні компоненти, адже кожна користувацька вимога описує модуль користувацької частини додатку:

1. Модуль автентифікації, що забезпечує реєстрацію, авторизацію за логіном і паролем, а також автоматично авторизує користувача, якщо його сесію не було завершено.
2. Модуль маршруту, що є однаковим як для нового, так і для

збереженого маршруту і виводить детальну інформацію про маршрут, а також використовує модуль карти для візуального відображення.

3. Модуль карти ініціалізує карту і маркери маршруту з допомогою Google Maps API та інформації із бази даних щодо конкретного маршруту.
4. Модуль виводу списку маршрутів. Завдяки компонентному підходу, один і той самий компонент виводу списку застосовується на сторінках “Збережені”, “Мої маршрути” та “Домашня сторінка”, змінюючи набір відображуваних даних залежно від вхідних параметрів.
5. Модуль App є вхідною точкою програми, забезпечує загальний зовнішній вигляд та є батьківським для усіх інших компонентів.
6. Модуль router відповідає за переключення між компонентами SPA під час взаємодії із програмою та змінює посилання у адресному рядку.

Кожен модуль може бути представлений одним компонентом, або містити у собі дочірні. Основну логіку взаємодії з клієнтом бере на себе модуль App, він же ініціалізується першим під час запуску додатку, перевіряє чи авторизовано користувача та виводить всі можливі шляхи роутеру в залежності від дій користувача.

Компоненти побудовано за принципом Single File Component, що визначається Vue.js як найкращий шаблон для створення компонентів. Таким чином всередині компоненту його шаблон, стилі та логіка нерозривно зв’язані між собою і робота постійно ведеться у межах одного компоненту. Це надає можливість ізольованої розробки, коли зміни у компоненти не можуть вплинути на працездатність цілої програми, а також масштабування і повторного використання модулю навіть за межами розробленої системи з мінімальними правками.

Наглядно компонентна архітектура програми передана через UML-діаграму компонентів (див. Додаток А).

3.2.2. Побудова архітектури взаємодії з сервером

В архітектурі взаємодії із сервером можна виділити три функціональні складові, необхідні для роботи програми:

1. Автентифікація з допомогою Firebase;
2. Обмін даними із базою даних Firestore;
3. Робота із функціоналом, що надає Google Maps API.

Ініціалізація сервісів Google Cloud, а саме Firebase і Firestore відбуваєтсья у головному файлі програми main.js, що також ініціалізує вхідну точку клієнтської частини програми. Вона містить ключі API та об’єкт конфігурації, завдяки якому встановлюється зв’язок між програмою і необхідними сервісами і повертає об’єкт, з яким можна надалі програмно взаємодіяти. Інтерфейси бази даних та автентифікації далі передаються і зберігаються до файлу модулю vuex, який одночасно є і сховищем даних клієнта, і забезпечує обмін даними із віддаленими сервісами.

У файлі модулю vuex, що за замовчуванням має назву store.js, описано стан системи і функції взаємодії із базою даних та локальними даними. Таким чином, будь який компонент системи може звернутися до store і отримати необхідні для виводу на екран дані, відправити запит до бази даних або зберегти нову інформацію.

Взаємодія із Google Maps API відбувається безпосередньо у компоненті карти. В ньому забезпечено ініціалізацію базової карти засобами API, вивід її на екран і визначення користувацьких маркерів маршруту. Компонент робить запит на збережені дані для конкретного маршруту у базу даних через глобальний store, передає отриману

інформацію сервісу Google Maps, і той врешті створює відповідний маршрут, що буде виведено користувачу для перегляду і взаємодії.

Завдяки Vue.js всі ці процеси є зручними і реактивними, а зміна стану одного компонента або запису у базі даних автоматично оновить стан усього додатку.

3.2.3. Проектування бази даних

Проектування структури БД має вирішальне значення при побудові масштабної програми, такої як розроблюваний у даній кваліфікаційній роботі веб-додаток. Коректно спроектована БД забезпечує легкий і зручний доступ до користувацьких даних. Оскільки процес роботи користувача із сайтом постійно включає в себе вивід великої кількості інформації, це також мінімізує неефективні запити на сервер і надмірне використання ресурсів.

У базі даних програми було визначено три колекції -- колекція користувачів, маршрутів і окремих маркерів маршрутів.

Детальну структуру бази даних наведено графічно на рис. 3.2.

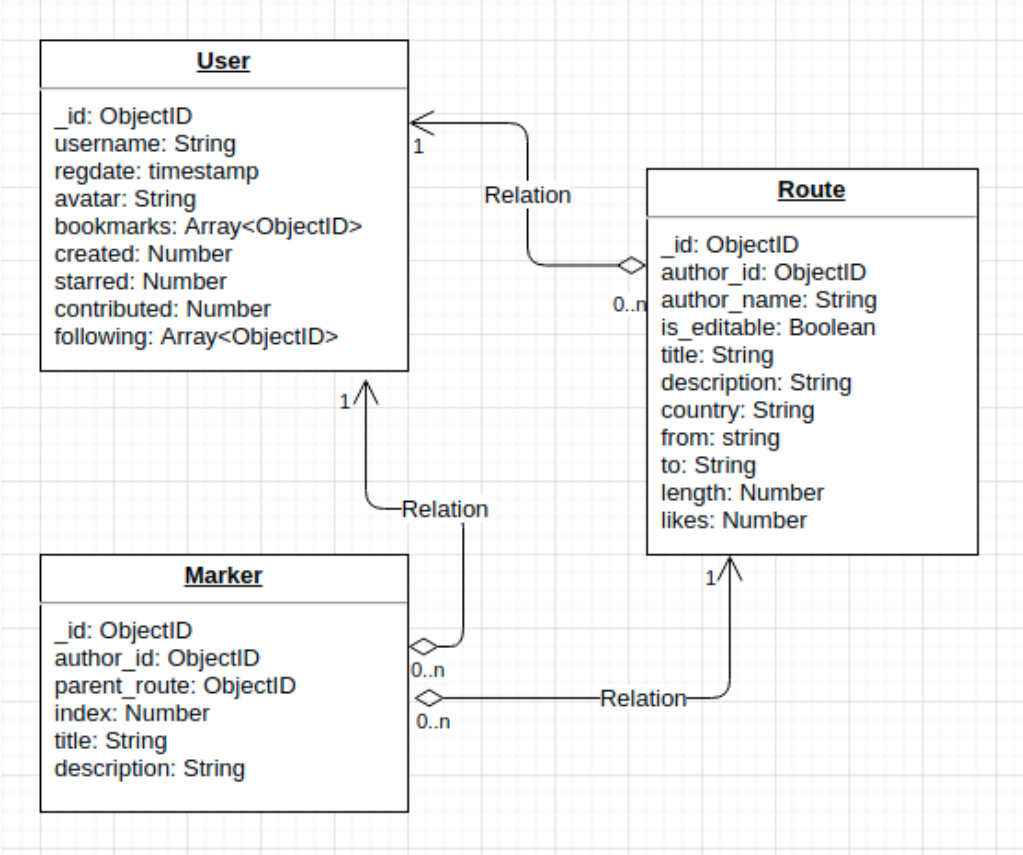


Рис. 3.2. Структура бази даних додатку

Архітектуру БД було спроектовано так, що кожна сутність, що є залежною від іншої, має посилання на батьківський об’єкт. Таким чином, сутності маршрутів та маркерів мають висхідний зв’язок “один до багатьох” і для кожного маршруту легко можна отримати перелік проставлених маркерів-зупинок, а для кожного користувача -- перелік створених ним маршрутів.

Також, вбудований стан програми Vue дозволяє не робити зайві запити до бази даних, зберігаючи тимчасові або незмінювані дані після первинної ініціалізації безпосередньо у клієнта.

3.3. Життєвий цикл програмного забезпечення

Для реалізації веб-додатку для сумісного планування туристичних маршрутів будо обрано інкрементну модель життєвого циклу ПЗ. Такий вибір обумовлено тим, що вимоги до розроблюваного ПЗ можуть змінюватись і доповнюватись під час процесу розробки, а також є необхідним уже з перших ітерацій отримати мінімальну працездатну версію розроблюваного ПЗ.

Інкрементна розробка являє собою процес часткової реалізації всієї системи і поступового нарощування функціональних можливостей. Цей підхід дозволяє зменшити витрати, понесені до моменту досягнення рівня вихідної продуктивності. За допомогою цієї моделі прискорюється процес створення функціонуючої системи. Цьому сприяє застосовуваний принцип компонування зі стандартних блоків, завдяки якому забезпечується контроль над процесом розробки змінюваних вимог.

Інкрементна модель діє за принципом каскадної моделі з перекриттями, завдяки чому функціональні можливості продукту, придатні до експлуатації, створюються раніше. Для цього може знадобитися повний заздалегідь сформований набір вимог, які виконуються у вигляді послідовних, невеликих за розміром проектів, або виконання проекту може початися з формулювання спільних цілей, які потім уточнюються і реалізуються групами розробників.

Переваги інкрементної моделі:

- Робоча програма існує вже на ранній стадії життєвого циклу продукту;
- Гнучкість. Змінити масштаби і вимоги проекту можна у будь який

момент часу з мінімальними затратами;

- Невеликі ітерації спрощують тестування і внесення правок;
- Простіше ідентифікувати ризики, впоратися з ними;
- Кожна ітерація - проста в управлінні контрольна точка проекту.

Недоліки інкрементної моделі:

- Кожна фаза ітерації нерухома;
- Можуть виникнути проблеми щодо архітектури системи, так як не всі вимоги зібрані заздалегідь для всього життєвого циклу ПЗ.

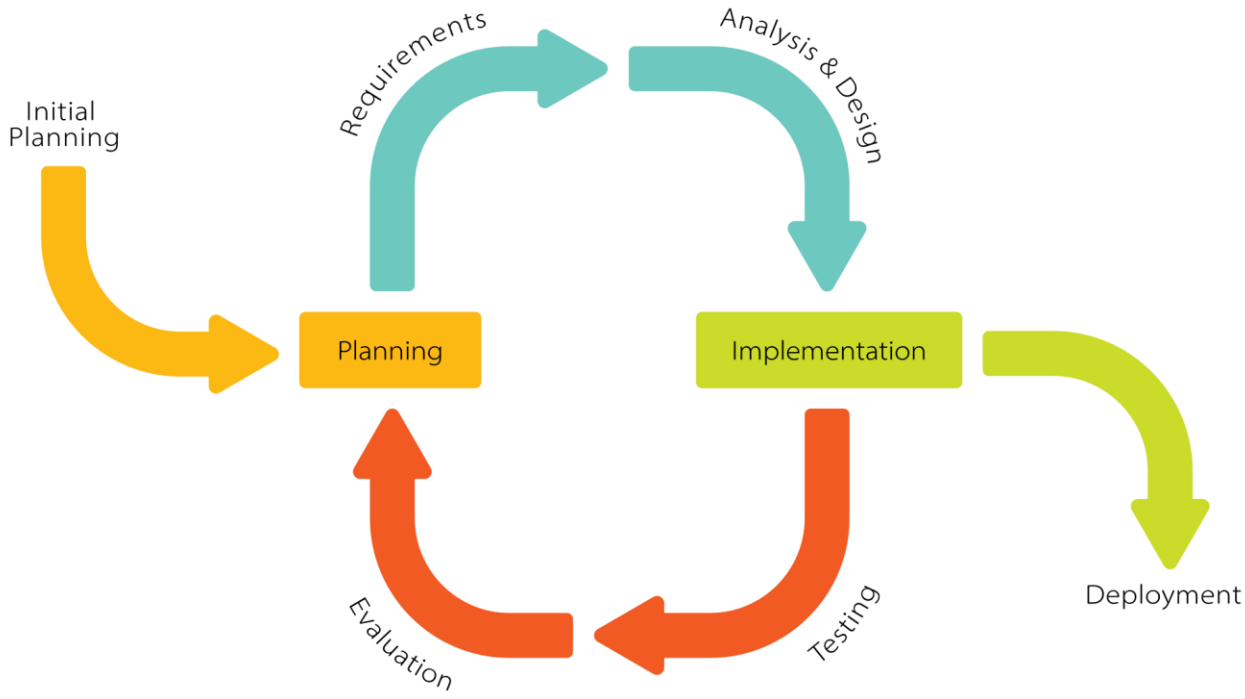


Рис. 3.3. Загальна схема інкрементної моделі життєвого циклу

Загалом, інкрементна модель життєвого циклу ПЗ є найбільш поширеним рішенням для процесу розробки веб додатків.

Розробка веб-додатку для сумісного планування і збереження туристичних маршрутів за інкрементною моделлю життєвого циклу велася

за наступними етапами:

1. Визначення та пріоритизація вимог - для розробки масштабного веб-додатку, що взаємодіє із великою кількістю користувачів, необхідно максимально повно визначити вимоги до майбутньої програми та провести аналіз цих вимог для пріоритизації елементів функціоналу.
2. Планування першої ітерації - після пріоритизації було визначено, який функціонал має бути створеним у першу чергу для швидкого отримання функціонуючого ПЗ, що виконує базово необхідні функції.
3. Розробка дизайну - було розроблено дизайн, що забезпечує просту навігацію і приємне користування веб-додатком, і є найбільш підходящим до архітектури клієнту.
4. Конструювання і тестування - розробка і тестування запланованого функціоналу першої ітерації.
5. Розгортання програми - запуск веб-додатку, аналіз поведінки та відгуків користувачів задля покращення ПЗ у наступних ітераціях розробки.
6. Планування наступних ітерації за пріоритетами вимог.

3.4. Реалізація

Точкою входу в програму є головний модуль, що перевіряє токен авторизації користувача, визначає усі компоненти, що приймають участь у роботі програми. Якщо токен авторизації валідний, користувач направляється на головну сторінку, а його ID ідентифікатор зберігається у сховищі програми для подальшої роботи із базою даних. Якщо токен

відсутній або час його валідності закінчився, користувач бачить вікно автентифікації, де в нього є можливість увійти в систему за своїми даними реєстрації, або зареєструватися із використанням адреси електронної пошти, логіну і пароля.

Головна сторінка містить меню програми, за кліком на пункти якого визначається, який з компонентів програми побачить користувач, та робоче середовище, яке рендерить поточно обраний компонент. Серед пунктів меню наявні: домашня сторінка, кнопка “створити маршрут”, сторінка зі списком маршрутів, створених даним користувачем, збережені ним маршрути, сторінка профілю та кнопка виходу із системи із закриттям поточної сесії автентифікації.

Всі компоненти виводу списку маршрутів працюють за єдиним принципом. При кліку на відповідний пункт меню і створенні компонента, програма надсилає у сховище параметри, щодо необхідного переліку маршрутів для виведення користувачу. Залежно від цих параметрів, у сховищі виконується функція, що надсилає запит до бази даних, зберігає отримані дані або передає їх безпосередньо компоненту. Користувач бачить деталі кожного з відображених маршрутів, може взаємодіяти із ними кнопками “зберегти”, “поширити”, тощо, або перейти до редагування того чи іншого маршруту, якщо стороннє редагування було дозволене автором маршруту.

Відкриття маршруту для редагування або створення нового відбувається із застосуванням основного модулю взаємодії із мапою. Робота із ним передбачає такі етапи:

- Ініціалізація мапи, шляхом з’єднання із Google Maps API;
- Візуалізація мапи у відведеному для неї вікні;
- Налаштування прослуховування подій, за допомогою якого мапа має

змогу інтерактивно реагувати на користувацькі натиснення і зчитувати координати обраного місця;

- Якщо маршрут не новий -- завантаження даних про маршрут із бд за його унікальним ID, завантаження списку маркерів для поточного маршруту, завантаження даних про автора маршруту;
- Налаштування передачі координат маркерів сервісу побудови маршрутів для візуального поєднання точок у цільний маршрут;
- Ініціалізація вікна для виводу деталей щодо кожної обраної точки на маршруті.

При натисканні на точку на мапі, новий маркер додається до маршруту і виводиться на екран. Після невеликої затримки, оновлені дані направляються у модуль сховища, звідки відбувається запит до бази даних для збереження оновленого переліку точок маршруту.

При створенні нового маршруту, після того, як користувач задасть усі бажані точки та заповнить деталі маршруту, він має натиснути на кнопку “зберегти”, після чого новий маршрут буде збережено у колекцію маршрутів бази даних. Вже після цього він зможе отримати унікальне посилання та сумісно редагувати маршрут.

3.5. Висновок до розділу 3

У розділі 3 було спроектовано та реалізовано веб-додаток для сумісного планування і збереження туристичних маршрутів, який надає можливість користувачам реєструватися і авторизуватися у системі, створювати, переглядати, оцінювати та змінювати туристичні маршрути, зберігати обрані у власний список для швидкого доступу, а також

переглядати маршрути із найбільшою кількістю користувацьких вподобань.

При розробці використовувались теоретичні основи, досліджені у розділі 1, та технології, описані у розділі 2. Таким чином, отримана система є гнучкою, функціональною, легкою у масштабуванні і підтримці, адже була створена із використанням сучасних методик і практик.

ВИСНОВКИ

У процесі виконання випускної кваліфікаційної бакалаврської роботи було досліджено вплив соціалізації туристичної системи на задоволення користувачів та рівень комфорту під час планування маршруту разом із однодумцями або друзями. Було розглянуто способи підвищити комфорт сумісного планування або полегшити вибір готового маршруту туристичної подорожі.

В ході виконання роботи я:

1. Дослідила теоретичні основи побудови соціальних веб-систем та систем туристичного спрямування;
2. Проаналізувала рішення і технології розробки веб-додатків;
3. Розробила і впровадила веб-додаток для сумісного планування і збереження туристичних маршрутів з використанням наступних технологій: JavaScript, Vue.js, Firebase.

Після реалізації програмного продукту я отримала практичний досвід роботи із сервісами Google Cloud, що дозволяють реалізувати автентифікацію за допомогою сесій та Cookie, а також налагодити real-time інтеракцію між різними користувачами системи.

Також я поглибила свої знання у побудові клієнт-серверних програм, проектуванні архітектури таких додатків та використанні хмарної бази даних.

За результатами роботи було реалізовано веб-додаток, що коректно працює у більшості операційних систем та браузерів, поєднує кращі практики розробки веб-додатків та виконує усі поставлені перед ним функціональні задачі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

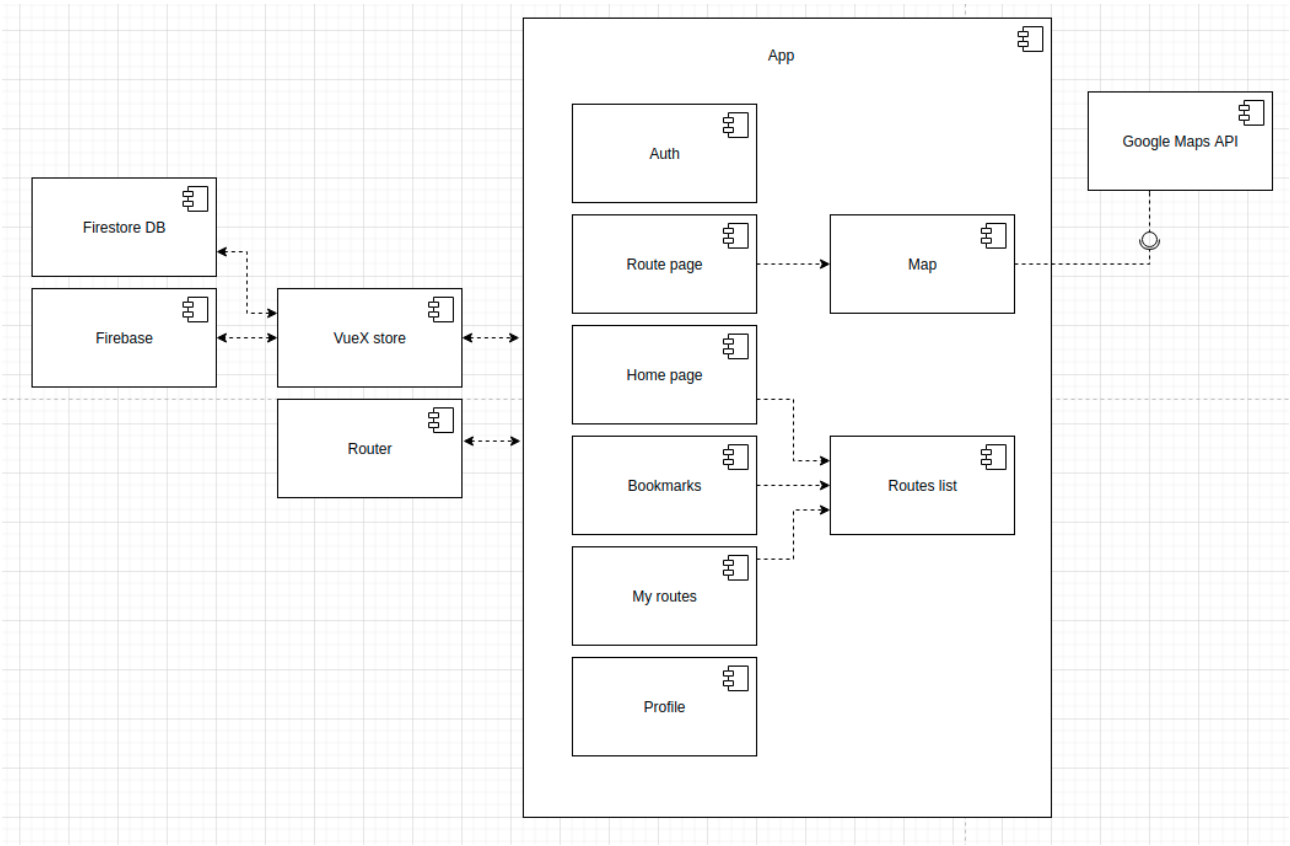
1. Архитектура веб-приложения [Електронний ресурс] – Режим доступу до ресурсу: https://itanddigital.ru/webapplications
2. Geisberger R., Sanders P., Schultes D., Delling D. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks, LNCS, 2008, Vol. 5038.
3. Route Planning [Електронний ресурс] – Режим доступу до ресурсу: https://patents.google.com/patent/US20140200807A1/en
4. Офіційний Блог Google Україна [Електронний ресурс] – Режим доступу до ресурсу: https://ukraine.googleblog.com/2020/02/google-15-10.html
5. angular vs react vs vue [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmtrends.com/angular-vs-react-vs-vue
6. Введение в Web API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/JavaScript/Client-side_web_APIs/Introduction
7. Andrew Lombardi, WebSocket: Lightweight Client-Server Communications — 2015.
8. Social Networking App Technology Stack [Електронний ресурс] – Режим доступу до ресурсу: https://yalantis.com/blog/social-networking-app-technology-stack-how-to-develop-social-apps/

9. I
ncremental Model [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/software-engineering-incremental-model

ДОДАТКИ

Додаток А

UML-діаграма компонентів



Додаток Б

Реалізація основного функціоналу системи

Д.1.

Реалізація функцій авторизації та реєстрації

```
async autologin({state, commit}) {
    state.currentUser = firebase.auth().currentUser;
  },
  async logIn({state, commit}, userData) {
    try {
      await firebase.auth().signInWithEmailAndPassword(userData.email,
userData.password);
      const user = firebase.auth().currentUser;
      state.currentUser = user;
    } catch (e) {
      alert(e);
    }
  },
  async register({state, commit}, userData) {
    try {
      await
firebase.auth().createUserWithEmailAndPassword(userData.email,
userData.password);
      const user = firebase.auth().currentUser;
      await user.updateProfile({displayName: userData.username});
      state.currentUser = user;
      alert('Welcome!');
```

```
      await db.collection('users').doc(user.uid).set({
        bookmarks: [],
        following: [],
        id: user.uid,
        regdate: new Date(),
        username: user.displayName,
        created: 0,
        contributed: 0,
        starred: 0,
      });
    } catch (e) {
      alert(e);
    }
  },
```

Д.2.

Ініціалізація мапи

```
data() {
    return {
      markers: [],
      labels: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
    }
  },
  created() {
    const loader = new Loader({
      apiKey: 'AIzaSyBRYFJeCBqzrPxWT5hqid9u5Nm******',
```

```
  version: "weekly",
  libraries: ["places"],
});

const mapOptions = {
  center: { lat: 50.450, lng: 30.523 },
  zoom: 8,
};

loader
    .load()
    .then(() => {
      const GoogleMaps = window.google?.maps;
      this.GM = GoogleMaps;

      const map = new GoogleMaps.Map(
        document.getElementById('map-view'),
        mapOptions,
      );
      this.map = map;

      map.addListener("click", (event) => {
        this.addMarker(event.latLng);
      });

      const input = document.getElementById("search");
      const searchBox = new GoogleMaps.places.SearchBox(input);

map.controls[GoogleMaps.ControlPosition.TOP_LEFT].push(input);
```

```
        });
    },
    methods: {
      addMarker(location) {
        const marker = new this.GM.Marker({
          position: location,
          label: this.labels[this.markers.length],
          map: this.map,
        });
        this.markers.push(marker);

        marker.addListener("click", () => {
          this.$emit('toggleInfo', 'info');
        });
      },
    },
```

Д.3.

Код компоненту “Маршрут”

```
components: {
    MapInit,
  },
  data() {
    return {
      id: 0,
      link: window.location.href,
      searchTerm: '',
```

```
    search: '',
    routeData: {},
    isNew: false,
  };
},
created() {
  this.id = this.$route.params.id;
  this.$store.dispatch('getRoutes')
      .then(a => {
        if (!this.$store.getters.getRouteById(this.id)) {
          this.routeData = {
            author_id: this.$store.state.currentUser.uid,
            author_name: this.$store.state.userData.username,
            country: '',
            is_editable: true,
            length: 0,
            likes: 0,
            title: 'Your awesome route title',
            from: null,
            to: null,
          };
          this.isNew = true;
          return;
        }
        this.routeData = this.$store.getters.getRouteById(this.id).data;
      });
},
methods: {
  handleInput() {
```

```
            this.search = this.searchTerm;
            this.searchTerm = '';
          },
          handleLink() {
            const area = document.querySelector('#copy');
            area.select();
            area.setSelectionRange(0, 99999);
            document.execCommand('copy');
            alert(`Your link - ${this.link} - is copied to clipboard!`);
            window.getSelection().removeAllRanges();
          },
          flipFav() {
            this.$store.dispatch('update', {field: 'bookmarks', value:
this.id}).then(() => {
              alert('Saved!');
            });
          },
          flipLike() {
            this.$store.dispatch('updateRoute', {routeObj: {likes:
++this.routeData.likes}, id: this.id});
          },
          handleSave() {
            if (this.isNew) {
              this.$store.dispatch('createRoute', this.routeData).then(id => {
                this.routeData.id = id;
                this.$router.push({ path: `/route/${id}` });
              });
              return;
            }
```

```
            this.$store.dispatch('updateRoute', {routeObj: this.routeData, id:
this.id}).then(() => {
              alert('Saved!');
            });
          }
        },
```

Д.4.

Деякі функції роботи із колекцією маршрутів у БД

```
async getMyRoutes({state, commit}) {
      const r = [];
      await db.collection('routes').where('author_id', "==",
state.currentUser.uid).get().then((res) => {
        res.forEach(i => r.push({id: i.id, data: i.data}));
      });
      return r;
    },
    async createRoute({state, commit}, routeObj) {
      let id;
      await db.collection('routes')
        .add(routeObj)
        .then(docRef => {
          id = docRef.id;
          return docRef.id;
        })
        .catch(e => {
          alert(e);
```

```
    });
  return id;
},
async updateRoute({state, commit}, {routeObj, id}) {
  await db.collection('routes').doc(id)
    .update(routeObj)
    .then(() => {
      console.log("Document successfully updated!");
    })
    .catch(e => {
      alert(e);
    });
},
```