МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІМ. Ю.М. ПОТЕБНІ ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Порівняльний аналіз фреймворків Angular та React на прикладі сайту туристичного агентства**

Виконав: студент 2 курсу, групи 8.1212-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
В. В. Малиш
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., І.А.Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел» П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра _____електроніки, інформаційних систем та програмного__ забезпечення

Рівень вищої освіти _____________другий (магістерський)________

Спеціальність ___121_Інженерія програмного забезпечення______
(код та назва)

Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри__________Т.В. Критська_
“_01_” __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

_____Малиш Вікторії Вадимівні_____
(прізвище, ім’я, по батькові)

1.Тема роботи __Порівняльний аналіз фреймворків Angular та React на прикладі сайту туристичного агентства__________________________

керівник роботи ___Ірина Анатоліївна Скрипник, к.ф.-м.н.._ доцент_
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від ___09.10. 2023 р. №1577-с

2. Строк подання студентом кваліфікаційної роботи________30.11.2023______

3. Вихідні дані магістерської роботи

- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми використання фреймворків для створення веб-сайтів;
- створення двох веб-сайтів з використанням фреймворків Angular та React, їх опис;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

15 слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання ______01.09.2022_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерсь-кої роботи	Примітка
1	Аналіз предметної області	02.09-10.09.23	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.23	виконано
3	Аналіз існуючих фреймворків та мов програмування	13.09-14.09.23	виконано
4	Дослідження фреймворку React	15.09-20.09.23	виконано
5	Дослідження фреймворку Angular	21.09-26.09.23	виконано
6	Порівняння фреймворків за характеристиками	27.09-28.09.23	виконано
7	Збір необхідної інформації для розробки застосунків	29.09-13.10.23	виконано
8	Аналіз моділув та вибір стеку для кожного застосунку	14.10-16.10.23	виконано
9	Розробка бекенд-застосунку	17.10-19.10.23	виконано
10	Розробка фронтенд-застосунку на React	20.10-09.11.23	виконано
11	Розробка фронтенд-застосунку на Angular	10.11-17.11.23	виконано
12	Порівняльний аналіз фреймворків React та Angular на базі практичного дослідження	15.11-19.11.23	виконано
13	Оформлення звіту	20.11-30.11.23	виконано

Студент ______________________ ______Малиш В. В.______
(підпис) (прізвище та ініціали)

Керівник роботи ______________ ______Скрипник І. А.______
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер ______________ ______Скрипник І. А.______
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок – 109

Рисунків – 12

Джерел – 26

Малиш В.В. Порівняльний аналіз фреймворків Angular та React на прикладі сайту туристичного агентства : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І.А. Скрипник Запоріжжя : ЗНУ, 2023. 109 с.

Мета і завдання дослідження полягають у порівнянні характеристик відомих фреймворків мови javascript, а саме React та Angular, проаналізувати їх переваги та недоліки, популярність використання для веб-застосунків, продуктивність виконання.

Для виконання дослідження було створено сайт турагентства, в якому у frontend-частині створені два застосунки: веб-застосунок, написаний з використанням фреймворку React, та веб-застосунок, написаний з використанням фреймворку Angular. Також розроблена backend-частина, написана на Node.js з використанням бази даних PostgreSQL.

У процесі дослідження було порівняно фреймворки Angular та React та досліджено, для якого рівня проектів їх краще використовувати. Як результат, були розроблені два сайти турагенства на обох фреймворках. Дані сайти мають однаковий функціонал та використовують одий бекенд. Окрім цього, було досліджено чимало бібліотек для обох фреймворків.

Ключові слова: фреймворк, веб-сайт, JavaScript, TypeScript, Python, React, Angular, Vue.js, Node.js, PostgreSQL,

SUMMARY

Pages: 109

Figures: 12

Sources: 26

Malysh V.V. Comparative analysis of Angular and React frameworks on the example of a travel agency website: master's qualification thesis of specialty 121 "Software engineering" / science. manager I.A. Skrypnyk Zaporizhzhia: ZNU, 2023. 109 p.

The aim and objectives of the research are to compare the characteristics of well-known frameworks of the javascript language, namely React and Angular, to analyze their advantages and disadvantages, the popularity of use for web applications, and the productivity of execution.

To carry out the research, a travel agency site was created, in which two applications were created in the frontend part: a web application written using the React framework, and a web application written using the Angular framework. A backend part written in Node.js using the PostgreSQL database has also been developed.

In the research process, Angular and React frameworks were compared and it was investigated for which level of projects it is better to use them. As a result, two travel agency sites were developed on both frameworks. These sites have the same functionality and use the same backend. In addition, many libraries for both frameworks have been explored.

Keywords: framework, website, JavaScript, TypeScript, Python, Re-act, Angular, Vue.js, Node.js, PostgreSQL,

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ОБГРУНТУВАННЯ ТЕМИ 15
1.1 Обґрунтування мови програмування 15
1.2 Порівняльний аналіз мов програмування 16
1.2.1 JavaScript 16
1.2.2 HTML/CSS 16
1.2.3 Python 17
1.3 Порівняльний аналіз фреймворків для JavaScript 18
1.3.1 React 18
1.3.2 Angular 19
1.3.3 Vue 20
1.4 Обґрунтування теми 21
1.4.1 Аналіз Аналогів 21
РОЗДІЛ 2 ПОРІВНЯННЯ ФРЕЙМВОРКІВ 27
2.1 Дослідження фреймворка React 27
2.2 Дослідження фреймворка Angular 28
2.3 Порівняння за популярністю 29
2.4 Легість у освоєнні 31
2.5 Підтримка спільноти 32
РОЗДІЛ 3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ 34
3.1 Технічне завдання 34
3.2 Стек технологій 34
3.2.1 Redux 34
3.2.2 Stripe 35
3.2.3 Semantic UI 36
3.2.4 PostgreSQL 37
3.2.5 Sequelize 38

3.2.6 Lodash ... 38
3.2.7 Day.js ... 39
3.3 Розробка бекенд застосунку ... 40
3.3.1 Налаштування оточення розробки ... 40
3.3.2 Створення застосунку ... 40
3.4 Розробка фронтенд застосунку ... 62
3.4.1 Створення застосунку ... 63
3.4.2 Налаштування маршрутизації ... 66
3.4.3 Створення сторінки авторизації ... 68
3.4.4 Налаштування стейт менеджмента ... 76
3.4.5 Інтеграція з бекендом ... 88
РОЗДІЛ 4 ПОРІВНЯННЯ ФРЕЙМВОРКІВ ... 95
4.1 Порівняння на базі розроблених веб-застосунків ... 95
4.2 Мова програмування ... 95
4.3 Інструменти та конфігурація ... 98
4.4 Архітектура та стейт менеджмент ... 99
4.5 Кількість бібліотек ... 101
4.6 Проблеми, що виникали при роботі з Angular ... 105
4.7 Проблеми, що виникали при роботі з React ... 105
ВИСНОВКИ ... 107
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 108

ВСТУП

Актуальність теми

Загалом, веб-сайти були і залишаються необхідною складовою сучасного інтернет-простору, виконуючи різноманітні функції та надаючи користувачам доступ до різноманітного контенту та сервісів. Вони стають платформою для взаємодії між користувачами та брендами, надаючи можливість обміну інформацією, отримання зворотного зв'язку та спілкування через різноманітні канали.

Веб-сайти є основним засобом розповсюдження інформації. Вони дозволяють компаніям, організаціям та особам представляти свою діяльність, надавати оновлену інформацію та взаємодіяти зі своєю аудиторією. Не менш необхідним інструментом вони є і для онлайн-торгівлі. Такі інтернет-магазини дозволяють компаніям продавати товари та послуги в інтернеті, що є надзвичайно важливим в сучасному економічному середовищі.

Крім цього, веб-сайти виступають як освітні та розважальні платформи, надаючи користувачам доступ до великої кількості відомостей, навчальних ресурсів, стрімів, відео та іншого контенту.

З розвитком мобільних технологій важливістю стає мобільна сумісність веб-сайтів. Забезпечення гнучкості та зручності використання на різних пристроях є ключовим аспектом для забезпечення актуальності. Веб-сайти, оптимізовані для пошукових систем, мають більше можливостей привертати увагу цільової аудиторії. Ефективна SEO-стратегія важлива для забезпечення відвідуваності та видимості в мережі.

Проте розробка Веб-сайту на чистому html javaScript та css потребує більше часу та терпіння, адже код більш важкий та неструктурований, виникають незручності та проблеми при додаванні функціоналу. Для облегшення створення сайтів є фреймворки. Використання фреймворків дозволяє розробникам фокусуватися на функціональності та високорівневих завданнях,

мінімізуючи тим самим час та зусилля, витрачені на рутинні операції. Це робить фреймворки незамінними інструментами для розробників у сучасній веб-розробці.

Фреймворки вже включають базовий код та інфраструктуру, що дозволяє розробникам уникнути потреби писати багато стандартного коду "з нуля". Це прискорює процес розробки і дозволяє швидше випускати готові продукти на ринок. Також фреймворки надають чітку структуру для організації коду. Це робить код більш читабельним та легше зрозумілим для розробників, сприяючи підтримці та розвитку проекту в подальшому.

Фреймворки часто оптимізовані для високої продуктивності, що дозволяє створювати веб-додатки, які працюють швидко і ефективно, особливо коли вони використовують кращі практики розробки. До того ж багато з них мають вбудовані засоби для захисту від різних видів атак, таких як впровадження SQL-запитів, міжсайтовий скриптинг (XSS), атаки на зміну сесій тощо. Це спрощує завдання забезпечення безпеки веб-додатка.

Не менш важливим фактором є те, що фреймворки мають активні спільноти розробників, що робить легшим знаходження рішень для проблем та отримання підтримки від інших фахівців.

Мета і завдання дослідження

Метою роботи є дослідження відомих фреймворків, а саме React та Angular, за наступними характеристиками:

- мова програмування;
- інструменти та конфігурація;
- архітектура;
- кількість бібліотек;
- продуктивність.

Відповідно до мети ставляться такі завдання:

1. Аналіз мов програмування для розробки сайтів.

2. Аналіз фреймворків для обраної мови програмування та вибір двох з них.
3. Попередній аналіз фреймворків стосовно теорії, пошук важливої інформації, перевірка підтримки спільноти щодо проблем, пов'язаних з фреймворком.
4. Аналіз реальних сайтів для створення турагенства.
5. Створення бекенду.
6. Створення клієнтського застосунку на React.
7. Створення клієнтського застосунку на Angular.
8. Порівняння фреймворків на готових застосунках за критеріями, описаними вище.
9. Написання висновків.

Об’єкт дослідження

Об’єктами дослідження є фреймворки React та Angular, існування яких полегшує розробку веб-сайтів.

Предмет дослідження

Веб-сайти, проєктування та розробка яких виконується за допомогою обраних фреймворків.

Методи дослідження

Для розв’язання представлених завдань використовуються такі методи дослідження:

1. Аналіз джерел про фреймворки та пошук інформації.
2. Створення застосунків на цих фреймворках.
3. Порівняльний аналіз фреймворків на основі програмних продуктів.
4. Синтез отриманих результатів досліджень.

Наукова новизна одержаних результатів

Удосконалення можливостей роботи з сайтом для користувача, особливості, що відрізняють власну розробку від аналогів.

Практичне значення одержаних результатів

На базі отриманих результатів можна визначити, для яких застосунків який фреймворк буде зручнішим, який з фреймворків легший в навчанні, ознайомитись з проблемами, що виникають при використанні фреймворку та визначити методи їх вирішення, визначити плюси та мінуси використання фреймворку.

Апробація результатів кваліфікаційної роботи магістра

Результати роботи було представлено на науково-технічних конференціях студентів, магістрантів, аспірантів, молодих вчених викладачів і опубліковані в збірнику наукових праць студентів, аспірантів і молодих вчених «Молода наука -2023» [12].

Глосарій

Фреймворк (Framework) — це структура або склад програмного забезпечення, яка надає загальний фундамент для розробки програм та включає у себе багато заздалегідь написаних, готових до використання компонентів, бібліотек і інструментів. Фреймворки забезпечують рамки для розробки, що допомагає розробникам прискорити процес та стандартизувати його.

Веб-сайт (Website) — це набір взаємопов'язаних веб-сторінок, доступних через Інтернет, що мають спільну адресу (URL) та часто містять текст, зображення, відео та інші мультимедійні елементи.

JavaScript — це високорівнева, інтерпретована мова програмування, яка використовується для створення інтерактивних веб-сайтів. Вона може використовуватися як на стороні клієнта (браузер), так і на стороні сервера (за допомогою Node.js).

TypeScript — це розширення мови JavaScript, яке додає статичні типи та інші функції до мови, що полегшує розробку великих та складних програм.

Python — це високорівнева, інтерпретована мова програмування, яка використовується для різноманітних завдань, включаючи веб-розробку, наукові обчислення, штучний інтелект та інші.

HTML (Hypertext Markup Language) — це стандартна мова розмітки для створення та відображення веб-сторінок у браузері.

CSS (Cascading Style Sheets) — це мова стилів, яка використовується для визначення зовнішнього вигляду веб-документів, написаних мовою HTML або XML.

React — це бібліотека для створення інтерфейсів користувача у веб-додатках. Вона розроблена компанією Facebook та дозволяє створювати ефективні та перевикористовувані компоненти.

Angular — це фреймворк для розробки веб-додатків, розроблений компанією Google. Він використовує TypeScript і пропонує компонентний підхід до розробки.

Vue.js — це прогресивний JavaScript-фреймворк для створення інтерфейсів користувача. Він легкий та простий у використанні.

Node.js — це середовище виконання для JavaScript на сервері, яке дозволяє виконувати код JavaScript поза браузером.

PostgreSQL — це об'єктно-реляційна система управління базами даних (ORDBMS), яка підтримує розширення SQL та забезпечує надійність та високу продуктивність.

Sequelize — це ORM (Object-Relational Mapping) для Node.js, яке дозволяє взаємодіяти з базами даних PostgreSQL та іншими, використовуючи JavaScript.

Redux — це бібліотека для керування станом веб-додатків у JavaScript, часто використовується з бібліотеками та фреймворками, такими як React.

Функція (Function) — це блок коду, який може виконуватися при виклику. Вона може приймати аргументи та повертати значення.

ORM (Object-Relational Mapping) — це технологія програмування, яка дозволяє зв'язувати об'єктно-орієнтовану модель даних з реляційною базою даних. Це спрощує взаємодію з базою даних, дозволяючи використовувати об'єктно-орієнтовані структури в коді.

UI (User Interface) — це інтерфейс користувача, який представляє собою всі елементи та взаємодію між користувачем та комп'ютерною системою. UI включає в себе графічний дизайн, текстові елементи, кнопки та інші елементи, які дозволяють користувачам взаємодіяти з програмами чи веб-сайтами.

Semantic (Semantic) — це властивість, пов'язана з семантикою, тобто змістом та значенням. У веб-розробці та HTML розмітці, термін "семантичний" означає використання HTML-тегів для їх природнього значення, що покращує якість коду та полегшує розуміння контенту як людьми, так і машинами.

Фронтенд (Frontend) — це частина веб-додатка або програми, яка відповідає за відображення та взаємодію з користувачем. Вона включає в себе інтерфейс, який відображається в браузері або іншому клієнтському пристрої.

Бекенд (Backend) — це частина веб-додатка або програми, яка відповідає за обробку логіки, взаємодію з базою даних та виконання серверних операцій. Бекенд працює на сервері та обслуговує запити від фронтенду.

API (Application Programming Interface) — це набір правил та інструкцій, які дозволяють різним програмам взаємодіяти одна з одною. У веб-розробці, API визначає спосіб комунікації між різними компонентами програмного забезпечення, такими як фронтенд та бекенд.

Stripe — це компанія та платіжна платформа, яка дозволяє компаніям та розробникам приймати онлайн-платежі та керувати фінансами.

Lodash — це бібліотека утиліт для JavaScript, яка надає корисні функції для роботи з об'єктами, масивами та іншими типами даних.

Інтерфейс (Interface) — це точка взаємодії між користувачем та системою, яка може бути графічною (GUI) або командною (CLI). У контексті програмування, інтерфейс може також вказувати на визначені конструкції для спілкування між компонентами програми.

Стилі (Styles) — це правила та визначення, які визначають зовнішній вигляд елементів на веб-сторінці або у додатку. У веб-розробці, стилі визначаються за допомогою CSS та використовуються для задання кольорів, шрифтів, розмірів тощо.

Сайт (Site) — це набір взаємопов'язаних веб-сторінок, що мають спільну тематику або призначення, і доступний через Інтернет.

Модуль (Module) — це самостійна частина програмного забезпечення, яка має певний функціонал та може бути використана незалежно від інших частин системи.

Бібліотека (Library) — це набір функцій або ресурсів, які можуть бути використані для розробки програм або веб-сайтів. Бібліотеки полегшують розробку, оскільки реалізують певні функціональності, які можна використовувати безпосередньо.

Авторизація (Authorization) — це процес перевірки прав доступу користувача до певних ресурсів або функцій системи. Це визначає, чи має користувач дозвіл використовувати конкретний ресурс чи виконувати певні операції.

Аутентифікація (Authentication) — це процес перевірки ідентичності користувача, який зазвичай включає в себе введення ідентифікаційних даних, таких як ім'я користувача та пароль. Після успішної аутентифікації користувач отримує доступ до системи або ресурсів.

РОЗДІЛ 1 ОБҐРУНТУВАННЯ ТЕМИ

1.1 Обґрунтування мови програмування

Для вибору мови програмування спочатку проведено аналіз рейтингу мов програмування у світі [1]. Спершу було досліджено рейтинги на Stackoverflow, які показали, що мова JavaScript вже більше 8 років знаходиться на першому місці серед запитів на платформі Stackoverflow [2], на другому місці — HTML/CSS, що є стандартними мовами розмітки та стилізації для створення структури веб-сторінок, і на третьому місці — Python.

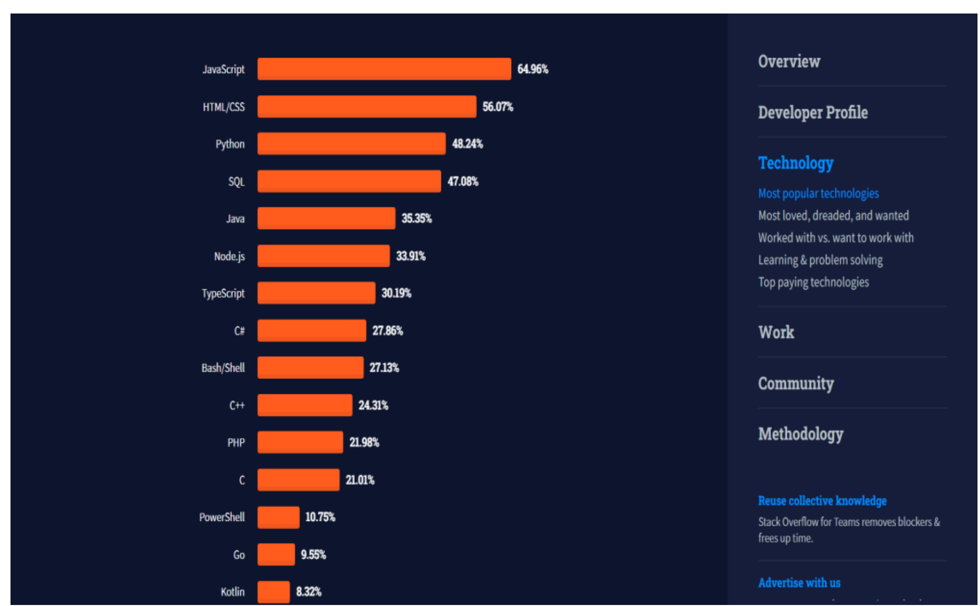

Рисунок 1 — Рейтинг запитів серед мов програмування за Stackoverflow

Далі було проаналізовано ці три мови шляхом проведення порівняльного аналізу та виділено їх переваги та недоліки.

1.2 Порівняльний аналіз мов програмування

1.2.1 JavaScript

Мова JavaScript [13] була створена компанією Netscape Communications в 1995 році. Перші версії мови були спрямовані на веб-браузери і використовувалися для додавання інтерактивності до веб-сторінок.

Переваги:

- Універсальність. JavaScript може використовуватися на фронтенді та бекенді (за допомогою Node.js).
- Широкий вибір фреймворків і бібліотек. Є багато інструментів, таких як React, Angular і Vue.js, що полегшують значно розробку веб-додатків.
- Асинхронний код. Підтримує асинхронне програмування, що робить його ідеальним для веб-серверів і взаємодії зі сторонніми API.
- Велика спільнота. Має велику та активну спільноту розробників.

Недоліки:

- Проблеми зі сумісністю. Різні браузери можуть інтерпретувати JavaScript по-різному, що може викликати проблеми сумісності.
- Потенційно погана безпека. Вразливості можуть виникнути через поганий код або недостатній захист.

1.2.2 HTML/CSS

HTML (HyperText Markup Language) і CSS (Cascading Style Sheets) були розроблені як стандарти для створення та стилізації веб-сторінок. HTML була вперше описана Тімоті Бернерс-Лі в 1991 році, а CSS була запропонована наступного десятиліття.

Переваги:

- Чіткість розділення завдань: HTML відповідає за структуру сторінки, а CSS за її стиль, що сприяє чіткості та обслуговуванню коду.
- Зручність для дизайнерів: CSS надає велику кількість можливостей для креативного дизайна сторінок.

- Підтримка всіма браузерами: HTML і CSS — це стандарти, і підтримуються всіма сучасними браузерами.

Недоліки:

- Не є загальнопризначеними мовами програмування: HTML і CSS призначені для створення і відображення вмісту в браузерах, вони не здатні виконувати складну бізнес-логіку.
- Обмежені можливості для інтерактивності: Без JavaScript важко створювати веб-додатки з високим рівнем інтерактивності.

1.2.3 Python

Python був створений Гвідо ван Россумом і вперше випущений у 1991 році. Він був розроблений як загальнопризначена мова програмування з акцентом на простоту і читабельність коду.

Переваги:

- Читабельність коду: Python славиться своєю простотою і легкістю читання, що сприяє швидкому розвитку і підтримці проектів.
- Велика спільнота і бібліотеки: Python має велику кількість бібліотек для різних завдань і активну спільноту розробників.
- Універсальність: Python використовується для веб-розробки, наукових досліджень, аналізу даних, штучного інтелекту, та інших галузей.
- Крос-платформеність: Python підтримується на багатьох операційних системах.

Недоліки:

- Продуктивність: У деяких випадках Python може бути менш продуктивним порівняно з мовами з низькорівневою маніпуляцією пам'яттю.
- Не такий швидкий, як C/C++: Для обчислювально інтенсивних завдань Python може бути повільним порівняно з мовами, які компілюються.

В результаті проведення аналізу зрозуміло, що JavaScript надає широкий набір інструментів і можливостей, що полегшують проведення досліджень у

веб-середовищі. Він дозволяє створювати інтерактивні додатки, обробляти дані, взаємодіяти зі сторонніми ресурсами та інтегрувати багато інших функціональних можливостей, тому було обрано саме цю мову.

1.3 Порівняльний аналіз фреймворків для JavaScript

Проаналізувавши рейтинги Stackoverflow [2] по фреймворкам JavaScript було виділено три найпопулярніші фреймворки для фронтенду, перше місце займає React, на другому місці Angular і на третьому — Vue.

Щоб обрати фреймворки для порівняння було проведено порівняльний аналіз цих трьох вреймворків.

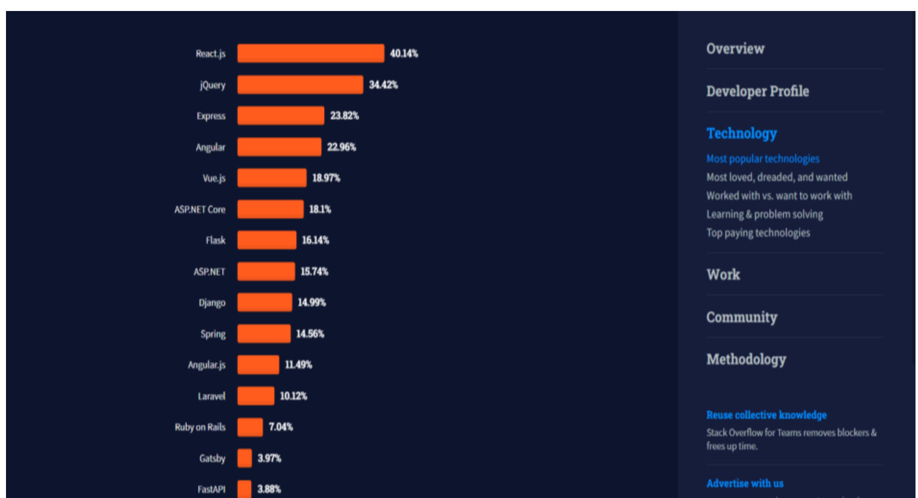

Рисунок 2 — Рейтинг запитів серед фреймворків JavaScript за Stackoverflow

1.3.1 React

React [3] був розроблений компанією Facebook і вперше випущений у 2013 році.

Цей бібліотечний фреймворк здобув популярності завдяки декларативній моделі програмування та відмінній швидкості.

Переваги:

- Висока продуктивність: React відомий своєю високою швидкістю завдяки віртуальному DOM (Document Object Model), що робить його відмінним вибором для розробки вимогливих за ресурсами додатків.
- Сильна спільнота: Має велику та активну спільноту розробників, що пропонує безліч ресурсів та допомогу.
- Широкий вибір бібліотек і розширень: Існує багато сторонніх бібліотек та компонентів, які допомагають прискорити розробку.
- Фокус на компонентах: React побудований навколо концепції компонентів, що сприяє модульності і перевикористанню коду.

Недоліки:

- Вимагає вибору інших інструментів: React це лише бібліотека для користувача, тому для повноцінного фронтенд-фреймворку потрібно вибрати більше інструментів, таких як Redux для керування станом додатку.
- Велика кількість концепцій: Для початківця може знадобитися час, щоб оволодіти всіма концепціями, такими як віртуальний DOM та життєвий цикл компонентів.

1.3.2 Angular

Angular (перша версія, також відома як AngularJS) [4] був розроблений Google і випущений у 2010 році.

У 2016 році вийшла версія Angular 2 і відтоді Angular відрізняється від AngularJS.

Переваги:

- Спрощений життєвий цикл компонентів: Angular надає ясний життєвий цикл для компонентів, що сприяє контролю і управлінню ресурсами;
- Повна платформа: Angular надає повністю інтегрований фреймворк для розробки веб-додатків, включаючи керування станом, маршрутизацію, інтернаціоналізацію та багато іншого.

- Спрощений життєвий цикл компонентів: Angular надає повністю інтегрований фреймворк для розробки веб-додатків, включаючи керування станом, маршрутизацію, інтернаціоналізацію та багато іншого.

Недоліки:

- Складність: Angular може бути важким для вивчення для новачків через велику кількість концепцій і обов'язків.
- Великі ресурсовитрати: Додатки на Angular можуть бути більшими за аналогічні додатки на інших фреймворках, що може вимагати більше ресурсів для роботи.

1.3.3 Vue

Vue.js [5] був створений китайським розробником Єваном Ю у 2014 році. Він швидко набув популярності завдяки своїй простоті та легкості використання.

Назва "Vue" походить від слова "view" (вигляд) і вказує на те, що цей фреймворк призначений для роботи з відображенням частин веб-сторінки.

Переваги:

- Легка інтеграція: Ви можете легко включити Vue.js у вже існуючі проекти або використовувати його для окремих компонентів на сторінці.
- Простота використання: Vue.js славиться своєю простотою і легкістю навчання, що робить його відмінним вибором для початківців і швидкого прототипування.
- Реактивність: Vue.js має просту та потужну систему реактивності, що сприяє автоматичному оновленню інтерфейсу при зміні даних.

Недоліки:

- Менша спільнота: Хоча спільнота Vue.js активно росте, вона все ще менша порівняно з React і Angular;
- Менше готових рішень: У порівнянні з React і Angular, кількість готових бібліотек і розширень для Vue.js менша.

1.4 Обґрунтування теми

Для вибору тематики проекту було проведено аналіз предметної області і обрано туристичне агенство.

Для вдосконалення сайту розглянуто кілька відомих сайтів турагенств та проаналізовано їх, виділено плюси та мінуси кожного за певними критеріями.

1.4.1 Аналіз Аналогів

1. «Join UP», виробник: Україна.

   Зображення сайту можна побачити на рисунку 3.

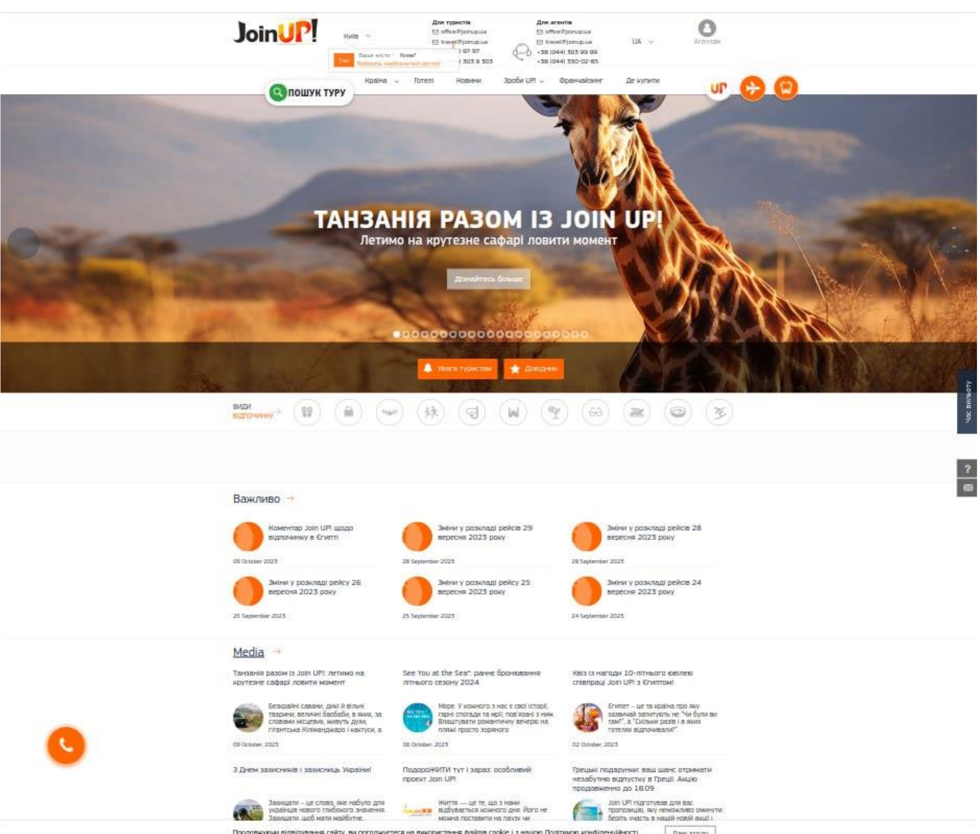

Рисунок 3 — Сайт «Join UP»

Основні функціональні можливості:

- Можливість забронювати квитки.

- Пошук турів.
- Розширений пошук по виду відпочинку, типу тура і т.д.
- Перегляд актуальної інформації та оголошень.
- Перегляд акцій та гарячих турів.
- Є зворотній зв'язок та служба підтримки.
- Локалізація, можливість зміни мови.

Переваги:

- Простий дизайн.
- Відсутня реклама.

Недоліки:

- Відсутня реєстрація особистого кабінету користувача.
- Некоректно працює фільтрація.
- Адаптивність. При деяких розмірах гаджета не дуже зручний інтерфейс для користувача.

2. «**Mouzenidis**» виробник: Україна

Зображення сайту можна побачити на рисунку 4.

Рисунок 4 — Сайт «Mouzenidis»

Основні функціональні можливості:

- Можливість забронювати квитки.
- Перегляд акцій та гарячих турів.
- Можливість створення особистого кабінету.
- Перегляд інформації про країну.

Переваги:

- Навігація.
- Локалізація, можливість зміни мови.
- Наявність відгуків та відео.

Недоліки:

- Адаптивність, сайт проблемно переглядати з маленького екрану.

- Переповнення зайвою інформацією.
- Реклама заважає переглядати сайт.

3. «**Tez tour**» виробник: Україна

Зображення сайту можна побачити на рисунку 5.

Основні функціональні можливості:

- Можливість забронювати квитки;
- Перегляд акцій та гарячих турів;
- Можливість створення особистого кабінету.

Переваги:

- Наявність навігації;
- Локалізація, можливість зміни мови;
- Сайт добре адаптований під будь-який розмір екрану;
- Наявність зворотнього зв'язку.

Недоліки:

- Не весь функціонал працює коректно;
- Сайт переповнений зайвою інформацією;
- Довгий час обробки запитів.

4. «**Anex tour**» виробник: Україна.

Зображення сайту можна побачити на рисунку 6.

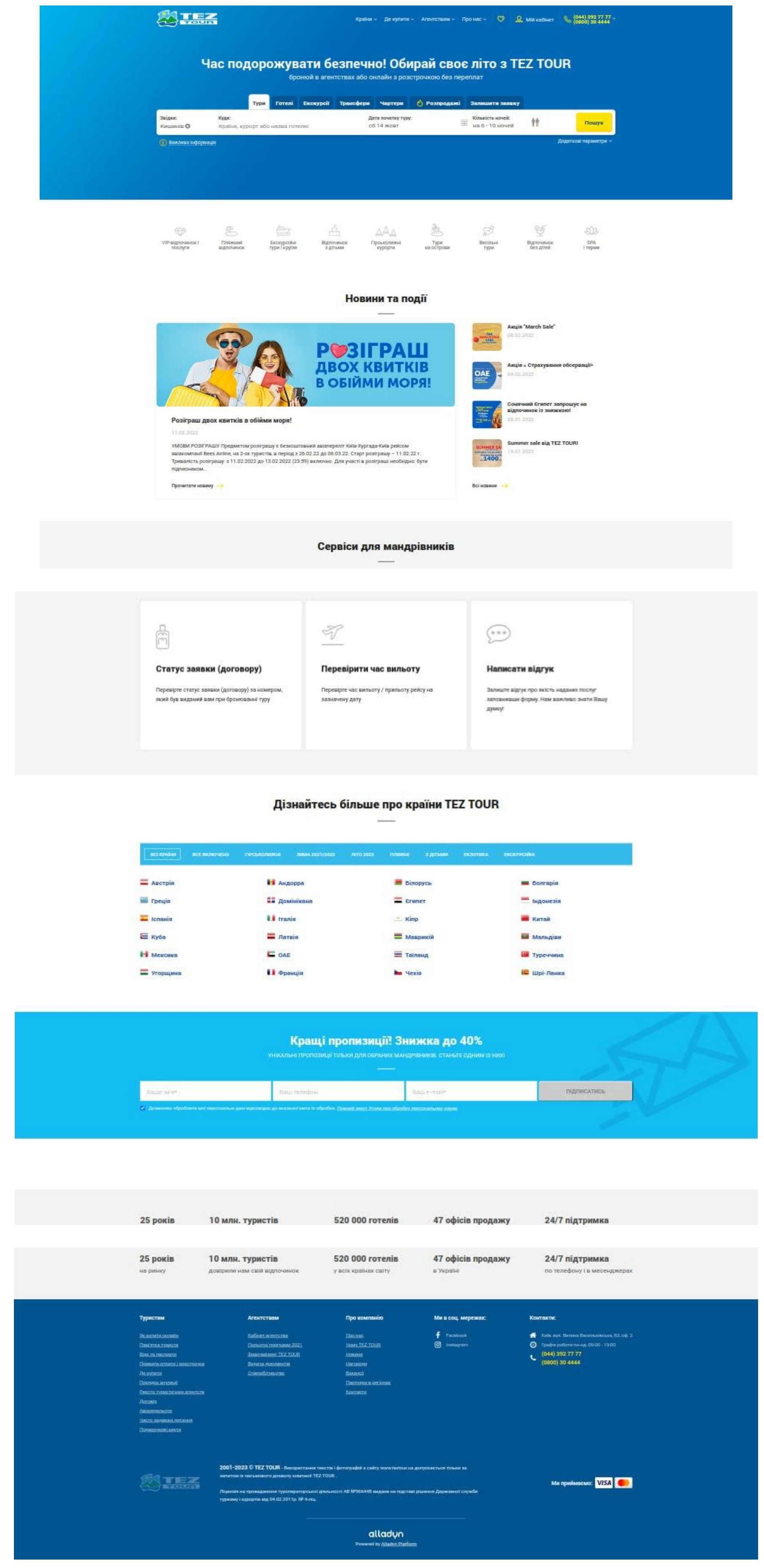

Рисунок 5 — Сайт «Tez tour»

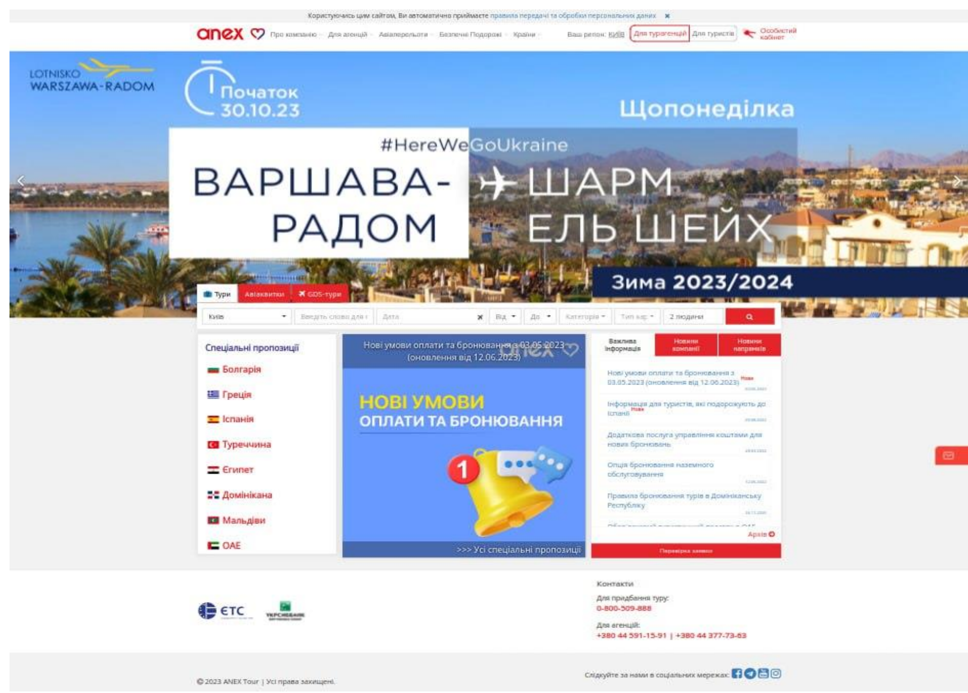

Рисунок 6 — Сайт «Anex tour»

Основні функціональні можливості:

- Можливість забронювати квитки;
- Перегляд акційних турів та гарячих путівок;
- Можливість створення особистого кабінету.

Переваги:

- Наявність зворотнього зв’язку;
- Наявність особистого кабінету.

Недоліки:

- Проблеми з адаптивністю на маленьких екранах;
- Багато вкладок, які не несуть важливої інформації.

Важливо розглянути особливості кожного фреймворку, проте для дослідницької роботи було обрано React та Angular, так як ці фреймворки більш популярні, розвинуті та мають більше можливостей та функціоналу.

РОЗДІЛ 2 ПОРІВНЯННЯ ФРЕЙМВОРКІВ

2.1 Дослідження фреймворка React

Реакт (React) — це потужна бібліотека JavaScript, створена командою Facebook, яка використовується для розробки веб-додатків з високою взаємодією та швидким оновленням інтерфейсу користувача. Одна з головних особливостей Реакт полягає у його вмінні ефективно взаємодіяти з віртуальним DOM, що призводить до високої продуктивності додатків.

Основний концепт Реакт — це "компоненти". Кожен компонент є незалежною частиною інтерфейсу та має свій власний стан і властивості. Це полегшує розробку та підтримку коду, особливо в масштабованих проектах. Компоненти можна вкладати один в одного, створюючи ієрархію, яка відображає структуру додатку.

Одна з сильних сторін Реакт — це використання JSX (JavaScript XML), що дозволяє вписувати HTML-подібний код безпосередньо в JavaScript. Це спрощує створення і розуміння структури компонентів та покращує читабельність коду.

Реакт також підтримує створення односторінкових додатків (SPA), де зміни сторінок відбуваються без перезавантаження сторінки, що забезпечує більш зручний та плавний досвід користувача.

За останні роки Реакт активно розвивався, додаючи нові функції та покращення, такі як хуки (hooks) [14] для роботи зі станом в функціональних компонентах. Хуки — це новий механізм в Реакт, метою якого є надання функціональним компонентам можливості використовувати стан та інші функціональності, які раніше були доступні лише в класових компонентах. Існує багато хуків, найпулярніші з них:

- *useState:* дозволяє функціональному компоненту використовувати локальний стан. Повертає масив, де перший елемент — це поточне значення стану, а другий — функція для його оновлення.

- *useEffect:* використовується для виконання побічних ефектів в компоненті. Приймає два аргументи: функцію, яка визначає ефект, і масив залежностей, що вказує, коли ефект повинен бути викликаний.
- *useContext:* використовується для отримання значення з контексту. Контекст дозволяє передавати значення глибоко в дерево компонентів, не передаючи їх через кожен компонент.
- *useReducer:* використовується для управління станом компонента за допомогою функції редуктора. Редуктор приймає поточний стан та дію, а повертає новий стан. Це особливо корисно для складних станів, де useState може стати громіздким.
- *useCallback* повертає мемоізовану версію функції, що корисно для оптимізації функцій, які передаються дочірнім компонентам.
- *useMemo* повертає мемоізоване значення та корисний при оптимізації важких обчислень.

Незважаючи на всі його переваги, важливо враховувати, що велика кількість концепцій та можливостей може зробити вивчення Реакт трошки складним для новачків.

Однак, коли розробник освоює основи, він може швидко стати продуктивним та використовувати всі можливості цієї потужної бібліотеки для створення вражаючих веб-додатків.

2.2 Дослідження фреймворка Angular

Angular — це фреймворк для розробки веб-додатків, створений компанією Google. Його можна розглядати як великий та повноцінний інструмент, який надає розробникам великий набір інструментів для побудови сучасних та масштабованих веб-додатків.

Однією з ключових концепцій Angular є використання TypeScript [15], що є розширенням JavaScript. TypeScript додає статичну типізацію до мови, що полегшує виявлення помилок на етапі розробки та поліпшує розуміння коду.

Основні складові Angular:

- **Компоненти:** додатки на Angular побудовані на основі компонентів. Компонент — це самодостатня одиниця, яка містить логіку та шаблон для відображення веб-сторінок. Компоненти дозволяють організувати код та взаємодіяти з іншими компонентами.
- **Модулі:** Angular використовує модульну архітектуру для організації коду. Модуль — це збірник компонентів, сервісів та інших ресурсів, які мають логічний зв'язок.
- **Сервіси:** сервіси використовуються для реалізації логіки, яка не пов'язана з відображенням. Сервіси дозволяють поділитися даними та функціональністю між різними частинами додатку.
- **Директиви:** директиви — це інструкції в шаблонах, які змінюють відображення DOM або надають новий функціонал.
- **Шаблони та двостороннє зв'язування:** Angular використовує шаблони для опису того, як компонент повинен відображати дані. Двостороннє зв'язування дозволяє автоматично оновлювати дані в шаблоні при їх зміні та навпаки.
- **Роутинг:** Angular має вбудований механізм для організації навігації між різними сторінками додатку без перезавантаження сторінки.

Angular є потужним інструментом для розробки веб-додатків, особливо для великих та складних проектів. Його інструменти та концепції дозволяють створювати ефективний та організований код.

2.3 Порівняння за популярністю

Обидва фреймворки є потужними інструментами для розробки веб-додатків, і вибір залежить від конкретних потреб проекту та особистих вподобань команди розробників.

Проте, порівнюючи ці два фреймворки, можна виділити деякі критерії, за якими легше визначити, для яких цілей ці фреймворки зручніші:

1. **Спрощення DOM:**

   React:

- Використовує віртуальний DOM, що дозволяє оптимізувати оновлення інтерфейсу шляхом оновлення тільки тих елементів, які змінилися.

  Angular:

- Використовує реальний DOM та власні механізми оптимізації оновлення інтерфейсу.

2. **Спрощення вивчення:**

   React:

- Вимагає вивчення концепцій, таких як JSX та віртуальний DOM. Навчання може бути більшим викликом для новачків.

  Angular:

- Є фреймворком з багатьма концепціями та термінологією, що може виявитися складнішим для освоєння, але при цьому надає однорідний підхід до розробки.

3. **Інтеграція та Екосистема:**

   React:

- Має велику та активну спільноту з багатьма сторонніми бібліотеками і рішеннями.

  Angular:

- Велика екосистема, але інтеграція із сторонніми бібліотеками може вимагати додаткового часу та зусиль.

4. **Трудомісткість вивчення та поріг вхідного бар'єру:**

   React:

- Має відносно низький поріг вхідного бар'єру, але може бути важче для вивчення новачкам через використання JSX та певних концепцій.

  Angular:

- Має велику кількість концепцій, і вивчення може взяти більше часу. Однак, коли освоїли основи, можна швидше та ефективніше розробляти.

Обираючи між React та Angular, важливо враховувати характер проекту, досвід команди, індивідуальні уподобання та можливості фреймворку в рамках конкретних завдань.

2.4 Легість у освоєнні

Обираючи фреймворк, перше питання, яке постає у розробника: "Чи складний цей фреймворк у навчанні?", адже в деяких ситуаціях є мало часу на освоєння. У випадку з React, починаючи, розробник стикаєтеся з приємною простотою. React використовує JavaScript, що вже знайоме, і відкриває двері до JSX — синтаксису, що дозволяє вбудовувати HTML-подібний код прямо в JavaScript. Це надає можливість описувати вигляд компонентів за допомогою знайомих HTML-подібних тегів, роблячи код більш читабельним та експресивним.

Потім потрібно вивчити бібліотеку маршрутизації, наприклад, react router v4, а далі бібліотеки керування станом — Redux чи MobX. У випадку Angular треба вивчити багато тем, починаючи від базових, таких як директиви, модулі, декоратори, компоненти, сервіси, введення залежності, pipes та шаблони. Після цього з’являються більш вдосконалені теми, такі як виявлення змін, зони, компіляція AoT та Rx.js. Отже вхідний бар’єр при навчанні для Angular дещо вищий, ніж для React.

Angular та React пропонують рішення для створення мобільних додатків. У випадку Angular це фреймворк Ionic, який використовує контейнер Cordova, який вбудований в Angular. Однак отриманий додаток — це просто веб-додаток всередині контейнера веб перегляду. React Native [16], з іншого боку, — це платформа для створення справді нативних мобільних додатків.

Angular дуже зорієнтований на модульність, і створенні модулів для організації коду та зручного його використання. Компоненти є будівельними блоками, які об'єднують в собі логіку та представлення, і подібно до React, створюється для будь-яких частин сторінки чи додатку.

Angular також впроваджує концепцію "сервісів", які використовуються для реалізації логіки, яка не пов'язана з відображенням. Вони дозволяють поділяти даними та функціональністю між різними частинами додатку.

Вивчаючи Angular, можна помітити значну відмінність роутинга, що дозволяє організовувати навігацію між різними сторінками додатку. Це робить ваш додаток більш користувацьки-дружелюбним та організованим.

Angular має вбудовані інструменти для роботи з HTTP, формами та іншими аспектами розробки, що дозволяє швидко вирішувати завдання.

Усе враховуючи, освоєння Angular — це глибока та структурована подорож, яка дозволяє вам вивчати багато концепцій, що полегшують розробку великих та високоякісних додатків. Навідміну від React, в якому можна поступово вивчати нові концепції та здобувати навички, необхідні для створення великих та високоякісних веб-додатків.

2.5 Підтримка спільноти

Підтримка спільноти грає важливу роль у визначенні успіху та продуктивності фреймворків.

React має велику та енергійну спільноту розробників. Це означає, що завжди є велика кількість ресурсів для навчання та розвитку, таких як блоги, відеоуроки та форуми. До того ж багато розробників та компаній активно вносять свій вклад у розвиток та підтримку фреймворку.

Можна знайти безліч сторонніх бібліотек та компонентів, що розширюють можливості React. Спільнота також активно використовує різні бібліотеки стану, які полегшують роботу з управлінням станом.

Розробники можуть користуватися різними джерелами для навчання, включаючи офіційну документацію, блоги, відеоуроки та форуми. Це полегшує вивчення та підтримку фреймворку.

Angular також має велику та живу спільноту. Це означає, що фреймворк отримує систематичні оновлення та підтримку від розробників у всьому світі. Вона включає в себе досвідчених розробників, які працюють з фреймворком вже довший час, і новачків, які вивчають його вперше.

Особливість Angular полягає в його єдності та інтегрованості. У ньому вже вбудовані рішення для багатьох аспектів розробки, включаючи роутинг, HTTP запити, форми та інші. Це полегшує розробку та спільну роботу над великими проектами.

До того ж офіційна документація Angular є великою та детальною. Вона надає вичерпні відомості щодо всіх аспектів фреймворку та стане хорошим джерелом для навчання та вирішення проблем.

Обидва фреймворки мають сильні спільноти, але підхід до розробки та типові сценарії використання може впливати на вибір між ними. React відзначається гнучкістю та багатством вибору, в той час як Angular надає стабільність та готовні рішення для більших проектів. Обидва фреймворки підтримуються широким спектром розробників, що робить їх відмінними інструментами для розробки веб-додатків, та не можна точно сказати який з них краще обирати та для яких цілей вони більше підходять.

Знайти ще переваги та недоліки, які допоможуть зробити більш точні висновки можна тільки за допомогою практичного дослідження, тому було вирішено розробити два веб-застосунки: один використовуючи react, другий — angular.

РОЗДІЛ 3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Технічне завдання

Головною метою створення веб-застосунків є порівняння фреймворків.

Для цього створено два веб-застосунки: один на React, інший — на Angular, з використанням одного бекенда на два застосунки з використанням node.js.

Для порівняння необхідно проаналізувати такі критерії:

- продуктивність;
- легкість вивчення для новачків;
- порівняння відладки;
- порівняння модулів та бібліотек, тобто кількість бібліотек з якими можна працювати, які можливості взагалі можна чи не можна використовувати в тому чи іншому фреймворку.

Для реалізації було обрано сайт турагентства, з вибором турів по містах України та інших країн, багатофункціональний та з використанням найпопулярніших технологій.

3.2 Стек технологій

3.2.1 Redux

Redux [6] — це бібліотека управління станом для JavaScript додатків, особливо популярна у веб-розробці. Вона дозволяє ефективно керувати станом додатка і зберігати його в одному місці, що полегшує розробку складних програм.

Основна ідея Redux полягає в тому, що стан додатка зберігається у вигляді одного об'єкту, який називається "store".

Store — це об'єкт, який зберігає весь стан додатка. Він є єдиним джерелом правди для стану додатка. Store підтримує методи для доступу до поточного стану, відправлення actions та підписки на зміни стану.

Actions — це прості об'єкти, які відображають події або взаємодії, що відбуваються у додатку. Вони описують, що саме сталося, і містять тип та необов'язкові дані, пов'язані з цим action.

Reducers визначають, які зміни слід внести до стану додатка відповідно до надісланих actions. Вони є чистими функціями, які отримують поточний стан і action, і повертають новий стан на підставі цих даних. Кожен редуктор відповідає за певну частину стану додатка.

Redux також має можливість підписки на зміни стану, що дозволяє компонентам додатка реагувати на зміни стану і оновлювати свої дані та інтерфейс відповідно.

Redux спрощує управління станом, особливо великих додатків зі складними взаємодіями між компонентами. Він сприяє односторонньому потоку даних, роблячи код більш передбачуваним і підтримує розширюваність та тестування. Також існують допоміжні бібліотеки, такі як React Redux, що дозволяють зручно інтегрувати Redux з бібліотеками, такими як React.

3.2.2 Stripe

Stripe [7] — це платіжна платформа, яка надає інструменти та послуги для обробки онлайн-платежів. Вона дозволяє підприємствам та розробникам приймати платежі через Інтернет з різних джерел, таких як кредитні карти, мобільні платежі та інші електронні платіжні системи.

Основні можливості Stripe включають:

- Інтеграція платежів: Stripe надає простий спосіб інтегрувати платіжну систему у ваш додаток чи веб-сайт. Вона має документацію та набори інструментів для різних платформ, включаючи мови програмування, такі як JavaScript, Python, Ruby та інші.

- Обробка різних типів платежів: Stripe підтримує оплату кредитними та дебетовими картками (Visa, Mastercard, American Express і багато інших), мобільними платежами (Apple Pay, Google Pay) та іншими електронними платіжними системами.
- Безпека та відповідність: Stripe дбає про безпеку ваших платежів. Вона використовує шифрування та інші заходи безпеки для захисту конфіденційної інформації. Крім того, Stripe дотримується вимог щодо відповідності з платіжними стандартами, такими як PCI DSS (Стандарт безпеки даних платіжної картки).

Stripe є популярним вибором для багатьох компаній та розробників, завдяки своїй простоті використання, гнучкості та надійності. Вона надає зручний спосіб приймати платежі в Інтернеті та спрощує процес обробки транзакцій для бізнесів будь-якого розміру.

3.2.3 Semantic UI

Semantic UI [8] — це сучасна бібліотека інтерфейсу користувача (UI), яка дозволяє розробникам швидко та легко створювати красиві веб-інтерфейси. Вона пропонує набір готових компонентів, стилів і шаблонів, які відповідають принципам зрозумілого та логічного найменування класів (semantic class names).

Основні особливості Semantic UI:

- Логічне та зрозуміле найменування класів: Semantic UI використовує зрозумілі назви класів для стилізації компонентів. Це полегшує розуміння та модифікацію стилів, а також сприяє швидкому впровадженню нових компонентів.
- Гнучкість та модульність: Semantic UI надає широкі можливості налаштування компонентів та стилів. Вона дозволяє вибирати лише потрібні компоненти, а також комбінувати їх для створення унікальних інтерфейсів. Крім того, Semantic UI підтримує респонсивний дизайн, що

дозволяє пристосовувати інтерфейс до різних пристроїв і розмірів екранів.

Загалом, Semantic UI є потужною бібліотекою UI, яка допомагає розробникам швидко створювати елегантні та сучасні веб-інтерфейси. Вона комбінує логічну структуру з гнучкістю та багатими можливостями налаштування, що робить її популярним вибором для багатьох проектів.

3.2.4 PostgreSQL

PostgreSQL [9] — це потужна та розширювана система управління базами даних (СУБД), яка пропонує високий рівень надійності, розширюваності та функціональності. Вона є однією з найпопулярніших відкритих джерел баз даних та використовується в різних веб-додатках та програмних системах.

Основні особливості PostgreSQL:

- Розширюваність: PostgreSQL надає розширені можливості для налаштування та розширення функціональності бази даних. Вона підтримує створення власних типів даних, функцій та процедур, а також може бути розширена за допомогою додаткових модулів.
- ACID-властивості: PostgreSQL гарантує дотримання ACID-властивостей для забезпечення цілісності, надійності та консистентності даних. Вона підтримує транзакції зі згортанням, розбиванням і заблокуванням, що дозволяє уникати конфліктів та забезпечує цілісність бази даних.
- Можливості реплікації: PostgreSQL підтримує реплікацію даних, що дозволяє створювати резервні копії бази даних та розподілити роботу між декількома серверами. Це забезпечує збереження даних та підвищує доступність системи.

PostgreSQL є вільним та відкритим програмним забезпеченням, що означає, що вона може бути безкоштовно використана, модифікована та розповсюджувана. Вона має активну спільноту розробників, яка підтримує її розвиток та надає підтримку користувачам.

3.2.5 Sequelize

Sequelize [10] — це об'єктно-реляційний мапер (ORM) для Node.js, який дозволяє легко та зручно взаємодіяти з реляційними базами даних, зокрема PostgreSQL, MySQL, SQLite та іншими. Він надає абстракцію бази даних, що дозволяє розробникам працювати з даними у вигляді об'єктів та використовувати звичні для них методи та парадигми програмування.

Основні особливості Sequelize:

- Моделі та міграції: Sequelize дозволяє визначати моделі даних, які відповідають таблицям бази даних. Використовуючи міграції, ви можете створювати, змінювати або видаляти таблиці та їх структуру. Це спрощує роботу зі схемою бази даних та забезпечує контроль версій.
- Запити та асоціації: Sequelize надає зручний спосіб виконання запитів до бази даних за допомогою вбудованих методів, таких як find, create, update, delete та інших. Він також підтримує визначення асоціацій між моделями, таких як один до одного, один до багатьох та багато до багатьох
- Підтримка транзакцій: Sequelize дозволяє використовувати транзакції для забезпечення атомарності та консистентності змін у базі даних. Ви можете групувати декілька запитів разом у межах однієї транзакції та контролювати коміт або відкат змін.

Sequelize є потужним інструментом для роботи з реляційними базами даних в середовищі Node.js, допомагаючи розробникам зосередитися на бізнес-логіці своїх додатків, а не на деталях взаємодії з базою даних.

3.2.6 Lodash

Lodash [11] — це бібліотека утиліт для JavaScript, яка надає набір функцій для спрощення та оптимізації роботи з масивами, об'єктами, рядками та іншими даними. Вона є популярним інструментом, який використовується для полегшення розробки веб-додатків та забезпечення більш чистого та ефективного коду.

Основні особливості Lodash:

- Функціональна програмування: Lodash пропонує потужні функції для маніпулювання та операцій над колекціями даних. Вона дозволяє використовувати функціональний підхід до програмування, такий як мапування, фільтрація, згортання (reduce) та багато інших, що полегшує обробку та трансформацію даних.
- Кросс-платформеність: Lodash підтримує різні середовища виконання, включаючи браузери, Node.js та React Native. Це робить її універсальним інструментом для розробки на різних платформах.
- Легка інтеграція: Lodash можна легко встановити за допомогою менеджера пакетів, такого як npm чи yarn, і використовувати в своєму проекті. Вона має чітку документацію та активну спільноту розробників, що допомагають у вирішенні питань та розумінні її функціональності.

Lodash є потужним інструментом для розробників JavaScript, який спрощує роботу з даними та полегшує написання ефективного та зрозумілого коду. Вона може значно збільшити продуктивність розробки та поліпшити якість вашого програмного забезпечення.

3.2.7 Day.js

Day.js [17] — це легковагова бібліотека для роботи з датами та часом в JavaScript. Вона пропонує простий та інтуїтивно зрозумілий API для роботи з датами, замінюючи важкі та складні функції з стандартного об'єкта Date в JavaScript.

Основні особливості Day.js:

- Легковаговість: Day.js має дуже малий розмір, що робить її ідеальним варіантом для використання в проектах з обмеженими ресурсами або там, де мінімізація розміру завантаження є важливою.
- Простота та зрозумілість: API Day.js дуже простий та легкий у використанні. Вона пропонує зрозумілі методи для роботи з датами, такі як створення, форматування, парсинг, додавання, віднімання та інші. Це дозволяє швидко та зручно працювати з датами без зайвих складнощів.

- Маніпуляція з датами: Day.js надає потужні методи для маніпулювання датами. Ви можете додавати, віднімати, порівнювати, заокруглювати та іншими способами змінювати значення дат. Вона також підтримує роботу з часовими поясами та локалізацію.

Day.js є гнучкою та простою бібліотекою для роботи з датами в JavaScript, яка забезпечує зручний та ефективний спосіб роботи з часом. Вона є чудовим вибором для проектів, де потрібна мінімізація розміру та простота використання.

3.3 Розробка бекенд застосунку

Для розробки бекенду було використано node.js [18] та express.js [19]. База данных — PostgreSQL, для полегшення роботи з базою даних, написання мутацій та моделей було використано Sequelize. Так як при пошуку турів зрозуміло що буде багато маніпуляцій з даними, для полегшення роботи з даними було використано Lodash, також багато обробляються дати тому для зміни формату дат використовувався модуль Day.js. Платіжною системою було обрано Stripe [20].

3.3.1 Налаштування оточення розробки

Перш за все, потрібно було встановити Node.js [18] на комп'ютері. Node.js включає в себе пакетний менеджер npm, який буде використовуватися для встановлення залежностей додатка.

3.3.2 Створення застосунку

Після створення робочої папки та ініціалізації проекту було завантажено необхідні модулі та бібліотеки для роботи застосунку. Далі створення необхідної структури, структура бекенду на рисунку 7.

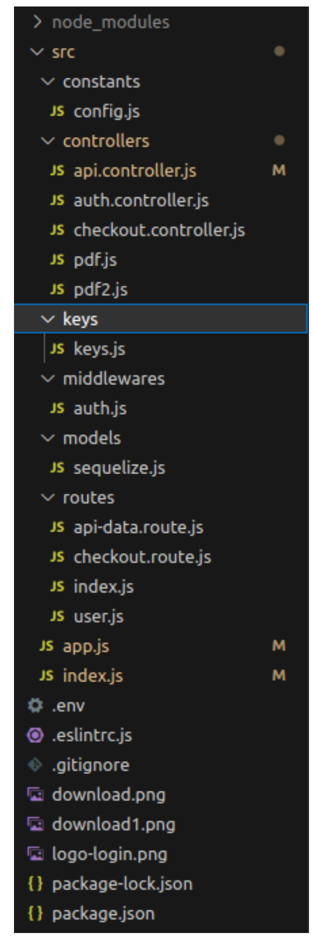

Рисунок 7 — Структура проекту

Після створення необхідної структури черга за підключенням бази даних та написанням моделей та міграцій.

Для підключення бази данних до бекенду використовувався модуль Sequelize [10], для цього було створено екземпляр Sequelize з передачею параметрів підключення до конструктора Sequelize. Підключення наведено у лістингу 1.

Лістинг 1

```
const { Sequelize, Model, DataTypes } = require("se-
quelize");
const sequelize = new Sequelize("travelox_db", "admin",
"admin", {
  host: "localhost",
  dialect: "postgres",
});
```

Далі створювалися моделі, для цього викликається функція sequelize.define(), яка створює нову модель "Users"(фрагмент коду цієї моделі наведений у лістингу 2). Ця функція отримує два аргументи: назву моделі ("users") і об'єкт, який описує поля моделі та їх типи. У цьому об'єкті опису полів моделі присутні наступні поля:

- id: поле типу INTEGER, яке виступає як унікальний ідентифікатор користувача. Встановлено автоматичне інкрементування значень (autoIncrement: true) та використання його як первинний ключ (primaryKey: true);
- email: поле типу TEXT, яке зберігає електронну пошту користувача;
- phone_number: поле типу BIGINT, яке зберігає номер телефону користувача;
- firstName: поле типу TEXT, яке зберігає ім'я користувача;
- lastName: поле типу TEXT, яке зберігає прізвище користувача;
- gender: поле типу TEXT, яке зберігає стать користувача;
- country: поле типу TEXT, яке зберігає країну користувача;
- dateOfBirth: поле типу TEXT, яке зберігає дату народження користувача;
- password: поле типу TEXT, яке зберігає пароль користувача.

У другому аргументі передається об'єкт з опціями для моделі. У даній моделі, встановлена опція timestamps: false, що означає, що Sequelize не буде автоматично додавати поля createdAt і updatedAt до таблиці "users", які використовуються для відстеження часу створення та оновлення запису.

Лістинг 2

```
const User = sequelize.define("users", {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true
  },
  email: DataTypes.TEXT,
  phone_number: DataTypes.BIGINT,
  firstName: DataTypes.TEXT,
  lastName: DataTypes.TEXT,
```

```
    gender: DataTypes.TEXT,
    country: DataTypes.TEXT,
    dateOfBirth: DataTypes.TEXT,
    password: DataTypes.TEXT,
  },
  {
    timestamps: false
  });
```

Таким чином були створені інші моделі, такі як:

1. PaymentCards — для збереження банківських карт користувача,

ця таблиця має зв'язок багато-до-одного з моделлю "User" через проміжну таблицю "UserPaymentCards". Спочатку викликається функція sequelize.define(), для створення нової моделі "UserPaymentCards"(фрагмент коду наведений у лістингу 3). Перший аргумент вказує назву моделі, а другий аргумент — об'єкт, який описує поля моделі і їх типи, в цій моделі лише одне поле: “id”: це поле типу INTEGER, яке виступає як унікальний ідентифікатор для зв'язку між користувачем і платіжною карткою. Встановлено автоматичне інкрементування значень (autoIncrement: true) та використання його як первинний ключ (primaryKey: true). Третім аргументом передається об'єкт з опціями “timestamps: false” для відключення автоматичного створення полів createdAt і updatedAt для цієї моделі.

Лістинг 3

```
  const UserPaymentCards = sequelize.define('UserPayment-
Cards', {
    id: {
      type: DataTypes.INTEGER,
      autoIncrement: true,
      primaryKey: true
    }
  }, {
    timestamps: false
  });
```

Далі відбувається встановлення зв'язку між моделями "User" і "UserPaymentCards":

- User.hasMany(UserPaymentCards): встановлює зв'язок один-до-багатьох (one-to-many) між моделями "User" і "UserPaymentCards", тобто один користувач може мати багато записів в моделі "UserPaymentCards". В результаті, у моделі "User" буде додано метод getUserPaymentCards, який дозволяє отримати усі записи з моделі "UserPaymentCards" для даного користувача.
- UserPaymentCards.belongsTo(User): встановлює зв'язок багато-до-одного (many-to-one) між моделями "UserPaymentCards" і "User", тобто кожен запис в моделі "UserPaymentCards" належить лише одному користувачу з моделі "User". В результаті, у моделі "UserPaymentCards" буде додано поле з назвою UserId, яке є зовнішнім ключем, що посилається на ідентифікатор користувача в моделі "User".

Далі відбувається встановлення зв'язку між моделями "PaymentCards" і "UserPaymentCards":

- PaymentCards.belongsTo(UserPaymentCards): встановлює зв'язок багато-до-одного (many-to-one) між моделями "PaymentCards" і "UserPayment-Cards", тобто кожна платіжна карта належить лише до одного запису в моделі "UserPaymentCards". В результаті, у моделі "PaymentCards" буде додано поле з назвою UserPaymentCardId, яке є зовнішнім ключем, що посилається на ідентифікатор запису в моделі "UserPaymentCards".
- UserPaymentCards.hasOne(PaymentCards): встановлює зв'язок один-до-одного (one-to-one) між моделями "UserPaymentCards" і "PaymentCards", тобто кожен запис в моделі "UserPaymentCards" може мати лише одну платіжну карту з моделі "PaymentCards". В результаті, у моделі "UserPaymentCards" буде додано метод getPaymentCard, який дозволяє отримати платіжну карту для даного запису.

Фрагмент коду з зв’язками наведений у лістингу 4.

Лістинг 4

```
User.hasMany(UserPaymentCards);
```

```
UserPaymentCards.belongsTo(User);

PaymentCards.belongsTo(UserPaymentCards);
UserPaymentCards.hasOne(PaymentCards);
```

2. Booking — для збереження даних про заброньовані тури. Ця таблиця аналогічна попередній, тобто вона також має зв’язок багато-до-одного з моделлю "User" через проміжну таблицю "UserBookings". Тобто спочатку викликається функція sequelize.define(), для створення нової моделі "UserBookings"(фрагмент коду наведений у лістингу 5). Перший аргумент вказує назву моделі, а другий аргумент — об'єкт, який описує поля моделі і їх типи, в цій моделі лише одне поле: “id”: це поле типу INTEGER, яке виступає як унікальний ідентифікатор для зв'язку між користувачем і заброньованим туром. Встановлено автоматичне інкрементування значень (autoIncrement: true) та використання його як первинний ключ (primaryKey: true). Третім аргументом передається об'єкт з опціями “timestamps: false” для відключення автоматичного створення полів createdAt і updatedAt для цієї моделі.

Лістинг 5

```
const UserBookings = sequelize.define('UserBookings', {
  id: {
    type: DataTypes.INTEGER,
    autoIncrement: true,
    primaryKey: true
  }
}, {
  timestamps: false
});
```

Далі відбувається встановлення зв'язку між моделями "User" і "UserBookings":

- User.hasMany(UserBookings): встановлює зв'язок один-до-багатьох (one-to-many) між моделями "User" і "UserBookings", тобто один користувач може мати багато записів в моделі "UserBookings". В результаті, у моделі "User" буде додано метод getUserBookings, який дозволяє отримати усі записи з моделі "UserBookings" для даного користувача;

- UserBookings.belongsTo(User): встановлює зв'язок багато-до-одного (many-to-one) між моделями "UserBookings" і "User", тобто кожен запис в моделі "UserBookings" належить лише одному користувачу з моделі "User". В результаті, у моделі "UserBookings" буде додано поле з назвою UserId, яке є зовнішнім ключем, що посилається на ідентифікатор користувача в моделі "User".

Далі відбувається встановлення зв'язку між моделями "Bookings" і "UserBookings":

- Bookings.belongsTo(UserBookings): встановлює зв'язок багато-до-одного (many-to-one) між моделями "Bookings" і "UserBookings", тобто кожний заброньований тур належить лише до одного запису в моделі "UserPaymentCards". В результаті, у моделі "Bookings" буде додано поле з назвою UserBookingId, яке є зовнішнім ключем, що посилається на ідентифікатор запису в моделі "UserBookings";
- UserBookings.hasOne(Bookings): встановлює зв'язок один-до-одного (one-to-one) між моделями "UserBookings" і "Bookings", тобто кожен запис в моделі "UserPaymentCards" може мати лише один заброньований тур з моделі "Bookings". В результаті, у моделі "UserBookings" буде додано метод getBooking, який дозволяє отримати заброньований тур для даного запису.

Фрагмент коду з зв'язками наведений у лістингу 6.

Лістинг 6

```
User.hasMany(UserBookings);
UserBookings.belongsTo(User);

Bookings.belongsTo(UserBookings);
UserBookings.hasOne(Bookings);
```

Наступним етапом було створення маршрутизації та самих ендпоінтів. Для створення маршрутизації використовується бібліотека Express.

Повний фрагмент коду налаштування маршрутизації наведений у лістингу 7.

Спочатку відбувається підключення бібліотеки Express, яка дозволяє створювати серверні додатки та обробляти HTTP-запити. Далі викликається express.Router(), який створює об’єкт маршрутизатора, який дозволяє визначити роути на бекенді. Після цього імпортуються функції getTours і getTour з модуля api.controller.js. Ці функції будуть викликатися при обробці відповідних HTTP-запитів. Далі встановлюються маршрути:

- router.post("/get-tours", getTours): маршрут POST на роуті "/get-tours", який отримує список турів за певними параметрами та повертає його клієнту, для виконання цього функціоналу викликається функція getTours з модуля api.controller.js;
- router.post("/get-tour", getTour): маршрут POST на шляху "/get-tour", який отримує дані про певний тур по id, та повертає його деталі, для виконання цього функціоналу викликається функція getTour з модуля api.controller.js.

І в кінці експортується об'єкт маршрутизатора router для використання в інших частинах бекенду.

Лістинг 7

```
const express = require("express");
const router = express.Router();
const { getTours, getTour } = require("../control-
lers/api.controller");
router.post("/get-tours", getTours);
router.post("/get-tour", getTour);
module.exports = router;
```

Таким самим чином було написано інші маршрути:

1. Маршрути для користувача (повний фрагмент цього коді наведений у лістингу 8):

- router.post("/sign-up", signUp): маршрут POST на роуті "/sign-up", який реєструє нового користувача, для виконання цього функціоналу викликається функція signUp з модуля auth.controller.js;

- router.post("/sign-in", signIn): маршрут POST на роуті "/sign-in", який авторизує зареєстрованого користувача, для виконання цього функціоналу викликається функція signIn з модуля auth.controller.js.

Лістинг 8

```
const express = require("express");
const router = express.Router();
const { signUp, signIn } = require("../controllers/auth.controller");
router.post("/sign-up", signUp);
router.post("/sign-in", signIn);
module.exports = router;
```

2. Маршрути для бронювання(повний фрагмент цього коді наведений у лістингу 9):

- router.post("/create-checkout-session", createCheckoutSession): маршрут POST на роуті "/create-checkout-session", який бронює тур та створює сесію для оплати, для виконання цього функціоналу викликається функція createCheckoutSession з модуля checkout.controller.js.

Лістинг 9

```
const express = require("express");
const router = express.Router();
const { createCheckoutSession } = require("../controllers/checkout.controller");
router.post("/create-checkout-session", createCheckoutSession);
module.exports = router;
```

Написання ендпоінтів:

1. Авторизація — так як в турагенстві для зручності відстеження турів та власних бронювань потрібно мати аккаунт, тому в першу чергу було створено ендпоінти для авторизації.

- signUp: спочатку імпортуються всі необхідні модулі, а саме: keys, який містить ключі і налаштування, такі як ключ для JWT підпису; модель User з модуля sequelize.js, який містить опис структури, моделей бази даних і взаємодію з нею; jsonwebtoken, який дозволяє створювати та перевіряти JWT (JSON Web Tokens) та bcrypt, для хешування паролів.

Далі створюється сама функція signUp, яка отримує параметри req (об'єкт запиту) і res (об'єкт відповіді). В тілі запиту ініціалізуються змінні, отримані за допомогою деструктуризації об'єкта req.body, а саме: firstName, lastName, email, password. Після цього необхідно перевірити чи існує такий email в базі даних, для цього достатньо виконати запит на базу даних, так як цей запит знадобиться і в інших компонентах, його винесено в окрему функцію, зміст якої наведений нижче у лістингу 10.

Лістинг 10

```
checkEmail = async function (email) {
  const result = await User.findOne({ where: { email: email }, raw: true });
  if (result === null) return false;
  else if (result.email === email) return result;
};
```

Далі перевіряється результат, якщо користувач з таким емейлом вже існує, то повертається відповідь з повідомленням про те, що електронна пошта вже використовується та статусом 404(не знайдено) та так як перед повідомленням стоїть return, це значить що функція закінчить своє виконання якщо потрапить у цей блок. Якщо користувач ще не зареєстрований то виконується синхронізація моделі User з базою даних. Опція alter: true дозволяє внести зміни в схему бази даних, якщо вони є потрібними, це потрібно на той випадок, якщо модель бази данних була змінена.

Після синхронізації відбувається хешування паролю, генерується так звана “сіль” для хешування паролю з використовується genSaltSync з модуля bcrypt для синхронної генерації солі з вартістю "10".

Після хешування паролю відбувається генерування тимчасового токену, тобто створення JWT з об'єктом, який містить дані, що включають електронну пошту та ім'я користувача. Ключ для підпису береться з keys.jwt. Опція expiresIn вказує, що токен буде дійсний протягом 300 секунд (5 хвилин), цього часу цілком достатньо для авторизації користувача.

Далі дані нового користувача записуються в базу даних, використовуючи модель User. Разом з даними про користувача передається тимчасовий токен, який знадобиться при першій авторизації. Також в базі даних для зменшення ризику взлому аккаунта пароль зберігається хешований.

Після цього клієнту повертається відповідь з JSON-об'єктом, який містить токен, деталі користувача та статус 200(успішно). У разі виникнення помилки виводиться повідомлення про помилку у консоль для зручності тестування і повертається відповідь з HTTP-кодом 500(Internal Server Error) і повідомленням про помилку.

Повний фрагмент коду цього едпоінта наведений у лістингу 11.

Лістинг 11

```
const keys = require("../keys/keys");
const { User, PaymentCards, UserPaymentCards } = require("../models/sequelize");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");
module.exports.signUp = async function (req, res) {
  try {
    const { firstName, lastName, email, password } = req.body;
    const isAlreadyExistEmail = await checkEmail(email);
    if (isAlreadyExistEmail) {
      return res.status(404).json({
        message: "Email already taken",
      });
    }
    await User.sync({ alter: true });
    const salt = bcrypt.genSaltSync(10);
    const bcryptPassword = bcrypt.hashSync(password, salt);
    const token = jwt.sign(
```

```
      {
        email: email,
        firstName: firstName,
      },
      keys.jwt,
      { expiresIn: 300 }
    );

    const newUser = await User.create({
      firstName: firstName,
      lastName: lastName,
      email: email,
      password: bcryptPassword,
      token: token,
    });
    await newUser.save();

    return res.json({
      token: token,
      userDetails: {
        id: newUser.dataValues.id,
        firstName: newUser.dataValues.firstName,
        lastName: newUser.dataValues.lastName,
        email: newUser.dataValues.email,
      },
    });
  } catch (exeption) {
    console.log("[Error]: " + exeption);
    return res.status(500).json({
      message: exeption,
    });
  }
};
```

- signIn — спочатку імпортуються всі необхідні модулі (такі ж як і у попередньому ендпоінті). Далі створюється функція signIn для авторизації, яка отримує параметри req (об'єкт запиту) і res (об'єкт відповіді).

Далі ініціалізуються дані користувача такі як електронна пошта і пароль користувача, за допомогою деструктуризації об'єкта req.body.

Після ініціалізації змінних викликається функція для перевірки чи існує користувач з вказаною електронною поштою. Отримані дані про користувача зберігаються у змінній user.

Після отримання даних користувача порівнюється хешований пароль, переданий у POST-запиті, зі збереженим хешованим паролем користувача в базі даних. Результат порівняння зберігається у змінній checkPassword. Далі перевіряється, чи співпадають паролі, якщо паролі не співпадають, повертається відповідь з повідомленням про помилковий пароль та статусом 404 (не знайдено), в іншому випадку генерується новий JWT токен з об'єктом, який містить дані користувача, такі як електронна пошта і ім'я(отримані з функції checkEmail). Ключ для генерування токену береться з keys.jwt. Опція expiresIn вказує, що токен буде дійсний протягом 3000 секунд (50 хвилин).

Після того як токен створений оновлюється запис користувача в таблиці User, оновлюючи поле token на нове значення token. Оновлення відбувається за умовою, що поле email у таблиці має значення, яке збігається з електронною поштою користувача.

Після виконання запиту повертається відповідь з JSON-об'єктом, який містить новий токен, деталі користувача та статус 200(успішно). У разі виникнення помилки виводиться повідомлення про помилку у консоль для зручності при тестуванні та повертається відповідь з HTTP-кодом 500(Internal Server Error) і повідомленням про помилку.

Повний фрагмент коду цього ендпоінта наведений у лістингу 12.

Лістинг 12

```
module.exports.signIn = async function (req, res) {
  try {
    const { email, password } = req.body;
    const user = await checkEmail(email);
    const checkPassword = bcrypt.compareSync(password,
user.password);

    if (!checkPassword) {
      return res.status(404).json({
        message: "Wrong password",
      });
    }
    const token = jwt.sign(
      {
        email: user.email,
        firstName: user.firstName,
```

```
    },
    keys.jwt,
    { expiresIn: 3000 }
  );
  await User.update(
    { token: token },
    {
      where: {
        email: user.email,
      },
    }
  );
  return res.json({
    token: token,
    userDetails: {
      id: user.id,
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email,
    },
  });
} catch (exeption) {
  console.log("[Error]:", exeption);
  return res.status(500).json({
    message: exeption,
  });
}
};
```

2. Апі — другим етапом було отримання списку турів та маніпуляції з ними. Тому були написані ендпоінти з підключенням до API та отриманням турів, для розширеного пошуку та фільтрації. І власне ще один — для отримання детальної інформації про певний тур.

- getTours — ендпоінт для отримання списку всіх турів за певними параметрами та фільтрами.

Спочатку імпортуються всі необхідні модулі: axios, який дозволяє виконувати HTTP-запити, reviewsList — об'єкт reviewsList з файлу config.js, X_RapidAPI_Key — значення ключа доступу до API з змінної середовища та X_RapidAPI_URL — значення URL-адреси API з змінної середовища. Далі створюється функція, яка отримує параметри req (об'єкт запиту) і res (об'єкт відповіді).

Після створення функції ініціалізується змінна, в яку кладеться об'єкт з даними з тіла POST-запиту.

Далі створюється об'єкт options, який містить налаштування для виконання HTTP-запиту (фрагмент коду з налаштуванням наведений у лістингу 13).

Лістинг 13

```
const options = {
   method: "POST",
   url: `${X_RapidAPI_URL}list`,
   headers: {
     "content-type": "application/json",
     "X-RapidAPI-Key": String(X_RapidAPI_Key),
     "X-RapidAPI-Host": "hotels4.p.rapidapi.com",
   },
   data: data,
 };
```

Далі виконується HTTP-запит з використанням налаштувань, заданих у об'єкті options, після успішного виконання запиту і отримання відповіді, клієнту повертається відповідь у форматі JSON з полем tours, яке містить список турів.

У разі виникнення помилки під час виконання запиту виводиться у консоль повідомлення про помилку для зручності при тестуванні та відправляється відповідь з HTTP-кодом 500 (Internal Server Error) і повідомленням про помилку.

Повний фрагмент коду цього ендпоінта наведений у лістингу 14.

Лістинг 14

```
const axios = require("axios");
const { reviewsList } = require("../constants/config");
const X_RapidAPI_Key = process.env.X_RapidAPI_Key;
const X_RapidAPI_URL = process.env.X_RapidAPI_URL;

module.exports.getTours = (req, res) => {
  const data = req.body.data;
  const options = {
    method: "POST",
```

```
    url: `${X_RapidAPI_URL}list`,
    headers: {
      "content-type": "application/json",
      "X-RapidAPI-Key": String(X_RapidAPI_Key),
      "X-RapidAPI-Host": "hotels4.p.rapidapi.com",
    },
    data: data,
  };

  axios
    .request(options)
    .then((response) => {
      console.log(response.data.data.proper-
tySearch.properties);
      res.status(200).json({
        tours: response.data.data.propertySearch.prop-
erties,
      });
    })
    .catch((error) => {
      console.log(error);
      res.status(500).json({ message: error });
    });
};
```

- getTour — ендпоінт для отримання деталей певного туру. Спочатку імпортуються всі необхідні модулі, ті ж самі що і в попередньому ендпоінті. Потім створюється функція getTour, яка отримує параметри req (об'єкт запиту) і res (об'єкт відповіді). Після створення функції ініціалізується змінна, в яку кладеться об'єкт з даними з тіла POST-запиту та оголошуються змінні images і result. Далі створюється об'єкт optionsGetImages, який містить налаштування для виконання HTTP-запиту. Повний зміст налаштувань наведений у лістингу 15.

Лістинг 15

```
const optionsGetImages = {
    method: "POST",
    url: `${X_RapidAPI_URL}detail`,
    headers: {
      "content-type": "application/json",
      "X-RapidAPI-Key": String(X_RapidAPI_Key),
      "X-RapidAPI-Host": "hotels4.p.rapidapi.com",
```

```
    },
    data: data,
  };
```

Далі виконується HTTP-запит з використанням налаштувань, заданих у об'єкті optionsGetImages. Після отримання відповіді, масив зображень туру перетворюється у новий масив об'єктів images, який містить URL-адресу і опис кожного зображення.

Після формування списку зображень у змінну result кладеться інформація про тур з відповіді.

Далі створюється об'єкт tourDetails, який містить розгорнуті деталі туру, такі як ідентифікатор, назва, рейтинг, координати, зображення, адреса тощо, всі ці дані витягуються з отриманих даних.

І нарешті клієнту повертається відповідь у форматі JSON з полем tourDetails, яке містить розгорнуті деталі туру та статус 200(успішно). У разі виникнення помилки виводиться у консоль повідомлення про помилку для зручності тестування і відправляється відповідь з HTTP-кодом 500 (Internal Server Error) і повідомленням про помилку.

Повний фрагмент коду цього ендпоінта наведений у лістингу 16.

Лістинг 16

```
module.exports.getTour = async (req, res) => {
  const data = req.body.data;

  let images;
  let result;

  const optionsGetImages = {
    method: "POST",
    url: `${X_RapidAPI_URL}detail`,
    headers: {
      "content-type": "application/json",
      "X-RapidAPI-Key": String(X_RapidAPI_Key),
      "X-RapidAPI-Host": "hotels4.p.rapidapi.com",
    },
    data: data,
  };
```

```
        try {
          result = await axios.request(optionsGetImages);

          images = result.data.data.propertyInfo.propertyGal-
lery.images.map(
            (item) => {
              return {
                url: item.image.url,
                description: item.image.description,
              };
            }
          );
          result = result.data.data.propertyInfo.summary;

          const tourDetails = {
            id: result.id,
            name: result.name,
            map: result.map,
            rating: result.overview.propertyRating.rating,
            city: result.location.address.city,
            coordinates: [
              result.location.coordinates.latitude,
              result.location.coordinates.longitude,
            ],
            images: images,
            imageMap: result.location.staticImage.url,
            addressLine: result.location.address.addressLine,
            reviews: reviewsList,
            totalCount: Math.floor(Math.random() * (500 —
100)) + 100,
          };

          return res.status(200).json({ tourDetails:
tourDetails });
        } catch (exeption) {
          console.error("[exeption]", exeption);
          return res.status(500).json({ message: exeption });
        }
      };
```

3. Бронювання — Наступним етапом було створення бронювання, для цього створено ендпоінт з підключенням оплати, перед цим в дашборді Stripe налаштовано всі необхідні параметри для підключення особистого аккаунту до бекенду.

- createCheckoutSession — ендпоінт для створення бронювання та оплати.

Спочатку імпортуються моделі баз даних: Booking, User, UserBookings. Далі імпортується бібліотека Stripe [20] і створюється об'єкт Stripe з використанням секретного ключа API, який отримано з дашборду Stripe, цей секретний ключ потрібен для того щоб підключити бекенд до аккаунту Stripe.

Далі створюється функція createCheckoutSession, яка отримує параметри req (об'єкт запиту) і res (об'єкт відповіді).

Після цього ініціалізуються змінні, отримані з тіла POST-запиту, зокрема поля id, cancelUrl, successUrl, amount і userId.

Далі для того щоб створити сесію для оплати потрібно створити в Stripe продукт, для його створення використовується метод “create” і передаються параметри: назва, ціна.

Після створення продукту створюється сесія оформлення замовлення в Stripe з використанням методу create і переданими параметрами, такими як елементи замовлення, режим платежу, URL-адреси успішного та відміненого замовлення. В результаті створення сесії отримується ідентифікатор платежу, який далі записується в базу даних разом з полями tourId та amount, при виконанні запиту функція спочатку шукає користувача за його userId за допомогою методу findByPk в моделі User.

Після успішного пошуку виконується функція з параметром user, в якій перевіряється, чи був знайдений користувач, якщо користувач знайдений то створюється бронювання за допомогою методу “create” в моделі Booking з відповідними параметрами, включаючи tourId, amount та paymentIntent.

Після створення бронювання виконується функція з параметром book, в якій створюється об'єкт UserBookings та встановлюються зв'язки з бронюванням та користувачем. В кінці функції повертається клієнту відповідь у форматі JSON з полем url, що містить URL-адресу сесії оформлення замовлення. Якщо виникла помилка то відправляється відповідь з HTTP-кодом статусу 500 (Internal Server Error) і повідомленням про помилку.

Повний фрагмент коду цього ендпоінта наведений у лістингу 17.

Лістинг 17

```
     const { Booking, User, UserBookings } = re-
quire("../models/sequelize");

     const stripe = re-
quire("stripe")("sk_test_51KUSyrKfW2ml5zvjWfV8DDFPisAct25UP
DDvkQqRkEWWimpAf11u3wE08axcmZxTU1PgVO-
bUMw0lvtDi9tW2kOgc00MnuxCRNu");

     module.exports.createCheckoutSession = async (req, res)
=> {
       try {
         const { id, cancelUrl, successUrl, amount, userId }
= req.body;

         const product = await stripe.products.create({
           name: id,
           images: [],
           default_price_data: { unit_amount: amount * 100,
currency: "usd" },
           expand: ["default_price"],
         });

         const session = await stripe.checkout.sessions.cre-
ate({
           line_items: [
             {
               price: product.default_price.id,
               quantity: 1,
             },
           ],
           mode: "payment",
           success_url: `${successUrl}&success=true`,
           cancel_url: `${cancelUrl}&canceled=true`,
         });

         User.findByPk(userId)
           .then((user) => {
             if (!user) {
               return res.status(404).json({
                 message: "User not found",
               });
             }
             return Booking.create({
               tourId: id,
```

```
              amount: amount,
              paymentIntent: session.payment_intent,
            }).then((book) => {
              return UserBookings.create().then((userBook-
ing) => {
                userBooking.setPaymentCard(book);
                userBooking.setUser(user);
              });
            });
          })
          .catch((error) => {
            return res.status(400).json({ message: error
});
          });

        return res.status(200).json({ url: session.url });
      } catch (exeption) {
        return res.status(500).json({
          message: exeption,
        });
      }
    };
```

Після написання усіх ендпоінтів було створено мідлвар. Ця функція використовується для захисту маршрутів, до яких потрібна аутентифікація. Вона перевіряє наявність та валідність токена в заголовку авторизації, а також оновлює прострочені токени.

Спочатку в мідлварі перевіряється наявність заголовка авторизації в запиті (req.headers.authorization). Якщо заголовок відсутній, повертається відповідь зі статусом 400 і повідомленням про відсутність заголовка авторизації. Якщо токен існує то його значення розбивається на дві частини за допомогою .split(" "). Перша частина є типом авторизації (Bearer), а друга частина є самим токеном. За допомогою токена виконується пошук користувача в базі даних.

Якщо користувача не знайдено, повертається відповідь зі статусом 401 і повідомленням про неавторизований доступ. Якщо користувач знайдений то перевіряється токен на його валідність, для цього використовується функція jwt.verify() з модуля jsonwebtoken. Якщо токен прострочений (expired), створюється новий токен з допомогою функції jwt.sign() і оновлюється поле token

в базі даних для користувача, це забезпечує безпеку інформації. Крім того, ця функція додає об'єкт користувача (user) до об'єкта req, щоб його можна було використовувати в наступних ендпоінтах для отримання інформації про авторизованого користувача, потім повертається відповідь зі статусом 201 і новим токеном.

Якщо сталась помилка під час перевірки аутентифікації, повертається відповідь зі статусом 401 і повідомленням про неавторизований доступ.

Повний фрагмент коду мідлвара наведено і лістингу 18.

Лістинг 18

```
const jwt = require("jsonwebtoken");
const { User } = require("../models/sequelize");

const auth = async function (req, res, next) {
  let user = {};
  try {
    console.log(req.body);
    if (!req.headers?.authorization) {
      return res.status(400).json({ message: "missing
authorization header" });
    }

    const bearerToken = req.headers.authorization;
    const [bearer, token] = bearerToken.split(" ");
    console.log(token);

    user = await User.findOne({ where: { token: token
}, raw: true });

    if (!user) {
      return res.status(401).json({ message: "unauthor-
ized" });
    }

    // --- check token if expired ---
    jwt.verify(token, process.env.TOKEN_KEY);

    // --- attach user information to req.user ---
    req.user = user;

    next();
  } catch (exception) {
    if (exception.name === "TokenExpiredError") {
```

```
          // --- token is expired, let's create a new one
and return it to the client ---
          const newToken = jwt.sign(
            {
              email: user.email,
              firstName: user.firstName,
            },
            process.env.TOKEN_KEY,
            { expiresIn: Number(process.env.JWT_EXPIRES_IN)
}
          );

          await User.update(
            { token: newToken },
            {
              where: {
                token: token,
              },
            }
          );

          return res.status(201).json({ token: newToken });
        }
        return res.status(401).json({ message: "unauthor-
ized" });
      }
    };
    module.exports = auth;
```

Після написання мідлварів настала черга для їх тестування, для цього була використана програма для тестування Postman, в якій можна робити запити на бекенд без використання фронтенд застосунку, отримувати відповідь та відстежувати статуси запитів та помилки.

Після проведення всіх тестувань бекенд застосунок був готовий та наступним етапом було створення фронтенд застосунку на React.

3.4 Розробка фронтенд застосунку

Для розробки фронтенд застосунку на React для стейт менеджменту було обрано Redux [6], для створення красивих та адаптивних компонентів використано Semantic UI [8], для полегшення роботи з даними було використано

Lodash, також для зміни формату та інших маніпуляцій з датами використовувався модуль Day.js [17]. Для Angular було обрано використовувати модулі, що входять до фреймворку.

3.4.1 Створення застосунку

Для створення React застосунку з готовою структурою виконано в консолі робочої папки команду “npx create-react-app reactTravelox”, після чого утворилася папка з необхідними залежностями, модулями та структурою, структура зображена на рисунку 8.

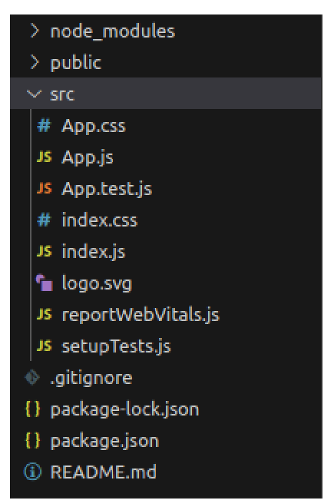

Рисунок 8 — Початкова структура застосунку

Зі сторони Angular для запуску застосунку необхідно встановити Angular CLI [22], який надає набір корисних команд для створення, розробки і збірки застосунків Angular. Встановлено цей модуль, використовуючи команду “npm install -g @angular/cli”.

Першим етапом було створення необхідних сторінок, для цього до структури додано папку “Pages”, але сторінки будуть складатися з певних компонентів тому додано папку “Components”. Для роботи з Redux треба створити actions та reducers та підключити їх, для цього була додана папка “Redux”. При написанні такого застосунку завжди використовуються константи, наприклад

адреса бекенду, адреса до API чи ключі або коди, для їх зберігання був створений файл config.js. Структура застосунку після змінення можна побачити на рисунку 9.

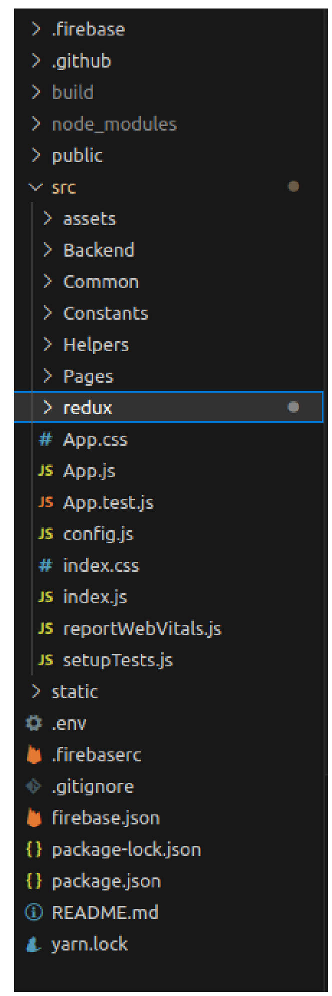

Рисунок 9 — Структура застосунку після змінення

Для створення Angular застосунку з готовою структурою виконано в консолі робочої папки команду “ng new angularTravelox”, після чого утворилася папка з необхідними залежностями, модулями та структурою. Структару можна побачити на рисунку 10.

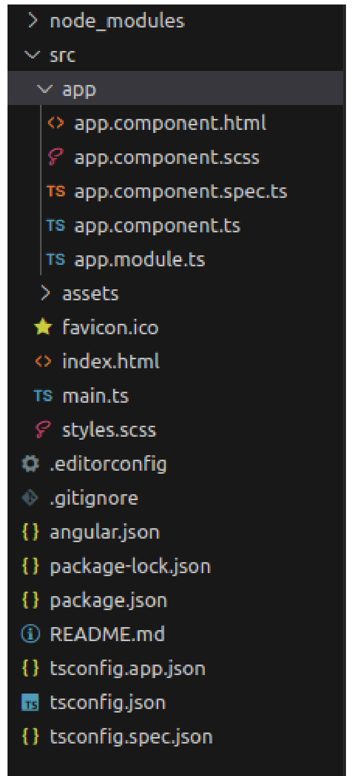

Рисунок 10 — Початкова структура застосунку

На стороні Angular для створення необхідних сторінок теж додано до структури папку “Pages”, але сторінки будуть складатися з певних компонентів тому додано папку “Components”. Для роботи з Redux треба створити actions та reducers та підключити їх, для цього була додана папка “Redux”.

При написанні такого застосунку завжди використовуються константи, наприклад адреса бекенду, адреса до API чи ключі або коди, для їх зберігання був створений файл config.js. Структуру застосунку після змін можна побачити на рисунку 11.

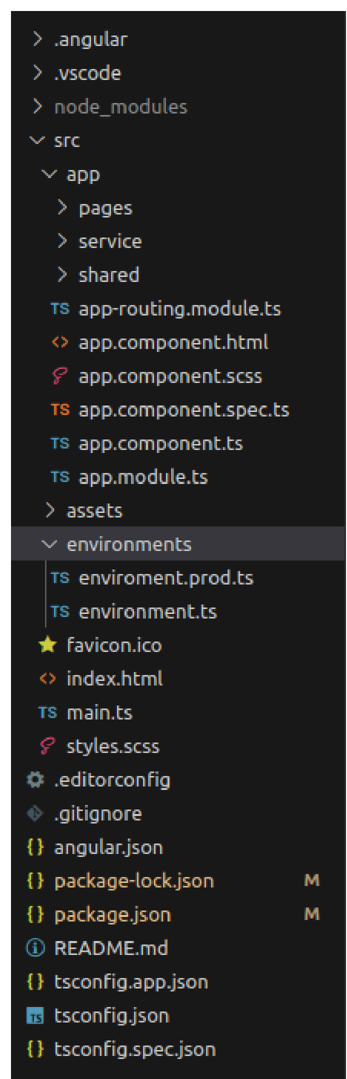

Рисунок 11 — Структура застосунку після змінення

3.4.2 Налаштування маршрутизації

Другим етапом було написано роути на налаштовано маршрутизацію, це у React застосунку можна зробити за допомогою модуля react-router-dom [23].

Спершу прописано всі необхідні роути (лістинг 19)

Лістинг 19

```
export const routes = [
  {
    path: "/err-404",
    component: <ErrPage />,
  },
  {
    path: "/login",
    component: <Login />,
```

```
        },
        {
          path: "/register",
          component: <Register />,
        },
        {
          path: "/forgot-password",
          component: <ForgotPassword />,
        },
         {
          path: "/privacy-policy",
          component: (
            <TermsOfServiceANDPrivacyPolicy
              data={privacyPolicy}
              title={"privacy policy"}
              link={"privacy policy"}
            />
          ),
        },
        {
          path: "/tour-package",
          component: <TourPackage />,
        },
        {
          path: "/tour-package/:id",
          component: <TourSingle />,
        },
        {
          path: "/tour-cart",
          component: <TourCart tourCartHeader={tour-
CartHeader} data={tourCart} />,
        },
        {
          path: "/tour-booking",

          component: <TourBooking data={tourBooking} />,
        },
        {
          path: "/booking-confirm",
          component: <BookingConfirm data={confirmData} />,
        },
         {
          path: "/",
          component: <Home />,
        },
      ];
```

Потім дадаю ці роути і циклі (Лістинг 20)

Лістинг 20

```
<Routes history={history}>
{routes.map((route, index) => (
<Route path={route.path} element={route.component}
key={index} />
))}
</Routes>
```

На відміну від React, Angular має вбудовану маршрутизацію що робить встановлення та використання системи маршрутизації досить простим. Angular використовує декларативний підхід до визначення маршрутів, що означає, що можна описувати структуру маршрутів за допомогою конфігурації, а не програмно. Також Angular дозволяє вкладати маршрути один в одного, що стає зручним для організації структури додатка.

3.4.3 Створення сторінки авторизації

Після ініціалізування роутів настав час верстати сторінки, перш за все це авторизація. Для цього була створена папка “Register” в якій були файли Register.module.scss для стилів та RegisterContainer.jsx для самого компоненту та відповідного функціоналу.

В компоненті RegisterContainer повертається сторінка з формою, з кожного поля в формі, зчитуються дані та зберігаються в змінну, яка являє собою об'єкт, а саме змінні firstName, lastName, email та password. Кожна ця змінна перед тим як записатися валідується відповідно своїй назві, наприклад поле password повинно містити мінімум 8 символів та повинен мати хоча б одну цифру. Код форми наведений у лістингу 21.

Лістинг 21

```
return (
  &lt;Segment className={styles.registerSegment} raised&gt;
    <div className={styles.registerHeader}>
      <img src={logo} />
      <p>create your travelox account</p>
    </div>
    <Form className={styles.registerForm} as="form">
```

```
            <Form.Field>
              <label>First Name</label>
              <Form.Input
                placeholder="Your First Name"
                type="text"
                name="firstName"
                error={
                  !candidateValid.firstName && !candi-
date.firstName
                    ? {
                        content: "Required field",
                        pointing: "below",
                      }
                    : !candidateValid.firstName && candi-
date.firstName
                    ? {
                        content: "First name must contain
from 3 to 30 letters",
                        pointing: "below",
                      }
                    : null
                }
                onChange={handleChange}
              />
            </Form.Field>

            <Form.Field>
              <label>Last Name</label>
              <Form.Input
                placeholder="Your Last Name"
                type="text"
                name="lastName"
                error={
                  !candidateValid.lastName && !candi-
date.lastName
                    ? {
                        content: "Required field",
                        pointing: "below",
                      }
                    : !candidateValid.lastName && candi-
date.lastName
                    ? {
                        content: "Last name must contain
from 3 to 30 letters",
                        pointing: "below",
                      }
                    : null
                }
                onChange={handleChange}
```

```
              />
            </Form.Field>
            <Form.Field>
              <label>Email Address</label>
              <Form.Input
                placeholder="Your Email"
                type="email"
                name="email"
                error={
                  !candidateValid.email && !candidate.email
                    ? {
                        content: "Required field",
                        pointing: "below",
                      }
                    : !candidateValid.email && candi-
date.email
                    ? {
                        content: "Invalid email",
                        pointing: "below",
                      }
                    : null
                }
                onChange={handleChange}
              />
            </Form.Field>
            <Form.Field>
              <label>Password</label>
              <Form.Input
                placeholder="Your Password"
                type="password"
                name="password"
                error={
                  !candidateValid.password && !candi-
date.password
                    ? {
                        content: "Required field",
                        pointing: "below",
                      }
                    : !candidateValid.password && candi-
date.password
                    ? {
                        content: "Password must contain at
least 8 characters",
                        pointing: "below",
                      }
                    : null
                }
                onChange={handleChange}
              />
```

```
          </Form.Field>
          <Form.Field>
            <div className={styles.registerCheckbox}>
              <Checkbox /> I agree to the{" "}
              <Link to="/terms-of-service" class-
Name={styles.registerLink}>
                Terms Of Service
              </Link>
            </div>
          </Form.Field>
          <Button as="button" onClick={() => handleSub-
mit()}>
            <Icon className={styles.registerIcon}
name="paper plane outline" />{" "}
            Register
          </Button>
        </Form>

        <div className={styles.registerFooter}>
          Already have an account?{" "}
          <Link to="/" className={styles.registerLink}>
            Login.
          </Link>
        </div>
      &lt;/Segment&gt;
    );
  };

  export default RegisterContainer;
```

Для стилізації форми, та інших компонентів в обох фреймворках використовувався scss стилі. Приклад одного з них наведено у Лістингу 22.

Лістинг 22

```
@import url("https://fonts.googleapis.com/css2?fam-
ily=Roboto:wght@400;700&display=swap");
:global(#app) {
  .register {
    display: flex;
    align-items: center;
    height: 100vh;
  }

  .registerSegment {
    padding: 50px;
    margin: auto;
```

```
border: 1px solid #fff;
border-radius: 10px;
box-shadow: 0 0 40px 5px rgb(0 0 0 / 5%);
height: max-content;
width: 483px;

.registerForm {
  label {
    font-weight: 400;
    font-family: "Roboto";
    font-style: normal;
    font-size: 16px;
    line-height: 1.8;
  }
  input {
    border: 1px solid #e8e8e8;
    border-radius: 8px;
    padding: 12px 18px;
    box-shadow: none;
    transition: 0.5s;
    height: 50px;

    &:focus {
      border: 1px solid #fca702;
    }

    &::placeholder {
      font-weight: 400;
      font-family: "Roboto";
      font-style: normal;
      font-size: 16px;
      line-height: 1.8;
      color: #6c757d;
    }
  }

  button {
    width: 100%;
    height: 51px;
    background: #fca702;
    color: #fff;
    padding: 12px 0;
    border: none;
    border-radius: 8px;
    font-weight: 600;
    box-shadow: 0 3px 24px rgb(0 0 0 / 10%);
    transition: 0.5s;
    font-family: inherit;
    font-style: normal;
```

```
            font-size: 16px;
            line-height: 1.8;
            margin-top: 15px;
            transition: background-color 0.5s linear;

            &:hover {
              background: #051242;
            }

            .registerIcon {
              color: #ffffff;
            }
          }

          .registerCheckbox {
            font-size: 16px;
          }
        }

        .registerFooter {
          text-align: center;
          margin-top: 40px;
          font-size: 16px;
        }
      }
    }
```

Для створення сторінки на Angular створюється папка з файлами з певною структурою.

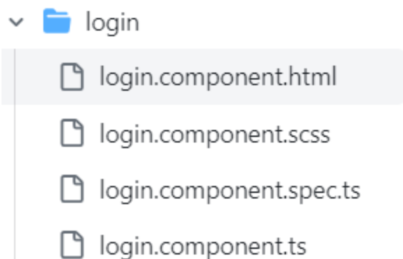

Рисунок 12 — Структура папки сторінки авторизації

У файлі login.component.html буде сама верстка, у Angular верстка за допомогою html-тегів. Тег <main> визначає головний блок сторінки для входу і має клас container-login, який може бути використаний для стилізації або розміщення блоків на сторінці. Блок <div> має атрибут [formGroup]="formLogin",

що зв'язує його з формою Angular. Крім того, він має класи ui, raised, segment, та block-form, які, використовуються для стилізації.

Далі йде блок, що містить зображення логотипу, яке відображається в центрі блока та параграф, який містить заголовок "login with your travelox account" і має класи center-block та title.

Після цього створюється форма, яка містить:

- блок, що містить введене поле для електронної пошти (Email Address) з відповідними міткою, підказкою про помилку та класами для відображення помилки.
- блок містить введене поле для паролю (Password) з відповідними міткою, підказкою про помилку та кнопкою для показу/приховування паролю.
- блок містить чекбокс для "Remember Me" та посилання на "Forgot Password?".
- кнопка "Sign In" з класами та властивістю [disabled], яка визначає, чи повинна кнопка бути вимкненою (disabled), залежно від того, чи є форма недійсною і чи була торкнута (touched).
- блок містить додаткові поради, такі як "Don't have an account? Register."

Цей код має класичну структуру форми входу з елементами управління, такими як поля вводу, чекбокси та кнопки, і використовує Angular для роботи з формою та взаємодії з компонентом. Зміст файлу login.component.html наведений у лістингу 23.

Лістинг 23

```
<main class="container-login">
  <div [formGroup]="formLogin" class="ui raised segment block-form">
    <div class="center-block">
      <img src="assets/logo-login.png" alt="">
    </div>
    <p class="center-block title">login with your trav-elox account</p>
    <div class="block-input ui input">
      <label>Email Address</label>
      <input
```

```
            [ngClass]="(formLogin.controls['email'].invalid
&& formLogin.controls['email'].touched) ? 'ui input error'
: 'ui input'"
            type="text" formControlName="email" place-
holder="Your Email">
          <p *ngIf="formLogin.controls['email'].invalid &&
formLogin.controls['email'].touched"
             class="ui pointing red basic label">Field is
required</p>
        </div>

        <div class="block-input ui input">
          <label>Password</label>
          <input [type]="hide ? 'password' : 'text'" form-
ControlName="password"  placeholder="Your Password">
          <a class="btn btn-default eye" (click)="shown-
Password()">
            <i *ngIf="hide" class="eye slash icon"></i>
            <i *ngIf="!hide" class="eye icon"></i>
          </a>
          <p class="ui pointing red basic label"
             *ngIf="formLogin.controls['password'].invalid
&& formLogin.controls['password'].touched">Field is re-
quired</p>
        </div>

        <div class="field">
          <div class="ui checkbox">
            <input class="hidden" readonly="" tabindex="0"
type="checkbox" value="">
            <label>Remember Me</label>
          </div>
          <a class="" href="/forgot-password">Forgot Pass-
word?</a>
        </div>

        <button class="ui button button-default" [disa-
bled]="formLogin.invalid && formLogin.touched"
(click)="singIn()">
          <i class="sign-in icon"></i>Sign In
        </button>

        <div class="help">
          <p>Don't have an account? <a href="/register">
Register.</a></p>
        </div>
      </div>
    </main>
```

3.4.4 Налаштування стейт менеджмента

Налаштування Redux [6] на React застосунок.

Спочатку встановлюються всі необхідні модулі та пакети, після чого створюється стор (фрагмент коду наведений у лістингу 24), за допомогою функції createStore з пакету Redux. Він приймає два аргументи: rootReducers і compose:

- rootReducers представляє кореневий редюсер, який є поєднанням всіх редюсерів додатка. Він визначений в папці Redux, у файлі rootReducers.js, і імпортуваний у корневому файлі App.js для створення стору;
- compose використовується для поєднання кількох підсилювачів (enhancers) Redux разом, в цьому випадку використовується applyMiddleware(thunk) для додавання middleware Redux Thunk, який дозволяє використовувати асинхронні дії у додатку. Також використовується window.__REDUX_DEVTOOLS_EXTENSION__ і window.__REDUX_DEVTOOLS_EXTENSION__() для підключення розширення Redux DevTools, яке дозволяє відстежувати стан сховища та відлагоджувати Redux, це дуже зручно при тестуванні застосунку.

Лістинг 24

```
const store = createStore(
  rootReducers,
  compose(
    applyMiddleware(thunk),
    window.__REDUX_DEVTOOLS_EXTENSION__ && window.__RE-
DUX_DEVTOOLS_EXTENSION__()
  )
);
```

Після створення стору, у кореневому компоненті використовується Provider з пакету react-redux, який надає доступ до сховища Redux всім компонентам в додатку. Store передається як властивість store для Provider. Також у цьому файлі використовується “BrowserRouter” з пакету react-router-dom, що дозволяє використовувати маршрутизацію в додатку. Всі компоненти

додатку вкладені всередину App, який буде рендеритися в залежності від поточного шляху. Повний фрагмент коду корневого файлу наведений у лістингу 25.

Лістинг 25

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import { BrowserRouter } from "react-router-dom";
import { rootReducers } from "./redux/combineReducers";
import { Provider } from "react-redux";
import { createStore, applyMiddleware, compose } from
"redux";
import thunk from "redux-thunk";

const store = createStore(
  rootReducers,
  compose(
    applyMiddleware(thunk),
    window.__REDUX_DEVTOOLS_EXTENSION__ && window.__RE-
DUX_DEVTOOLS_EXTENSION__()
  )
);
const root =
ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
);
```

Папці Redux має кілька папок та файлів:

1. actions — папка, яка має файли з екшенами (actions), в кожному екшені виконується запити, розглянемо на прикладі файла actionApi.js, в якому зберігаються екшени для маніпуляцій з турами:

- getTours — це функція, яка виконує запит до сервера за списком турів. Вона отримує дані (data) в якості аргументу і повертає асинхронну функцію, яка приймає диспетчер як аргумент. Виконується запит до сервера за допомогою axios.post, і якщо результат має статус 200, дані про тури передаються до

диспетчера за допомогою дії (dispatch({ type: TOURS, payload: result.data.tours })), таким чином дані кладуться у стор і можуть бути доступними у будь-якому компоненті;

- getTopTours — ця функція виконує запит до сервера за топовими турами. Вона не приймає аргументів і також повертає асинхронну функцію, яка приймає диспетчер як аргумент. Запит до сервера виконується за допомогою axios.post, і при успішному результаті дані про тури передаються до диспетчера за допомогою дії (dispatch({ type: TOP_TOURS, payload: result.data.tours })) та також зберігаються у сторі;
- getTour — ця функція виконує запит до сервера за деталями конкретного туру. Вона отримує параметри (options) в якості аргументу і повертає асинхронну функцію, яка приймає диспетчер як аргумент. Запит до сервера виконується за допомогою axios.post, і при успішному результаті дані про деталі туру передаються до диспетчера за допомогою дії (dispatch({ type: TOUR, payload: result.data.tourDetails })) та також зберігаються у сторі.

Повний зміст файлу actionApi.js наведений у лістингу 26.

Лістинг 26

```
import axios from "axios";
import { API_URL, BACKEND_URL, X_RapidAPI_Key } from
"../../config";
import { TOURS, TOUR, TOP_TOURS, DESTINATIONS, REVIEWS
} from "../types";

const headers = {
  "authorization": `Bearer ${localStorage.getItem("to-
ken")}`,
};

export const getTours = (data) => {
  return async (dispatch) => {
    try {
      console.log(`Bearer ${localStorage.getItem("to-
ken")}`);

      const result = await axios.post(
        `${BACKEND_URL}api/get-tours`,
```

```
            {
              data,
            },
          );
          if (result.status === 200) dispatch({ type:
TOURS, payload: result.data.tours });
        } catch (err) {
          dispatch(alert(err));
        }
      };
    };

    export const getTopTours = () => {
      return async (dispatch) => {
        const options = {
          method: "POST",
          url: `${API_URL}list`,
          params: {
            destinationId: "1506246",
            pageNumber: "1",
            pageSize: "3",
            checkIn: "2022-09-20",
            checkOut: "2022-09-25",
            adults1: "1",
            sortOrder: "PRICE",
            locale: "en_US",
            currency: "USD",
          },
          headers: {
            "content-type": "application/json",
            "X-RapidAPI-Key": X_RapidAPI_Key,
            "X-RapidAPI-Host": "hotels4.p.rapidapi.com",
          },
        };

        try {
          const result = await ax-
ios.post(`${BACKEND_URL}api/get-tours`, options, {
            headers: headers,
          });
          if (result.status === 200) {
            dispatch({ type: TOP_TOURS, payload: re-
sult.data.tours });
          }
        } catch (err) {
          console.log("Error", err);
          dispatch(alert("404 status (can not get top
tours)"));
        }
```

```
    };
  };

  export const getTour = (options) => {
    return async (dispatch) => {
      try {
        const result = await axios.post(
          `${BACKEND_URL}api/get-tour`,
          {
            data: options,
          },
        );
        if (result.status === 200) {
          dispatch({ type: TOUR, payload: re-
sult.data.tourDetails });
        }
      } catch (err) {
        console.log("Error", err);
        dispatch(alert("404 status (can not get tour de-
tails)"));
      }
    };
  };
```

2. reducers — папка, яка зберігає редуктори. Розглянемо редуктор на прикладі турів. Цей редуктор використовується для збереження та оновлення даних, отриманих з сервера, у стані Redux. Кожна дія впливає на відповідне поле стану, дозволяючи компонентам React отримувати та відображати актуальні дані.

Спочатку імпортуються необхідні константи та дані з імпортів, потім визначається початковий стан initialState, який містить поля для збереження отриманих даних. Початкові значення встановлюються, включаючи пусті масиви для tours, topTours, reviews, tour для зберігання деталей конкретного туру, та destinations для зберігання локалізованих даних.

Потім визначається функція reducerApi, яка приймає поточний стан state і дію action. Усередині функції виконується перевірка дії за допомогою switch: наприклад для дії TOURS встановлюється нове значення tours у стані, для дії TOUR встановлюється нове значення tour у стані тощо. Для дії

CLEAR_TOURS повертається початковий стан initialState. За замовчуванням, якщо ні одна з дій не відповідає, повертається поточний стан без змін.

Повний зміст файлу reducerApi.js наведений у лістингу 27.

Лістинг 27

```
import { locale } from "../../Backend/Data";
import {
  CLEAR_TOURS,
  DESTINATIONS,
  REVIEWS,
  TOP_TOURS,
  TOUR,
  TOURS,
} from "../types";

const initialState = {
  tours: [],
  tour: {},
  topTours: [],
  destinations: locale,
  reviews: [],
};

export const reducerApi = (state = initialState, ac-
tion) => {
  switch (action.type) {
    case TOURS:
      return { ...state, tours: action.payload };
    case TOUR:
      return { ...state, tour: action.payload };
    case TOP_TOURS:
      return { ...state, topTours: action.payload };
    case DESTINATIONS:
      return { ...state, destinations: action.payload
};
    case REVIEWS:
      return { ...state, reviews: action.payload };

    case CLEAR_TOURS:
      return initialState;

    default:
      return state;
  }
};
```

3. combineReducers.js — в цьому файлі використовується функція combineReducers з пакету Redux для комбінування різних редукторів у один кореневий редуктор. Спочатку імпортується функція combineReducers з пакету Redux та два редуктори reducerApi і reducerUser з відповідних файлів. Далі викликається функція combineReducers і передаються об'єкти редукторів, які потрібно комбінувати. У цьому випадку, об'єкт має два поля: api і user, які відповідають редукторам reducerApi і reducerUser відповідно. Отриманий кореневий редуктор rootReducers використовується для створення store, створення якого описано раніше.

Повний зміст файлу combineReducers.js наведений у лістингу 28.

Лістинг 28

```
import { combineReducers } from "redux";
import { reducerApi } from "./reducers/reducerApi";
import { reducerUser } from "./reducers/reducerUser";

export const rootReducers = combineReducers({
api: reducerApi,
user: reducerUser,
});
```

4. types.js — у цьому файлі оголошуються константи, які використовуються як типи дій (action types) в Redux. Повний зміст файлу наведений у лістингу 29.

Лістинг 29

```
//api
export const TOURS = "API/TOURS";
export const TOUR = "API/TOUR";
export const TOP_TOURS = "API/TOP_TOURS";
export const DESTINATIONS = "API/DESTINATIONS";
export const REVIEWS = "API/REVIEWS";
export const CLEAR_TOURS = "USER/CLEAR_TOURS";

//user
export const USER = "USER/USER";
export const CLEAR_USER = "USER/CLEAR_USER";
```

У Angular є своя бібліотека для керування станом @ngrx/store [24], за допомогою redux-підходу, тому було вирішено використовувати саме її.

Спочатку необхідно встановити @ngrx/store та його залежності за допомогою npm, також можна використовувати yarn. Встановлення модулів наведено у лістингу 30.

Лістинг 30

```
npm install @ngrx/store @ngrx/effects @ngrx/store-devtools
```

Далі налаштовується стор, для цього створюється файл app.store.ts, в якому в першу чергу імпортуються всі необхідні модулі та залежності, а саме:

- клас NgModule з модуля @angular/core. NgModule використовується для оголошення Angular модулів.
- StoreModule з бібліотеки @ngrx/store. StoreModule використовується для налаштування та підключення store до Angular додатку.
- EffectsModule з бібліотеки @ngrx/effects. EffectsModule використовується для обробки побічних ефектів в Redux.
- редуктор з файлу './reducers/tours.reducer'. Редуктор визначає, як змінюється стан для конкретного slice (частини) вашого додатку.
- клас TourEffects з файлу './effects/tours.effects'. TourEffects містить логіку для обробки побічних ефектів, таких як HTTP запити, обробка даних тощо.

Далі налаштовується сам стор, вказується, що модуль ToursModule використовуватиме @ngrx/store для управління станом (з використанням tourReducer) та @ngrx/effects для обробки побічних ефектів (з використанням TourEffects) в контексті фічі з ім'ям 'tours'. Це дозволяє організувати код та логіку, пов'язану з конкретним функціоналом вашого додатку. Вміст файлу app.store.ts наведений у лістингу 31.

Лістинг 31

```
import { NgModule } from '@angular/core';
```

```
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';

import { tourReducer } from './reducers/tours.reducer';
import { TourEffects } from './effects/tours.effects';

@NgModule({
  imports: [
    StoreModule.forFeature('tours', tourReducer),
    EffectsModule.forFeature([TourEffects]),
  ],
})
export class ToursModule { }
```

Після налаштування стора визначається редуктор для обробки дій, пов'язаних з турами в контексті @ngrx/store. Для цього створюється файл tours.reducer.ts, в якому в першу чергу імпортуються залежності:

- функції createReducer та on з бібліотеки @ngrx/store. createReducer використовується для створення редуктора, а on визначає обробники для конкретних дій.
- всі експортовані об'єкти з файлу дій tours.actions та присвоює їм псевдонім TourActions. Це дозволяє використовувати дії, визначені в tours.actions, в цьому файлі.

Далі визначається інтерфейс TourState, який представляє стан для фічі турів. Стан має властивості result (масив турів) і error (помилка, якщо вона виникає). Визначається початковий стан для фічі турів, де result — порожній масив, а error — null.

Після цього визначається сам редуктор за допомогою функції createReducer. Визначаються два обробники за допомогою on:

- Перший обробник (on(TourActions.toursLoaded, ...)) виконується, коли дія toursLoaded відбувається. Він оновлює стан, замінюючи result на новий масив турів.
- Другий обробник (on(TourActions.toursLoadError, ...)) виконується, коли дія toursLoadError відбувається. Він оновлює стан, замінюючи error на нове значення помилки.

Цей редуктор визначає, як має змінюватися стан фічі турів відповідно до відбутих дій. Зміст файлу tours.reducer.ts наведений у лістингу 32.

Лістинг 32

```
import { createReducer, on } from '@ngrx/store';
import * as TourActions from '../actions/tours.ac-
tions';

export interface TourState {
  result: any[];
  error: any;
}

export const initialState: TourState = {
  result: [],
  error: null,
};

export const tourReducer = createReducer(
  initialState,
  on(TourActions.toursLoaded, (state, { result }) => ({
...state, result })),
  on(TourActions.toursLoadError, (state, { error }) =>
({ ...state, error }))
);
```

Наступним етапом є визначення дій для роботи зі станом турів в контексті @ngrx/store. Для цього створюється файл tours.actions.ts, в якому імпортуються функції createAction та props з бібліотеки @ngrx/store. createAction використовується для визначення дій, а props допомагає визначити властивості, які можуть бути передані дії.Далі визначаються дiЇ:

- loadTours. Дія має тип [Tours] Load Tours, і вона може приймати об'єкт з властивістю data типу any. Ця дія представляє запит на завантаження турів.
- toursLoaded. Дія має тип [Tours] Tours Loaded, і вона може приймати об'єкт з властивістю result типу any. Ця дія викликається, коли тури успішно завантажені.

- toursLoadError. Дія має тип [Tours] Tours Load Error, і вона може приймати об'єкт з властивістю error типу any. Ця дія викликається, коли виникає помилка під час завантаження турів.

Ці дії будуть використовуватися в редукторі (tourReducer), який вже визначений, для визначення того, як стан повинен змінюватися відповідно до виконаних дій. Наприклад, коли викликається loadTours, редуктор може реагувати на цю дію і виконати запит на завантаження турів. Після успішного завантаження турів може бути викликана toursLoaded, і редуктор змінить стан, оновивши його новим списком турів. А якщо сталася помилка, буде викликана toursLoadError, і редуктор обробить цю помилку в стані. Вміст файлу tours.actions.ts наведений у лістингу 33.

Лістинг 33

```
     import { createAction, props } from '@ngrx/store';
     export const loadTours = createAction('[Tours] Load
Tours', props<{ data: any }>());
     export const toursLoaded = createAction('[Tours] Tours
Loaded', props<{ result: any }>());
     export const toursLoadError = createAction('[Tours]
Tours Load Error', props<{ error: any }>());
```

І останнім етапом налаштування є визначння ефектів для обробки побічних ефектів, пов'язаних з турами в контексті @ngrx/effects. Для цього створено файл tours.effects.ts, в якому імпортуються всі необхідні модулі та залежності:

- декоратор Injectable з Angular, який вказує, що клас TourEffects може бути внедрений (інжекційний) в інші класи;
- класи і функції з бібліотеки @ngrx/effects. Actions представляє потік всіх дій, що відбуваються в системі. createEffect використовується для створення ефектів, які обробляють побічні ефекти. ofType дозволяє фільтрувати лише ті дії, які відповідають певному типу;

- оператори з бібліотеки RxJS для обробки потоків. mergeMap використовується для об'єднання результатів декількох Observable, map — для трансформації значень в потоці, а catchError — для обробки помилок;
- функцію of з RxJS, яка створює Observable з переданими їй значеннями;
- всі експортовані об'єкти з файлу дій tours.actions та присвоює їм псевдонім TourActions.

Декоратор вказує, що клас TourEffects є Angular сервісом, який може бути інжектований в інші компоненти або сервіси. Визначається ефект loadTours$. Він використовує createEffect, щоб визначити ефект за допомогою потоку дій (this.actions$), який фільтрується за допомогою ofType(TourActions.loadTours) для отримання лише дій типу loadTours. Далі використовується mergeMap для об'єднання результатів Observable, що повертається з this.apiService.getTours(action.data). action.data — це дані, передані в дію loadTours.

Після цього використовується map для трансформації результатів отриманого Observable у дію TourActions.toursLoaded({ result }). Ця дія вказує, що тури успішно завантажені. Для обробки помилок використовує catchError, які можуть виникнути під час виклику this.apiService.getTours(action.data). Якщо виникає помилка, створюється новий Observable за допомогою of з дією TourActions.toursLoadError({ error }). Далі закриваються оператори та ефект.

Конструктор класу TourEffects, який інжектує Actions (потік дій) та ApiService.

Це дозволяє використовувати ці сервіси у ефектах. Ці ефекти будуть відповідати на дії, які визначені в tours.actions, та виконувати відповідні побічні ефекти, такі як виклик API для завантаження турів. Вміст файлу tours.effects.ts наведений у лістингу 34.

Лістинг 34

```
    import { Injectable } from '@angular/core';
    import { Actions, createEffect, ofType } from
'@ngrx/effects';
```

```
import { mergeMap, map, catchError } from 'rxjs/opera-
tors';
import { of } from 'rxjs';

import * as TourActions from '../actions/tours.ac-
tions';
import { ApiService } from '../../../service/api.ser-
vice';

@Injectable()
export class TourEffects {
  loadTours$ = createEffect(() =>
    this.actions$.pipe(
      ofType(TourActions.loadTours),
      mergeMap((action) =>
        this.apiService.getTours(action.data).pipe(
          map((result) => TourActions.toursLoaded({ re-
sult })),
          catchError((error) => of(Tou-
rActions.toursLoadError({ error })))
        )
      )
    )
  );
  constructor(private actions$: Actions, private
apiService: ApiService) {}
}
```

3.4.5 Інтеграція з бекендом

Після створення та стилізації всіх необхідних блоків настав час для інтеграції та написанні функціоналу. Для цього у застосунку Angular було створено файл login.component.ts, а якому імпортуються всі необхідні залежності:

- декоратор Component з Angular для визначення компонента.
- класи FormBuilder, FormControl, FormGroup та Validators з Angular для створення та валідації форми.
- сервіс Router для переходу між різними сторінками вашого додатку.
- сервіс автентифікації (AuthService), який, ймовірно, відповідає за взаємодію з сервером для автентифікації користувача.

Далі декоратор @Component визначає компонент з ім'ям app-login, шаблоном та стилями. Потім оголошується властивість formLogin, яка представляє

форму для введення даних користувача. Оголошується властивість hide, яка використовується для визначення, чи приховувати пароль в полі вводу.

Конструктор компонента, в якому відбувається ініціалізація форми (formLogin) за допомогою FormBuilder, а також встановлення початкового значення для hide. Також в конструкторі відбувається ін'єкція сервісів AuthService та Router.

Далі створюється метод singIn(), який викликається при натисканні на кнопку входу. Метод викликає markAllAsTouched() для позначення всіх полів форми як "доторкані" (touched) та перевіряє, чи форма є недійсною. Якщо форма є недійсною, метод завершується. В іншому випадку він витягує значення електронної пошти та пароля з форми та викликає метод signIn сервісу автентифікації (AuthService).

Також передбачено обробку успішного та невдалого входу з виведенням відповідних повідомлень у консоль.

Метод shownPassword(), який викликається при натисканні на кнопку для показу або приховування пароля. Метод змінює значення властивості hide, яка використовується для управління типом введеного поля пароля (text або password).

Зміст файлу login.component.ts наведений у лістингу 35.

Лістинг 35

```
import { Component } from '@angular/core';
import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '../../../../service/auth.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent {

  public formLogin: FormGroup;
```

```
  public hide: boolean = true;

  constructor(
    private authService: AuthService,
    private formBuilder: FormBuilder,
    private router: Router
  ) {
    this.formLogin = this.formBuilder.group({
      email: ['', [Validators.required, Valida-
tors.email]],
      password: ['', [Validators.required]],
    });
  }

  singIn() {
    this.formLogin.markAllAsTouched();
    if (this.formLogin.invalid) {
      return;
    }

    const email = this.formLogin.get('email')?.value;
    const password = this.formLogin.get('pass-
word')?.value;
    this.authService.signIn(email, password)
      .subscribe(
      (value) => {
        console.warn('Successfully login value',
value);
        // this.router.navigate(['/']).then();
      },
      (value) => {
        console.warn('Error login value', value);
      }
    );
  }

  shownPassword() {
    this.hide = !this.hide;
  }
}
```

В React застосунку все виконується трохи простіше.

Нижче форми реєстрації розміщена кнопка, на натиск якої визивається функція, в якій діспачится action з запитом на бекенд на ендпоінт для реєстрації нового користувача (цей action наведен у лістингу 36).

Лістинг 36

```
export const signUp = (user) => {
  return async (dispatch) => {
    try {
      const result = await ax-
ios.post(`${BACKEND_URL}auth/sign-up`, user);
      if (result.status < 300) {
        dispatch({ type: USER, payload: re-
sult.data.userDetails });
        localStorage.setItem("token", result.data.to-
ken);
        localStorage.setItem("isUser", true);
      }
    } catch (err) {
      console.log("Error", err);
      dispatch(alert(`Email ${user.email} already
taken`));
    }
  };
};
```

Змінна BACKEND_URL, яка представляє собою url до бекенду, так як це константа, то вона винесена в окремий файл config.js, про який розповідається при описі структури.

Виконується запит, і якщо він успішний в store кладуться дані юзера та виводиться користувачу відповідне повідомлення про статус реєстрації, після чого користувач перенаправляється на головну сторінку, і в хедері з моменту реєстрації буде відображатися username нового користувача. Весь зміст компонента RegisterContainer розміщений у лістингу 37.

Лістинг 37

```
import { Button, Checkbox, Form, Icon, Segment } from
"semantic-ui-react";
import styles from "./Register.module.scss";
import logo from "../../assets/logo-login.png";
import { Link, useNavigate } from "react-router-dom";
import { useState } from "react";
import { validate } from "../../Helpers/validation";
import { signUp } from "../../redux/actions/actionUser";
import { useDispatch } from "react-redux";
```

```
const RegisterContainer = ({}) => {
  const dispatch = useDispatch();
  const navigate = useNavigate();

  const [candidateValid, setCandidateValid] = useState({
    firstName: true,
    lastName: true,
    email: true,
    password: true,
  });

  const [candidate, setCandidate] = useState({
    firstName: "",
    lastName: "",
    email: "",
    password: "",
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setCandidateValid({ ...candidateValid, [name]: val-
idate(value, name) });
    setCandidate({ ...candidate, [name]: value });
  };

  const handleSubmit = () => {
    dispatch(signUp(candidate)).then(navigate("/"));
  };
```

На стороні Angular для виконання запита створюються сервіси. Для цього створюється папка service, в якій файли з сервісами. Розглянемо один з них на прикладі сервіса, який відповідає за взаємодію з сервером для автентифікації користувачів. Спочатку імпортуються залежності:

- декоратор Injectable з Angular, який вказує, що клас AuthService є сервісом, який може бути інжектований в інші компоненти або сервіси.
- об'єкт environment з файлу конфігурації, де міститься BACKEND_URL. BACKEND_URL вказує на базову URL-адресу сервера.
- клас Observable з бібліотеки RxJS для роботи з асинхронними потоками даних.
- HttpClient, який дозволяє виконувати HTTP-запити.
- інтерфейс User з файлу, де визначений користувач.

Після імпортів визначається константа API, яка представляє собою повний URL-адресу для HTTP-запитів до сервера автентифікації. Далі декоратор вказує, що сервіс AuthService повинен бути внедрений на рівні всього додатку (provided в root). Конструктор сервісу, який інжектує HttpClient для виконання HTTP-запитів. Далі створюється два методи:

- signUp(user: User): Observable<any> Метод signUp, який приймає об'єкт користувача (User) і виконує POST-запит на сервер для реєстрації нового користувача.
- signIn(email: string, password: string): Observable<any> Метод signIn, який приймає електронну пошту та пароль, і виконує POST-запит на сервер для автентифікації користувача.

Цей сервіс використовує HttpClient для взаємодії з сервером за допомогою HTTP-запитів. Методи signUp та signIn використовують http.post для відправлення даних на сервер. Також використовується константа API, яка дозволяє визначити базову URL-адресу сервера.

Зміст файлу auth.service.ts наведений у лістингу 38.

Лістинг 38

```
import { Injectable } from '@angular/core';
import { environment } from '../../environments/envi-
ronment';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';
import { User } from '../shared/interfaces/user';

const API = environment.BACKEND_URL + 'auth';
@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor(
    private http: HttpClient,
  ) { }

  signUp(user: User): Observable<any> {
    const body = {...user}
```

```
    return this.http.post(`${API}/sign-up`, body);
  }

  signIn(email: string, password: string): Observa-
ble<any> {
    const body = {
      email,
      password
    }
    return this.http.post(`${API}/sign-in`, body);
  }
}
```

Таким самим чином було інтегровано всі інші сторінки.

В результаті після інтеграції вийшов готовий повноцінний сайт, який можна використовувати на ринку.

РОЗДІЛ 4 ПОРІВНЯННЯ ФРЕЙМВОРКІВ

4.1 Порівняння на базі розроблених веб-застосунків

Під час створення веб-застосунків на різних феймворках було помічено деякі відмінностей, особливості створення певного функціоналу, принципів роботи сайту, відмінності мов програмування та інших. Для зручності порівняння можна виділити критерії, по яким можна порівняти розробку на фреймворках React та Angular.

4.2 Мова програмування

React, навідміну від Angular, використовує JavaScript [13] для розробки компонентів. JavaScript є мовою програмування високого рівня, яка використовується для розробки динамічних веб-сайтів та взаємодії з користувачем на стороні клієнта. Вона є інтерпретованою та мовою скриптів, що використовується для створення динамічного контенту на веб-сторінках.

Angular використовує більш строго типізовану версією JavaScript — TypeScript. TypeScript [15] дозволяє виявити помилки на етапі компіляції, що допомагає у підвищенні надійності і обслуговуваності коду. Це робить код більш структурованим та полегшує роботу над великими проектами. Наприклад якщо є функція, яка приймає деякі пропси то для цієї функції можна прописати інтерфейс, який типізує всі ці змінні, приклад написанні інтерфеу та його використання при типізації пропсів можна побачити у лістингу 39.

Лістинг 39

```
interface TourCardProps {
  title: string;
  description: string;
  duration: number;
  price: number;
}
```

```
function TourCard(props: TourCardProps): JSX.Element {
  return (
    <div>
      <h2>{props.title}</h2>
      <p>{props.description}</p>
      <p>Duration: {props.duration} days</p>
      <p>Price: ${props.price}</p>
    </div>
  );
}
```

TypeScript компілюється до чистого JavaScript, тому код, написаний в TypeScript, може бути використаний на будь-якому середовищі, яке підтримує JavaScript.

Основні переваги TypeScript:

1. *Статична типізація:* TypeScript дозволяє визначати типи змінних, що полегшує виявлення помилок на етапі розробки.
2. *Розширена підтримка ООП:* TypeScript підтримує класи, інтерфейси та інші концепції ООП, що полегшує роботу з об'єктно-орієнтованим програмуванням.
3. *Багатий функціонал:* TypeScript має багатий набір функціональних можливостей, таких як перечислення, генерики, індексовані типи, які полегшують розробку складних застосунків.
4. *Краща підтримка для великих проектів:* TypeScript створений з урахуванням потреб великих проектів. Статична типізація та інші концепції, такі як модулі та простори імен, дозволяють покращити структуру коду та забезпечити чітку організацію, що спрощує роботу великих розробницьких команд.
5. *Легша рефакторинг та підтримка коду:* Благодаря статичній типізації, рефакторинг коду в TypeScript стає менш вразливим до помилок. Велика підтримка у редакторах коду (наприклад, у Visual Studio Code) дозволяє швидко знаходити та виправляти помилки.
6. *Широкі можливості інтеграції:* TypeScript інтегрується з існуючим JavaScript кодом без проблем, оскільки компілятор TypeScript перетворює

код у валідний JavaScript. Це дозволяє поетапно впроваджувати TypeScript у проекти, не вносячи суттєвих змін.

7. *Сучасні стандарти ECMAScript:* TypeScript підтримує сучасні стандарти ECMAScript, що дозволяє використовувати нові фічі та покращення мови, не чекаючи їх широкого прийняття браузерами.

Основні переваги JavaScript:

1. *Універсальність:* JavaScript є мовою, яка використовується не лише для розробки веб-сторінок, але і для реалізації серверного коду (за допомогою Node.js), розробки мобільних додатків (за допомогою фреймворків, таких як React Native та Cordova), а також для розробки розширень для браузерів.
2. *Широка підтримка браузерами:* JavaScript підтримується всіма сучасними веб-браузерами, що робить його основною мовою для розробки фронтенду веб-додатків.
3. *Динамічна природа:* Динамічна типізація JavaScript дозволяє легко маніпулювати об'єктами та змінювати їх типи під час виконання коду, що полегшує роботу з динамічною структурою даних.
4. *Велика кількість бібліотек і фреймворків:* JavaScript має великий екосистему бібліотек та фреймворків, які значно полегшують розробку. Наприклад, React, Angular, та Vue.js для фронтенду, або Express.js для розробки серверної частини.
5. *Асинхронний код:* JavaScript підтримує асинхронне програмування, що дозволяє ефективно взаємодіяти з серверами та іншими джерелами даних без блокування виконання інших операцій.
6. *Легкість вивчення:* JavaScript є відносно простою для вивчення мовою, що дозволяє новачкам швидко розпочати розробку.

Вибір між TypeScript та JavaScript залежить від конкретних потреб проекту. Якщо необхідно більше контролю над типами та структурою коду, а також якщо розробляється великий проект, TypeScript може бути більш

відповідним вибором. З іншого боку, для швидкої розробки простих веб-сайтів або скриптів JavaScript може бути достатнім.

4.3 Інструменти та конфігурація

React надає більше свободи у виборі інструментів і конфігурації. Наприклад можна використовувати Create React App або налаштовувати свій власний інструментарій за необхідності.

Create React App [25] — це інструмент, який дозволяє легко створювати нові React додатки без необхідності налаштовування конфігурації. Він включає в себе заздалегідь налаштований Webpack, Babel та інші інструменти, що дозволяє вам швидко почати розробку. Можна легко оновлювати версії React та інших пакетів, використовуючи команди, які надає CRA. Приклад команд для створення і запуска застосунку за допомогою CRA наведено і лістингу 40.

Лістинг 40

```
npx create-react-app my-app
cd my-app
npm start
```

Але якщо потрібен повний контроль над конфігурацією свого проекту, можна створити його власноруч, налаштувавши Webpack, Babel та інші інструменти самостійно. Цей підхід дозволяє додавати додаткові інструменти, плагіни та налаштовувати їх за вашими потребами. Наприклад, можна створити свій package.json, встановити необхідні пакети та створити файл конфігурації для Webpack (наприклад, webpack.config.js) і Babel (.babelrc). Цей підхід може бути більш складним для новачків, але він надає більше можливостей для налаштувань і розширення.

Angular має власну інфраструктуру, включаючи Angular CLI [22], яка допомагає створювати та керувати проектами. Це спрощує старт і забезпечує стандартну конфігурацію.

Angular CLI — це інтерфейс командного рядка, який надає розробникам набір інструментів для роботи з Angular-проектами. Основні можливості Angular CLI включають:

- **Створення проекту:** за допомогою команди **ng new**, можна легко створити новий проект Angular. CLI автоматично налаштує необхідні файли та структуру проекту, щоб не довелося розглядати велику кількість деталей конфігурації.
- **Розробка та тестування:** Angular CLI надає команди для розробки та тестування застосунку. Наприклад, команда **ng serve** запускає розробницький сервер, що автоматично відслідковує зміни у коді та надає гарячу перезавантаження для швидкого перегляду змін.
- **Керування компонентами та сервісами:** За допомогою Angular CLI, можна створювати нові компоненти, сервіси, директиви та інші елементи Angular за допомогою команд, таких як **ng generate component**, **ng generate service**, тощо.
- **Збірка та розгортання:** Angular CLI надає команди для збірки застосунку в оптимізований бандл і для розгортання цього бандлу. Команда **ng build** збирає проект, а результат можна розгорнути на сервері.
- **Тестування:** CLI також допомагає вам у виконанні тестів застосунку за допомогою інтеграції з фреймворком тестування, наприклад, Jasmine та Karma.

4.4 Архітектура та стейт менеджмент

Після налаштування обох стейт менеджментів можна виділити плюси та мінуси їх використання.

Аналіз використання @ngrx/store [24] в Angular:

Плюси:

1. Інтеграція з Angular: Якщо розробляти додаток на Angular, використання @ngrx/store забезпечить вам нативну інтеграцію з цим фреймворком.
2. Сильна типізація: TypeScript, який часто використовується в проектах Angular, надає сильну підтримку для типів, і @ngrx/store може використовувати цю перевагу для забезпечення безпечної роботи зі станом.
3. Широкий функціонал екосистеми: @ngrx має інші корисні пакети, такі як @ngrx/effects для управління побічними ефектами, що робить розробку додатків більш гнучкою.

Мінуси:

1. Більше "обгорток" коду: Зазвичай, для декількох базових операцій потрібно написати більше коду, порівняно з іншими бібліотеками для управління станом.
2. Вивчення кривої: Новачкам може знадобитися час для вивчення концепцій Redux та Angular, особливо якщо вони ще не мають досвіду роботи з управлінням станом.

Аналіз використання Redux [6] в React:

Плюси:

1. Простота використання: Redux славиться своєю простотою та стислістю. Він легко включається в проекти React і не вимагає багато коду.
2. Широка підтримка спільноти: Redux має велику та активну спільноту розробників, яка може бути корисною для отримання допомоги та ресурсів.
3. Розширюваність: Багато плагінів та інтеграцій для Redux дозволяють легко розширювати його функціонал.

Мінуси:

1. Boilerplate-код: Деякі розробники вважають, що Redux може вимагати багато boilerplate-коду для налаштування та роботи зі станом.

2. Вивчення кривої: Як і в разі Angular, новачкам може знадобитися час для вивчення концепцій Redux.
3. Необхідність включення в кожен проект: Не для всіх проектів із невеликою складністю Redux може бути оптимальним вибором, і додавання його може здаватися надто складним.

Обидві технології є потужними і широко використовуються у великих проектах, при виборі треба враховувати вимоги проекту.

4.5 Кількість бібліотек

Якщо порівнювати Angular і React з точки зору кількості та екосистеми сторонніх бібліотек, то обидва фреймворки мають значний екосистемний потенціал, але є деякі відмінності. Якщо порівняти ці два фреймворки за 4 основними параметрами можна зробити такі висновки:

React:

1. React Core: Основний функціонал React має лише кілька ключових пакетів, таких як react та react-dom, які відповідають за основний функціонал бібліотеки та рендеринг на веб-сторінці.
2. State Management: Для керування станом, можна використовувати useState та useReducer, які входять в основний пакет React. Або як і розповідалось у попередньому розділі використовувати Redux чи інші бібліотеки.
3. Routing: React не включає в себе нативні засоби для маршрутизації, тому розробники часто вибирають сторонні бібліотеки, такі як react-router.
4. HTTP Requests: React не має вбудованих інструментів для виконання HTTP-запитів, тому розробники можуть використовувати сторонні бібліотеки, такі як axios або fetch.
5. React Bootstrap, Material-UI: Для React існують сторонні бібліотеки, які надають компоненти в стилі Bootstrap або Material Design. Наприклад,

Material-UI — це популярна бібліотека для React, яка надає готові компоненти, відповідні дизайну Material Design.

Angular:

1. Angular Core: Angular — це повноцінний фреймворк, який включає в себе багато модулів для всієї функціональності, включаючи @angular/core, @angular/common, @angular/router і т.д.
2. State Management: Angular надає свій власний механізм для керування станом через сервіси, observable та RxJS.
3. Routing: Angular має вбудований модуль маршрутизації, який дозволяє легко визначати маршрути та їхні обробники.
4. HTTP Requests: Angular має свій власний модуль для виконання HTTP-запитів, що називається HttpClient, який включений в пакет @angular/common/http.
5. Angular Material [26]: Це офіційна бібліотека компонентів, розроблених командою Angular. Вона надає готові компоненти із стильним дизайном, які можна легко інтегрувати в Angular-додатки.

Під час створення застосунків не тільки при налаштуванні стейт менеджмента були виявлені особливості а і під час створення та стилізації самих компонентів. В Angular застосунку використовувався Angular Material, детальне описання створення компонентів описано у розділі 3. У React застосунку так як немає офіційної бібліотеки була використана стороння Semantic UI [8] бібліотека, детальний опис створення компонентів за допомогою цієї бібліотеки описаний у розділі 3. Після використанні обох бібліотек можна визначити плюси та мінуси кожної з них.

Semantic UI в React:

Плюси:

1. Гнучкість та Стиль: Semantic UI пропонує гнучку та елегантну систему компонентів, яка дозволяє легко стилізувати і налаштовувати інтерфейс додатка.

2. Добра Документація: Semantic UI має добру документацію та приклади, що сприяє швидкому вивченню та використанню компонентів.

Мінуси:

1. Залежність від CSS-класів: Використання Semantic UI може вимагати додавання специфічних CSS-класів до елементів, що може збільшити обсяг коду та вплинути на зрозумілість розмітки.

Angular Materials в Angular:

Плюси:

1. Інтеграція з Angular: Angular Materials розроблено з урахуванням інтеграції з Angular, що полегшує використання його компонентів в проектах Angular.
2. Матеріальний Дизайн: Angular Materials базується на принципах матеріального дизайну, що сприяє створенню інтерфейсу з сучасним та привабливим виглядом.
3. Потужна Темізація: Angular Materials надає потужні інструменти для темізації, що дозволяє швидко змінювати стиль та вигляд компонентів.

Мінуси:

1. Розмір бібліотеки: Angular Materials може мати більший розмір бібліотеки, порівняно з іншими рішеннями, що може вплинути на завантаження додатка.
2. Можливі Проблеми з Інтеграцією: У деяких випадках може виникнути проблема інтеграції або адаптації компонентів Angular Materials до конкретних потреб проекту.

Також слід розглянути особливості налаштування маршрутиації для обох фреймворків.

Маршрутизація в Angular:

Плюси:

1. Вбудована Рішення: Angular має вбудовану бібліотеку для маршрутизації (RouterModule), що робить встановлення та використання системи маршрутизації досить простим.

2. Декларативний Підхід: Angular використовує декларативний підхід до визначення маршрутів, що означає, що розробник описує структуру маршрутів за допомогою конфігурації, а не програмно.
3. Підтримка Child Routes: Angular дозволяє вкладати маршрути один в одного, що стає зручним для організації структури додатка.

Мінуси:

1. Більше Конфігурації: Іноді конфігурація маршрутів в Angular може здаватися складнішою порівняно з деякими іншими рішеннями.

Маршрутизація в React:

Плюси:

1. Багато Варіантів Бібліотек: Для React існує багато бібліотек для маршрутизації, таких як React Router, Reach Router, Next.js та інші. Вибір залежить від потреб проекту.
2. Гнучкість та Контроль: Розробник має більше гнучкості та контролю над тим, як саме працює маршрутизація, вибираючи бібліотеку, яка найкраще підходить вимогам.
3. Динамічне Завантаження Компонентів: Деякі бібліотеки для маршрутизації в React підтримують динамічне завантаження компонентів, що може покращити продуктивність додатка.

Мінуси:

1. Не Вбудована Реалізація: У відміну від Angular, React не має вбудованої системи маршрутизації, і потрібно обрати бібліотеку, яка більше підходить (зазвичай, React Router).
2. Програмний Підхід: Деякі бібліотеки, такі як React Router, використовують програмний підхід до визначення маршрутів, що може здаватися складнішим для новачків.

Проаналізувавши бібліотеки для обох фреймворків можна визначити, що React надає більш гнучкий підхід, де можна вибирати, які бібліотеки використовувати для додаткового функціоналу. Angular же приходить з "все включено" підходом, що дозволяє розробникам використовувати внутрішні

засоби фреймворку для багатьох аспектів розробки. Це питання вибору залежить від потреб та вподобань.

4.6 Проблеми, що виникали при роботі з Angular

З самого початку виникла проблема з розумінням структури, бо Angular володіє розширеним набором концепцій, таких як модулі, сервіси, інжектори та інші. Це призвело до збільшення часу для вивчення та розуміння цих концепцій але всеодно не все було зрозуміло. Рішенням цієї проблеми стало правильне планування та розбиття додатка на модулі, це полегшило розуміння структури. Також, стала в нагоді документація Angular та інші навчальні ресури, що допомогли зрозуміти кращі практики.

Наступною проблемою стала велика кількість boilerplate коду. Angular вимагає велику кількість boilerplate коду, особливо для створення компонентів та сервісів. Це може значно збільшило обсяг написаного коду, який до того ж був не дуже зрозумілим. Для цього було використано генерацію коду Angular CLI для створення компонентів, сервісів тощо. Також, можна розглянути використання шаблонів та паттернів проектування для зменшення кількості необхідного коду, але під час створення застосунку це не використовувалось.

4.7 Проблеми, що виникали при роботі з React

Першою проблемою при створенні застосунку на React постала проблема з управління станом додатку. Так як в React є можливість використання хуків для управління станом звісно облегшує роботу, але при використанні хуків виникає проблема їх використання в іншому компоненті бо викорисатовувати змінну, створену хуками можна тільки в одному компоненті і, наприклад, якщо у батьківському компоненті є змінна і вона також повинна використовуватись у дочірньому компоненті використовуватись вона не може. Для цього є два рішення, перше це передавати пропсами в дочірній компонент цю

змінну, це допоможе використовувати її в дочірньому компоненті, але при цьому може виникнути інша проблема, якщо велика вкладеність змінну потрібно буде передавати пропсами через декілька компонентів і це не зручно і тим паче може загубитись змінна, і це ускладнює код. Дреге рішення, що є більш зручним це використання бібліотек для глобального управління станом, таких як Redux або Context API. При розробці було обрано використовувати Redux, але в деяких випадках було використано і змінні хуків і передані через пропси.

Наступною проблемою був вибір додаткових бібліотек, адже React являється лише бібліотекою для розробки інтерфейсів, тому для деяких функціональностей (наприклад, маршрутизація або управління станом) може бути потрібно вибирати додаткові бібліотеки. Для вибору бібліотеки маршрутизації був проведен невеликий аналіз і обрано React Router, бо він виявився найпопулярнішим серед розробників та має доволі велику та детальну документацію. Щодо вибора UI бібліотеки були різні варіанти і було вибрано Semantic UI [8], бо він має набір цікавих та гарно стилізованих компонентів, які легко кмізувати, та до того ж Semantic UI має детальну документацію і приклади до кожного компонента. Взагалі при виборі бібліотек дуже допомагають різні блоги та спільнота розробників, там можна знайти багато потрібної інформації і не тільки рекомендації для виборі біблітек, а й можна знайти рішення до деяких пробле наприклад при встановленні модулів.

Ще однією проблемою при розробці стало питання перерендеринга та оптимізації швидкодії. React може перерендерювати компоненти навіть при незначних змінах у стані, що може призвести до зайвих операцій та зниження швидкодії. Рішенням було використання shouldComponentUpdate або PureComponent для оптимізації перерендерингу. Також, на форумах спільноти, була порада розглянути використання бібліотек, таких як React.memo або Reselect, для кешування та уникнення непотрібних перерендерингів, але ця порада не була досліджена.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи можна зробити висновки, для яких проектів та з якими потребами краще використовувати фреймворки.

Використання React:

- для малих і середніх проектів: Якщо проект невеликий, або потрібне швидке вирішення, React може бути кращим вибором. Він надає більше гнучкості у виборі інструментів та бібліотек.
- для проектів, які потребують високої продуктивності і швидкості: React має високу продуктивність завдяки віртуальному DOM і доброму підходу до оптимізації.
- для компонентно-орієнтованих проектів: React ідеально підходить для створення багатокомпонентних інтерфейсів.

Використання React:

- для великих та складних проектів: Якщо потрібно створити великий корпоративний веб-додаток, Angular може бути кращим вибором завдяки своїй інтегрованій системі та стандартизації.
- для проектів, де безпека та строга типізація мають вагоме значення: TypeScript дозволяє виявити багато помилок на ранньому етапі, що поліпшує надійність додатку.
- для проектів, що вимагають багато функціональності і сторонніх бібліотек: Angular має багатий набір вбудованих інструментів і дірективів, що полегшує роботу зі сторонніми компонентами та бібліотеками.

Обираючи між React і Angular, слід враховувати вимоги та специфіку проекту, власні навички розробника та переваги та недоліки кожного фреймворку. У кожного з них є свої місця, і вони можуть бути продуктивними в різних контекстах. То ж вибір між React та Angular залежить від конкретних потреб проекту та доступності ресурсів та інструментів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рейтинг State of JS 2020. URL: [https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/] (дата звернення: 10.09.2023).

2. Рейтинг Stackoverflow 2020 Developer Survey. URL: [https://insights.stackoverflow.com/survey/2020#technology-web-frameworks] (дата звернення: 10.09.2023).

3. Офіційний сайт React. URL: [https://reactjs.org/] (дата звернення: 15.09.2023).

4. Офіційний сайт Angular. URL: [https://angular.io/] (дата звернення: 21.09.2023).

5. Офіційний сайт Vue. URL: [https://vuejs.org/] (дата звернення: 21.09.2023).

6. Redux. URL: [https://redux-toolkit.js.org/] (дата звернення: 29.03.2023).

7. Stripe. URL: [https://stripe.com/] (дата звернення: 02.10.2023).

8. Semantic UI. URL: [https://semantic-ui.com/] (дата звернення: 02.10.2023).

9. PostgreSQL. URL: [https://www.postgresql.org/docs/] (дата звернення: 17.10.2023).

10. Sequelize. URL: [https://sequelize.org/docs/v6/core-concepts/model-basics/] (дата звернення: 17.10.2023).

11. Книга про особливості React: React: Up & Running. European Conference on Computer Vision : conference materials, 6-12 Sept., 2016. P. 140–155.

12. Малиш В.В., Скрипник І.А. Порівняльний аналіз фреймворків Angular та React на прикладі сайту туристичного агенства. Молода наука-2023 : Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених. Т.5, 17-22 квітня 2023 р. Запоріжжя : ЗНУ, 2023. С. 103-104.

13. JavaScript Tutorial. URL: [https://www.javascripttutorial.net/] (дата звернення: 13.09.2023).

14. React hooks. URL: [https://legacy.reactjs.org/docs/hooks-reference.htm] (дата звернення: 18.09.2023).

15. Typescript. URL: [https://www.typescriptlang.org/docs/] (дата звернення: 12.09.2023).

16. Офіційний сайт React Native. URL: [https://reactnative.dev/] (дата звернення: 12.09.2023).

17. Day.js. URL: [https://day.js.org/docs/en/parse/parse] (дата звернення: 10.10.2023).

18. Офіційний сайт Node.js. URL: [https://nodejs.org/en/docs] (дата звернення: 19.10.2023).

19. Офіційний сайт Express.js. URL: [https://expressjs.com/en/starter/installing.html] (дата звернення: 08.07.2023).

20. Stackoverflow. URL: [https://stackoverflow.com/] (дата звернення: 19.10.2023).

21. Порівняння React та Angular, блог. URL: [https://radixweb.com/blog/react-vs-angular] (дата звернення: 19.10.2023).

22. Angular CLI. URL: [https://www.npmjs.com/package/@angular/cli] (дата звернення: 10.11.2023).

23. React Router Dom. URL: [https://reactrouter.com/en/main] (дата звернення: 21.10.2023).

24. Ngrx.io. URL: [https://ngrx.io/] (дата звернення: 11.11.2023).

25. Create React APP. URL: [https://create-react-app.dev/] (дата звернення: 07.11.2023).

26. Material Angular. URL: [https://material.angular.io/] (дата звернення: 17.11.2023).

Декларація

академічної доброчесності

здобувача вищої освіти ЗНУ

Я___________Малиш Вікторія Вадимівна______________,

студент(ка)___2__курсу,

форми здобуття освіти ___денної__________,

Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ

Спеціальності 121 Інженерія програмного забезпечення,

адреса електронної пошти__ipz18bd-5@stu.zsea.edu.ua_____________,

підтверджую, що написана мною кваліфікаційна робота на **тему «Порівняльний аналіз фреймворків Angular та React на прикладі сайту туристичного агентства»**

відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою Інтернет-системи, а також на архівування роботи в базі даних цієї системи.

Дата_30.11.2023_ Підпис_________ Малиш Вікторія Вадимівна

Дата_30.11.2023_ Підпис_________ Скрипник Ірина Анатоліївна