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 **Особливості розробки сайтів для онлайн реєстрації послуг**

Виконав: студент 2 курсу, групи 8.1212-іпз-2-дн
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Нікітенко О. О.
(ініціали та прізвище)
Керівник доцент, к.т.н. В. І. Заяц
(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

Нікітенко Олексій Олександрович
(прізвище, ім’я, по батькові)

1.Тема роботи Особливості розробки сайтів для онлайн реєстрації послуг
керівник роботи доцент, к.т.н. Заяц Валерій Іванович
(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 19.01.2024
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 є особливості бази даних, її взаємодії та роботи на фреймворку Angular дослідження можливостей, що надаються;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

6. Консультанти розділів магістерської роботи

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

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Переваги та недоліки наітивного підходу. Дослідження продуктивністі. Додатки розроблені на мові, специфічної для платформи (в мене JavaScript/TypeScript), працює швидше та оптимізована під конкретну платформу.	03.05.2023	виконано
2	Програмування на TypeScript, для розробки сайту. Розробка клієнтської Сторони (Frontend)	02.06.2023	виконано
3	Розробка серверної сторони (Backend). TypeScript використаний для розробки на платформі Angular.	10.07.2023	виконано
4	Реалізація повної функціональності застосунку. Розробка сайту (Full Stack). Робота з API та взаємодія з сервером.	01.12.2023	виконано
5	Тестування коду.	19.01.2024	виконано
6	Оформлення дипломної роботи.	26.02.2024	виконано
7	Попередній захист.	05.03.2024	виконано

Студент _____ Нікітенко О. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ доцент к.т.н. Заяц В. І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ доцент к.ф.-м.н. Скрипник І.А,__
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 105

Рисунків — 5

Джерел — 29

Нікітенко О. О. Особливості розробки сайтів для онлайн реєстрації послуг: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В.І Заяц. Запоріжжя : ЗНУ, 2023. 100 с.

Мета — розробити план реалізації проекту, визначити стек технологій та створити стратегію вибору технологій. Завдання дослідження полягає в аналізі аналогів і кращих практик, вивченні існуючих аналогів систем онлайн-бронювання та виявленні їхніх переваг і недоліків. Дослідження поведінки користувачів, включаючи аналіз звичок та очікувань користувачів при реєстрації на онлайн-сервісах. Оцінка безпеки та конфіденційності шляхом вивчення питань, пов'язаних із захистом персональних даних користувачів та забезпеченням безпеки в процесі роботи на сайті. Визначення функціональності шляхом підготовки переліку можливостей системи, таких як вибір послуг, вибір спеціалістів, опції календаря тощо. Оцінка технічної реалізації, з огляду на можливість використання технічних рішень (фреймворків, баз даних, мов програмування) для реалізації функціоналу. Ці етапи допомагають визначити ключові аспекти розробки та забезпечити успішну розробку і ефективне використання сайту. Для дослідження було створено веб-сайт для надання послуг онлайн, написаний на фреймворку Angular. Крім того, була розроблена бекенд-частина з використанням бази даних Firebase від Google та використовувалась мова програмування TypeScript написана на основі JavaScript.

Ключові слова: фреймворк, веб-сайт, JavaScript, TypeScript, HTML, CSS, Angular, Firebase, Google, функція, клас, компонент, фронтенд, бекенд, API, інтерфейс, стилі, сайт, модуль, бібліотека, авторизація, автентифікація, послуга.

ABSTRACT

Pages — 105

Drawings — 5

The source — 29.

Research goals and objectives for the development of an online service registration site are varied. Understanding user needs, researching the needs and expectations of the target audience regarding the online booking process. Determination of technical requirements by establishing functional and technical requirements for the reservation and registration system.

The goal is to develop a project implementation plan, define a technology stack, and create a technology selection strategy.

The task of the research is to analyze analogues and best practices, study existing analogues of online booking systems and identify their advantages and disadvantages. Research of user behavior, including analysis of user habits and expectations when registering on online services. Assessment of security and privacy by studying issues related to the protection of personal data of users and ensuring security in the process of working on the site.

Defining functionality by preparing a list of system capabilities, such as service selection, specialist selection, calendar options, etc. Evaluation of the technical implementation, considering the possibility of using technical solutions (frameworks, databases, programming languages) to implement the functionality.

These stages help identify key aspects of development and ensure successful development and effective use of the site.

For the study, an online service delivery website written in the Angular framework was created. In addition, the back-end part was developed using the Firebase database

from Google and the TypeScript programming language was used, written on the basis of JavaScript.

Keywords: framework, website, JavaScript, TypeScript, HTML, CSS, Angular, Firebase, Google, function, class, component, frontend, backend, API, interface, styles, site, module, library, authorization, authentication, service .

ЗМІСТ

ВСТУП

Актуальність теми

Популярність та популяризація Інтернету змінює багато сфер життя, в тому числі й сферу надання послуг. Реєстрація послуг стає важливим елементом для клієнтів, які шукають швидкий і простий спосіб підписатися на послуги. Тому існує потреба в розробці веб-сайтів для бронювання послуг в Інтернеті, які надають зручну та ефективну платформу для клієнтів та постачальників.

Зручності, які надають інтернет-технології у світі, який дедалі більше оцифровується, люди шукають способи полегшити своє життя за допомогою онлайн-технологій. Онлайн-бронювання послуг дозволяє клієнтам зручно та швидко планувати роботу та послуги.

Системи онлайн-бронювання, доступні цілодобово, дозволяють планувати зустрічі в будь-який момент. Це особливо важливо в сучасному світі, де люди мають різні графіки роботи і можуть бути доступними у різний час.

Миттєвість означає, що клієнти можуть вибрати зручний час і забронювати зустріч без зайвого клопоту. Це економить час, усуваючи необхідність чекати по телефону або перевіряти години роботи.

Простота використання, інтуїтивно зрозумілий і дружній інтерфейс веб-сайту для бронювання робить цей процес доступним для всіх категорій користувачів, навіть якщо вони не є технічними експертами.

Вичерпна інформація, що надається через системи онлайн-бронювання, може включати всю необхідну інформацію про доступні дати та час, спеціалістів та послуги. Це дозволяє клієнтам приймати обґрунтовані рішення та обирати найкращий варіант.

Нагадування та календарі є корисними завдяки сповіщенням та нагадуванням через SMS та електронну пошту гарантують, що клієнти не пропустять зустрічі та підвищують відвідуваність.

Мобільність, що дозволяє записуватись на прийом через мобільні додатки, робить процес більш гнучким і дозволяє записуватись на прийом на ходу, що важливо для сучасних користувачів.

Ефективність операцій з надання послуг може бути досягнута завдяки системам онлайн-реєстрації за допомогою декількох ключових механізмів зокрема:

Оптимізація робочого часу на підприємстві для підприємців та фахівців, які можуть ефективно керувати своїм графіком і встановлювати робочі години відповідно до власних можливостей та потреб клієнтів. Це може допомогти максимізувати продуктивність та уникнути втрат часу.

Синхронізація календаря як додаткова послуга, яка може автоматично синхронізувати графік роботи з особистим календарем підприємця. Це дозволяє уникнути дублювання розкладу та покращує планування.

Покращене обслуговування клієнтів це можливість призначати зустрічі на зручний для клієнта час, що підвищує гнучкість обслуговування та покращує задоволеність клієнтів. Скорочення часу очікування також сприяє підвищенню задоволеності клієнтів.

Онлайн-системи автоматизують багато рутинних процесів, таких як нагадування про заплановані зустрічі, їх підтвердження та скасування. Це дозволяє власникам бізнесу ефективніше використовувати свій час, спрямовуючи його на більш важливі завдання.

Аналітика та звітність про записи на прийом надають підприємцям інформацію про різні аспекти їхнього бізнесу, наприклад, про попит або популярність послуг у певний час доби. Це дозволяє постачальникам послуг приймати обґрунтовані рішення для оптимізації робочого процесу.

Збільшення аудиторії клієнтів завдяки онлайн-системі, так вона може залучати нових клієнтів завдяки зручності та доступності електронних записів.

Мета дослідження

Метою дослідження є розробка функціонального та інтуїтивно зрозумілого веб-сайту для онлайн-бронювання послуг. Розробка такого веб-сайту передбачає створення системи, що забезпечує зручну та швидку реєстрацію клієнтів, вибір необхідних послуг, доступ до програм та їх доступність, а також можливість здійснення платежів через онлайн-платформу. Веб-сайт сприятиме ефективній організації процесу реєстрації та покращенню взаємодії між клієнтами та постачальниками послуг.

Для досягнення цієї мети будуть використані сучасні методи та інструменти розробки, зокрема фреймворк Angular 14 для фронтенду, база даних Firebase для зберігання даних та мова програмування TypeScript для функціональності. Для ефективного управління процесом розробки та забезпечення гнучкості у вирішенні проблем також буде застосована методологія Agile.

Очікуваними результатами цієї роботи є розробка функціонального прототипу веб-сайту для реєстрації, тестування та валідації онлайн-сервісу відповідно до потреб та очікувань користувачів. Крім того, результати тестування та впровадження будуть проаналізовані для оцінки ефективності та переваг використання такого веб-сайту.

Дослідження та розробка цього проекту має практичне значення, оскільки покращує процес подання заявок на отримання послуг, забезпечує зручну та ефективну взаємодію між клієнтами та постачальниками послуг, а також сприяє розвитку сектору онлайн-послуг.

Наприклад, традиційне бронювання послуг пов'язане з такими незручностями, як численні телефонні дзвінки та тривалий час очікування, а також обмежені можливості відстеження та оновлення розкладу.

Розробка базується на принципах зручності, доступності та ефективності. Мета — надати користувачам простий та інтуїтивно зрозумілий інтерфейс, що дозволяє їм швидко і легко обирати необхідні послуги, перевіряти розклад і наявність вільних місць, а також здійснювати оплату безпосередньо на сайті. Для постачальників послуг пропонуються корисні інструменти для планування, управління замовленнями та комунікації з клієнтами.

Підхід базується на новітніх технологіях та інструментах розробки: Використовуючи Angular 14, я можу створити чуйний і швидкий фронтенд, який забезпечує чудовий користувацький досвід. Firebase, надійний бекенд-сервіс, забезпечує швидке зберігання даних і доступ до них із вбудованими можливостями автентифікації та оплати; TypeScript дозволяє писати чистий, безпечний код, що відповідає найсучаснішим стандартам програмування.

Це також забезпечить масштабованість і гнучкість рішення для задоволення зростаючих потреб клієнтів і провайдерів. Використання модульної архітектури та компонентного підходу дозволить швидко впроваджувати нові функції та розширювати функціонал сайту в майбутньому.

Загалом, розробка веб-сайту для онлайн-реєстрації послуг має великий потенціал для спрощення процесу реєстрації послуг та покращення взаємодії між клієнтами та провайдерами сервісу.

Об'єкт дослідження

Об'єктом дослідження є процес розробки веб-сайту для надання послуг онлайн та розробка таблиць як основного виду контенту. Розробка сайту, де можна отримати інформацію про послуги, а також корисні навички та поради, є актуальною темою, оскільки організація, заснована на розвитку нових технологій та інформації про знання, стає все більш популярною.

Предмет дослідження

Предметом дослідження розробки сайту для онлайн реєстрацій послуг є процес створення веб-додатку, який дозволяє користувачам зареєструватися на потрібну послугу за допомогою Інтернету. Це може включати в себе створення зручного інтерфейсу для користувачів, забезпечення безпеки та захисту персональних даних, автоматизацію процесу реєстрації та інтеграцію з базою даних організації, що надає послугу, що дозволить їм зареєструватись на необхідну послугу в онлайн-режимі, без необхідності відвідувати офіси чи дзвонити по телефону.

Методи досліджень

Аналіз конкурентів. Дослідження конкурентів на ринку послуг може допомогти визначити, які функції та можливості ринку вже є популярними серед користувачів, які є сильні та слабкі сторони конкурентів.

Аналіз потреб користувачів. Дослідження потреб та очікування користувачів може допомогти дізнатися, які функції та можливості повинен мати сайт, щоб задовольнити потреби користувачів.

Розробка прототипів. С–творення прототипів сайту дозволяє перевірити роботу сайту та його функцій у реальному часі, що дозволяє зробити помилки та вдосконалити дизайн та функціональність сайту.

Тестування користувачів. Тестування сайту за допомогою користувачів може допомогти вибрати фідбек і відгуки про роботу сайту, зрозуміти, які функції є корисними, а які не використовуються, щоб вдосконалити дизайн та функціональність сайту.

Аналіз даних. Збір даних про використання сайту та їх аналіз може допомогти зрозуміти, які функції та можливості вдалі для комп’ютера, а які ні, що дозволяють зробити кращою роботу з користувачами.

Задачі дослідження можуть бути різними, але зазвичай вони пов'язані з вирішенням конкретних проблем або знаходженням відповідей на питання. При дослідженні можуть виникати різні проблеми, такі як відсутність відповідності

потребам цільової аудиторії, недостатній трафік на сайті, невідповідність вмісту сайту його тематиці та ін.

Наукова новизна отриманих результатів

Методи розробки та інтерфейсні рішення для вивчення та впровадження нових методів проектування веб-інтерфейсів, які полегшують користувачам реєстрацію в сервісах. Сюди входять дослідження зручних для користувача елементів дизайну, взаємодії та психології користувачів для покращення загального досвіду.

Методи оптимізації робочого часу як розробка нових алгоритмів для ефективного розподілу робочого графіка фахівців відповідно до попиту та доступності також може бути темою дослідження. За мету є підвищення продуктивності та оптимізація робочих процесів.

Автоматизовані системи управління бізнес-процесами, а точніше розробка інноваційних систем для автоматизації бізнес-процесів, таких як нагадування клієнтам, обробка платежів, звітність та аналіз, дозволить підприємцям більш ефективно управляти своїми послугами.

Аналітика даних та інтелектуальна обробка, тобто використання технологій аналізу даних та штучного інтелекту для прогнозування попиту на послуги, надання рекомендацій користувачам та оптимізації робочих графіків може забезпечити велику цінність для бізнесу.

Практичне значення отриманих результатів

Отримані результати можуть бути використані для визначення того, які показники є корисними для яких типів клієнтів або людей, що надають послуги, які сфери розробки є простими у вивченні та вживанні, знати проблеми, що виникають

при використанні фреймворку та визначити шляхи їх вирішення, а також виявити переваги та недоліки використання його.

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» [2], та на ІІІ Всеукраїнській науково-практичній конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» [3].

Глосарій

Фреймворк (Framework) — це програмна структура або конфігурація, яка забезпечує загальну основу для розробки додатків і включає набір заздалегідь написаних, готових до використання компонентів, бібліотек та інструментів.

Фреймворки забезпечують основу для розробки та допомагають прискорити і стандартизувати процес розробки.

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

JavaScript — це високорівнева інтерпретована мова програмування, яка використовується для створення інтерактивних веб-сайтів. Її можна використовувати на стороні клієнта (браузер) або на стороні сервера.

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

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.

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

Google — організація, що була заснована 4 вересня 1998 року як приватна компанія, що займається проектуванням, розробкою та розповсюдженням найпопулярнішої у світі пошукової системи.

Firebase — це платформа для розробки мобільних додатків з дуже широким спектром можливостей. Вона починалася як стартап-компанія і в даний час використовується для розробки додатків.

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

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-додатку або програми, що відповідає за обробку логіки, взаємодію з базами даних і виконання серверних операцій. Бекенд працює на сервері і відповідає на запити від фронтенду.

API (Application Programming Interface) — це набір правил та інструкцій для взаємодії різних додатків між собою. У веб-розробці API визначає спосіб взаємодії різних компонентів програми, таких як фронтенд і бекенд, між собою.

Інтерфейс (Interface) — це точка взаємодії між користувачем і системою і може бути графічним інтерфейсом користувача (GUI) або інтерфейсом командного рядка (CLI). У контексті програмування інтерфейс може також означати структуру, визначену для комунікації між компонентами програми.

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

Сайт (Site) — це сукупність взаємопов'язаних веб-сторінок, які мають спільну тему або мету і доступні через Інтернет.

Модуль (Module) — це незалежна частина програмного забезпечення, яка виконує певну функцію і може використовуватися незалежно від інших частин системи.

Бібліотека (Library) — це набір функцій і ресурсів, які можна використовувати для розробки програми або веб-сайту. Бібліотеки полегшують розробку, надаючи конкретні функції, які можна використовувати безпосередньо.

Авторизація (Authorization) — це процес контролю прав доступу користувача до певних ресурсів або функцій системи. Вона визначає, чи має користувач дозвіл на використання певного ресурсу або виконання певної операції.

Аутентифікація (Authentication) — це процес перевірки особи користувача, зазвичай шляхом введення ідентифікаційної інформації, такої як ім'я користувача та пароль. Якщо автентифікація пройшла успішно, користувач отримує доступ до системи та ресурсів.

РОЗДІЛ 1 РОЗРОБКА САЙТУ ПО ЗАПИСУ ОНЛАЙН ДЛЯ ПОСЛУГ

1.1 Характеристика проекту

Сучасний світ безперервно змінюється, а технології розвиваються з кожним роком все швидше. З поширенням Інтернету все більше людей віддає перевагу онлайн-послугам, замість традиційних. Тому, якщо ви є постачальником якоїсь послуги, то варто розглянути можливість розробки сайту для онлайн-реєстрації на ваші послуги.

У цьому контексті, цей проект з розробки сайту може стати дуже актуальним і корисним для бізнесу. Використовуючи сучасні технології, ми можемо розробити зручний та інтуїтивно зрозумілий сайт, що дозволяє клієнтам зареєструватися на послуги в декілька простих кроків.

У роботі детально розглянуто процес розробки, від постановки завдань до введення сайту в експлуатацію. Також огляну теоретичні основи та практичні аспекти роботи з відповідним програмним забезпеченням та технологіями, які допоможуть створити оптимальний сайт, та зробити процес реєстрації на послуги максимально простим та зручним для клієнтів.

Першим кроком отримання послуги є онлайн-реєстрації на сайті.

Онлайн-запис — це нова, але вже налагоджена система. Записуючись на прийом до фахівця, клієнт отримує підтвердження про зустріч на електронну пошту, або найкращим для нього способом. Це сприяє підвищенню ефективності та мінімізує ще й оптимізує час, що витрачається на процес створення стандартів.

Система дозволяє записуватись на прийом з будь-якого терміналу з доступом до системи Інтернет [6].

1.2 Вимоги до сайту

Вимоги до сайту це елементи, що допоможуть забезпечити ефективну та безпечну роботу сайту для онлайн реєстрацій послуг, ці особливостей розробки сайту можуть включати:

1. Формування бази даних: для зберігання даних користувачів та інформації про послуги.
2. Система реєстрації та авторизації: для забезпечення безпеки та відслідковування дій користувачів на сайті.
3. Функції пошуку та фільтрації: для швидкого та зручного пошуку необхідної інформації та послуг.
4. Модуль оплати: для зручної оплати послуг онлайн.
5. Модуль відгуків: для збору та відображення відгуків користувачів про послуги.
6. Адаптивний дизайн: для зручного відображення сайту на різних пристроях та розширеннях екранів.

1.3 Сфера використання

Розробка веб-сайту для онлайн-реєстрації послуг стає все більш актуальною і потрібною ідеєю, оскільки зростає попит на дистанційні сервіси та роботу з віддаленим доступом. Завдяки таким сайтам люди можуть легко та зручно записатися на потрібні їм послуги онлайн, уникнувши витрати часу на черги та особистий візит до офісів. Розробка сайту може стати ефективним інструментом для підприємств та організацій, які надають різноманітні послуги, що потребують

реєстрації. Це дозволить забезпечити зручність та швидкість процесу реєстрації, зменшити навантаження та збільшити задоволеність клієнтів. Цей сайт може забезпечити зручну та ефективну можливість реєстрації на різноманітні послуги, від медичних до освітніх та багато інших.

Оскільки більшість людей мають доступ до Інтернету та використовують його для здійснення покупок, виконання різних операцій та отримання послуг, сайт є важливим елементом віртуального простору. Інноваційна ідея може стати важливим елементом взаємодії між компанією та клієнтами, та посприяє збільшенню числа клієнтів та збільшенню обсягів продажів в епоху цифрових технологій та зростаючого інтернет-користування. Швидко та зручно зареєструватися (та ефективніше контролювати процес реєстрації та зменшити кількість помилок та непорозумінь) на потрібну послугу, значно зекономити час та зусилля.

1.4 Постановка проблеми

Постановка проблеми при розробці сайту полягає у тому, що існує необхідність створити зручний та функціональний інструмент, який дозволить користувачам реєструватись на потрібні послуги швидко та без зайвих зусиль. Для досягнення цієї мети необхідно розробити ефективну систему реєстрації, яка буде простою та зрозумілою для користувачів будь-якого рівня підготовки. Врахувати безпеку та конфіденційність даних користувачів, що будуть вводити свою особисту інформацію на сайті. При реєстрації на певні послуги, клієнти зазвичай змушені відвідувати офіси або інші місця обслуговування, що може вимагати значних зусиль, часу та коштів. Це може бути особливо не зручно в умовах пандемії або воєнного стану, коли люди намагаються обмежити особистий контакт та перебувати в безпеці. Також, відвідування офісів може бути проблемою для людей з інвалідністю або обмеженою мобільністю.

Один з варіантів розв'язання цієї проблеми — створення сайту для онлайн-реєстрацій на послуги. Проте виникає низка технічних та дизайнерських питань, таких як оптимальний інтерфейс користувача, зручна оплата, захист від шахрайства, інтеграція з системами баз даних тощо.

Постановка проблеми для розробки сайту для онлайн реєстрацій послуг може включати наступне:

1. Відсутність зручного та ефективного способу реєстрації на послуги в режимі онлайн для клієнтів.
2. Недостатнє залучення клієнтів через традиційні канали маркетингу.
3. Потреба у забезпеченні безпеки та конфіденційності особистої інформації клієнтів при реєстрації.
4. Необхідність розробки зручного інтерфейсу для реєстрації та оплати послуг.
5. Потреба у забезпеченні підтримки та обслуговування клієнтів у разі технічних проблем з сайтом або процесом реєстрації.

1.5 Можливі шляхи вирішення проблем

Історичний аналіз — дозволяє зрозуміти, які події та тенденції призвели до виникнення проблеми, допомогти досліднику зрозуміти, які зміни в Інтернеті та поведінці користувачів сприяли виникненню проблем на сайті.

Аналіз цільової аудиторії — дозволяє встановити, які потреби та очікування мають користувачі, що мають інтерес до сайту. Цей метод зручний для формулювання запитань та відповідей, які допоможуть вирішити проблеми на веб-сайті [7].

Вивчення нових тенденцій та ідей. Це допоможе дослідникам зрозуміти, які нові можливості можуть бути використані на веб-сайті, які ідеї можуть викликати інтерес та які тренди можуть допомогти вирішити проблеми на веб-сайті.

Розробка сайту для онлайн реєстрацій послуг може бути вирішена різними способами. В залежності від потреб і вимог клієнта та специфіки послуг, які надаються є кілька варіантів вирішення:

1. Створення веб-сайту з можливістю онлайн реєстрації. Передбачається розробка спеціальної сторінки, на якій користувачі можуть заповнити необхідні дані для реєстрації на певну послугу.
2. Використання інструментів розробки програмного забезпечення для створення власного рішення.
3. Використання блокчейн технологій для забезпечення безпеки даних користувачів та унеможливлення фальсифікації. Використовувати Ethereum для створення розумних контрактів, для автоматичної верифікації та підтвердження даних.
4. Використання інтеграції з існуючими системами, такими як CRM або ERP, для автоматизації процесу реєстрації та отримання даних про клієнтів
5. Розробка власного сайту від початку до кінця — це означає розробку всіх етапів сайту від збору вимог та дизайну до програмування та тестування. Це дозволяє максимально відповідати всім вимогам та потребам.
6. Гібридний варіант — комбінація розробки власного сайту та використання зовнішніх сервісів або готових платформ для спрощення процесу реєстрації та керування подіями.

Обираючи необхідно враховувати потреби та вимоги клієнта, а також час, які можливо вкласти в розробку сайту.

1.6 Вибір найкращого варіанту

Обрати найкращий варіант вирішення проблеми розробки можливо за наступними кроками:

1. Проаналізувавши потреби клієнтів. Вивчення потреб та очікувань майбутніх користувачів, щодо функції та можливостей сайту для того, щоб він був зручним та привабливим.

2. Враховувавши вартість. Аналіз вартості розробки, підтримки та поновлення використовуваних технологій.

3. Перевірка власного досвіду. Достатній досвід та знання для використання обраних технологій

4. Розгляд різних технології. Дослідити різні технології та інструменти, що можливо використовувати для розробки сайту (фреймворки для фронтенду, бази даних та серверні технології для бекенду). Порівняйте їх переваги та недоліки, обрати найкращий варіант.

5. Зробити прототип та протестувати. Розробити прототип та провести тестування, щоб переконатися, що він відповідає потребам та працює без помилок.

Тому було обрані такі рішення.

Angular 16 (Angular) є одним з найпопулярніших фреймворків фронтенд-розробки веб-додатків. Деякі з причин, чому Angular може вважатися найкращим фронтенд-фреймворком на даний момент, включають:

1. Повна функціональність. Angular забезпечує високу функціональність для розробки веб-додатків, включаючи маршрутизацію, компоненти, сервіси, форми та інші компоненти.

2. Швидка розробка. Angular забезпечує швидку розробку завдяки широкому набору інструментів, таких як Angular CLI та Angular Material, що дозволяють швидко створювати різноманітні компоненти та розширювати функціональність.

3. Велика спільнота розробників. Angular має велику спільноту розробників, яка допомагає вирішувати проблеми та надає підтримку для нових розробок.

4. Масштабованість. Angular дозволяє розробляти великі веб-додатки, які можна масштабувати під велику кількість користувачів та різні пристрої.

5. Сумісність з іншими інструментами. Angular сумісний з іншими інструментами, такими як TypeScript, RxJS та Webpack, що дозволяє легко інтегрувати його з іншими частинами стеку технологій.

Ці причини роблять Angular привабливим вибором для розробки веб-додатків, як і потреби проекту та навички розробника.

Firebase може вважатися одним з найкращих бекендів для розробки веб-додатків, особливо для тих, що потребують реального часу та інших функцій, таких як автентифікація користувачів, збереження та відображення даних в режимі реального часу та інше. Основні переваги Firebase, які роблять його привабливим для розробників, наступні:

1. Швидкість та ефективність. Firebase має швидкий та ефективний механізм збереження та відображення даних в режимі реального часу, що дозволяє зменшити час розробки та покращити продуктивність додатку.
2. Аутентифікація користувачів. Firebase надає потужні засоби для автентифікації користувачів, що дозволяє зробити додаток безпечним та надійним.
3. Легка інтеграція. Firebase може бути легко інтегровано з різноманітними фреймворками та інструментами розробки, такими як Angular, React, Vue та інші.
4. Хмарне збереження. Firebase забезпечує безпечне та надійне зберігання даних в хмарі, що дозволяє зменшити час настроювання серверів та забезпечити швидкий доступ до даних.
5. Широкий функціонал. Firebase має широкий функціонал, що дозволяє розробникам використовувати різноманітні функції, такі як аналітика, реклама, тестування та інші, що полегшує розробку та підтримку веб-додатків.

Firebase не є повноцінним бекендом, це платформа для розробки та розгортання веб-додатків та мобільних додатків, яка надає широкі можливості для роботи з базою даних, автентифікацією користувачів, зберіганням файлів та ін. Перевага Firebase — вона є повністю хмарною платформою, можливо легко масштабувати додаток, не треба встановлення та налаштування додаткових

серверів. Firebase має простий та зрозумілий інтерфейс для роботи зі зберіганням даних та аутентифікацією користувачів [8].

TypeScript є надмножиною JavaScript, що дозволяє використовувати статичну типізацію, додає багато інших корисних функцій, які роблять код більш структурованим і зрозумілим. TypeScript дозволяє виявляти помилки під час розробки та знизити кількість помилок, що можуть виникнути в реальному світі. TypeScript є популярним і широко використовуваним, тому є багато ресурсів та документації, що допоможуть вирішувати проблеми. TypeScript є потужним і зручним інструментом для розробки веб-додатків [7].

До переваг TypeScript відноситься:

1. Статична типізація. Дозволяє виявляти помилки на етапі компіляції, а не під час виконання програми. Це зменшує кількість помилок в коді та поліпшує його стабільність.
2. Підтримка орієнтованого на об'єкти програмування. Підтримує класи та об'єктно-орієнтоване програмування, що дозволяє створювати більш складні та структуровані програми.
3. Підтримка модульності. Підтримує модульну архітектуру, що дозволяє розділити код на логічні модулі та зменшити його залежності.
4. Сумісність з JavaScript: TypeScript є надмножиною JavaScript, що дозволяє використовувати існуючий JavaScript-код без змін.
5. Підтримка IDE. Більшість популярних інтегрованих середовищ розробки (IDE), таких як Visual Studio Code, підтримують TypeScript, що дозволяє отримувати підказки та допомогу при написанні коду.
6. Підтримка TypeScript у Angular. Angular є однією з найпопулярніших фреймворків для розробки веб-додатків, і він підтримує TypeScript як головну мову програмування. Це забезпечує більш просту та зручну розробку веб-додатків. Узагалі, TypeScript є потужним інструментом для розробки веб [9].

Висновки до розділу 1

Зважаючи на використання Angular та Firebase, можна зробити припущення, що сайт буде швидким та стабільним. Firebase дозволяє автоматизувати багато процесів, що спрощує процес розробки та підтримки проекту.

TypeScript є інструментом для розробки веб-додатків, що розширює функції JavaScript. Додатки, написані на TypeScript, можуть працювати будь-де та використовуватися з серверною платформою. Він дозволяє розробникам зменшити кількість помилок та збільшити швидкість розробки за рахунок перевірки типів та автоматичної підказки.

Використання Angular, Firebase та TypeScript дозволить створити мобільний та мультиплатформний веб-додаток зі структурованою генерацією коду та автоматичним запуском серверного коду.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ САЙТУ ПО ЗАПИСУ ОНЛАЙН ДЛЯ ПОСЛУГ

Для забезпечення ефективної взаємодії між клієнтами та послугами на сайті, а також для забезпечення постійного контролю та доступу до повної інформації, потрібно налаштовувалась система моніторингу та аналізу. Ця система повинна надавати можливість відстежувати всі аспекти взаємодії користувачів з сайтом, зокрема їхні дії, вибори та відгуки. Крім того, важливо забезпечити доступ до цієї системи з будь-якої точки світу, щоб забезпечити глобальну доступність та підтримку. Для цього можна використовувати хмарні рішення, такі як хмарні сервери або послуги хмарного моніторингу.

2.1 Загальні відомості про дослідження розробки сайту по запису онлайн для послуг

Розробка веб-сайту для онлайн-бронювання — це процес створення веб-сайту, який дозволяє користувачам реєструватися на послуги в режимі онлайн. Цей процес зазвичай передбачає створення форми заявки, де користувачі можуть ввести свої дані, вибрати дату і час зустрічі, вибрати тип послуги, заповнити форму зі своїми особистими даними та іншими необхідними даними, а також отримати підтвердження про успішну реєстрацію на послугу.

Клієнти можуть легко та зручно подати заявку на отримання послуг без необхідності приходити до офісу чи телефонувати. Це забезпечить ефективний

облік, розподіл замовлень і належний контроль за замовленнями та роботою співробітників.

Розробка сайту для онлайн-заявки на послугу включає в себе створення форми заявки, розробку бази даних для зберігання даних про замовлення і користувачів, створення адміністративного екрану для управління замовленнями і клієнтами, створення системи нагадувань і підтвердження замовлень, інтеграцію з системою онлайн-платежів для прийому платежів, серед інших основних елементів.

Розробка сайтів для онлайн-реєстрації послуг може бути виконана для різних типів бізнесу, таких як здоров'я і краса, освіта, ресторани і готелі, автосервіси, електронна комерція, здоров'я, косметика, освіта і спорт.

Основне завдання веб-сайту для онлайн-реєстрації на послуги — дати можливість клієнтам зручно і швидко зареєструватися на певну послугу. Для цього можуть бути використані такі функції, як онлайн-календар, що показує доступність, система нагадувань і підтвердження бронювання по електронній пошті або SMS.

Щоб розробити сайт для онлайн-бронювання послуг, слід врахувати наступні кроки:

1. Визначення функціональних вимог до веб-сайту та підготовка технічного завдання (ТЗ);
2. Вибір технології розробки (фронтенд, бекенд, база даних, хостингова інфраструктура);
3. Розробка дизайну та інтерфейсу веб-сайту, а також мобільної версії веб-сайту;
4. Розробка функціональних можливостей сайту, таких як реєстрація клієнтів, онлайн-календар, нагадування про зустрічі, підтвердження зустрічей тощо;
5. Тестування та дослідження сайту на різних пристроях та перевірка його безпеки;
6. Розміщення сайту на хостингу та налаштування його інфраструктури.

Процес розробки повинен враховувати різні потреби клієнтів і забезпечувати високу задоволеність користувачів шляхом надання зручного і простого інтерфейсу.

При розробці сайту для сервісних додатків важливо враховувати потреби користувачів, забезпечити зручну і просту навігацію по сайту, а також забезпечити швидкість і безпеку роботи сайту. Також слід розглянути можливість інтеграції з іншими системами, такими як платіжні шлюзи та системи управління контентом.

Розробка сайту онлайн-бронювання послуг — одне з найактуальніших завдань у сфері веб-розробки.

Дослідження сайту також вимагає досвіду у створенні баз даних для зберігання інформації про послуги, клієнтів та їхні бронювання.

Крім технічних навичок, дослідники повинні розуміти потреби і бажання користувачів і максимально підвищити зручність сайту. Також важливо дослідити безпеку сайту і захистити його від шахраїв і зловмисників, які використовують сайт у своїх цілях.

Таким чином, дослідження сайту онлайн-бронювання послуг є складним процесом і вимагає від розробників знань і досвіду в різних сферах, а також уважного ставлення до потреб і побажань користувачів.

2.2 Дослідження інструментів для розробки веб-сайту для онлайн-реєстрації послуг

Для розробки веб-сайтів для онлайн-реєстрації послуг можна використовувати різні інструменти, залежно від потреб та вимог. Деякі з найпоширеніших інструментів є наступними:

1. Фронтенд-фреймворки Angular, React, Vue.js — вони дозволяють створювати динамічні користувацькі інтерфейси, які покращують взаємодію з користувачем;
2. Бази даних. Firebase — надають можливість зберігати, організовувати та отримувати доступ до даних користувачів;

3. Інтегровані сервіси. Google Maps — розширити функціональність вашого сайту, додають можливості та покращують користувацький досвід;

4. Редактори коду. Visual Studio Code — надають можливість редагувати та зберігати код, допомагають підтримувати сайт у правильному форматі та зберігати код у безпечному місці;

5. Хмарні платформи Google Cloud — надають можливість зберігати та обробляти дані користувачів, виконувати завдання на сервері та забезпечувати масштабованість додатку [4].

Для розробки сайтів онлайн-бронювання послуг можна використовувати низку інструментів. Основними з них є фреймворки, які прискорюють процес розробки, дозволяючи використовувати готові компоненти та інструменти.

Деякі з переваг Angular:

1. Модульність та компонентна архітектура: Angular забезпечує модульність коду, що дозволяє організувати код і легко розширювати його; Angular базується на компонентній архітектурі, яка дозволяє розробникам розбивати веб-додатки на багаторазові та нові компоненти.

2. Швидкість розробки: Angular має широкий набір компонентів та інструментів для розробки фронтенду, що дозволяє прискорити процес розробки. Angular має вбудовану підтримку для динамічних даних, що дозволяє розробникам створювати швидкі та ефективні веб-додатки;

3. Кросплатформеність: Angular дозволяє розробникам створювати крос-платформні додатки, які працюють як на веб-сайтах, так і на мобільних пристроях;

4. Директиви: Angular дозволяє розробникам використовувати директиви для розширення HTML-коду додатковими можливостями, такими як обробка подій, маніпуляції зі стилями тощо;

5. Сервіси: в Angular сервіси можна використовувати для обміну даними між компонентами;

6. Підтримка: оскільки Angular є дуже популярним фреймворком, він має велику

спільноту розробників та підтримку Google, що дозволяє розробникам швидко вирішувати проблеми та використовувати нові функції [5].

За допомогою Angular можливо створювати якісні, швидкі веб-сайти зі зручним інтерфейсом та багатою функціональністю. Це відмінний вибір для розробників, які хочуть створювати сучасні веб-додатки.

Firebase можна вважати одним з найкращих бекендів для розробки веб-додатків Firebase — це хмарна платформа для розробки мобільних і веб-додатків, яка надає набір інструментів для таких функцій, як зберігання даних, автентифікація користувачів, автентифікація в режимі реального часу та розробка веб-додатків. Firebase має кілька переваг:

1. Швидкість та масштабованість. Firebase має простий, інтуїтивно зрозумілий користувацький інтерфейс та широкий спектр інструментів розробки. Це дозволяє розробникам швидко розробляти і розгортати веб-додатки на Firebase Firebase можна легко масштабувати від невеликих проектів до великих підприємств, що робить її ідеальним вибором для будь-якої команди розробників Firebase працює на серверах Google і забезпечує швидкість і масштабованість, необхідні для веб-додатків;
2. Простота використання. Firebase має простий, інтуїтивно зрозумілий інтерфейс, який дозволяє швидко створювати та розгортати веб-додатки;
3. Гнучкість. Firebase підтримує багато мов програмування і має гнучку структуру, що дозволяє розробникам працювати у власному середовищі розробки;
4. Продуктивність. Firebase забезпечує швидке зберігання даних в режимі реального часу, що дозволяє користувачам миттєво бачити зміни;
5. Надійність та безпека Firebase має сильні функції безпеки, такі як захист від злому, шифрування даних та аутентифікація користувачів, щоб гарантувати, що конфіденційна інформація зберігається надійно;
6. Інтеграція Firebase можна інтегрувати з іншими інструментами, такими як Angular, React та Vue.js, що полегшує розробку веб-додатків;

7. Інтеграція з іншими продуктами Google. Оскільки Firebase належить Google, її можна успішно інтегрувати з іншими продуктами Google, такими як Google Analytics, Google AdWords та Google Cloud Platform;
8. Простота використання. Firebase має простий і легкий інтерфейс, що дозволяє розробникам зосередитися на розробці додатків, а не на налаштуванні інфраструктури;
9. Розширюваність. Firebase пропонує ряд інструментів і сервісів, які дозволяють легко розширювати функціональність програми [6].

Ці переваги роблять Firebase дуже популярним вибором для розробки веб-додатків, особливо для стартапів і малих та середніх підприємств, які хочуть розробляти і розгортати веб-додатки швидко і ефективно.

TypeScript — одна з провідних мов програмування для розробки сайтів реєстрації онлайн-сервісів та інших веб-додатків загалом. TypeScript базується на синтаксисі JavaScript, змінних і параметрах функцій, написанні інтерфейсів, класів, декораторів і т.д. Це строго типізована мова програмування з багатьма новими можливостями. Ці можливості покращують якість коду, підвищують читабельність і розширюваність, а також зменшують кількість помилок під час розробки та супроводу проектів. TypeScript також дуже популярна серед розробників веб-додатків, оскільки вона підтримується багатьма інструментами розробки, такими як Angular, React та Vue.js [12].

TypeScript — це мова програмування, яка розширює JavaScript. Вона дозволяє розробникам писати більш безпечний і структурований код, що особливо важливо для веб-додатків, які мають справу з великими обсягами коду:
1. Підтримка статичної типізації, що зменшує кількість помилок на ранніх стадіях розробки;
2. Можливість використовувати сучасний синтаксис JavaScript з новими можливостями та функціоналом, недоступними в чистому JavaScript;
3. Забезпечує відмінну масштабованість і підтримку кодової бази;
4. Підтримує інструменти рефакторингу та автоматичного виявлення помилок.

В останні роки TypeScript став дуже популярним серед розробників веб-додатків, особливо для сайтів реєстрації онлайн-сервісів. Ось деякі з причин, чому TypeScript є однією з найпопулярніших мов програмування для цієї мети:

1. Сумісність з JavaScript. TypeScript є надмножиною JavaScript, що дозволяє розробникам використовувати всі можливості JavaScript та додаткові можливості TypeScript;
2. Підтримка сучасних функцій. TypeScript підтримує сучасні функції JavaScript, такі як асинхронні функції, обіцянки та функції-стрілки;
3. Статична типізація. TypeScript використовує статичну типізацію, що дозволяє розробникам виявляти помилки у своєму коді на етапі розробки, а не під час виконання;
4. Зрозумілість коду. TypeScript дозволяє розробникам писати більш чіткий, структурований код, який легше зрозуміти і підтримувати в майбутньому;
5. Інструменти для розробників: TypeScript надає ряд інструментів розробки, таких як редактори коду, які дозволяють розробникам писати код більш ефективно [12].

При розробці сайтів онлайн-бронювання послуг TypeScript корисна тим, що дозволяє розробникам створювати більш безпечний і структурований код, а також скорочує час, який витрачається на налагодження коду. Вона також сумісна з широким спектром фреймворків і бібліотек.

TypeScript — одна з найпопулярніших мов програмування для розробки веб-додатків; основні переваги TypeScript полягають у тому, що це строго типізована мова, що робить її надійною і дозволяє виявляти помилки під час компіляції, а не під час виконання. TypeScript має багато інших переваг, таких як розширена підтримка IDE, зручне маніпулювання об'єктами та модулями, а також широка підтримка веб-фреймворків. У контексті розробки сайтів онлайн-бронювання послуг TypeScript допомагає забезпечити високу якість коду і зробити розробку більш ефективною [11].

2.3 Дослідження проблеми при розробці веб-сайту для онлайн-бронювання послуг

Під час розробки веб-сайту для онлайн бронювання послуг може виникнути низка проблем. Нижче наведено перелік можливих проблем:

1. Безпека даних. Високий рівень захисту необхідний для запобігання неправомірному використанню або крадіжці конфіденційної інформації, такої як персональні дані. Побудова захисту від хакерів, зловмисників і шахраїв може бути проблемою;
2. Відповідністю Веб-сайти повинні відповідати правовим нормам та стандартам інтернет-безпеки;
3. Інтеграція з платіжними системами та сервісами. Якщо на сайті є можливість оплати послуг, слід забезпечити інтеграцію з платіжними системами, щоб убезпечити функціонування та уникнути можливих проблем з оплатою;
4. Інтеграція з іншими системами. У деяких випадках сайти онлайн-реєстрації на послуги потребують інтеграції з іншими системами, такими як системи управління базами даних, системи електронного документообігу тощо;
5. Функціональність. Сайт повинен мати достатній функціонал, щоб користувачі могли легко реєструвати свої послуги та знаходити необхідну інформацію;
6. Надійність. Веб-сайт повинен бути надійним і стійким, щоб користувачі могли отримати доступ до послуг у будь-який час. Використання серверів у різних географічних регіонах дозволяє забезпечити доступність сайту до користувачів, навіть у випадку проблем з мережевим зв'язком у певних областях. [14].

2.4 Загальна інформація про розробку та дослідження веб-сайтів для онлайн-бронювання послуг

Розробка сайту для онлайн-бронювання послуг — важливе завдання, яке полягає у створенні сайту, що дозволяє клієнтам зручно бронювати послуги в

режимі онлайн. Основними етапами розробки такого сайту можна вважати наступні:

1. Дослідження вимог. Перш за все, необхідно провести аналіз вимог, щоб зрозуміти, які саме послуги будуть пропонуватися, які функції повинні бути включені на сайті і які вимоги має замовник;

2. Дизайн інтерфейсу. Створіть зручний і привабливий інтерфейс, який дозволить користувачам легко реєструватися для отримання послуг. Це включає такі елементи, як реєстраційна форма, календарний планувальник і список доступних послуг;

3. Дослідження функціональності. Це означає реалізацію функцій, які дозволяють клієнтам вибирати доступні послуги, переглядати графіки роботи, вибирати дати і час, підтверджувати зустрічі та виконувати інші дії, пов'язані з процесом бронювання [16];

4. Інтеграція з платіжними системами. При оплаті послуг необхідна інтеграція з платіжними системами для забезпечення зручного та безпечного способу оплати;

5. Дослідження адміністрування. Окрема адміністративна панель повинна бути створена для того, щоб власник сервісу міг керувати програмами, послугами, записами клієнтів, аналітикою тощо;

6. Тестування та налагодження. Після розробки веб-сайту важливо протестувати його, щоб переконатися, що він працює належним чином [13].

2.5 Методи досліджень сайту по запису онлайн для послуг

Існує кілька способів дослідити сайт онлайн-бронювання послуг:

1. Гнучка методологія. Це ітеративний процес дослідження, де команда розробників працює в невеликих фазах розробки або спринтах, що дозволяє змінювати дизайн і функціональність сайту на кожному етапі на основі зворотного зв'язку та вимог користувачів;

2. Scrum. Це гнучкий підхід, при якому робота ділиться на короткі спринти. Під час спринтів команда розробників працює над конкретними завданнями, планує зустрічі, вирішує проблеми та забезпечує поетапну доставку продукту;

3. Модель водоспаду. Процес розробки та дослідження, де кожен етап відбувається лінійно. Спочатку веб-сайт проектується, потім розробляється, тестується, досліджується та публікується. Кожен етап завершується перед тим, як перейти до наступного;

4. Метод прототипування. Метод, при якому швидко розробляється прототип веб-сайту з мінімальною функціональністю, тестується на відповідність вимогам користувачів і для досліджень збирається зворотній зв'язок. Прототипи — це швидкі ітерації, які можна змінювати та вдосконалювати на ранніх стадіях розробки;

5. Використовуйте фреймворки та платформи. Готові фреймворки та платформи можна використовувати для забезпечення базової функціональності, необхідної для розробки й дослідження веб-сайту для онлайн-реєстрації. До фреймворків належать Angular, React і Vue.js, а до платформ — WordPress і Shopify. Вони забезпечують швидку розробку та дослідження та готові компоненти для реалізації основної функціональності.

6. Ощадлива розробка. Метод, спрямований на мінімізацію витрат і використання ресурсів шляхом постійного вдосконалення процесу. Він базується на циклі планування, розробки, вимірювання та вдосконалення і має на меті забезпечити максимальну цінність для користувачів, зосереджуючись на критично важливих функціональних можливостях;

7. DevOps. Методологія розробки, яка інтегрує розробку, дослідження та експлуатацію. Вона дозволяє командам розробників миттєво контролювати продуктивність сайту та вносити зміни, коли це необхідно;

8. Швидка розробка додатків (RAD). Методологія, яка використовує вже існуючі модулі та інструменти для розробки веб-сайту за дуже короткий проміжок часу для подальшого дослідження [17].

Ці методи можуть використовуватися окремо або в комбінації, залежно від потреб та уподобань проекту. Важливо вибрати підхід, який найкраще відповідає характеристикам проекту і дозволяє досягти бажаних результатів.

Висновки до розділу 2

При виконанні роботи було розглянуто літературу, що має відношення до теми цього дослідження, а саме: великі дані, аналітика. Вивчено джерела інформації про дослідження комерції подібних сайтів та їх розробку, а також проаналізовано аналогічні продукти.

Було проаналізовано та порівняно існуючі серверні та клієнтські технології за низкою критеріїв, та підібрані найоптимальніші для виконання роботи.

При проектуванні архітектури була прийнята модульна структура, яка дозволяє розділити функції системи на незалежні компоненти. Це полегшує процеси розробки, тестування та підтримки.

РОЗДІЛ 3 РОЗРОБКА ВЕБ-САЙТУ

3.1 Технічне завдання

Головною метою цього дослідження є розробка функціонального та інтуїтивно зрозумілого веб-сайту для послуг онлайн-бронювання. Розробка такого веб-сайту включатиме створення системи, яка гарантує зручну та швидку реєстрацію клієнтів, вибір необхідних послуг, доступ та доступність програми, а також можливість здійснення платежів через онлайн-платформу. Веб-сайти сприяють ефективній організації процесу реєстрації та покращують взаємодію між клієнтами та постачальниками послуг.

Для реалізації було зроблено сайт надання послуг, з вибором послуг по найбільших містах України, сайт багатофункціональний і використовує найпопулярніші технології, платіжний сервіс, підтримка гугл мапс тощо.

3.2 Використовувані технології

Підхід базується на новітніх технологіях та інструментах розробки, таких як Angular, TypeScript, Firebase.

3.2.1 Angular

Використовуючи Angular, створюється чуйний і швидкий фронтенд, який забезпечує чудовий користувацький досвід. Основними перевагами використання Angular є:

— Структурована архітектура. Angular надає модульну структуру, яка полегшує розробку та підтримку великих проектів. Компоненти, модулі та сервіси структурують код і полегшують спільну роботу команд розробників.

— Універсальність. Angular фреймоврк має вбудований функціонал для реалізації маршрутизації, форм, HTTP-запитів та багато іншого. Це дозволяє працювати з широким спектром функціональності без необхідності використання додаткових бібліотек або фреймворків.

— Шаблонізація та двонаправлене зв'язування. Angular використовує двонаправлене прив'язування даних, що полегшує розробку інтерактивних та динамічних інтерфейсів. Зміни в моделі автоматично відображаються у поданні і навпаки.

— Підтримка тестування. Angular надає вбудовану підтримку для тестування; фреймворк Angular має власні інструменти для тестування модулів, що дозволяє легко писати та запускати тести на коді.

— Кросплатформеність. Angular дозволяє розробляти крос-платформні додатки; Angular дозволяє створювати як веб-, так і мобільні додатки для Android та iOS з використанням фреймворків Ionic або Angular NativeScript. Додатки для Android та iOS з використанням фреймворків Ionic або Angular NativeScript.

— Співпраця з іншими інструментами. Angular легко інтегрується з іншими інструментами та бібліотеками, такими як RxJS для асинхронного коду та NgRx для управління станом додатку [18].

3.2.2 Typescript

TypeScript — це мова програмування, яка є засіб розробки застосунків, що розширює можливості JavaScript. Це означає, що весь коректний JavaScript-код є коректним TypeScript-кодом, але TypeScript має додаткові можливості та функціонал для покращення розробки великих та складних проектів [15].

TypeScript дозволяє писати чистий, безпечний код, що відповідає найсучаснішим стандартам програмування.

Ключові особливості TypeScript:

— Статична типізація. TypeScript дозволяє визначати типи даних для змінних, параметрів функцій та об'єктів. Це дозволяє виявляти та усувати помилки на етапі розробки.

— Мова програмування TypeScript має всі можливості JavaScript з додаванням нових концепцій, таких як інтерфейси, класи та аксесори.

— Компіляція. Код на TypeScript повинен бути скомпільований у JavaScript, перш ніж він може бути виконаний браузером або серверним середовищем виконання. Це робиться за допомогою компілятора TypeScript (tsc).

— Об'єктно-орієнтоване програмування: TypeScript підтримує об'єктно-орієнтоване програмування, включаючи такі поняття, як класи, успадкування та інтерфейси.

— Підтримка ECMAScript: TypeScript включає нові функції, визначені в стандарті ECMAScript, такі як функція стрілки та async/await.

— Сумісність з екосистемою JavaScript: більшість бібліотек і фреймворків JavaScript можна використовувати в TypeScript без змін.

— TypeScript дозволяє розробникам писати більш надійний і зручний код для великих проектів, і його використання особливо рекомендується командам, які планують створювати складні і масштабовані веб-додатки [16].

3.3 Розробка бекенд застосунку

Для розробки бекенду було використано Firebase, надійний бекенд-сервіс, він забезпечує швидке зберігання даних і доступ до них із вбудованими можливостями автентифікації та оплати [11]. При роботі з сайтом зрозуміло що буде багато роботи з даними, Firebase допоможе з цим.

Крім того, оскільки обробляється багато дат, формат дати було змінено за допомогою модуля Day.js. Платіжною системою було обрано Stripe тому що він безкоштовний [19].

3.3.1 Firebase

Firebase — це платформа від Google, яка пропонує ряд сервісів для розробки веб— та мобільних додатків. Firebase включає в себе інструменти та сервіси, які спрощують розробку, тестування та публікацію додатків. Основними компонентами Firebase є:

— База даних в режимі реального часу. Хмарна база даних, яка зберігає дані у форматі JSON і дозволяє миттєво обмінюватися даними між клієнтом та сервером.

— Аутентифікація це служба аутентифікації користувачів. Підтримуються різні методи входу, включаючи електронну пошту, паролі, Google, Facebook і Twitter. Конкретно у нашому разі ми будемо використовувати тільки Google.

— Хмарне сховище це хмарна база даних, яка підтримує документи та колекції, дозволяючи організовувати дані та робити до них відповідні запити.

— Хостинг це статичні та динамічні веб-сайти можуть бути розгорнуті на інфраструктурі Firebase.

— Хмарні функції це функції на стороні сервера, що викликаються у відповідь на інші служби Firebase або події HTTP-запитів.

— Хмарні повідомлення це служби, які надсилають повідомлення на мобільні пристрої.

— Сховище. Хмарне сховище для зберігання та обробки медіафайлів, таких як зображення та відео [16].

Firebase використовується як основа для багатьох рутинних завдань і готова до використання з іншими технологіями, такими як Angular, React і Vue.js, що дозволяє розробникам швидко публікувати і розширювати додатки.

3.3.2 Day.js

Day.js — це легка бібліотека для роботи з датою та часом у JavaScript. Вона надає зручний і простий інтерфейс для маніпуляцій з датою і часом і є альтернативою великим бібліотекам, таким як Moment.js.

Основними особливостями Day.js є:

— Легкість: Day.js має невеликий розмір, що робить його ідеальним для використання в проектах, де важлива оптимізація розміру. — Простий інтерфейс. API Day.js простий і зрозумілий, що робить його придатним для роботи з датами і часом.

— Методи локалізації. Day.js підтримує методи локалізації, що важливо при інтернаціоналізації додатків.

— Ланцюговий синтаксис. Можливо легко використовувати ланцюговий синтаксис для послідовного виклику методів.

— Підтримка плагінів. Day.js підтримує плагіни для розширення функціональності за потреби.

Бібліотека підходить для проектів, які мають справу з датами і часом, але не хочуть збільшувати розмір бібліотеки.

3.3.3 Stripe

Stripe — це платіжна платформа, яка надає інфраструктуру для обробки онлайн-платежів. Дозволяючи компаніям і розробникам приймати онлайн-платежі, Stripe користується великою популярністю серед розробників, оскільки пропонує зручний API, який дозволяє легко інтегрувати платіжну функціональність у веб-додатки [19].

Основними особливостями Stripe є:

— Простота використання. Простий API для легкої інтеграції у веб-сайти та додатки.

— Підтримка різних типів платежів. Підтримуються кредитні картки, дебетові картки та інші способи оплати.

— Захист від шахрайства — вбудовані функції безпеки для захисту від шахрайства та несанкціонованих транзакцій.

— Підтримка кастомізації Налаштування платіжні форми та кнопки відповідно до ваших потреб.

— Періодичні платежі Періодичні платежі можна налаштувати для підписок та інших послуг.

— Локалізація та глобальна підтримка. Доступна оплата у багатьох країнах і різних валютах.

Ці особливості роблять Stripe популярним вибором для онлайн-платежів для багатьох компаній і стартапів.

3.3.4 Налаштування оточення розробки

Перш за все, потрібно було встановити Angular [4] на комп'ютері. Angular включає в себе пакетний менеджер npm, який буде використовуватися для встановлення залежностей додатка. Встановлення залежностей проекту: Після створення проекту потрібно встановити всі залежності, такі як бібліотеки та інструменти, необхідні для розробки. Це можна зробити за допомогою команди npm install. Встановлення редактору коду. Вибір відповідного редактора коду є важливим кроком. БагатоAngularрозробниківвикористовуютьтакіредактори,як Visual Studio Code або WebStorm.

Створення компонентів модулю. Після того, як створився проект, почніть розробляти його,створюючи компоненти, модулі, сервіси та інші елементи за допомогою Angular CLI або вручну.

Тестування та налагодження. Після написання коду важливо протестувати і налагодити додаток, щоб переконатися, що він працює правильно і відповідає вимогам.

3.3.5 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 1.

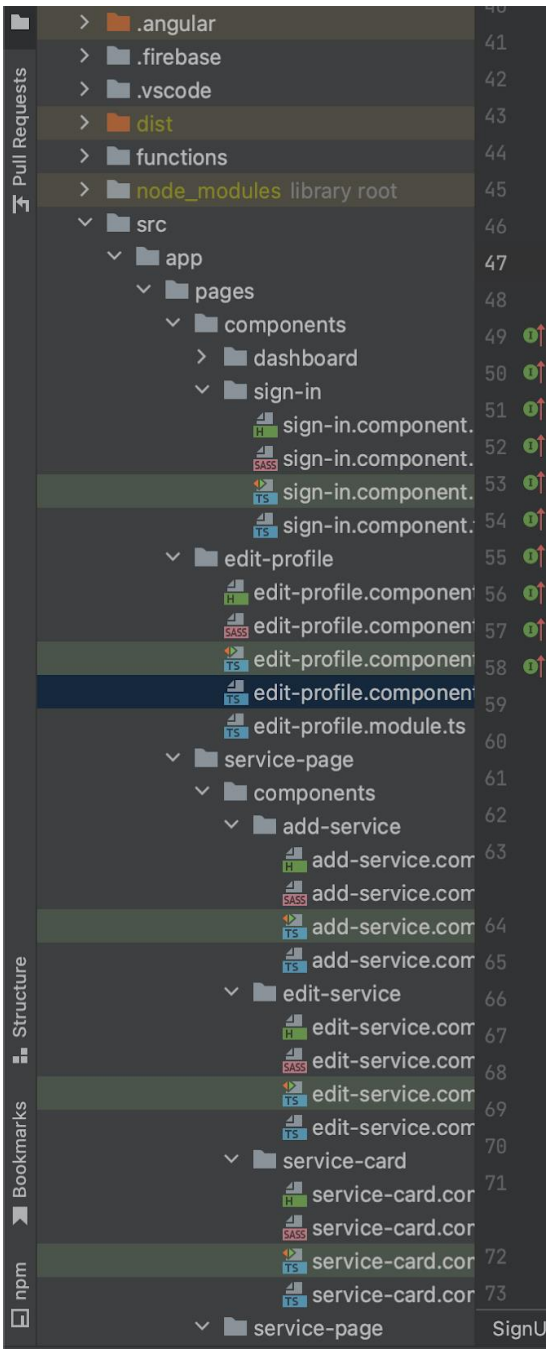

Рисунок 1 — Структура проекту

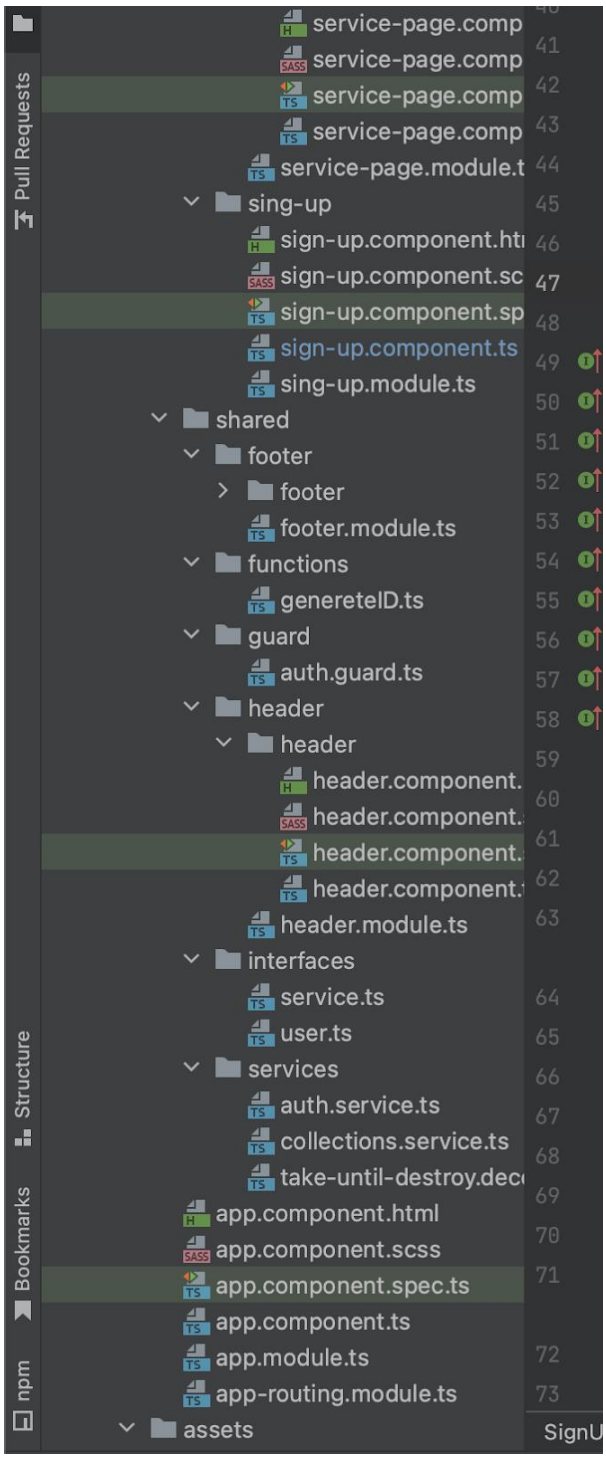

Рисунок 2 — Структура проекту (продовження)

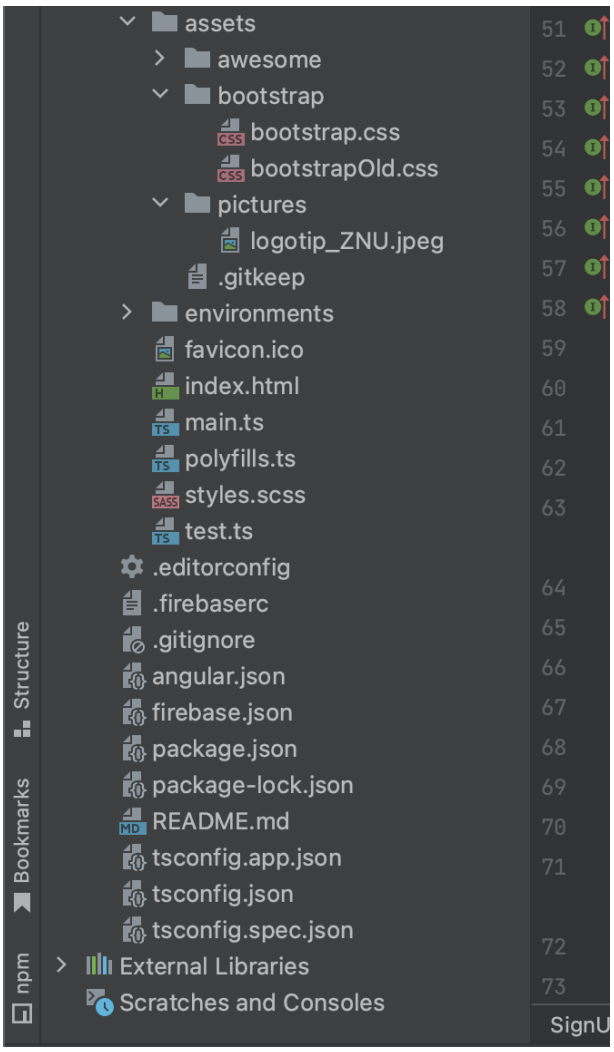

Рисунок 3 — Структура проекту (продовження)

Наступним кроком є підключення бази даних та створення моделей бази даних проекту. Тут є перелік таблиць, що створено — Appointment, Order, Services, Users і Worker_Hours. Кожна з цих таблиць буде мати свою відповідну модель, яка визначає поля та їх типи даних.

Наприклад, модель Appointment містить поля, такі як id, date, time, customer_id (ідентифікатор клієнта), service_id (ідентифікатор послуги) і т.д. Аналогічно, модель Order може містити поля, такі як id, date, customer_id, total_amount і т.д.

База даних включає таблиці для запису інформації про призначення (Appointment), замовлення (Order), послуги (Services), користувачів (Users) і робочі години працівників (Worker_Hours).

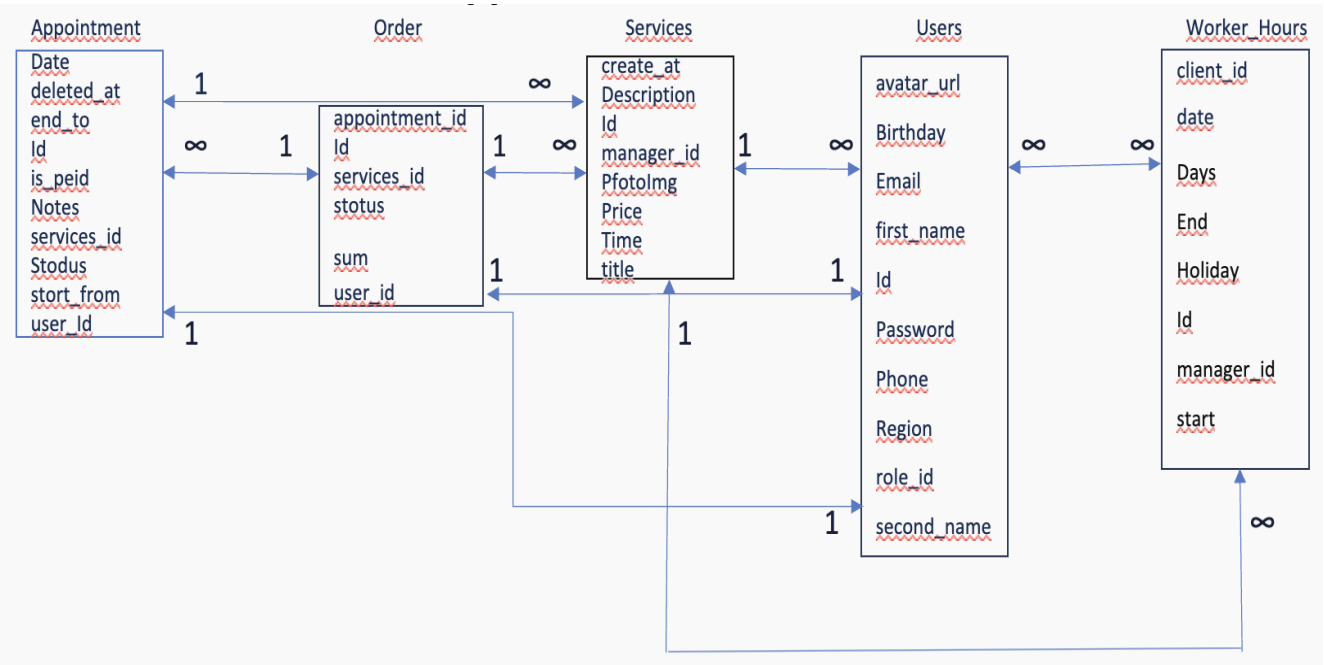

Рисунок 4 — Структура бази даних

Взаємозв'язки між таблицями:

Таблиці Appointment, Order і Services пов'язані між собою за допомогою ідентифікатора послуги (service_id). Це означає, що кожне замовлення (Appointment) пов'язане з однією послугою (Service).

Таблиця Appointment також пов'язана з таблицею Users за допомогою ідентифікатора користувача (user_id). Це означає, що кожне замовлення пов'язане з одним користувачем.

Таблиця Worker_Hours пов'язана з таблицею Users за допомогою ідентифікатора працівника (id). Це означає, що робочий час кожного працівника (Worker_Hours) пов'язаний з одним користувачем (Users).

Базу даних використовуємо для наступного:

1. Реєстрація користувачів. Таблиця Users містить інформацію про користувачів, які можуть робити замовлення.

2. Оформлення замовлень. Таблиці Order і Appointment містять інформацію про замовлення послуг.

3. Управління послугами. Таблиця Services містить інформацію про послуги, які можна замовити.

4. Управління робочим часом. Таблиця Worker_Hours містить інформацію про робочий час працівників.

Перелік полів, як частини структури даних в програмуванні. Розглянемо ці поля.

Users

avatar_url

Birthday

Email

first_name

Id

Password

Phone

Region

role_id

second_name

— Users це означає користувача, і це може об'єкт, що представляє користувача в програмному коді.

— id це ідентифікатор користувача. Унікальний номер, що ідентифікує конкретного користувача.

— fn це скорочення від "first name" (ім'я). Це поле містить ім'я користувача.

— sn це скорочення від "second name" (прізвище). Це поле може містити прізвище користувача.

— role_id це ідентифікатор ролі користувача в системі.

— e-mail це адреса електронної пошти користувача.

— phone це номер телефону користувача.

— avatar_url це ідентифікатор зображення (аватара) користувача. Приклад створення об'єкту користувача наведений у лістингу 1

```
interface User {
  id: string;
  fn: string;
  sn: string;
  birthday: Date;
  vole_id: string;
  email: string;
  phone: string;
  avatar_id: string;
}
```

Лістинг 1 — Створення об'єкту користувача

Приклад запису для структури Day надається у лістингу 2

```
const user: User = {
  id: "12345",
  fn: "Ім'я",
  sn: "Прізвище",
  birthday: new Date("1990-01-01"),
  vole_id: "user",
  email: "user@example.com",
  phone: "+1234567890",
  avatar_id: "avatar123",
};
```

Лістинг 2 — Приклад створення об'єкту користувача

Фрагмент структури записів мовою програмування. Ось опис у формі TypeScript.

Тут використано TypeScript для оголошення типів. Використання enum

дозволяє обмежити значення day конкретними днями тижня, а тип WorkHours описує структуру об'єкта з робочим графіком.

Таким чином, були створені наступні моделі, такі як PaymentCards, що зберігає банківські картки користувача. Ця таблиця має зв'язок "один-до-багатьох" з моделлю "User" через проміжну таблицю "UserPaymentCards". Спочатку створюється нова модель "UserPaymentCards". Перший аргумент задає ім'я моделі, а другий — об'єкт, який визначає поля і типи моделі: 'id': це поле типу integer і слугує унікальним ідентифікатором для зв'язку між користувачем та платіжною карткою. Його значення автоматично інкрементується (autoIncrement: true) і використовується як первинний ключ (primaryKey: true). Третій аргумент — об'єкт з опцією "timestamps: false" для відключення автоматичної генерації полів createdAt та updatedAt цієї моделі.

Потім встановлюється зв'язок між моделями "User" та "UserPaymentCards": User.hasMany(UserPaymentCards)": Між моделями "User" і "UserPaymentCards" встановлюється зв'язок "один-до-багатьох". В результаті до моделі "User" додається метод getUserPaymentCards для отримання всіх записів в моделі "UserPaymentCards" конкретного користувача.

UserPaymentCards.belongsTo(User): Встановлює зв'язок "один-до-багатьох" між моделлю "UserPaymentCards" і моделлю "User". Це означає, що кожен запис в моделі "UserPaymentCards" належить тільки одному користувачеві в моделі "User". В результаті модель "UserPaymentCards" матиме додаткове поле UserId, яке є зовнішнім ключем, що виражає ідентифікатор користувача моделі "User".

Після цього встановлюється зв'язок між моделлю "PaymentCards" і моделлю "UserPaymentCards":

PaymentCards.belongsTo(UserPaymentCards): Між моделями "PaymentCards" і "UserPaymentCards" встановлюється зв'язок "один-до-багатьох". В результаті до моделі PaymentCards додається поле з назвою UserPaymentCardId, яке є зовнішнім ключем, що виражає ідентифікатор запису моделі UserPaymentCards.

UserPaymentCards.hasOne(PaymentCards): Встановлює зв'язок один-до-одного між моделлю "UserPaymentCards" і моделлю "PaymentCards". Це означає, що кожен запис в моделі "UserPaymentCards" може мати тільки одну платіжну картку в моделі "PaymentCards". В результаті до моделі "UserPaymentCards" додається метод getPaymentCard, який дозволяє отримати платіжну картку для цього запису.

Авторизація — оскільки постачальники послуг потребують відповідного облікового запису для власних послуг, то спочатку було створено кінцеву точку для авторизації.

— signUp. Спочатку були імпортовані всі необхідні модулі. А саме: keys, який містить ключі та налаштування, в тому числі ключі підпису.

Далі створюється функція signUp, яка отримує параметри. Тіло запиту ініціалізує змінні, отримані при руйнуванні об'єкта, а саме: ім'я, прізвище, email та пароль тощо. Цей запит винесено в окрему функцію, оскільки він буде потрібен іншим компонентам. Його вміст показано у лістингу 3.

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from
'../../shared/services/auth.service';
import { FormBuilder, FormControl, FormGroup, Validators } from
'@angular/forms';
import { Router } from '@angular/router';
import { User } from '../../shared/interfaces/user';
@Component(
  {
    selector: 'app-sign-up',
    templateUrl: './sign-up.component.html',
    styleUrls: ['./sign-up.component.scss'],
  }
)
```

Лістинг 3 — Створення функції signUp

Це початок класу для компонента "sign-up" у Angular. Тут використовуються деякі Angular модулі та сервіси для створення реєстраційної сторінки.

Тут вказується селектор компонента ('app-sign-up'), шлях до файлу шаблону та файли стилів.

Клас компонента. Тут оголошуються змінні та об'єкти, необхідні для роботи компонента. Конструктор отримує необхідні сервіси (в даному випадку, AuthService, FormBuilder та Router).

Залишається реалізувати логіку реєстрації та інші методи, а також зв'язати їх з шаблоном та формами, що показано у лістингу 4.

```
export class SignUpComponent implements OnInit {
  public formRegistration: FormGroup;
  public hide: boolean = false;
  public hide1: boolean = false
  public equal: boolean = false;
  public role: number = 0;
  public region: string = 'Kyiv';
  constructor(
    private formBuilder: FormBuilder,
    private authService: AuthService,
  ) {
    this.formRegistration = this.formBuilder.group({
      email: ['', [Validators.required,Validators.email]],
password: ['', [Validators.required, Validators.min(6)]],
first_name: ['', [Validators.required]],
second_name: ['', [Validators.required]],
phone: ['', [Validators.required]],
birthday: ['', [Validators.required]],
confirm: ['', [Validators.required]],
        });
      }
      ngOnInit(): void { }
      onSubmit(): void {

    console.warn('this.role: ', this.role);
    console.warn('this.region: ', this.region);
    this.formRegistration.markAllAsTouched();
    if (this.formRegistration.invalid) {
      return;
      }
      const data: User = {
        id: '',
email: this.formRegistration.get('email')?.value,
first_name: this.formRegistration.get('first_name')?.value,
second_name: this.formRegistration.get('second_name')?.value,
avatar_url: '',
```

```
birthday: this.formRegistration.get('birthday')?.value,
phone: this.formRegistration.get('phone')?.value,
role_id: this.role,
region: this.region,
password: this.formRegistration.get('password')?.value,
        }
console.warn('User data: ', data);
// this.authService.SignUp(data);
      }
```

Лістинг 4 — Реалізація логіки реєстрації

Використовується FormGroup для групування форми, а FormBuilder для створення екземпляру форми у конструкторі. Були оголошені деякі змінні та флаги, які, пов'язані з роботою форми та логікою компонента. Конструктор і ініціалізація форми. У конструкторі ініціалізуєте formRegistration з допомогою FormBuilder, додаючи необхідні валідатори для кожного поля форми.

Метод onSubmit:Метод onSubmit викликається при поданні форми. Викликається markAllAsTouched, щоб позначити всі поля як "доторкнуті" (щоб вивести повідомлення про помилку, якщо вони є), а потім перевіряєте, чи форма є валідною. Якщо форма валідна, ви створюєте об'єкт User і виводите його у консоль. HTML-шаблон:

Верхній фрагмент HTML-шаблону (який не наведений тут, покажу його у розділи фронтенду) містить розмітку для форми, ймовірно, використовуючи formGroup, formControlName і т. д.

Загалом, цей компонент є готовим для реєстрації користувачів, проте я планую розширити його логіку для вирішення своїх потреб.

Додатково показую це у лістинг 5.

```
checkPassword(): void {
    if (this.formRegistration.get('password')?.value ===
this.formRegistration.get('confirm')?.value) {
      this.equal = false;
      return;
    }
    this.equal = true;
  }
  GoogleAuth(): void {
```

```
    this.authService.GoogleAuth(true, this.role).then(
      (data): void => {
        console.warn('data: ', data);
      }
    ).catch(
      (error): void => {
        console.debug('error', error)
      }
    )
  }
  showPassword(): void {
    this.hide = !this.hide;
  }
  showConfirmPassword(): void {
    this.hide1 = !this.hide1;
  }
}
```

Лістинг 5 — Розширення компоненту логіки

Метод checkPassword. Цей метод перевіряє, чи поле "password" співпадає з полем "confirm". Якщо так, то this.equal встановлюється в false (означає, що паролі різняться), в іншому випадку — встановлюється в true.

Метод GoogleAuth:Цей метод викликає GoogleAuth сервісу AuthService. Це метод для автентифікації користувача за допомогою Google. Він використовує обіцянки (Promise) для обробки успішного та неуспішного результатів автентифікації.

Методи showPassword та showConfirmPassword. Ці методи відповідають за відображення/приховування символів пароля та підтвердження пароля. Вони встановлюють значення this.hide і this.hide1 в обернене значення їх поточних значень.

Код виглядає логічно та готовий для використання. Методи у коду дозволяють контролювати різні аспекти реєстрації користувачів та спрощують взаємодію з сервісом автентифікації.

Роботу компоненту sing-up.module.ts показана у лістингу 6.

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

```
import { RouterModule } from '@angular/router';
import { SignUpComponent } from './sign-up.component';
import { FormsModule, ReactiveFormsModule } from
'@angular/forms';
@NgModule({
  declarations: [
    SignUpComponent
  ],
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    RouterModule.forChild([
      { path: '', component: SignUpComponent, pathMatch: 'full'
}
    ])
  ]
})
export class SingUpModule { }
```

Лістинг 6 — Компонент sing-up.module.ts

— NgModule декоратор. Визначає компоненти, директиви та інші видимі для цього модулю ресурси.

— imports. Визначає інші модулі, які потрібні для цього модулю. У цьому випадку використовуються CommonModule для загальних Angular директив, FormsModule та ReactiveFormsModule для роботи з формами, і RouterModule для налаштування маршрутів.

— RouterModule.forChild([...]). Ця частина визначає маршрут для компонента SignUpComponent. Коли URL відповідає шляху '' (пустий рядок), компонент SignUpComponent буде відображений. pathMatch: 'full' означає, що шлях повинен збігатися повністю для відображення компонента.

— CommonModule, FormsModule, ReactiveFormsModule:

CommonModule. Модуль, який містить загальні для Angular директиви, такі як ngIf, ngFor.

FormsModule. Модуль для роботи з шаблонами та двосторонніми прив'язками в Angular.

ReactiveFormsModule. Модуль для роботи з реактивними формами в Angular.

Визначає компоненти, які належать до цього модулю. У цьому випадку, лише SignUpComponent.

Цей модуль дозволяє логічно групувати та ізолювати функціональність реєстрації користувачів та підключає необхідні модулі та компоненти для його роботи.

— signIn. Спочатку імпортуються всі необхідні модулі (так само, як і в попередній кінцевій точці). Потім створюється функція signIn для авторизації, яка отримує параметри.

Далі ініціалізуються дані користувача, такі як електронна пошта та пароль, шляхом реконструкції об'єкта.

Після ініціалізації змінних викликається функція, яка перевіряє, чи існує користувач із зазначеним email. Отримані дані користувача зберігаються у змінній user.

Після отримання даних користувача хешований пароль, переданий у POST-запиті, порівнюється з хешованим паролем, що зберігається в базі даних. Результати порівняння зберігаються у змінній checkPassword. Потім перевіряється, чи збігається пароль, і якщо ні, то повертається відповідь з повідомленням про неправильний пароль і статусом 404 (не знайдено), інакше створюється новий JWT-токен з об'єктом, що містить дані користувача, такі як електронна пошта (отримана з функції checkEmail) та ім'я.

Після створення токена запис користувача в таблиці User оновлюється шляхом оновлення поля новим значенням токена. Це оновлення відбувається за умови, що поле email таблиці має значення, яке відповідає електронній пошті користувача.

Після виконання запиту у відповідь повертається JSON-об'єкт, що містить новий токен, дані користувача і статус 200 (успіх). Якщо виникає помилка, для зручності тестування в консолі відображається повідомлення про помилку з HTTP-кодом 500 (внутрішня помилка сервера), а у відповідь повертається повідомлення про помилку.

Повний фрагмент коду цієї кінцевої точки наведено в лістингу 7

```
import { Component, OnInit } from '@angular/core';
import { AuthService } from
'../../../shared/services/auth.service';
import { FormBuilder, FormControl, FormGroup, Validators } from
'@angular/forms';
import { Router } from '@angular/router';
import { Store } from '@ngrx/store';
import * as AuthActions from
'../../../store/auth/actions/auth.actions';
import {signInRequest} from
"../../../store/auth/actions/auth.actions";
@Component({
  selector: 'app-sign-in',
  templateUrl: './sign-in.component.html',
  styleUrls: ['./sign-in.component.scss']
})
```

Лістинг 7 — Відображення сторінки входу користувача

Цей компонент Angular відповідає за відображення сторінки входу користувача. Основні частини цього компонента:

— Component декоратор, selector CSS селектор, який визначає, як компонент буде вибиратися в HTML-документах.

— templateUrl. Шлях до зовнішнього файлу шаблону для цього компонента.

— styleUrls. Масив шляхів до зовнішніх файлів стилів для цього компонента.

Імпорт модулів та сервісів:

— AuthService. Сервіс, який надає функціональність для автентифікації.

— FormBuilder. Клас для конструювання об'єктів форм у Angular.

— Router. Сервіс для навігації між різними частинами додатка.

— Store. Сервіс NgRx Store для управління станом додатка.

— AuthActions. Модуль, що містить дії для автентифікації в контексті NgRx.

Клас компонента:

— formLogin. Об'єкт форми для обробки введених даних користувача.

— hide. Змінна, яка використовується для показу/приховування пароля.

— onSubmit(). Метод, який викликається при натисканні на кнопку "Увійти". Він отримує дані форми та викликає дію для автентифікації через NgRx.

Цей компонент відповідає за відображення форми входу користувача, обробку введених даних та взаємодію зі службою автентифікації та NgRx для керування станом. Продовження цього є у наступному лістингу 8

```
export class SignInComponent implements OnInit {
  public loginForm: FormGroup;
  public hide: boolean = false;
  public errorMessage: boolean = false;
  constructor(
    private authService: AuthService,
    private formBuilder: FormBuilder,
    private router: Router,
    private store: Store<any>
  ) {
    this.loginForm = this.formBuilder.group({
    email: ['', [Validators.required, Validators.email]],
    password: ['', [Validators.required]],
    })
  }
  ngOnInit() { }
  onSubmit() {
    if (this.loginForm.invalid) {
      this.errorMessage = true;
      return;
    }
    this.errorMessage = false;
    const email = this.loginForm.get('email')?.value;
  const password = this.loginForm.get('password')?.value;
this.store.dispatch(AuthActions.signInRequest({email, password
}));
this.store.select('auth').subscribe(data) => {
console.log('auth data: ', data);
      }
    )
```

Лістинг 8 — Автефікація та валідація користувача

— Router. Сервіс для навігації між різними частинами додатка;

— Store. Сервіс NgRx Store для управління станом додатка;

— errorMessage. Змінна, яка слугує для відображення повідомлення про помилку;

— onSubmit(). Метод, який викликається при натисканні на кнопку "Увійти". Перевіряє, чи форма валідна, і відправляє дані до NgRx через відповідну дію.

В загальному, цей компонент відповідає за відображення форми входу, перевірку введених даних, відправку їх для автентифікації через NgRx та відображення стану аутентифікації згідно зі змінами. Додаткові зміни надаємо у лістингу 9

```
GoogleAuth() {    this.authService.GoogleAuth().then().catch(
           (error) => {
            console.error('error', error)        }
       )
     }
showPassword() {
    this.hide = !this.hide;
  }
testLink() {
  this.router.navigate(['dashboard']).then();
  }
}
```

Лістинг 9 — Обробка подій

Цей фрагмент коду представляє методи компонента Angular, який відповідає за обробку подій та взаємодію з іншими частинами додатка. Розглянемо кожен метод:

— GoogleAuth(). Цей метод викликається при спробі аутентифікації через Google. Він викликає відповідний метод GoogleAuth() з сервісу автентифікації (AuthService). Якщо операція успішна, не виконується жодних дій (then() пустий). Якщо виникає помилка, виводиться повідомлення про помилку в консоль;

— showPassword(). Цей метод викликається при натисканні кнопки або елемента для переключення видимості пароля. Змінна hide (вона оголошена в іншому місці в компоненті) змінюється на протилежне значення, вказуючи на зміну стану видимості пароля;

— testLink(). Цей метод викликається при спробі навігації до сторінки "dashboard". Використовує сервіс Router для навігації.

Загалом, ці методи відображають типові завдання взаємодії з віддаленим сервісом автентифікації (у випадку Google), управління видимістю пароля та навігацію між сторінками додатка. Розглянемо sign-in.component.spec.ts у лістингу.

```
import { ComponentFixture, TestBed } from
'@angular/core/testing';
import { SignInComponent } from './sign-in.component';
describe('SignInComponent', () => {
  let component: SignInComponent;
  let fixture: ComponentFixture<SignInComponent>;
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [SignInComponent]
    });
    fixture = TestBed.createComponent(SignInComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Лістинг 10 — Створення компоненту sign-in.component.spec.ts

Це оголошення тесту. Використовується describe для групування тестів. У цьому випадку група називається 'SignInComponent'.

— Ініціалізація компонента і фікстури. Використовуються beforeEach і TestBed.configureTestingModule, щоб налаштувати середовище тестування. До кожного тесту створюється компонент (SignInComponent) і відповідний ComponentFixture.

— Тест "should create". Здійснюється перевірка, чи був компонент створений без помилок за допомогою expect(component).toBeTruthy();.

Цей тестовий набір перевіряє базовий сценарій створення компонента. Якщо компонент успішно створений, тест вважається успішним.

— edit-profile, вказує на сторінку та її функціонал, де користувач може редагувати свій профіль. Це може включати в себе можливість змінювати

інформацію про користувача, таку як ім'я, прізвище, адреса електронної пошти, фотографія та інші аспекти профілю.

На практиці реалізація "edit-profile" є частиною веб-сайту, де користувач може авторизуватися та виконувати редагування свого профілю. Це включає в себе використання форм, валідацію даних і використання зображень та інших мультимедійних елементів для оновлення профілю. Код редагування профілю надається у лістингу 11

```
    import { Component, OnInit, ViewChild, ElementRef } from
'@angular/core';
    import { FormBuilder, FormControl, FormGroup, Validators }
from '@angular/forms';
    import { AngularFireStorage } from
'@angular/fire/compat/storage';
    import { finalize } from 'rxjs/operators';
    import { User } from '../../shared/interfaces/user';
    import { CollectionsService } from
'../../shared/services/collections.service';
    import { take } from 'rxjs';
    @Component({
      selector: 'app-edit-profile',
      templateUrl: './edit-profile.component.html',
      styleUrls: ['./edit-profile.component.scss'],
    })
```

Лістинг 11 — Редагування профілю

Цей Angular компонент є частиною додатка, який відповідає за редагування профілю користувача. Давайте розглянемо деякі ключові елементи цього коду: Залежності та імпорти:

— Component, OnInit: Декоратор та інтерфейс Angular для створення компоненту та реалізації методу життєвого циклу ngOnInit.

— ViewChild, ElementRef: Angular дозволяє отримувати доступ до DOM-елементів за допомогою ViewChild та ElementRef.

— FormBuilder, FormControl, FormGroup, Validators. Використовується для створення та валідації форм.

AngularFireStorage. AngularFireStorage здійснює взаємодію з Firebase Storage. Використовується для завантаження та збереження файлів, зображень.

RxJS та Angular сервіси:

— finalize. Оператор RxJS, який викликає певний код, коли вказана Observable завершується (зазвичай використовується для закриття ресурсів, таких як завантаження файлів).

— CollectionsService: Це сервіс, який надає методи для взаємодії з колекціями, користувачами або іншими даними.

Компонент:

— Маємо FormGroup (formProfile) для об'єднання полів форми.

Є метод onSubmit(), який, обробляє відправку форми, та зберігає дані в базі даних.

Використовуються методи showPassword і showConfirmPassword для відображення або приховання паролів.

Є можливість вибору фотографії профілю, через інтерфейс вибору файлу.

Цей компонент є частиною більшого додатка, і для повного розуміння потрібно більше контексту, зокрема вивчення того, як цей компонент взаємодіє з іншими частинами додатка та в якому середовищі він використовується. Продовження додаю у лістингу 12

```
export class EditProfileComponent implements OnInit {
  // @ts-ignore
  @ViewChild('fileInput') fileInput: ElementRef = new
ElementRef<T>;
  public formEditUser: FormGroup = new FormGroup({});
  public avatarUrl: string = '';
  public hide: boolean = false;
  public hideConfirm: boolean = false;
  public equal: boolean = false;
  public loading: boolean = false;
  public role: number = 0;
  public region: string = 'Kyiv';
  private userID = '';
  constructor(
    private formBuilder: FormBuilder,
    private storage: AngularFireStorage,
```

```
    private collection: CollectionsService,
  ) { }
```

Лістинг 12 — Взаємодія компонента з іншими частинами додатка

Компонент EditProfileComponent містить наступні елементи:

— ViewChild. Використовується для доступу до елементу DOM (зображення або іншого файлового вводу) за допомогою @ViewChild('fileInput') fileInput: ElementRef.

Зауважую, що використовується @ts-ignore, щоб заблокувати тип (анотація типу T). Краще визначити тип, якщо це можливо, для отримання покращеної безпеки типів.

Змінні та властивості:

— avatarUrl: Рядок, що представляє URL зображення користувача;

— hide, hideConfirm. Властивості, що вказують, чи прихований пароль та підтвердження пароля відповідно;

— equal. Логічна властивість, що вказує, чи рівні пароль та його підтвердження;

— loading. Логічна властивість, що вказує на наявність завантаження (можливо, фотографії або інших даних);

— role. Номер ролі користувача;

— region. Рядок, що представляє регіон користувача;

— userID. Закритий ключ для ідентифікації користувача.

Конструктор:

— Ініціалізує об'єкт EditProfileComponent.

Приймає об'єкти formBuilder, storage (AngularFireStorage) та collection (CollectionsService) через ін'єкцію залежностей.

Для повного розуміння, як цей компонент використовується та взаємодіє з іншими частинами сайту, може бути корисно дослідити його використання в контексті шаблонів та інших частин. Продовження коду у лістингу 13

```
ngOnInit(): void {
```

```
  this.loading = true;
  let userData: any = localStorage.getItem('user') || '';
    if (userData) {
      userData = JSON.parse(userData);
      this.userID = userData['uid'];
this.collection.getOneItemToCollection('Users',
this.userID).pipe(
        take(1)
      ).subscribe(
        (usr) => {
          const user: User = usr as User;
if (user.avatar_url) {this.avatarUrl = user.avatar_url;
          }
this.role = user.role_id;          if (user.region) {
this.region = user.region;
          }
this.formEditUser = this.formBuilder.group({
email: [user.email, [Validators.required, Validators.email]],
password: [user.password, [Validators.required,
Validators.min(6)]],
first_name: [user.first_name, [Validators.required]],
second_name: [user.second_name, [Validators.required]],
phone: [user.phone, [Validators.required]],
birthday: [user.birthday, [Validators.required]],
avatar_url: [user.avatar_url],
confirm: [user.password, [Validators.required]],
               });
this.loading = false;
             }
           )
         }
       }
```

Лістинг 13 — Шаблони компоненту редагування профілю

Цей метод ngOnInit у Angular компоненті виконується при ініціалізації компонента. Давайте розглянемо його частини.

Отримання інформації з локального сховища:

Обираються дані користувача із локального сховища браузера. Припускається, що userData містить об'єкт з інформацією про користувача.

Отримання даних про користувача з колекції:

Використовує метод для отримання даних про користувача з колекції, можливо, з використанням Firestore. Оператор take(1) вказує на те, що після отримання одного значення потік буде завершено.

Обробка даних користувача та ініціалізація форми:

Обробляє інформацію про користувача та ініціалізує форму formEditUser. Якщо у користувача є URL аватара та регіон, вони також ініціалізуються. Збірка валідаторів використовується для валідації полів форми.

Цей метод готує компонент до відображення інформації про користувача в шаблоні та забезпечує, що форма буде заповнена потрібними даними. Продовження коду треба дивитися у наступному лістингу 14

```
uploadImage(event: any) {
    let file = event.target.files[0];
    let filePath = 'avatars/' + file.name;
    let fileRef = this.storage.ref(filePath);
    let task = this.storage.upload(filePath, file);
    task.snapshotChanges().pipe(
      finalize(() => {
fileRef.getDownloadURL().pipe(take(1)).subscribe(downloadURL =>
{
      this.avatarUrl = downloadURL;
            });
          })
        ).subscribe();
       }
       openEditAvatar() {
this.fileInput.nativeElement.click();
       }
showPassword(): void {
this.hide = !this.hide;
       }
       showConfirmPassword(): void {
         this.hideConfirm = !this.hideConfirm;
       }
checkPassword(): void {
if (this.formEditUser.get('password')?.value ===
this.formEditUser.get('confirm')?.value) {
this.equal = false;
      return;
    }
    this.equal = true;
  }
```

Лістинг 14 — Відображення зображень, паролей, редагування аватарів

Цей фрагмент коду відповідає за ряд операцій, пов'язаних із завантаженням та редагуванням аватара користувача:

uploadImage(event: any)

Отримання вибраного файлу з події завантаження (event.target.files[0]).

Формування шляху до файлу в хмаровому сховищі (Firebase Storage) та отримання посилання на файл.

Завантаження файлу в хмарове сховище.

Після завершення завантаження отримання URL-адреси завантаженого файлу.

Оновлення avatarUrl для відображення нового аватара користувача.

openEditAvatar()

Виклик методу прихованого введення файлу (виглядиме як клік на невидимий інпут файлу).

showPassword() та showConfirmPassword()

Зміна значень змінних hide та hideConfirm для відображення або приховання паролів у формі.

checkPassword()

Перевірка, чи пароль збігається з підтвердженням паролю.

Встановлення значення equal відповідно до результату перевірки.

Ці методи служать для взаємодії з формою та обробки різних аспектів користувацького інтерфейсу, таких як завантаження аватара, переключення відображення паролів тощо. Закінчення коду у лістингу 15

```
    onSubmit() {
        this.formEditUser.markAllAsTouched();
        console.warn('formEditUser.value: ',
this.formEditUser.value)
        if (this.formEditUser.invalid) {
          return;
        }
```

```
        console.info('formEditUser.value: ',
this.formEditUser.value)
        const data = {
          ...this.formEditUser.value,
          avatar_url: this.avatarUrl,
          role_id: this.role,
          region: this.region,
        }
    this.collection.updateItemToCollection('Users',
this.userID, {...data}).then(
          (value) => {
            console.info('update value:', value);
            alert('Update Data user!');
          }
        ).catch(
          (error) => {
            console.error('Error  value:', error);
            alert(`Error Data user!\n ${error}`);
          }
        );
      }
    }
```

Лістинг 15 — Подання форми при редагуванні профілю

Цей фрагмент коду відповідає за обробку подання форми при редагуванні профілю користувача:

— this.formEditUser.markAllAsTouched();

Позначає всі поля форми як "доторкані" (touched). Це важливо для валідації та показу помилок користувачеві.

— if (this.formEditUser.invalid) {...}. Перевірка на валідність всіх полів форми. Якщо форма не є валідною, викликається вихід з функції.

const data = { ...this.formEditUser.value, avatar_url: this.avatarUrl, role_id: this.role, region: this.region };

Формування об'єкту data, який містить дані, введені у форму редагування профілю. До цих даних додаються URL аватара, ідентифікатор ролі та регіон користувача.

— this.collection.updateItemToCollection('Users', this.userID, {...data}).then(...) .catch(...);

Виклик сервісного методу для оновлення даних користувача в колекції 'Users' на основі ідентифікатора користувача (this.userID).

Виведення повідомлення про успішне чи невдачне оновлення.

Цей метод викликається при поданні форми редагування профілю користувача і робить оновлення даних відповідно до введених користувачем значень.

Розглянемо код файлу edit-profile.component.spec.ts. Код надається у лістингу 16

```
import { ComponentFixture, TestBed } from
'@angular/core/testing';
import { EditProfileComponent } from './edit-
profile.component';
describe('EditProfileComponent', () => {
  let component: EditProfileComponent;
  let fixture: ComponentFixture<EditProfileComponent>;
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [EditProfileComponent]
    });
    fixture = TestBed.createComponent(EditProfileComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });
  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Лістинг 16 — Компонент edit-profile.component.spec.ts

Цей код є частиною тестування юніт-тестів для компонента Angular. Давайте розглянемо його детальніше:

— TestBed.configureTestingModule. TestBed — це Angular Testing Utility, який забезпечує середовище для запуску тестів Angular;

— configureTestingModule визначає модуль тестування, який містить компонент EditProfileComponent;

— fixture = TestBed.createComponent(EditProfileComponent);

— createComponent створює екземпляр компонента EditProfileComponent для подальших тестів;

— fixture — це об'єкт типу ComponentFixture, який дозволяє вам отримати доступ до компонента та його DOM елементів під час тестування;

— it('should create', () => {...}). Це фактичний тест, який перевіряє, чи компонент був створений успішно;

— expect(component).toBeTruthy(); перевіряє, чи змінна component має значення (не є null, undefined або false), що означає, що компонент був успішно створений.

При виконанні цього тесту, якщо компонент EditProfileComponent ініціалізується і не має помилок у процесі створення, тест буде успішним. Такі тести використовуються для перевірки базового функціоналу компонента та його коректності при створенні.

3.4 Розробка фронтенд застосунку

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

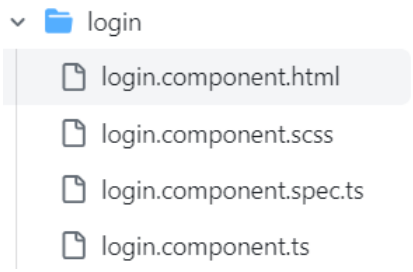

Рисунок 5 — Структура папки сторінки авторизації

Файл login.component.html містить сам макет, але використовує html-теги Angular. Тег <main> визначає основний блок сторінки входу і має клас container-login. Блок має атрибут [formGroup]="formLogin" і прив'язаний до форми Angular. Він також має класи ui, raised, segment і block-form, які використовуються для стилізації.

Далі йде блок із зображенням логотипу, що відображається в середині блоку, та абзац із заголовком "Увійдіть за допомогою свого облікового запису", клас центрального блоку та клас заголовка.

Потім створюється форма.

Блок, що містить поле для введення адреси електронної пошти, відповідну мітку, підказку про помилку і клас для відображення помилки.

Блок, що містить поле для введення пароля (Password), відповідні мітки, повідомлення про помилки та кнопку для показу/приховування пароля.

Блок містить прапорець "Запам'ятати мене" та посилання "Забули пароль?".

Є кнопка "Вхід" з класом і властивістю [disabled], яка визначає, чи є форма відключеною, залежно від того, чи торкалися кнопки, чи ні.

Цей блок має напис "Не маєте облікового запису? Будь ласка, зареєструйтесь".

Код має структуру класичної форми входу з елементами керування для полів введення, чекбоксами, кнопками тощо і використовує Angular для відображення форми та взаємодії з компонентом.

Це форми веб-сайту, написаний з використанням Angular та Bootstrap.
Основні енразглядені елементи та їх пояснення:
— <input type="text" class="form-control" placeholder="Email" formControlName="email">: Поле вводу для введення email. formControlName="email" вказує на те, що це поле пов'язане з властивістю 'email' у FormGroup, що дозволяє Angular Forms взаємодіяти з цим полем.

У цілому, цей код представляє базову форму входу з полем для введення email, яке має відповідну мітку та може відображати помилки валідації за допомогою Bootstrap класу 'has-error'. Продовження коду у лістингу 28

— : Рядок Bootstrap для центрування тексту;

— <p *ngIf="loginForm.controls['email'].invalid && loginForm.controls['email']. touched" class="text-danger"> Error email : Елемент з умовою виведення помилки. Використовується Angular директива *ngIf для визначення, чи елемент має бути відображений. Умова вказує на те, що елемент буде відображений, якщо поле email є недійсним і йому вже доторкнулися (touched). Якщо ця умова виконується, виводиться текст помилки "Error email";

— : Ще один рядок Bootstrap для розміщення наступного елемента форми;

— : Контейнер для групування елементів форми;

— <label class="col-md-2 control-label text-info">Password *</label>: Мітка для поля вводу паролю. Знову використовується Bootstrap для визначення ширини колонки;

— <div class="col-md-6 input-group" [ngClass]="(loginForm.controls['password']. invalid && loginForm.controls['password'].touched) ? 'has-error' : "">: Контейнер для вводу паролю і кнопки переключення видимості паролю. Використовується Bootstrap для визначення ширини колонки і Angular директива [ngClass] для умовного додавання класу 'has-error', якщо поле паролю є недійсним і йому вже доторкнулися;

— <input [type]="hide ? 'text' : 'password'" class="form-control" placeholder="Password" formControlName="password">: Поле вводу для паролю з умовним встановленням типу вводу (text або password) в залежності від значення hide. Також використовує formControlName="password", щоб пов'язати поле з відповідним елементом у FormGroup;

— : Контейнер для кнопки, яка переключає видимість паролю;

— <button class="btn btn-default" type="button" (click)="showPassword()">: Кнопка, яка викликає метод showPassword() при натисканні;

— <i *ngIf="!hide" class="fa fa-eye-slash" aria-hidden="true"></i>: Іконка "закритого ока", яка відображається, якщо hide має значення false;

— <i *ngIf="hide" class="fa fa-eye" aria-hidden="true"></i>: Іконка "відкритого ока", яка відображається, якщо hide має значення true.

Загально, цей код доповнює попередню форму вводу, додаючи поле для паролю, перевірку його валідації, та кнопку для переключення видимості паролю.

Цей фрагмент коду представляє іншу частину HTML-шаблону для форми входу. Давайте розглянемо кожен елемент окремо:

— : Рядок Bootstrap для центрування тексту;

— <p *ngIf="loginForm.controls['password'].invalid && loginForm.controls['password']. touched" class="text-danger"> Error password : Елемент з умовою виведення помилки. Використовується Angular директива *ngIf для визначення, чи елемент має бути відображений. Умова вказує на те, що елемент буде відображений, якщо поле password є недійсним і йому вже доторкнулися (touched). Якщо ця умова виконується, виводиться текст помилки "Error password".

Ще один рядок Bootstrap для центрування тексту.

— <p *ngIf="errorMessage" class="text-danger"> Invailed Email or Password : Елемент з умовою виведення тексту про помилку. Виводиться, якщо змінна errorMessage має значення true. Ця змінна, ймовірно, встановлюється в коді компонента під час обробки помилок під час входу;

— : Рядок Bootstrap для розміщення кнопок у формі;

— : Контейнер для групування елементів форми;

— : Порожня колонка ширини 2 Bootstrap grid. Використовується для вирівнювання кнопок по центру;

— : Колонка ширини 6 Bootstrap grid;

— Sign In: Кнопка для входу. Викликає метод onSubmit() при натисканні. Класи Bootstrap (btn, btn-primary, btn-block) використовуються для стилізації і розміщення.

Ще одна аналогічна колонка для іншої кнопки.

— Continue with Google: Кнопка для входу через Google. Викликає метод GoogleAuth() при натисканні. Класи Bootstrap використовуються для стилізації і розміщення.

Загалом, цей код додає рядок тексту для виведення помилки паролю, рядок тексту для виведення загальної помилки, та дві кнопки для входу, одна з яких призначена для звичайного входу, а інша для входу через Google.

Цей код представляє HTML-шаблон для форми реєстрації, і його можна розглядати поетапно:

— : Рядок для введення адреси електронної пошти;

— : Колонка Bootstrap шириною 2, в якій розміщено мітку "Email";

— <div class="col-md-6 form-group" [ngClass]="(formRegistration.controls['email']. invalid && formRegistration.controls['email'].touched) ? 'has-error' : "">: Колонка шириною 6 для введення адреси електронної пошти. Використовується директива [ngClass] для динамічного додавання класу 'has-error', якщо поле електронної пошти недійсне і йому вже доторкнулися (touched);

— <input type="text" class="form-control" placeholder="Email" formControlName="email">: Поле введення для адреси електронної пошти. formControlName вказує, що це поле пов'язане з полем 'email' у формі.

Знову є рядок Bootstrap для центрування тексту:

— : Рядок для центрування тексту;

— <p *ngIf="formRegistration.controls['email'].invalid && formRegistration. controls['email'].touched" class="text-danger"> Error email </p>: Виведення тексту про помилку, якщо поле електронної пошти недійсне і йому вже доторкнулися (touched). Використовується директива *ngIf Angular для умовного відображення.

Це загальний огляд коду для форми реєстрації, який містить поле введення електронної пошти і текст про можливу помилку.

Цей код представляє розділ форми для введення паролю у вигляді HTML-шаблону. Розглянемо його поетапно:

— <label class="control-label">Password*</label>: Мітка для поля введення паролю, де * вказує на обов'язкове поле і відображається червоною зіркою;

— <div class="col-md-6 input-group";

— <input [type]="hide ? 'text' : 'password'" class="form-control" placeholder="Password" formControlName="password">. Поле введення для паролю. Використовує атрибут [type] для зміни типу поля введення між текстом і паролем залежно від значення змінної hide. formControlName вказує, що це поле пов'язане з полем 'password' у формі;

— : Група елементів у вигляді кнопки;

— <button class="btn btn-default" type="button" (click)="showPassword()">. Кнопка для переключення видимості паролю. Коли клікається, викликається метод.

— <div class="col-md-6 input-group" [ngClass]="(formRegistration.controls['confirm']. touched && equal) ? 'has-error' : "">: Колонка шириною 6, яка містить поле введення підтвердження пароля та кнопку переключення видимості підтвердження паролю. До неї також застосовано директиву [ngClass], яка додає клас 'has-error', якщо поле підтвердження пароля є недійсним і йому вже доторкнулися, а також якщо пароль і підтвердження пароля не співпадають (що перевіряється за допомогою equal);

— : Група кнопок для кнопки переключення видимості підтвердження паролю;

— <i *ngIf="!hide1" class="fa fa-eye-slash" aria-hidden="true"></i>: Іконка "Eye-slash" для прихованого підтвердження паролю, яка відображається, коли пароль видно;

— <i *ngIf="hide1" class="fa fa-eye" aria-hidden="true"></i>: Іконка "Eye" для видимого підтвердження паролю, яка відображається, коли пароль приховано;

— <p *ngIf="equal" class="text-danger"> Error confirm password </p>: Повідомлення про помилку, яке виводиться, якщо пароль і підтвердження пароля не співпадають. Використовується директива *ngIf для умовного відображення.

Цей код представляє блок для введення імені та прізвища у формі реєстрації. Давайте розглянемо його по кроках:

Створено рядок Bootstrap для групування елементів у ряд. Колонка шириною 2 містить мітку "First name" з червоним астериском для відзначення обов'язкового поля. Колонка шириною 6 містить поле введення для імені. Також застосовано клас помилки 'has-error', якщо поле не є валідним і до нього вже доторкнулися. Додано рядок у центрі для відображення повідомлення про помилку, якщо ім'я невірне та вже було доторкано. Схожий код, але для другого імені (Second Name). Обидва блоки мають розпізнавання помилок та відповідні повідомлення, якщо введені дані є неправильними або не були введені.

Цей код також представляє блок для введення номера телефону та дати у формі реєстрації. Створено рядок Bootstrap для групування елементів у ряд. Колонка шириною 2 містить мітку "Phone" з червоним астериском для відзначення обов'язкового поля. Колонка шириною 6 містить поле введення для номера телефону. Також застосовано клас помилки 'has-error', якщо поле не є валідним і до нього вже доторкнулися. Додано рядок у центрі для відображення повідомлення про помилку, якщо номер телефону невірний та вже був введений. Схожий код, але для дати народження (Birthday). Обидва блоки мають розпізнавання помилок та відповідні повідомлення, якщо введені дані є неправильними або не були введені.

Колонка шириною 2 містить мітку "Make your choice" з червоним астериском для відзначення обов'язкового поля. Колонка шириною 6 містить два елементи radio, які представляють собою варіанти вибору ролі користувача. Використано директиву [(ngModel)], яка забезпечує двостороннє зв'язування між властивістю role в компоненті та значенням вибору. Кожен елемент radio має своє значення ([value]="0" для "I am user" та [value]="1" для "I am provider"). Цей код дозволяє

користувачеві обрати свою роль з двох можливих варіантів: "I am user" або "I am provider". Обране значення буде збережено у властивості role у компоненті.

Розглянемо кожну частину.

Вибір місця (поштового індексу) для постачальника:

Вибір місця доступний тільки для конкретно постачальників (роль 1). Використовується директива *ngIf для умовного відображення цього елемента форми. Використовується випадаючий список (<select>) для вибору місця. Вибране значення буде збережено властивості region у вашому компоненті. Кнопка "Sign Up":

Кнопка для відправки форми реєстрації. При натисканні на кнопку викликається метод onSubmit() з вашого компоненту. Кнопка "Continue with Google":

Кнопка для авторизації через Google. При натисканні на кнопку викликається метод GoogleAuth() з вашого компоненту. Роздільна лінія "Or":

Графічний елемент, який розділяє дві кнопки та створює візуальний ефект "або".

Посилання для входу: Посилання, яке перенаправляє користувача на сторінку входу при натисканні. Використовується директива routerLink. Цей код визначає фінальну частину форми реєстрації з можливістю вибору місця для постачальника, кнопками відправки та авторизації через Google, а також додатковими елементами для оформлення форми.

По аналогії робимо коди для edit-profile.component.html, add-service.component.html, service-card.component.html

3.5 Розробка header компоненту

У лістингу показую header компонент сайту, лістинг 17

```
isAuth(): boolean {
    let user: any = localStorage.getItem('user') || '';
    if(!user) {
```

```
      return false;
    }
    if (!this.isUpdateData) {
      user = JSON.parse(user);
      this.collection.getOneItemToCollection('Users',
user['uid']).pipe(
        take(1)
      ).subscribe(
        (data: any) => {
          this.avatar = data.avatar_url;
          this.name = data.first_name;
          this.isUpdateData = true;
        }
      );
      return true;
    }
    return true;
  }
}
```

Лістинг 17 — Header компонент сайту

Цей фрагмент коду виглядає як метод isAuth() в Angular-компоненті. Давайте розглянемо кожен рядок коду:

— isAuth(): boolean {: Цей рядок починає оголошення методу. Вказує, що метод повертатиме значення типу boolean.

— let user: any = localStorage.getItem('user') || '';: Змінна user отримується з локального сховища (LocalStorage) за ключем 'user'. Якщо значення відсутнє, user ініціалізується порожнім рядком.

— if (!user) { return false; }: Якщо значення user не існує (порожній рядок), метод повертає false. Це означає, що користувач не автентифікований.

— if (!this.isUpdateData) {: Цей блок виконується, якщо isUpdateData не встановлено в true. Це може слугувати флагом, щоб визначити, чи вже оновлені дані користувача.

— user = JSON.parse(user);: Якщо isUpdateData ще не встановлено, значення user, яке є рядком, розшифровується в об'єкт за допомогою JSON.parse.

— this.collection.getOneItemToCollection('Users', user['uid']).pipe(: Виклик сервісного методу getOneItemToCollection для отримання одного об'єкта з колекції 'Users' за ідентифікатором user['uid'].

— .subscribe((data: any) => { this.avatar = data.avatar_url; this.name = data.first_name; this.isUpdateData = true; }: Оператор .subscribe реагує на результат запиту. Якщо запит вдалий, дані, такі як avatar_url і first_name, витягаються і присвоюються змінним this.avatar і this.name, а isUpdateData встановлюється в true.

— return true;: Після оновлення даних, метод повертає true. Це вказує, що користувач успішно автентифікований та дані оновлені.

— }: Кінець умовного блоку, який відбувається, якщо isUpdateData не встановлено в true.

— return true;: Якщо isUpdateData вже встановлено в true, метод повертає true. Це означає, що користувач автентифікований, і дані оновлені.

— }: Кінець методу.

Отже, загальною метою цього методу є перевірка автентифікації користувача та оновлення деякої інформації про користувача, якщо це необхідно. Якщо користувач не автентифікований, метод повертає false. Якщо користувач вже автентифікований або його дані оновлені, метод повертає true.

Для html. Цей HTML-код представляє форму реєстрації на веб-сайті.

Давайте розглянемо кожен елемент окремо:

— <label class="control-label">Email*</label>: Мітка для введення email. Текст "Email" вказує на те, що це поле для введення email, і символ * вказує, що це обов'язкове поле;

&& formRegistration.controls['email'].touched) ? 'has-error' : "">: Колонка шириною 6 одиниць Bootstrap grid, яка включає формову групу та застосовує клас has-error (якщо введення недійсне та вже торкнуте) для стилізації;

— <input type="text" class="form-control" placeholder="Email" formControlName="email">: Поле введення тексту для email.

formControlName="email" забезпечує зв'язок з конкретним полем у формі Angular Reactive Forms;

— : Новий рядок для централізованого розташування тексту;

— <p *ngIf="formRegistration.controls['email'].invalid && formRegistration. controls['email'].touched" class="text-danger"> Error email </p>: Параграф, який виводить помилку "Error email", якщо поле для введення email недійсне і вже торкнуте (за допомогою Angular structural directive *ngIf);

— : Закриваючий тег для рядка text-center.

Цей код відображає основну структуру і стилізацію форми реєстрації з полям для введення email та повідомленням про помилку, яке відображається при недійсному введенні email.

3.6 Налаштування стейт менеджменту

Це лише один із способів реалізації стейт-менеджменту в Angular. Для більших та складніших додатків можуть використовуватися бібліотеки стейт-менеджменту, такі як NgRx, які надають більше можливостей для керування станом додатка. Код з модулю auth.action.ts Дивитесь у лістингу 18

```
    import { createAction, props } from '@ngrx/store';
    export const signInRequest = createAction('[Auth] Load
User', props<{email: string, password: string}>());
    export const signInSuccess = createAction('[Auth] User
Loaded Success', props<{ user: any }>());
    export const signInFailure = createAction('[Auth] User Load
Failure', props<{ error: any }>());
```

Лістинг 18 — Код з модулю auth.action.ts

У цьому фрагменті коду використовується NgRx, яка є бібліотекою для управління станом додатків в Angular.

— createAction створює об'єкт дії (action). Дії — це об'єкти, які вказують, що щось сталося в додатку. Вони є чистими об'єктами з типом та необов'язковими додатковими властивостями;

— '[Auth] Load User', '[Auth] User Loaded Success', '[Auth] User Load Failure' — це типи дій. Тип дії — це просто рядок, який описує, яку подію вони представляють. У цьому випадку, ці дії пов'язані із авторизацією;

— props<{email: string, password: string}>() — це властивість, яка визначає структуру даних, які пов'язані із цією дією. У випадку signInRequest, це об'єкт з електронною поштою та паролем.

За допомогою createAction створюються три різні дії: signInRequest, signInSuccess, signInFailure:

— signInRequest використовується для ініціювання процесу авторизації. Вона має дані електронної пошти та пароля;

— signInSuccess викликається, коли авторизація пройшла успішно. Вона передає об'єкт користувача, який був завантажений;

— signInFailure викликається, якщо є помилка під час авторизації. Вона передає об'єкт помилки.

Ці дії потім обробляються відповідним чином у редукторах (reducers), які змінюють стан додатку в залежності від типу дії та даних, які вона містить.

Інфо з модуля auth.effects.ts показана у лістингу 19

```
    import { Injectable } from '@angular/core';
    import { Actions, createEffect, ofType } from
'@ngrx/effects';
    import { catchError, map, mergeMap } from 'rxjs/operators';
    import { of } from 'rxjs';
    import * as AuthActions from '../actions/auth.actions';
    import { AuthService } from
'../../../shared/services/auth.service';
    @Injectable()
    export class AuthEffects {
      signIn$ = createEffect(() =>
        this.actions$.pipe(
          ofType(AuthActions.signInRequest),
          mergeMap((action) =>
```

```
            this.authService.singIn(action.email,
action.password).pipe(
              map((result) => AuthActions.signInSuccess({ user:
result })),
              catchError((error) =>
of(AuthActions.signInFailure({ error })))
            )
          )
        )
      );
```

Лістинг 19 — Код з модулю auth.effects.ts

Цей фрагмент коду представляє собою ефекти NgRx для обробки дій, пов'язаних із авторизацією, у тому числі signInRequest, signInSuccess, та signInFailure. Ефекти NgRx використовуються для виконання асинхронних операцій, таких як взаємодія з сервером, і використовуються разом із сервісами:

— @Injectable(). Цей декоратор вказує, що клас AuthEffects є Angular сервісом, і його можна вмікнути в інші класи (наприклад, компоненти);

— createEffect. Цей метод створює NgRx ефект. Ефекти в NgRx відповідають за асинхронні операції. В цьому випадку, signIn$ — це ефект для авторизації;

— this.actions$.pipe(...). Тут визначається обробка дій (actions). this.actions$ — це об'єкт, який представляє потік дій NgRx;

— ofType (AuthActions.signInRequest): Фільтрує потік дій, обираючи лише ті, які є діями авторизації (signInRequest);

— mergeMap((action) => ...). Об'єднує потік дій з іншим потоком (результатом виклику сервісу);

— this.authService.singIn (action.email, action.password): Виклик сервісу авторизації для входу з переданими параметрами (електронна пошта та пароль);

— .pipe(...). Визначення додаткових операцій, які будуть застосовані до результату сервісного виклику;

— map((result) => AuthActions.signInSuccess({ user: result })): Якщо виклик успішний, генерується нова дія signInSuccess з результатами входу (об'єкт користувача);

— catchError((error) => of(AuthActions.signInFailure({ error }))): Якщо виклик завершується помилкою, генерується дія signInFailure з включенням об'єкта помилки.

В результаті, ефекти NgRx signIn$ будуть реагувати на дії signInRequest, виконуючи відповідний виклик сервісу і видаючи нові дії в залежності від результатів цього виклику.

Код та опис модулю функції стейт менеджменту auth.reducers.ts показано у лістинг 20

```
import { createReducer, on } from '@ngrx/store';
import * as AuthActions from '../actions/auth.actions';
export interface AuthState {
  user: any;
  error: any;
}
export const initialState: AuthState = {
  user: null,
  error: null,
};
export const authReducer = createReducer(
  initialState,
  on(AuthActions.signInSuccess, (state) => ({
    ...state,
    error: null,
  })),
  on(AuthActions.signInFailure, (state, { error }) => ({
    ...state,
    error,
    user: null,
  }))
);
```

Лістинг 20 — Опис модулю функції стейт менеджменту

Цей фрагмент коду представляє собою редуктор NgRx для обробки дій авторизації та оновлення стану.

Імпорт бібліотек. createReducer та on — це функції NgRx, які дозволяють створювати редуктори та оброблювати дії.

Опис інтерфейсу стану. Визначається інтерфейс AuthState, який включає в себе user (користувач) та error (помилка). Стан має початкові значення, визначені в initialState.

Створення початкового стану. initialState представляє собою об'єкт, що містить user (початково null) і error (початково null).

Створення редуктору. authReducer створюється за допомогою createReducer. Він приймає початковий стан та набір обробників дій, які змінюють цей стан.

Обробник on(AuthActions.signInSuccess, ...): Вказує, що редуктор реагує на дію signInSuccess. Коли ця дія виникає, стан оновлюється, встановлюючи error в null (бо вона вказує на успішний вхід).

Обробник on(AuthActions.signInFailure, ...): Вказує, що редуктор реагує на дію signInFailure. Коли ця дія виникає, стан оновлюється, встановлюючи error в значення, отримане з дії, і user в null (бо вхід невдало завершився).

Таким чином, редуктор authReducer слідкує за діями, пов'язаними із входом, та змінює стан відповідно до результатів цих дій.

Код за модулем auth.module.ts лістинг 21 до нього і опис дії коду показан нище

```
import { NgModule } from '@angular/core';
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { authReducer } from './reducers/auth.reducers'
import { AuthEffects } from './effects/auth.effects'
    @NgModule({
imports: [
        StoreModule.forFeature('auth', authReducer),
        EffectsModule.forFeature([AuthEffects]),
      ],
    })
    export class AuthModule { }
```

Лістинг 21 — Код з модуля auth.module.ts

— StoreModule.forFeature('auth', authReducer): Вказує, що для функціональності, пов'язаної з авторизацією, використовується окремий редуктор, який названий 'auth'. authReducer — це редуктор, який ми визначили раніше;

— EffectsModule.forFeature([AuthEffects]): Вказує, що для даної функціональності використовуються окремі ефекти, описані в AuthEffects;

— Експорт модулю: Модуль може бути використаний в інших частинах Angular додатка. Зазвичай він експортується, щоб інші модулі могли використовувати його функціональність;

— Отже, цей Angular модуль встановлює конфігурацію для NgRx Store та Effects для обробки стану та ефектів, пов'язаних із функціональністю авторизації в вашому додатку.

Цей фрагмент коду представляє собою Angular модуль, який використовує NgRx для керування станом та ефектами в контексті авторизації.

Імпорт бібліотек. В модулі імпортуються необхідні Angular та NgRx бібліотеки.

— Ініціалізація NgRx: В @NgModule додаються StoreModule.forFeature та EffectsModule.forFeature.

3.7 Налаштування маршрутизації

Angular має вбудовану маршрутизацію, і система маршрутизації дуже проста у налаштуванні та використанні; Angular використовує декларативний підхід до визначення маршрутів, так що структура маршрутів може бути визначена за допомогою конфігурацій, а не програмування. Angular також дозволяє вкладати маршрути, що корисно для організації структури програми [14].

Декларативний підхід. Angular визначає маршрути декларативно. Це означає, що розташування і поведінка компонентів визначаються конфігурацією маршруту, а не програмуванням.

Конфігураційний підхід. Маршрути в Angular конфігуруються за допомогою об'єктів конфігурації. Ці об'єкти визначають, які компоненти можуть бути пов'язані з певною URL-адресою, які параметри можуть бути передані URL-адресі та інші важливі налаштування.

Простота налаштування та використання. Angular робить маршрутизацію простою у використанні. Щоб додати маршрут, просто визначте конфігурацію в RouterModule. Після цього Angular візьме на себе завдання відстеження URL-адреси та відображення відповідних компонентів.

Вкладеність маршрутів. В Angular існує можливість вкладати маршрути. Це означає, що можна створювати складні структури маршрутів, де деякі компоненти внутрішньо генеруються іншими компонентами. Це корисно для організації багаторівневих ієрархій сторінок і компонентів.

Робота з параметрами та запитами Angular також дозволяє працювати з параметрами маршруту, які можна використовувати для передачі додаткової інформації між компонентами.

Такий підхід до маршрутизації робить код більш прозорим і зручним для обслуговування, особливо при роботі зі складними веб-додатками з мільйонами сторінок і функцій [14].

3.8 Інтеграція з бекендом

Після створення та стилізації всіх необхідних блоків ми інтегрували та визначили функціонал. Для цього ми створили файл login.component.ts у додатку Angular та імпортували всі необхідні залежності:

- Angular's Component decorator для визначення компонента.
- Angular's Component decorator для визначення компонентів. Класи Angular FormBuilder, FormControl, FormGroup та Validators для створення та перевірки форм.
- Служба маршрутизації для навігації між різними сторінками додатку.

- AuthService: взаємодіє з сервером для автентифікації користувачів.

Пояснимо це на прикладі лістингу 22

```
User
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'Angular1';
}
```

Лістинг 22 — Компонент Angular

Цей код представляє основний компонент Angular. Давайте розглянемо його складові:

— @Component Декоратор: Це спеціальний тип аннотації, який вказує Angular, що клас AppComponent є компонентом;

— selector: Це властивість декоратора @Component і визначає, як компонент буде вибиратися в HTML-документі. У цьому випадку, якщо ви вставите <app-root></app-root> в HTML, Angular замінить цей тег на вміст компонента AppComponent;

— templateUrl: Це властивість декоратора, яка вказує шлях до зовнішнього HTML-файлу, який містить шаблон для цього компонента. У вашому випадку це файл з ім'ям app.component.html;

— styleUrls: Це властивість декоратора, вказуючи масив шляхів до зовнішніх файлів стилів, які будуть застосовуватися до цього компонента. У вашому випадку, це файли з ім'ям app.component.scss;

— class AppComponent: Це сам клас компонента, який містить логіку та дані для цього компонента;

— title: Це властивість класу AppComponent, яка містить рядок "Angular1". Це значення використовується в шаблоні для відображення заголовку компонента.

Отже, основна ідея полягає в тому, що Angular використовує декоратор @Component, щоб зробити звичайний клас TypeScript AppComponent розпізнаваним як компонент Angular. Потім визначається, як компонент повинен взаємодіяти з іншими частинами програми за допомогою властивостей, методів та шаблонів. Html версія буде виглядати так, як у лістингі 23 нище

```
<app-header></app-header>
<div class="container-full">
  <router-outlet></router-outlet>
</div>
```

Лістинг 23 — Html компонент

Цей код представляє структуру основного макету вашого Angular додатка.

— <app-header></app-header>. Це виглядає як власний елемент або компонент <app-header>. Вибірка "header" може містити компонент, який відповідає за відображення верхньої частини вашого інтерфейсу, таку як назва додатка, навігаційні посилання тощо.

— . Це блочний контейнер , який має клас container-full. Він може використовуватися для розділення різних секцій вашого інтерфейсу чи для надання стилів.

— <router-outlet></router-outlet>. Це спеціальний елемент <router-outlet>, який використовується для маршрутизації в Angular. Всі компоненти, які ви визначаєте для різних шляхів у вашому додатку, будуть відображатися в цьому місці. Його вміст змінюється в залежності від того, який компонент пов'язаний із поточним маршрутом.

Отже, загалом, цей код створює структуру додатка, де є місце для заголовка (<app-header>), а також контейнер для відображення різних компонентів залежно від маршруту, який вибрав користувач.

Висновки до розділу 3

На основі досвіду розробки сайтів для онлайн реєстрації послуг можна зробити наступні висновки:

1. Використання надійних протоколів безпеки, таких як HTTPS, є обов'язковим.
2. Для забезпечення масштабованості слід використовувати архітектуру, яка дозволяє легко додавати нові сервери або ресурси.
3. При проектуванні інтеграцій з іншими системами слід враховувати вимоги цих систем.

Крім того, слід враховувати наступні фактори:

— Кількість користувачів. Від кількості користувачів залежить необхідна пропускна здатність системи.

— Тип послуг, які надаються. Від типу послуг залежить складність системи та необхідність використання додаткових функцій, таких як, наприклад, система бронювання.

— Бюджет. Розробка сайтів для онлайн реєстрації послуг може бути досить витратною, тому важливо правильно оцінити бюджет проекту.

РОЗДІЛ 4 АНАЛІЗ ВЕБ-САЙТУ

4.1 Спільні риси

Якщо досліджувати Angular в контексті з іншими програмами для створення сучасного контенту, то можливо порівняти його з професійними програмами розробки, такими, як Wordpress.

Angular і WordPress мають деякі спільні риси, які роблять їх популярними інструментами для розробки веб-додатків.

Обидва інструменти дозволяють розробникам створювати веб-додатки, які виглядають і поводяться сучасно; Angular і WordPress пропонують широкий спектр функцій і можливостей, які дозволяють розробникам створювати привабливі та інтуїтивно зрозумілі інтерфейси.

Обидва інструменти мають великі спільноти користувачів та розробників, які надають підтримку та ресурси Angular та WordPress мають великі та активні спільноти, які надають підтримку, ресурси та навчання для розробників.

Обидва інструменти можна використовувати для створення веб-додатків будь-якого розміру і складності; Angular і WordPress мають гнучкість, що дозволяє використовувати їх для створення веб-додатків будь-якого розміру і складності.

Як Angular, так і WordPress є веб-орієнтованими технологіями, і обидва вони використовуються для розробки веб-додатків та веб-сайтів. Однак вони мають різні підходи та ролі в процесі створення веб-проектів.

4.1.1 Веб-орієнтовані технології

Як Angular, так і WordPress використовуються для розробки веб-додатків та веб-сайтів.

Angular:

Тип: Angular — це фронтенд-фреймворк, який дозволяє розробникам створювати потужні односторінкові додатки (SPA) та динамічні веб-сайти.

Мова: Використовує TypeScript або JavaScript для програмування клієнтської частини додатка.

Функціонал: Надає багатий інструментарій для створення компонентів, роботи з HTTP-запитами, маршрутизації та взаємодії з API.

WordPress:

Тип: WordPress — це веб-система управління контентом (CMS), яка полегшує створення та управління веб-сайтами.

Мова: Використовує мову програмування PHP для створення серверної частини та MySQL для зберігання даних.

Функціонал: Забезпечує готовий функціонал для створення та управління веб-сайтами, включаючи систему плагінів, тем та можливість розширення функціоналу.

Іншими словами, Angular забезпечує інструменти для створення фронтенду, тоді як WordPress — це рішення для створення та управління контентом, а також для організації серверної логіки. Багато веб-сайтів використовують обидві технології, комбінуючи Angular для фронтенду та WordPress для управління контентом на сервері.

4.1.2 Компонентний підхід

Angular базується на компонентний архітектурі, де додаток розбивається на компоненти, які об'єднують логіку та представлення. У WordPress також є концепція плагінів та тем, які можна розглядати як компоненти.

Angular та Компоненти:

Компоненти: Angular використовує компонентну архітектуру, де додаток будується з невеликих та самостійних компонентів. Кожен компонент включає в себе логіку та представлення, що дозволяє легко розбити складний додаток на більш прості та управляємі частини.

WordPress та Плагіни/Теми як Компоненти:

Плагіни та теми: У WordPress плагіни та теми можна розглядати як еквівалент компонентів. Плагіни додають нові функціональні можливості, а теми визначають вигляд та стиль веб-сайту. Це дозволяє створювати великі та розширювані веб-сайти шляхом комбінування різних плагінів та тем.

Обидві технології прагнуть до розділення функціональності на компоненти, що полегшує розробку, тестування та розширення коду.

Отже, хоча вони можуть використовувати різні терміни (компоненти для Angular, плагіни/теми для WordPress), концепція розділення додатку на невеликі, повторно використовувані блоки є загальною для обох технологій.

4.1.3 Розширення через модулі/плагіни

Обидва фреймворки можуть бути розширені за допомогою модулів (в Angular) або плагінів (в WordPress), що дозволяє розширювати функціональність додатків.

Давайте порівняємо це більш детально.

Angular та Розширення через Модулі:

Модулі: У Angular модуль — це механізм для групування компонентів, директив, сервісів та інших ресурсів, які спільно використовуються у додатку. Використовуючи модулі, ви можете організувати та розділити функціональність вашого додатку на логічні частини, а також легко додавати або вилучати функціональність.

WordPress та Розширення через Плагіни:

Плагіни: У WordPress плагін — це невеликий шматок коду, який можна додати до вашого сайту для додавання нових функцій або модифікації існуючих. Використовуючи плагіни, ви можете легко розширювати функціональність свого веб-сайту, додаючи нові можливості без втручання в основний код.

І Angular, і WordPress розглядають розширення через модулі/плагіни як важливий аспект архітектури, оскільки це дозволяє розробникам додавати нові функції або змінювати поведінку без необхідності редагування основного коду

додатку або сайту. Це робить код більш модульним, легко підтримуваним і розширюваним.

4.2 Відмінності

Існують також деякі ключові відмінності між Angular і WordPress, які слід враховувати при виборі інструменту для розробки веб-додатків:

Angular — це JavaScript-фреймворк, а WordPress — система управління контентом (CMS) Angular — це JavaScript-фреймворк, а WordPress — система управління контентом (CMS). Angular використовується для створення клієнтських веб-додатків, а WordPress — для створення серверних веб-сайтів.

Angular є більш складним інструментом, ніж WordPress; Angular вимагає від розробників знання JavaScript та інших пов'язаних з ним технологій; WordPress є більш зручним інструментом, який не вимагає від розробників таких же знань.

Angular більш масштабований, ніж WordPress; Angular можна використовувати для створення складних веб-додатків, які вимагають великих обсягів коду; WordPress менш масштабований і більше підходить для створення простих веб-сайтів.

Angular, фреймворк веб-додатків з відкритим вихідним кодом, розроблений Google, використовує TypeScript, розширення JavaScript, яке забезпечує додаткову функціональність і продуктивність. добре задокументований і підтримується великою спільнотою розробників.

Angular підходить для створення складних веб-додатків, які потребують сучасного інтерфейсу та високої продуктивності; Angular також підходить для створення веб-додатків, які потребують масштабованості.

WordPress — це популярна система управління контентом (CMS), яка використовується для створення веб-сайтів. WordPress — це безкоштовне програмне забезпечення з відкритим вихідним кодом, яке підтримується великою спільнотою розробників.

WordPress підходить для створення простих веб-сайтів, які не потребують складних інтерфейсів або високої продуктивності; WordPress також підходить для створення веб-сайтів, які потрібно швидко розгортати та налаштовувати.

Angular — це фронтенд-фреймворк, тобто він використовується для розробки клієнтської частини веб-додатків. WordPress — це система управління контентом (CMS), яка використовується для створення та управління веб-сайтами. Angular та WordPress використовуються для різних типів додатків:

Angular:

Тип додатка: Angular є фронтенд-фреймворком, орієнтованим на розробку клієнтської частини веб-додатків. Це означає, що Angular використовується для розробки користувацького інтерфейсу (UI) та логіки взаємодії для веб-додатків.

WordPress:

Тип додатка: WordPress є системою управління контентом (CMS), і його основним призначенням є створення та управління вмістом веб-сайтів. WordPress дозволяє користувачам легко публікувати, редагувати та організовувати контент, включаючи тексти, зображення, відео тощо.

Angular та WordPress використовуються для різних завдань у веб-розробці: Angular для розробки клієнтської сторони додатків, а WordPress для управління контентом на веб-сайтах. Іноді їх комбінують разом, наприклад, використовуючи Angular як фронтенд для веб-сайту, побудованого на WordPress як CMS.

4.2.1 Мови програмування

Angular використовує TypeScript (або JavaScript, но не в моему випадку) для програмування клієнтської частини, тоді як WordPress використовує PHP для програмування серверної частини.

Angular:

Мова програмування: Angular використовує TypeScript або JavaScript для програмування клієнтської частини веб-додатків. TypeScript є розширенням JavaScript, яке додає статичні типи та інші функції для поліпшення розробки.

WordPress:

Мова програмування: WordPress використовує мову програмування PHP для розробки та виконання серверної частини. PHP є мовою, спеціально призначеною для веб-розробки, і вона використовується для взаємодії з базою даних та генерації вмісту відповідно до запитів від користувачів.

При розробці веб-додатків на Angular використовуються мови програмування, спрямовані на клієнтську сторону (TypeScript, ну або JavaScript), тоді як WordPress використовує мову програмування PHP для обробки запитів на серверній стороні.

4.2.2 Шаблони

Angular використовує шаблони, які забезпечують динамічне відображення даних на клієнтській стороні. У WordPress використовуються шаблони PHP для генерації HTML-коду на сервері.

Angular:

Шаблони: Angular використовує шаблони, які описують структуру HTML разом із директивами та вбудованими об'єктами. Ці шаблони використовуються для створення динамічного вмісту та забезпечення відображення даних на стороні клієнта.

WordPress:

Шаблони PHP: У WordPress шаблони використовують мову програмування PHP для генерації HTML-коду на сервері. Кожен тип сторінки (наприклад, пост, сторінка, архів) може мати свій власний шаблон, який визначає структуру та вигляд сторінки.

В Angular шаблони використовуються для відображення даних на клієнтській стороні, тоді як у WordPress шаблони PHP використовуються для генерації HTML на сервері.

Спосіб управління контентом: WordPress має вбудовану систему управління контентом, що дозволяє користувачам з легкістю додавати та оновлювати контент

на веб-сайті. Angular не надає такого вбудованого механізму управління контентом і зазвичай використовується для створення односторінкових додатків (SPA). відмінності у способі управління контентом між WordPress і Angular.

WordPress:

Система управління контентом (CMS): WordPress славиться своєю інтегрованою системою управління контентом, яка дозволяє користувачам легко додавати, редагувати та видаляти контент, такий як текстові статті, зображення, відео тощо, за допомогою інтуїтивного веб-інтерфейсу.

Angular:

Фронтенд-фреймворк: Angular, навпаки, є фронтенд-фреймворком, який часто використовується для створення односторінкових додатків (SPA). Його фокус на сторінці клієнта робить його не таким придатним для управління контентом на сервері. Для співпраці з серверним контентом Angular зазвичай використовують узгоджені API або взаємодіють із серверною частиною.

Використання WordPress виправдовується, коли потрібна потужна система управління контентом, тоді як Angular більше підходить для створення високопродуктивних SPA та веб-додатків.

Приклад коду на Angular (TypeScript/HTML/CSS) у лістингу 24

```
// app.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <h1>{{ title }}</h1>
    <p *ngFor="let item of items">{{ item }}</p>
  `,
  styles: [`
    h1 { color: blue; }
    p { font-size: 16px; }
  `]
})
export class AppComponent {
  title = 'My Angular App';
  items = ['Item 1', 'Item 2', 'Item 3'];
}
```

Лістинг 24 — Приклад коду Angular

Приклад коду на WordPress (PHP/HTML) у лістингу 25

```
<!-- header.php -->
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
<head>
  <meta charset="<?php bloginfo('charset'); ?>">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title><?php bloginfo('name'); ?></title>
  <?php wp_head(); ?>
</head>
<body <?php body_class(); ?>>
<!-- index.php -->
<?php get_header(); ?>
<div id="content">
  <?php if (have_posts()) : while (have_posts()) : the_post();
?>
    <h1><?php the_title(); ?></h1>
    <div><?php the_content(); ?></div>
  <?php endwhile; endif; ?>
</div>
<?php get_footer(); ?>
<!-- footer.php -->
<?php wp_footer(); ?>
</body>
</html>
```

Лістинг 25 — Приклад коду WordPress

4.2.3 Стилізація

У Angular можна використовувати препроцесори стилів, такі як SCSS. У WordPress для стилізації використовується CSS, а можливо також і SCSS за допомогою додаткових конфігурацій.

Angular:

В Angular використовується мова програмування SCSS для стилізації компонентів. SCSS — це препроцесор, що дозволяє використовувати розширений синтаксис CSS, такий як змінні, вкладені селектори, міксини та інші.

WordPress:

У WordPress для стилізації використовується мова CSS. Однак можна розширити можливості за допомогою препроцесора, наприклад, SCSS або LESS, якщо це необхідно. Це може вимагати додаткових інструментів або налаштувань залежно від конфігурації проекту.

Вибір між цими підходами також може залежати від зручності та вимог проекту. Angular надає більшу гнучкість у використанні препроцесорів стилів.

4.2.4 Маршрутизація

Angular має вбудовану систему маршрутизації, яка конфігурується декларативно. У WordPress, маршрутизація зазвичай контролюється WordPress Rewrite API.

Angular:

В Angular маршрутизація є вбудованою та декларативною. Ви можете визначити маршрути через конфігурацію і вказати, які компоненти повинні бути відображені для різних шляхів. Це особливо корисно для односторінкових додатків (SPA), де зміна контенту відбувається без повного перезавантаження сторінки.

WordPress:

У WordPress маршрутизація керується Rewrite API. Ви можете використовувати WordPress Rewrite API для переписування URL-адрес і визначення, які сторінки чи записи повинні відповідати певним URL-шляхам. Це особливо важливо для систем управління контентом (CMS), де кожна сторінка чи запис може мати свій унікальний URL.

Обидва підходи мають свої переваги залежно від типу додатка чи веб-сайту, який ви розробляєте. Angular забезпечує декларативну та добре налаштовувану систему маршрутизації, тоді як WordPress дозволяє гнучко налаштовувати маршрути через Rewrite API, що особливо важливо для CMS.

4.2.5 Взаємодія з сервером

У Angular взаємодія з сервером може здійснюватися через HTTP-запити. У WordPress використовується система фільтрів та хуків для взаємодії з базою даних та обробки запитів.

Angular:

У Angular для взаємодії з сервером використовуються HTTP-запити, такі як GET, POST, PUT, DELETE. Angular надає модуль HttpClient, який дозволяє робити HTTP-запити до сервера і обробляти їхні відповіді та помилки. Це може включати отримання або надсилання даних, таких як JSON, між клієнтом і сервером.

WordPress:

У WordPress взаємодія з сервером відбувається за допомогою системи фільтрів та хуків, які дозволяють розширювати функціональність WordPress. PHP-фільтри і хуки використовуються для втручання в роботу ядра WordPress, обробки запитів, зміни даних у базі даних тощо. Це робиться на серверному рівні і часто включає в себе обробку даних в базі даних MySQL, яка використовується WordPress.

Обидва підходи відображають те, що Angular спеціалізується на клієнтській стороні і використовує стандартні протоколи HTTP для взаємодії з сервером, тоді як WordPress, як система управління контентом, використовує серверні техніки для обробки запитів та управління даними на сервері.

4.3 Вибір між Angular і WordPress

Якщо ви хочете створювати складні веб-додатки з сучасним інтерфейсом, Angular — хороший вибір. Якщо ви хочете створити простий веб-сайт з невеликою кількістю функцій, WordPress — хороший вибір. Ось деякі додаткові фактори, які слід враховувати при виборі між Angular і WordPress.

Бюджет. Angular безкоштовний, але можуть бути додаткові витрати на сторонні плагіни та інструменти; WordPress також безкоштовний, але деякі

розширення і теми можуть коштувати грошей; і WordPress безкоштовний, але можуть бути додаткові витрати на деякі розширення і теми.

Ваш досвід. Якщо ви новачок у веб-розробці, WordPress може бути простішим у використанні; Angular вимагає більше досвіду і знань від розробників.

Ваші потреби. Якщо ви хочете створити веб-сайт зі специфічними функціями або вимогами, вам потрібно з'ясувати, що є більш сумісним — Angular чи WordPress.

Висновки до розділу 4

Angular і WordPress — це два потужні інструменти, які можна використовувати для створення веб-додатків. Однак вони мають різні переваги та недоліки, які слід враховувати при виборі правильного інструменту для ваших потреб.

Angular. Переваги:

1. Компонентна архітектура.
2. Широкий функціонал.
3. Односторінкові додатки (SPA).

Недоліки:

1. Великий обсяг коду. Angular може вимагати написання більшого обсягу коду порівняно з іншими фреймворками.

WordPress. Переваги:

1. Простота використання.
2. Широкий вибір плагінів і тем.

WordPress. Недоліки:

1. Безпека. Як популярна платформа, WordPress може бути піддаються атакам, і для забезпечення безпеки потрібно використовувати додаткові заходи.

ВИСНОВКИ

1. Проведено аналіз існуючих підходів і технологій для розробки сайту послуг.

Для цього створено сайт на Angular, з використанням TypeScript як мови програмування. Для порівняння проаналізовано такі критерії:

- продуктивність;
- легкість входження та розуміння для новачків.

2. Проведено аналіз структур та програмної реалізації. В результаті в якості інструментів розробки була обрано мова TypeScript з використанням фреймворку Angular з хмарним бекендом. Це дозволяє дозволяє створювати ефективний та добре структурований код для клієнтської частини веб-сайту. Використання Firebase як хмарної платформи для бекенду покращує масштабованість та доступність застосунку.

3. Розроблено сайт для онлайн надання послуг. Розроблено функціонал з використанням потужних інструментів Angular та Firebase для створення повноцінного веб-додатка.

Розробка такого веб-сайту також включає в себе розробку допоміжного функціоналу:

— реєстрація користувачів;

— аутентифікація користувача;

— вибір послуги та дата реєстрації;

— оплата послуг онлайн;

— перегляд та корекція записів користувача.

4. Календарь. Реалізовано функціонал для роботи з датами, зображеннями користувачів.

Застосування відкритих технологій та сервісів свідчить про відкритість для інновацій та готовність до розширення функціоналу в майбутньому. Тому й було використано Angular з хмарним бекендом.

5. Проведено порівняльний аналіз технологій Angular, Firebase та Typescript, та традиційним підходом на основі WordPress.

Angular та Firebase можуть забезпечити велику гнучкість і розширюваність для веб-додатка. Він пропонує компонентну архітектуру, яка спрощує розширення та обслуговування. Angular, завдяки своїй декларативний природі та великій кількості готових компонентів.

Firebase надає хмарне зберігання даних та автентифікацію, що дозволяє швидко розгортати та масштабувати додаток. надає інструменти, такі як реальний час бази даних та зручна автентифікація. А також має вбудовану систему безпеки, ізольовану від клієнтського коду.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Бібліотека додаткових функцій для TypeScript, 2023. 11р. https://lodash.com/. URL: HYPERLINK "https://lodash.com/" \t "_blank" https://lodash.com/ (дата звернення: 11.05.2023).

2. Нікітенко О. О. Розробка сайту для онлайн-реєстрації послуг. Молода наука-2023 : ЗБ. наук. пр. студентів, аспірантів, докторантів і молодих вчен., м. Запоріжжя, 26 трав. 2023 р. Запоріжжя, 2023. С. 109. URL: HYPERLINK "https://www.znu.edu.ua/ii_znu/nauka/conf5/tom_5_2023_26_05_2023.pdf" \t "_blank" https://www.znu.edu.ua/ii_znu/nauka/conf5/tom_5_2023_26_05_2023.pdf (дата звернення: 20.06.2023).

3. Нікітенко О. О., Заяц В. І. Особливості розробки сайтів для онлайн-реєстрації послуг : III Всеукраїнської науково-практичної конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України», 17-20 жовтня 2023 року, Запоріжжя, 2023. С. 118. https://cutt.ly/twMGPRZd (дата звернення: 13.01.2024).

4. Бібліотека компонентів для Angular, 2023. 30р. https://material.angular.io/. URL: HYPERLINK "https://material.angular.io/" \t "_blank" https://material.angular.io/ (дата звернення: 11.05.2023).

5. Бібліотека розширень для Angular, 2023. 8р. https://ng-bootstrap.github.io/. URL: HYPERLINK "https://ng-bootstrap.github.io/" \t "_blank" https://ng-bootstrap.github.io/ (дата звернення: 11.05.2023).

6. Використання хмарних технологій, 2023. 217р. http://ekhsuir.kspu.edu. URL: HYPERLINK "http://ekhsuir.kspu.edu/handle/123456789/796" \t "_blank" http://ekhsuir.kspu.edu/handle/123456789/796 (дата звернення: 11.05.2023).

7. Загальні практичні поради щодо налаштування проектів Firebase, 2023. 12р. https://firebase.google.com. URL: HYPERLINK "https://cutt.ly/fwMGPDZa" \t "_blank" https://cutt.ly/awMGPC5Khttps://cutt.ly/awMGPC5K (дата звернення: 11.05.2023).

8. Онлайн-ресурси для вивчення TypeScript, 2023. 44р. https://www.typescriptlang.org. URL: HYPERLINK "https://cutt.ly/hwMGAh8O" \t "_blank" https://cutt.ly/qwMGAmjx (дата звернення: 11.05.2023).

9. Офіційна документація TypeScript, 2023. 361р. https://www.typescriptlang.org. URL: HYPERLINK "https://cutt.ly/LwMGAIJF" \t "_blank" https://www.typescriptlang.org/docs (дата звернення: 11.05.2023).

10. Ресурси для дизайну інтерфейсу, 2023. 68р. https://www.behance.net/. URL: HYPERLINK "https://www.behance.net/" \t "_blank" https://www.behance.net/ (дата звернення: 11.05.2023).

11. Ресурси для розробки веб-інтерфейсу, 2023. 40р. https://cutt.ly/GwMGAH3M/. URL: HYPERLINK " https://cutt.ly/GwMGAH3M " \t "_blank" https://cutt.ly/GwMGAH3M (дата звернення: 11.05.2023).

12. Ресурси для розробки графічних елементів, 2023. 334р. https://www.canva.com/. URL: HYPERLINK "https://www.canva.com/" \t "_blank" https://www.canva.com/ (дата звернення: 11.05.2023).

13. Сайт для пошуку бібліотек і пакетів npm, 2023. 11р. https://www.npmjs.com/. URL: HYPERLINK "https://www.npmjs.com/" \t "_blank" https://www.npmjs.com/ (дата звернення: 11.05.2023).

14. Статті з веб-розробки на сайті Medium, 2023. 202р. https://medium.com. URL: HYPERLINK "https://medium.com/topic/web-development" \t "_blank" https://medium.com/topic/web-development (дата звернення: 11.05.2023).

15. Concrete Types for TypeScript, 2023. 97p. https://drops.dagstuhl.de. URL: HYPERLINK "https://cutt.ly/5wMGAMJP" \t "_blank" https://cutt.ly/5wMGAMJP (date of access: 11.05.2023).

16. Firebase essential, 2023. 54p. https://books.google.com.ua/. URL: HYPERLINK "https://cutt.ly/QwMGStWO" \l "v=onepage&q=Firebase% 20Cloud% 20Function&f=false" \t "_blank" https://cutt.ly/awMGSsE1 (date of access: 11.05.2023).

17. Https://angular.io, 2023. 33p. Introduction to the Angular docs. URL: HYPERLINK "https://angular.io/docs" \t "_blank" https://angular.io/docs (date of access: 11.05.2023).

18. JavaScript frameworks: angular, react, vue, 2023. 216p. https://www.theseus.fi/. URL: HYPERLINK "https://cutt.ly/AwMGSVeb" \t "_blank" https://cutt.ly/zwMGS7mR (date of access: 11.05.2023).

19. Learning javascript design patterns: a javascript and jquery developer's guide, 2023. 412p. https://books.google.com.ua. URL: HYPERLINK "https://cutt.ly/uwMGDq1M" \l "v=onepage&q=js%20learning&f=false" \t "_blank" https://cutt.ly/FwMGDuYZ (date of access: 11.05.2023).

20. TypeScript: modern javascript development, 2023. 303p. https://books.google.com.ua. URL: HYPERLINK "https://cutt.ly/rwMGDdET" \l "v=onepage&q=typescript%20javascript&f=false" \t "_blank" https://cutt.ly/fwMGDWtn (date of access: 11.05.2023).

21. Ресурси для вивчення розробки на express.js, 2023. 139p. https://expressjs.com/. URL: HYPERLINK "https://expressjs.com/" \t "_blank" https://expressjs.com/ (дата звернення: 11.05.2023).

22. Ресурси для вивчення розробки на MongoDB, 2022. 88p. https://docs.mongodb.com/. URL: HYPERLINK "https://docs.mongodb.com/" \t "_blank" https://docs.mongodb.com/ (дата звернення: 11.05.2023).

23. Ресурси для вивчення розробки на MySQL, 2022. 243p. https://dev.mysql.com. URL: HYPERLINK "https://dev.mysql.com/doc/" \t "_blank" https://dev.mysql.com/doc/ (дата звернення: 11.05.2023).

24. Ресурси для вивчення розробки на node.js, 2023. 113p. https://nodejs.org/. URL: HYPERLINK "https://nodejs.org/en/docs/" \t "_blank" https://nodejs.org/en/docs/ (дата звернення: 11.05.2023).

25. Ресурси для вивчення розробки на PostgreSQL, 2021. 412p. https://www.postgresql.org. URL: HYPERLINK "https://www.postgresql.org/docs/" \t "_blank" https://www.postgresql.org/docs/ (дата звернення: 11.05.2023).

26. Сайт для вивчення HTML, CSS та JavaScript, 2023. 48p. https://www.w3schools.com/. URL: HYPERLINK "https://www.w3schools.com/" \t "_blank" https://www.w3schools.com/ (дата звернення: 11.05.2023).

27. Project Management / Adrienne Watt : FreeComputerBooks, 2022. 120 p. URL: https://cutt.ly/VwMGD1CD (дата звернення 03.04.2023 р.).

28. Special Topics in Information Technology / B. Pernici (ed.). Cham : Springer, 2020. 131p. URL: http://ebooks.znu.edu.ua/files/Bibliobooks/Inshi61/0045794.pdf. (дата звернення 03.04.2023 р.).

29. Nydelman G. Android Design Patterns : Interaction Design Solutions for Developers. Indianapolis : John Wiley & Sons, 2023. 458p. URL: http://ebooks.znu.edu.ua/files/Bibliobooks/Borysovska/0039958.pdf (дата звернення 03.04.2023 р.).

Декларація
академічної доброчесності
здобувача вищої освіти ЗНУ

Я Нікітенко Олексій Олександрович, студент _2_курсу, форми здобуття освіти денної, Інженерного навчально-наукового інституту ім. Ю. М. Потебні ЗНУ, спеціальності 121 Інженерія програмного забезпечення, адреса електронної пошти nikitenkoalexey@ukr.net

- підтверджую, що написана мною кваліфікаційна робота на тему **«Особливості розробки сайтів для онлайн реєстрації послуг»** відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;

- 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_____ Підпис_____ ПІБ (студент) _Нікітенко О. О._____

Дата_____ Підпис_____ ПІБ (науковий керівник) _Заяц В. І._____