МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АНАЛІЗ РЕАЛІЗАЦІЙ ТИПОВИХ КОМПОНЕНТІВ САЙТУ З ВИКОРИСТАННЯМ JAVASCRIPT БІБЛІОТЕК ANGULAR, REACT І VUE.JS»**

Виконав(ла): студент(ка) 2 курсу, групи 8.1219
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.І. Тихий
(ініціали та прізвище)

Керівник професор кафедри програмної інженерії, доцент, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

Лісняк А.О.

(підпис)

« » 2020 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Тихому Дмитру Івановичу

(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Аналіз реалізацій типових компонентів сайту з використанням JavaScript бібліотек Angular, React і Vue.JS

керівник роботи (проекту) Чопоров Сергій Вікторович, д.т.н., доцент

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » травня 2020 року № 576-с

2. Строк подання студентом роботи 30.11.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

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

1. Постановка задачі.
2. Основні теоретичні відомості.
3. Аналіз реалізацій типових компонентів сайту

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

Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 20.05.2020

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	10.06.2020	Виконано
2.	Збір вихідних даних.	15.07.2020	Виконано
3.	Обробка методичних та теоретичних		
	джерел.	21.08.2020	Виконано
4.	Розробка першого та другого розділу.	11.09.2020	Виконано
5.	Розробка третього та четвертого розділу.	15.11.2020	Виконано
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи.	26.11.2020	Виконано
7.	Захист кваліфікаційної роботи.	16.12.2020	Виконано

Студент _____ Д.І. Тихий
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Аналіз реалізацій типових компонентів сайту з використанням JavaScript бібліотек Angular, React і Vue.JS»: 83 с., 101 рис., 8 табл., 12 джерел.

АРХІТЕКТУРА, ФРЕЙМВОРК, БІБЛІОТЕКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБ-ІНТЕРФЕЙС, КОМПОНЕНТИ САЙТУ.

Об’єкт дослідження – компоненти сайту.

Мета роботи: проаналізувати реалізацію компонентів сайту написаних на JavaScript бібліотеках.

Метод дослідження – аналітичний, порівняльний.

У кваліфікаційній роботі розглядаються елементи веб-інтерфейсу, основні принципи асинхронного програмування та архітектура компонентів сайту. Розглянуто реалізації основних компонентів сайту з використанням JavaScript бібліотек, а саме: отримання даних з серверу, відображення інформації на веб-сторінках, системи маршрутизації, робота з класами та стилями, валідація даних, візуалізація даних, інтеграція бібліотек, асинхронне оновлення даних, інтернаціоналізація. Проаналізовано основні проблеми та переваги при розробці компонентів сайту за допомогою цих інструментів.

Результати роботи викладені у вигляді таблиць та висновків, що демонструють можливості обраних для аналізу JavaScript фреймворків і бібліотек, а також їх порівняння.

SUMMARY

Master’s Qualification Thesis «Analysis of Typical Site Components Implementations Using Angular, React and Vue.JS JavaScript Libraries»: 83 pages, 101 figures, 8 tables, 12 references.

ARCHITECTURE, FRAMEWORK, LIBRARY, SOFTWARE, WEB INTERFACE, COMPONENTS OF THE SITE.

The object of the study are site components.

The aim of the study is to analysis the implementation of site components written on JavaScript libraries.

The methods of research are analytical, comparative.

The qualification work considers the elements of the web interface, the basic principles of asynchronous programming and the architecture of the site components. Implementations of the main components of the site using JavaScript libraries are considered, namely: retrieving data from the server, displaying information on web pages, routing systems, working with classes and styles, data validation, data visualization, library integration, asynchronous data update, internationalization. The main problems and advantages of developing site components with the help of these tools are analyzed.

The results are presented in the form of tables and conclusions that demonstrate the capabilities of selected frameworks for analysis of JavaScript and libraries, as well as their comparison.

ЗМІСТ

ВСТУП

Починаючи роботу над створенням сайту, перед програмістом стає нелегкий вибір: який JavaScript фреймворк чи бібліотеку обрати для написання клієнтської частини web-додатка – Angular, Vue.js або React? Дані технології є найпопулярнішими у своєму класі та мають величезну підтримку як зі сторони творців, так і зі сторони розробників. В чомусь вони схожі, а в чомусь сильно відрізняються. Кожен з них підходить для вирішення всіх типових завдань, які потребують використання мови JavaScript, але є й такі завдання, з якими легче буде справитись обравши ту, чи іншу технологію.

Саме тому, ця робота є актуальною на сьогодні, адже важливим кроком в розробці сайту є вибір інструменту для написання клієнтської частини web-додатка.

Метою кваліфікаційної роботи є аналіз реалізацій типових компонентів сайту з використанням JavaScript бібліотек Angular, React і Vue.JS, а саме:

1) навігація: впроваджує багатосторінкову структуру та динамічний перехіжд між сторінками сайту без перезавантаження поточної сторінки;
2) авторизація: включає в себе можливість реєстрації, відновлення паролю та входу до системи, а також відправку даних до серверу;
3) блог: дозволяє переглядати списки постів з пагінацією та переходити на один із них;
4) візуалізація даних: відображає інформацію про курс гривні, який береться по API, за допомогою графіків;
5) локалізація: надає можливість обрати мову відображення контенту сайту.
6) чат: дає можливість відправляти та отримувати повідомлення в реальному часі.

7) карти Google: відображають інформацію про місцезнаходження з можливістю встановлення маркерів.

Ознайомившись з даними технологіями та проаналізувавши всі компоненти, потрібно обрати який із фреймворків буде кращим при розробці конкретного компоненту. При аналізі необхідно виділити плюси та мінуси конкретної бібліотеки, відобразити їх показники у вигляді таблиці, а також порівняти та обрати найкращу із них.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 JavaScript фреймворк

JavaScript є мовою з декількома парадигмами. Він підтримує об’єктно-орієнтовані, імперативні, прототипні та функціональні стилі програмування, а також керування подіями [1].

Зараз JavaScript є важливою частиною інтернету, він використовується на 95% сторінках усіх веб-сайтів, а інтернет є важливою частиною сучасного життя. У JavaScript є багате середовище з великою кількістю інструментів, бібліотек та фреймворків, кожен з яких створений для вирішення різної складності завдань.

Незважаючи на те, що jQuery була найпопулярнішою бібліотекою JavaScript для веб-інтерфейсів, вона не мала засобів для послідовної обробки даних у спільних представленнях. Для вирішення цієї проблеми були придуманні так звані фреймворки, вдосконалені бібліотеки. Backbone, Knockout та Ember – були одними з найбільш ранніх таких фреймворків JavaScript, які швидко завоювали популярність.

JavaScript фреймворк – це інструмент для створення динамічних веб додатків на мові JavaScript. Як правило ними користуються при виконанні складних задач, потребуючих великих витрат часу. Переважно, фреймворки застосовуються для написання SPA (Single Page Applications) додатків, де web-додаток розміщений на одній web-сторінці, яка для забезпечення роботи завантажує весь необхідний код разом із завантаженням самої сторінки.

Вслід за Ember, у жовтні 2010 року було випущенно AngularJS, який задав нову планку у розробці сайтів на JavaScript [2].

Використовуючи JavaScript фреймворки, можна розробляти повноцінні сайти та функціональні модулі. Але слід зазначити, що вони створенні для

середніх та великих по розміру сайтів, а для написання більш простих додатків, рекомендується використовувати невеликі бібліотеки. Поява сучасних JavaScript фреймворків значно полегшило створення високодинамічних інтерактивних додатків.

Фреймворки та бібліотеки надають чітку структуру програми та використовують свої патерни проектування. Найбільш популярними патернами є: MVC, MVP і MVVM [3].

JavaScript фреймворки надають розробнику уже готові функції та рішення, а саме:

- маршрутизація;
- анімації;
- взаємодія з сервером через протокол HTTP;
- синхронізація стану і перегляду;
- локалізація;
- шаблони;
- CEO оптимізація;

Із суттєвих переваг написання клієнтської частини web-додатка на JavaScript фреймворку є:

- легка реалізація SPA;
- сувора структура програми;
- скорочення коду;
- генерація коду;
- модульність;
- підтримка усіма браузерами;
- висока швидкість;
- тестування;
- управління розмірами бандлів;
- документація;
- відкритий код;

– віртуальний DOM;

Із недоліків можна виділити неповну підтримку пошуковими системами, але для вирішення цієї проблеми можна застосувати SSR (Server-Side Rendering) – рендеринг додатку на сервері клієнтської частини. Це дасть змогу пошуковими системам зчитувати інформацію з усіх динамічних сторінок сайту.

Наразі існує велика кількість фреймворків та бібліотек на JavaScript, кожен з яких має як плюси, так і мінуси. Але найпопулярнішими та найфункціональнішими з них є: Vue.js, Angular та React (див. рис. 1.1). Саме ці фреймворки мають найбільшу підримку зі сторони творців та розробників, а також велику кількість прикладів реалізацій різних компонентів та інтеграцій найпоширеніших бібліотек.

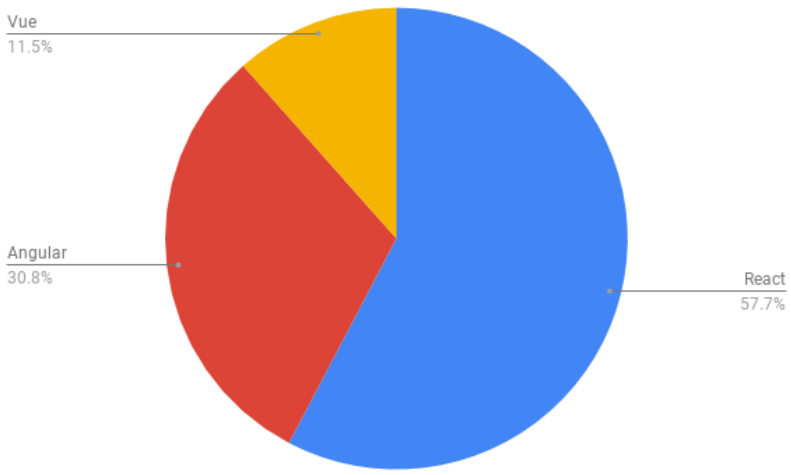

Рисунок 1.1 – Статистика завантажень Vue.js, Angular та React

1.2 Структура проектів

Створення Angular проекту відбувається через пакет @angular/cli та команду ng new app, де ng – це звернення до пакету, new – команда для створення, app – назва проекту.

В результаті виконання команди, в папці app було згенеровано новий проект з усіма необхідними папками та файлами (див. рис. 1.2).

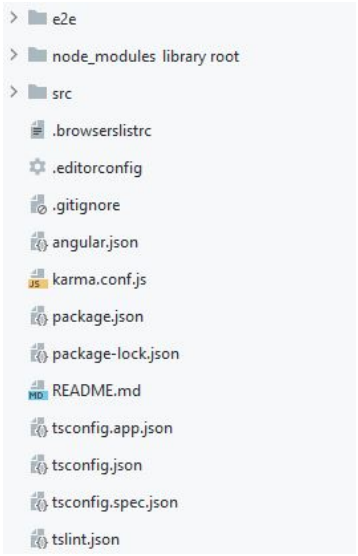

Рисунок 1.2 – Структура Angular проекту

Основним файлом проекту є angular.json, який потрібен для налаштування та компіляції додатка з використанням Angular CLI.

Файл має наступну структуру:

– projectType: тип проекту;
– root: вказує на папку файлів проекту щодо робочого середовища;
– sourceRoot: визначає кореневу папку файлів з вихідним кодом;
– architect: задає налаштування для побудови проекту;
– options: задає параметри побудови файлів;

Оскільки Angular використовує мову TypeScript, в кореневій папці проекту є файл tsconfig.json. Він визначає налаштування для компілятора TypeScript.

Для встановлення пакетів і залежностей, які будуть використовуватися проектом, використовується файл package.json.

Секція scripts описує команди (див. рис. 1.3), які можна виконувати для різних цілей. Зокрема, команда ng serve запускає веб-сервер для розробки програми і сам додаток, а команда ng build компілює додаток.

```
"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
```

Рисунок 1.3 – Список команд package.json

У секції dependencies в основному визначаються пакети angular, які необхідні додатку для роботи, а у секції devDependencies прописані тільки ті пакети, які будуть використовуватися для розробки. Зокрема, це пакети для роботи з мовою, а також пакети, необхідні для компіляції (див. рис. 1.4).

```
"dependencies": {
  "@angular/animations": "~10.2.0",
  "@angular/common": "~10.2.0",
  "@angular/compiler": "~10.2.0",
  "@angular/core": "~10.2.0",
  "@angular/forms": "~10.2.0",
  "@angular/platform-browser": "~10.2.0",
  "@angular/platform-browser-dynamic": "~10.2.0",
  "@angular/router": "~10.2.0",
  "rxjs": "~6.6.0",
  "tslib": "^2.0.0",
  "zone.js": "~0.10.2"
},
"devDependencies": {"@angular-devkit/build-angular": "~0.1002.0"...}
```

Рисунок 1.4 – Список основних пакетів Angular

Кореневою папкою проекту на Angular є папка src, яка має наступну структуру:

- папка app: містить в собі усі додатки та компоненти проекту;
- папка assets: слугує як сховище для публічних файлів;
- папка environments: містить в собі файли змінних;
- файл index.html: головна сторінка;
- файл main.ts: формує платформу, яка запускає додаток;

– файл polyfills.ts: визначає поліфіли;

– файл styles.scss: відповідає за імпорт стилів у проекті;

– файл test.ts: містить в собі налаштування для тестування модулів;

Створення Vue.js проекту відбувається через пакет @vue/cli та команду vue create app, де vue – це звернення до пакету, create – команда для створення, app – назва проекту.

Після завершення виконання команд, усі необхіді папки та файли будуть розташовані в папці app (див. рис. 1.5).

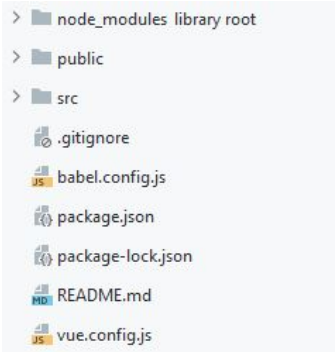

Рисунок 1.5 – Структура Vue.js проекту

Файлом для налаштування та компіляції додатку у Vue.js є файл vue.config.js. Він автоматично завантажує @vue або cli-service, якщо знаходить його в кореневому каталозі проекту. Цей файл є опціональним, налаштування також можна задавати в package.json, але це не є правильно, адже в такому випадку ви будете обмежені тільки JSON-сумісними значеннями.

Файл експортує об'єкт module.exports з налаштуваннями:

– publicPath: базовий URL-адресу збірки застосунка;

– outputDir: каталог для файлів збірки production;

– assetsDir: каталог для зберігання згенерованих статичних ресурсів;

– indexPath: шлях до згенерованого index.html;

Vue.js використовує нативний JavaScript, тому для статичного аналізу коду для виявлення проблемних шаблонів було згенеровано файл eslint.

Для встановлення усіх необхідних залежностей для коректної роботи додатку та виконання команд, використовується файл package.json (див. рис. 1.6).

```
"scripts": {
  "serve": "vue-cli-service serve",
  "build": "vue-cli-service build",
  "lint": "vue-cli-service lint"
},
"dependencies": {
  "core-js": "^3.6.5",
  "vue": "^2.6.11"
},
"devDependencies": {
  "@vue/cli-plugin-babel": "~4.5.0",
  "@vue/cli-plugin-eslint": "~4.5.0",
  "@vue/cli-service": "~4.5.0",
  "babel-eslint": "^10.1.0",
  "eslint": "^6.7.2",
  "eslint-plugin-vue": "^6.2.2",
  "vue-template-compiler": "^2.6.11"
},
```

Рисунок 1.6 – Список основних пакетів та команд Vue.js

Кореневою папкою проекту на Vue.js є папка src, вона має структуру:

– папка assets: слугує для зберігання статичних ресурсів;

– папка components: містить у собі компоненти сайту;

– файл App.vue: головний файл відображення;

– файл main.js: відповідає за експорт сторонніх бібліотек;

Папка public містить файл публічні файли, які доступні користувачам, а також файл index.html.

Альтернативною версією створення проекту на базі Vue.js є підключення CDN версії фреймворку.

Створення React проекту відбувається через команду npx create-react-app app, де npx – це звернення до пакету, create-react-app – команда для створення, app – назва проекту.

Після того як команда була виконана, у піпці app було згенеровано усі необхідні файли та папки (див. рис. 1.7).

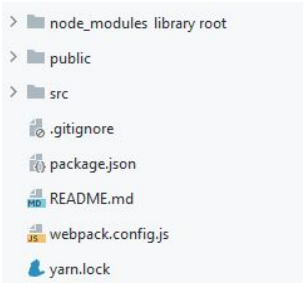

Рисунок 1.7 – Структура React проекту

Для налаштування та компіляції додатку React використовує файл webpack.config.js. Цей файл є опціональним і використовується для генерації кінцевої версії додатка.

Бібліотека використовує нативний JavaScript та транс компілятор Babel, який надає можливість для використання новітніх функцій JavaScript. Налаштування Babel також відбувається у файлі webpack.config.js.

Як і Angular та Vue.js, React має файл package.json, який містить у собі скрипти для вбудованих сценаріїв та їх попередньо встановлених подій життєвого циклу, а також набір залежностей, потрібних для нормального функціонування сайту на цій бібліотеці (див. рис. 1.8).

```
"dependencies": {
  "@testing-library/jest-dom": "^5.11.4",
  "@testing-library/react": "^11.1.0",
  "@testing-library/user-event": "^12.1.10",
  "react": "^17.0.1",
  "react-dom": "^17.0.1",
  "react-scripts": "4.0.0",
  "web-vitals": "^0.2.4"
},
```

Рисунок 1.8 – Список основних пакетів React

Папка src є кореневою та має наступну структуру:

– папка assets: містить статичні ресурси;

– файл App.js: компонент, який слугує для відображення контенту;

– файл index.js: основний файл, який відповідає за рендеринг додатку;

У папці public містяться публічні файли, які доступні користувачам, а також файл index.html, в якому є головний блок .

Результат аналізу структур проектів з використанням Angular, Vue.js, React та порівняння їх базових показників, можна побачити на таблиці 1.1.

Таблиця 1.1 – Порівняння базових показників Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Розмір папки	565 мб	187 мб	394 мб
Кількість файлів у проекті	54.364	24.428	57.233
Кількість залежностей	29	9	7
Швидкість компіляції	4.644 с	4.4 с	4.56 с
Вага скомпілованих файлів	5.1 м	624 кб	532 кб
Вага головного модуля	58.4 кб	32.04 кб	41.2 кб
Вага усіх модулів	2.5 мб	33.9 кб	44 кб
Швидкість завантаження сторінки	70 мс	50 мс	60 мс

Проаналізувавши таблицю, можна зробити висновок, що Angular має найбільш високі розміри файлів та менш швидші показники завантаження сторінки в порівнянні з іншими технологіями. Але все це обумовлено тим, що він має набагато більший функціонал та інтегрований TypeScript.

Vue та React, в першу чергу, доволі близькі за показникам, але все ж Vue.js має кращі результати, як в розмірах модулів, так і в швидкості завантаження сторінки. Але React має кращі показники розмірів скомпільованих файлів.

2 ОГЛЯД ТЕХНОЛОГІЙ

2.1 Angular

Angular – це JavaScript фреймворк, який використовується для розробки веб-додатків, а також платформа для створення ефективних та складних односторінкових додатків.

Angular був розроблений під керівництвом Angular Team у компанії Google [4]. Його перша версія була випущена в 2010 році, що робить його найстарішим із набору, а також він єдиний використовує мову TypeScript [5].

Суттєві зміни відбулися у 2016 році, коли було випущено оновлену версію, Angular 2 [5]. Саме ця версія набула найбільшої популярності серед розробників, хоча AngularJS (версія 1) все ще отримує оновлення.

На даний момент, останньою стабільною версією фреймворку є Angular 10, яка вийшла в червні 2020 року [5].

В Angular присутнє двостороннє зв'язування, яке дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому. Також в цьому фреймворкі інтегровано велику кількість дрібних бібліотек, які вирішили проблеми першої версії, наприклад Zone.js.

Для реактивного програмування, Angular використовує бібліотеку RxJS. Реактивне програмування - це асинхронна парадигма програмування, що стосується потоків даних та поширення змін.

RxJS (Reactive Extensions for JavaScript) – це бібліотека для реактивного програмування з використанням спостерігачів, що полегшує складання асинхронного коду або коду на основі зворотного виклику. Ця бібліотека забезпечує реалізацію типу Observable. RxJS також надає утилітні функції для створення та роботи зі спостерігачами. Ці утилітні функції можна використовувати для:

– перетворення існуючого коду для асинхронних операцій у спостерігачі;
– ітерація через значення в потоці;
– зіставлення значень з різними типами;
– фільтрування потоків;
– складання декількох потоків;
– зберігання поточного стану;
– зміна стану;
– обробка різних дій;
– виконання асинхронних запитів;
– правильна обробка помилок;

Фреймворк використовує Observable (див. рис. 2.1), який широко використовується при взаємодії з сервером по протоколу http.

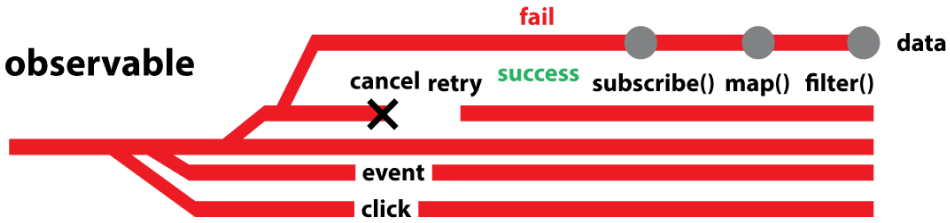

Рисунок 2.1 – Життєвий цикл Observable

Observable забезпечує підтримку передачі повідомлень між частинами вашої програми. Він часто використовуються в Angular і є рекомендованою технікою для обробки подій, асинхронного програмування та обробки декількох значень.

Основними перевагами цього фреймворку є:

– багатоплатформність: за допомогою Angular можна розробляти прогресивні веб-додатки, що дає змогу візуально і функціонально трансформувати сайт в додаток на різних платформах;

- висока швидкість та продуктивність: Angular перевизначає сучасну віртуальну машину JavaScript, перетворюючи шаблони в код;
- система тестування: завдяки інтегрованим інтелектуальним модулям для тестування, таким як Jasmine та Karma, можна будь-коли відстежувати помилки та виправляти непрацюючий код;

Технічними перевагами цього фреймворку є:

- архітектура: при побудові архітектури, Angular використовує шаблон проєктування MVC, що дає розробникам можливість досить просто розділити програму на моделі та компоненти, а решту фреймворк робить самостійно;
- двостороння прив'язка даних: найефективнішою функцією Angular є її двостороння система прив'язки даних за допомогою якої, системи, предтавлення і моделі завжди залишаються в ідеальній синхронізації, а при внесені змін в модель, користувачі можуть автоматично бачити усі зміни в представленні;
- інтерфейс командного рядка: Angular CLI відповідає найкращим галузевим практикам розробки веб-інтерфейсів із вражаючими вбудованими функціями підтримки та маршрутизації SCSS та дозволяє з легкістю генерувати готові функці;
- комплект для розробки компонентів та Angular Material: Angular вдосконалює свій комплект розробки компонентів (CDK) завдяки модернізації версій, а оновлення та віртуальна прокрутка – найважливіші особливості останнього Angular CDK;
- інтегрований TypeScript: за допомогою TypeScript, фреймворк пропонує розробникам безперебійну роботу, адже він високоефективний у виявленні помилок і тим самим скорочує час розробки [6];

- ін’єкція залежностей: Angular має власну структуру DI, яка зазвичай використовується при розробці програм для підвищення їх ефективності та модульності;
- директиви: це найскладніші Angular особливості, які дозволяють створювати власні теги HTML, що слугують власними віджетами, які розробники можуть використовувати в любій частині проекту та передавати в них різні дані за допомогою декораторів @Input() та @Output();

Недоліками використання даного фреймворку є:

- доволі великі розміри модулів;
- високий поріг входження;
- велика кількість інтегрованих допоміжних бібліотек;
- рендеринг відбувається через DOM, а не VDOM;
- нема можливості інтеграції через CDN;

2.2 Vue.js

Vue.js – це прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, він створений придатним для поступового впровадження [7].

Творцем Vue.js є Evan You, колишній співробітник Google і Meteor Dev Group [7]. Почав він розробляти фреймворк в 2013 року, а в лютому 2014-го відбувся перший публічний реліз [8]. Vue широко використовується серед китайських компаній, наприклад: Alibaba, Baidu, Xiaomi, Sina Weibo. Він входить в ядро Laravel і PageKit. Нещодавно вільна система управління репозиторіями GitLab теж перейшла на Vue.js.

В кінці вересня 2016 року вийшов в реліз Vue.js 2.0, який мав істотні зміни в продуктивності фреймворку [8]. У новій версії використовується віртуальний DOM, підтримується серверний рендеринг, а також можливість

використовувати JSX. На даний момент, останньою стабільною версією фреймворку є Vue.js 2.6.11, яка вийшла в грудні 2019 року [7].

Vue використовує синтаксис шаблонів на основі HTML, що дозволяє декларативно зв'язувати рендеринг DOM з основними екземплярами даних в Vue. Всі Vue шаблони валідні HTML, і можуть бути розпарсені браузерами та HTML парсерами. Всередині Vue компілює шаблони в рендерингові функції віртуального DOM. В поєднанні з реактивною системою, Vue здатний розумно обчислити кількість компонентів для ре-рендингу та застосувати мінімальну кількість маніпуляцій з DOM, коли стан застосунку зміниться.

В Vue можна напряму писати рендерингові функції використовуючи JSX. Для того, щоб це зробити просто замініть шаблон на рендерингову функцію. Рендерингова функція відкриває можливості для потужних патернів базованих на компонентах – для прикладу, нова транзитна система тепер повністю базована на компонентах, що використовує рендерингові функції всередині.

Для управління станом веб-додатку, фреймворк використовує бібліотеку Vuex. Vuex – це бібліотека та патерн управління станом додатків [9]. Він служить централізованим сховищем даних для всіх компонентів програми з правилами, які гарантують, що стан може бути змінено тільки передбачуваним чином. Vuex інтегрується з офіційним розширенням vue-devtools, надаючи «з коробки» такі потужні можливості.

На відміну від інших патернів, Vuex реалізований у вигляді бібліотеки, спеціально призначеної для Vue.js, щоб використовувати його систему реактивності (див. рис. 2.2). Vuex допомагає керувати спільно використовуваним станом ціною привнесення нових концепцій і допоміжного коду.

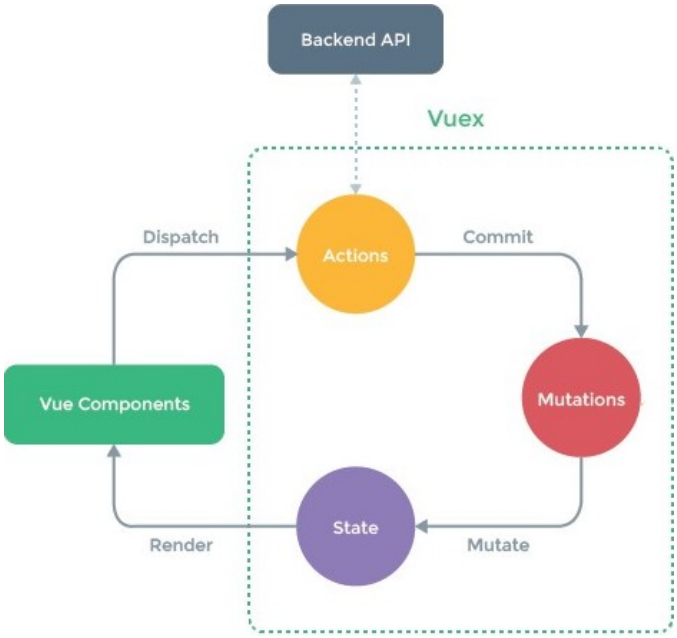

Рисунок 2.2 – Основний потік даних Vuex

Із переваг цього фреймворку можна виділити:

– віртуальний DOM: Vue.js використовує віртуальний DOM, який поглинає всі зміни, призначені для DOM, а також присутній у вигляді структур даних JavaScript;
– прив'язка даних: ця функція допомагає зв’язувати компоненти, маніпулювати або призначати значення атрибутам HTML, змінювати стилі, призначати класи за допомогою директиви прив'язки, яка називається v-bind;
– CSS анімації: Vue має вже готові методи застосування переходу до елементів HTML, коли вони додаються, оновлюються або видаляються з DOM, а також фреймворк має вбудований компонент переходу, який обгортає елемент, відповідальний за повернення ефекту переходу;
– шаблони: фреймворк пропонує шаблони на основі HTML, які пов'язують DOM з даними екземпляра Vue.js, який компілює шаблони у функції віртуального DOM Render та дозволяє передавати дані між компонентами;

- обчислювані властивості: однією з найважливіших особливостей фреймворку є обчислювані властивості, вони допомагають прослуховувати зміни, внесені в елементи інтерфейсу користувача, і виконувати необхідну логіку, що дає змогу зменшувати розмір коду;
- спостерігачі: спостерігачі застосовуються до даних, які будуть часто змінюватися, що дає можливість розробнику не робити жодних додаткових заходів;
- гнучкість та модульність: важливим аспектом будь-якої нової технології є можливість інтеграції з існуючими додатками, а шаблон веб-пакета Vue.js не обмежує доступ до його розширених функцій, таких як гаряче перезавантаження модулів, CSS файлів, підключення тощо, що дає можливість додавання будь-якого стороннього пакета у проект;

Недоліками використання даного фреймворку є:

- невелика підтримка зі сторони творців;
- складність реактивності;
- відсутність підтримки масштабних проектів;
- обмеженість ресурсів;

2.3 React

React – це бібліотека на мові JavaScript, яка поєднує його швидкість та використовує новий спосіб рендерингу веб-сторінок, роблячи їх надзвичайно динамічними та чутливими до дій користувача [10]. Після випуску бібліотеки вона стала надзвичайно популярною завдяки своєму революційному підходу до програмування користувальницьких інтерфейсів, що встановлює тривалу конкуренцію між іншим популярним інструментом веб-розробки.

Бібліотеку було створено Джорданом Волком (Jordan Walke), програмістом з Facebook. Автор працював над проектом під впливом XHP,

фреймворку HTML для PHP [11]. 2011-го року реліз з'явився у новинах Facebook [11]. Також фреймворк був представлений як проект з відкритим початковим кодом на конференції розробників JSConf US, що проходила у Сполучених Штатах у травні 2013 року [10]. На конференції React.js Conf, влаштовану Фейсбуком у березні 2015-го, проект було представлено як відкрите програмне забезпечення [11]. На даний момент, останньою стабільною версією фреймворку є React 16.13.1, яка вийшла в березні 2020 року [10].

React реалізує концепції реактивного програмування: зміна результату роботи залежить від стану компонентів. Віртуальний DOM у React покращує взаємодію з користувачем, а розробник пришвидшує роботу.

DOM (об'єктна модель документа) – це логічна структура документів у форматах HTML, XHTML або XML. Основне занепокоєння щодо традиційної конструкції DOM полягає в тому, як він обробляє зміни, тобто вводи користувачів, запити тощо. Сервер постійно перевіряє різницю, спричинену цими змінами, щоб надати необхідну відповідь. Щоб правильно відповісти, йому також потрібно оновити дерево DOM у цілому документі, що не є ергономічним та практичним рішенням, оскільки дерева DOM сьогодні досить великі та містять тисячі елементів. Команді React вдалося збільшити швидкість оновлень за допомогою віртуального DOM. На відміну від інших фреймворків, які працюють із Real DOM, ReactJS використовує свою абстрактну копію – Virtual DOM. Він оновлює навіть мінімалістичні зміни, застосовані користувачем, але не впливає на інші частини інтерфейсу.

React знаменитий своїми нетрадиційними рішеннями. Однією з якої є створення елементів інтерфейсу в JavaScript замість шаблонів, а також створення суперсету мови JavaScript – JSX.

Для побудови архітектури проекту та управління станом програми, React використовує Redux (див. рис. 2.3). Redux – це контейнер для управління станом додатку і багато в чому нагадує Flux [12].

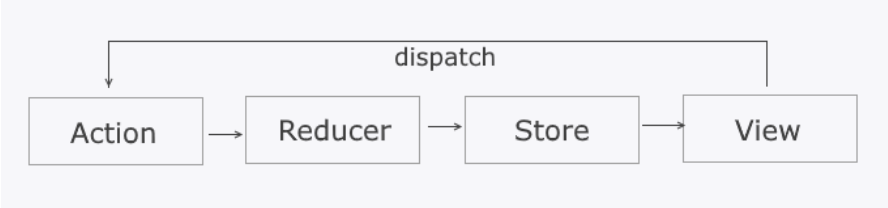

Рисунок 2.3 – Загальна схема взаємодії елементів архітектури Redux

Ключові моменти Redux:

- творці дій (action creators): функції, які створюють дії;
- сховище (store): зберігає стан додатка;
- дії (actions): деякий набір інформації, який виходить від додатка до сховища і який вказує, що саме потрібно зробити;
- reducer: функція (або кілька функцій), яка отримує дію і відповідно до цього дією змінює стан сховища;

Із переваг цієї бібліотеки можна виділити:

- віртуальний DOM: як і Vue, React зберігає в пам'яті полегшене уявлення про "справжній" DOM, який відомий як "віртуальний" DOM (VDOM);
- продуктивність: React використовує VDOM, завдяки чому веб-програми працюють дуже швидше, а також розбиває складний користувальницький інтерфейс на окремі компоненти, дозволяючи декільком користувачам працювати над кожним компонентом одночасно, тим самим прискорюючи час розробки;
- розширення: бібліотека виходить за рамки простого дизайну інтерфейсу та має безліч розширень, які пропонують повну підтримку архітектури додатків, а також забезпечує візуалізацію на стороні сервера, що передбачає рендеринг веб-програми, яка зазвичай виконується лише на стороні клієнта, на сервері, а потім надсилає клієнтові повністю відтворену сторінку;
- однонаправлена прив'язка даних;

- компоненти: компоненти є будівельними блоками будь-якої програми React, які діляться на два типи, компоненти класу та функціональні компоненти;
- тестування: для пошуку несправностей в коді, React використовує методи модульного тестування;

Недоліками використання даного фреймворку є:

- темпи розвитку;
- погана документація;
- відсутність підтримки масштабних проектів;

3 РЕАЛІЗАЦІЯ БАЗОВИХ КОМПОНЕНТІВ: АВТОРИЗАЦІЯ ТА НАВІГАЦІЯ НА САЙТІ

3.1 Навігація

Більшість SPA додатків мають кілька сторінок, на які можна переходити динамічно, тобто без перезавантаження головної сторінки.

В даному випадку, компонентом навігації є спеціальне меню (див. рис. 3.1).

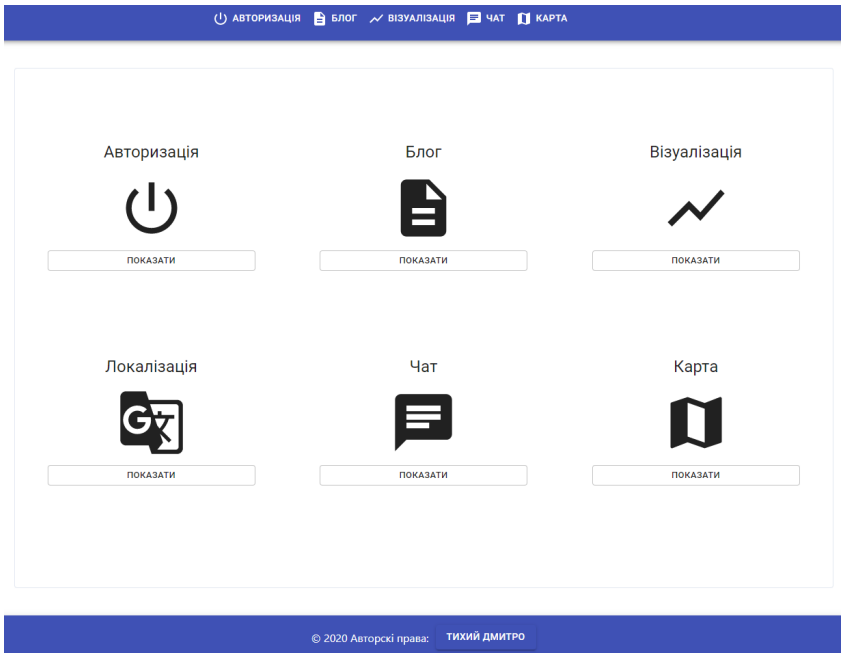

Рисунок 3.1 – Навігаційне меню сайту

3.1.1 Angular

Для роботи з маршрутизацією в Angular, спочатку потрібно визначити базову адресу програми. Для цього потрібно додати у веб-сторінку index.html в секцію <head> елемент <base href="/"/> (див. рис. 3.2).

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8"/>
        <title>Angular</title>
        <base href="/"/>
    </head>
    <body>
        <app-root></app-root>
    </body>
</html>
```

Рисунок 3.2 – Файл index.html в Angular

Щоб мати можливість задавати маршрути сторінок, було створено клас AppRoutingModule у файлі app-routing.module.ts. В ньому, за допомогою константи routes, яка є масивом, перечислено усі потрібні маршурути у вигляді об'єктів (див. рис. 3.3).

```
const routes: Routes = [
    {
        path: '',
        loadChildren: () => import('./pages/pages.module').then((mod :{PagesModule: PagesModule} ) => mod.PagesModule)
    },
    {
        path: 'auth',
        loadChildren: () => import('./modules/auth/auth.module')
            .then((mod :{AuthModule: AuthModule} ) => mod.AuthModule)
    }
];
```

Рисунок 3.3 – Маршрути навігації

Об'єкти routes мають наступні параметрами:

– path: маршрут по якому буде завантажуватись компонент;

– component: компонент, який буде завантажуватись;

– loadChildren: колбек функція для лінивої загрузки модулів;

Для переходу між сторінками було створено компонент header з навігаційним меню (див. рис. 3.4).

```
<div fxLayout="row" fxShow="false" fxShow.gt-sm>
    <button routerLinkActive="mat-stroked-button" mat-button routerLink="/">
        <mat-icon>power_settings_new</mat-icon>
        Головна сторінка
    </button>
    <button routerLinkActive="mat-stroked-button" mat-button routerLink="/auth">
        <mat-icon>power_settings_new</mat-icon>
        Авторизація
    </button>
</div>
```

Рисунок 3.4 – Навігаційне меню

Перехід відбувається за допомогою директиви routerLink="/ ", яка приймає параметр маршруту.

Компонент <router-outlet></router-outlet> приймає та підміняє на своє місце компонент, обраний для обробки запиту.

Для переадресації потрібно задати параметр redirectTo, а також ми можемо поставити критерій відповідності рядка запиту маршруту за допомогою параметра pathMatch. Його значення представляє шлях переадресації.

Таким чином, при переході на сторінку /auth, буде здійснена переадресація на сторінку /auth/login.

Для здійсненя програмної навігації, використовується клас Router, який потрібно підключити в конструкторі, а потім визвати метод navigate(), який приймає параметр маршруту (див. рис. 3.5).

```
constructor(
    private router: Router
) {
    this.router.navigate( commands: ['/auth']).then();
}
```

Рисунок 3.5 – Приклад використання програмної навігації

3.1.2 Vue.js

За роботу навігації у Vue.js відповідає спеціальна бібліотека маршрутизації vue-router, яка дуже схожа на Angular Router і має з ним спільні параметри та методи.

Спочатку було створено файл index.js (див. рис. 3.6) у каталозі router, та визначено там константу routes типу масив, який в свою чергу, буде містити у собі об'єкти з параметрами path, component, name:

```
const routes = [
    {
        path: "/", name: "Home",
        component: () => import(/* webpackChunkName: "home" */ "../views/Home.vue")
    },
    {
        path: "/auth", name: "Auth", redirect: "/auth/login",
        component: () =>
            import(/* webpackChunkName: "auth" */ "../modules/auth/Auth.vue")
    }
];
```

Рисунок 3.6 – Маршрути навігації

Далі потрібно імпорутвати router до основного файлу main.js та підключити його до проекту як параметр екземпляра класу Vue, а саме – const app = new Vue({router, render: h => h(App)}).

Наступним кроком було створено навігаційне меню у компоненті header (див. рис. 3.7).

```
<div id="nav">
    <router-link to="/">
        <md-button>
            <md-icon>power_settings_new</md-icon>
            Головна сторінка
        </md-button>
    </router-link>
    <router-link to="/auth">
        <md-button>
            <md-icon>account_box</md-icon>
            Авторизація
        </md-button>
    </router-link>
</div>
```

Рисунок 3.7 – Навігаційне меню Vue.js

Перехід між сторінками відбувається за допомогою компоненту <router-link to=""></router-link>. Він має параметр to в який потрібно передати шлях маршруту.

Компонент <router-view></router-view> приймає та підміняє на своє місце компонент, обраний для обробки запиту.

Програмна навігація у Vue.js відбувається за допомогою маршрутизатора $router. Достатньо визвати метод push(), а саме this.$router.push({ path: 'auth' }).

3.1.3 React

У React є своя система маршрутизації, яка дозволяє зіставляти запити до додатка з певними компонентами. Ключовою ланкою в роботі маршрутизації є модуль react-router, який містить основний функціонал по роботі з маршрутизацією. Кожен Router створює об'єкт history який зберігає шлях до поточного location і перемальовує інтерфейс сайту коли відбуваються якісь зміни шляху.

Спочатку потрібно імпортувати потрібні компоненти до файлу App.js за допомогою методу lazy (див. рис. 3.8).

```
const Home = React.lazy( factory: () => import(/* webpackChunkName: "Home" */'./pages/HomePage'));
const AuthLogin = React.lazy( factory: () => import(/* webpackChunkName: "AuthLogin" */'./modules/auth/pages/AuthLogin'));
```

Рисунок 3.8 – Лінивий імпорт компонентів

Далі потрібно добавити у компонент <Switch></Switch> потрібні маршрути (див. рис. 3.9).

```
<Switch>
    <Route path="/" exact>
        <Home/>
    </Route>
    <Route path="/auth">
        <Redirect to="/auth/login"/>
    </Route>
</Switch>
```

Рисунок 3.9 – Маршрути навігації

Перехід між сторінками відбувається за допомогою компоненту < Route path=""></Route>. Він має параметр path, в який передається шлях маршруту.

Наступним кроком є додання компонентів <Link></Link> навігаційного меню у компонент Header (див. рис. 3.10).

```
<div className="nav">
    <Link to="/" className={classes.link}>
        <Button className={classes.navBtn}>
            <HomeIcon/>
            Головна сторінка
        </Button>
    </Link>
    <Link to="/auth" className={classes.link}>
        <Button className={classes.navBtn}>
            <AuthIcon/>
            Авторизація
        </Button>
    </Link>
</div>
```

Рисунок 3.10 – Навігаційне меню

Програмна навігація у React відбувається за допомогою функції useNavigation(), а для переходу, достатньо визвати метод navigate().

3.1.4 Результати аналізу

Результати аналізу компонентів навігації сайту з використанням Angular, Vue.js та React можна побачити на таблиці 3.1.

Таблиця 3.1 – Порівняння показників компонентів навігації у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Легка	Легка
Кількість файлів компонентів	5	3	2
Розмір файлів компонентів	7.93 кб	6 кб	5 кб
Швидкість завантаження сторінки	17 мс	17 мс	19 мс
Розмір модулю	5.6 кб	3.2 кб	2.1 кб

Завершуючи аналіз, можна зробити висновок, що найскладнішим та найдовшим рішенням є розробка цього компоненту на Angular. Але слід зазначити, що даний фреймворк надає більше функцій кастомізації маршрутів, ніж інші.

Навігація у Vue дуже схожа на Angular, але в деяких моментах, логіка побудови даного компоненту легша для розуміння та написання коду, ніж у другого. Також Vue має доволі потужний маршрутизатор, який має багато параметрів для кастомізації.

У React, побудова навігації є найпростішою та потребує менше всього коду та часу на розробку.

3.2 Авторизація

Одним із найголовніших компонентів сайту є авторизація, а основним стандартом є аутентифікація JWT.

Для аналізу цих компонентів було створено три форми, які мають валідацію усіх полей та кнопку відправки данних, а саме:

– форма реєстрації (див. рис. 3.11);
– форма входу (див. рис. 3.12);
– форма відновлення паролю (див. рис. 3.13);

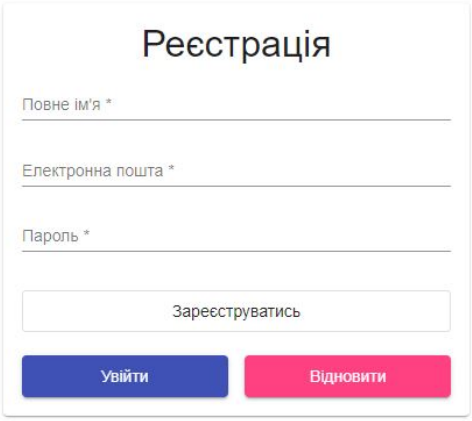

Рисунок 3.11 – Форма реєстрації

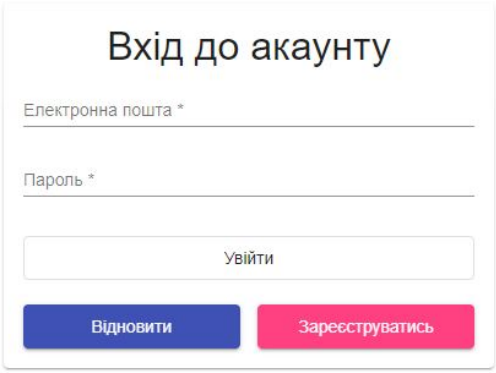

Рисунок 3.12 – Форма входу

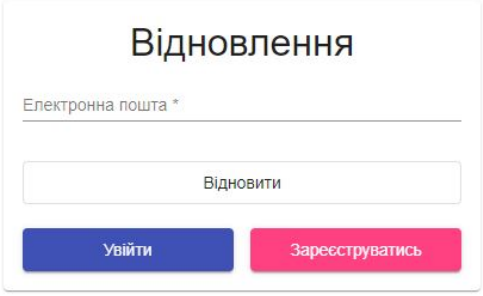

Рисунок 3.13 – Форма відновлення

3.2.1 Angular

Так як Angular використовує MVC патерн, структура компонентів складається з основного модулю, сервісу даних, дочірніх компонентів та моделей.

Далі йде підключення усіх сторінок до головного модуля із зазначенням маршрутів навігації (див. рис. 3.14);

```
path: '',
component: AuthComponent,
children: [
    {path: '', redirectTo: 'login', pathMatch: 'full'},
    {path: 'login', component: LoginComponent,},
    {path: 'register', component: RegisterComponent,},
    {path: 'recovery', component: RecoveryComponent,}
]
```

Рисунок 3.14 – Маршрути авторизації у Angular

Для роботи з формами, Angular має спеціальну бібліотеку @angular/forms. Щоб використовувати потрібні класи та сервіси цієї бібліотеки, їх потрібно імпортувати до компоненту використовуючи інструкцію import, а саме – import { FormBuilder, FormGroup, Validators } from '@angular/forms'.

За зберігання стану відповідає об'єкт form типу FormGroup, а для ініціалізації полів форми використовується сервіс FormBuilder (див. рис. 3.15).

```
form: FormGroup;
constructor(
    private fb: FormBuilder,
) {
    this.form = this.fb.group( controlsConfig: {
        username: ['', Validators.email],
        password: ['', Validators.required],
        full_name: ['', Validators.required]
    });
}
```

Рисунок 3.15 – Ініціалізація форми у Angular

За валідацію елементів форми відповідає абстрактний клас Validators.

Форма авторизації використовує наступні валідації:

– Validators.email: перевіряє правильність вводу електронної адреси;

– Validators.required: перевіряє заповненість поля;

Для відправки данних було створено сервіс AuthService, який підключається до компоненту через конструктор у вигляді приватної змінної authService типу AuthServic.

Сервіс містить у собі захищену від запису змінну apiUrl строкового типу, яка ініціалізується у конструкторі класу, та використовується для зберігання кінцевої точки API.

Для зв'язку з API, був використаний вбудований в Angular сервіс HttpClient, а обробку запросів робить клас бібліотеки rxjs Observable.

Для проведення авторизації сервіс має три методи (див. рис. 3.16).

```
login(params: {}): Observable<any> {
    return this.http.post( url: `${this.apiUrl}/login`, params).pipe(map( project: (
        response: { user: User, sessionData: { token: string, permissions: string }
        }) => {
            this.setSession(response.sessionData.token, response.sessionData.permissions);
            return new User(response.user)
        }
    ));
}
register(params: {}): Observable<any> {
    return this.http.post( url: `${this.apiUrl}/register`, params);
}
recover(params: {}): Observable<any> {
    return this.http.post( url: `${this.apiUrl}/recover`, params);
}
```

Рисунок 3.16 – Методи сервісу авторизації у Angular

Щоб підтримувати доступ до акаунту, було написано метод setSession(), який приймає в якості параметрів токен та права доступу, які будуть записуватися у локальне сховище браузера за допомогою localStorage (див. рис. 3.17).

```
setSession(token: string, permissions: string) {
    localStorage.setItem('token', token);
    localStorage.setItem('permissions', permissions);
}
```

Рисунок 3.17 – Метод запису даних у локальне сховище браузера

HTML частина складається з тегу <form></form> в яку підключається директива [formGroup] і метод (ngSubmit) (див. рис. 3.18).

```
<form [formGroup]="form" (ngSubmit)="onSubmit()">
    <h2>{{'auth-register-title' | transloco}}</h2>
    <mat-form-field class="full-width-input">
        <input matInput placeholder="{{'label-name' | transloco}}" formControlName="full_name">
    </mat-form-field>
    <mat-form-field class="full-width-input">
        <input matInput placeholder="{{'label-email' | transloco}}" formControlName="username">
    </mat-form-field>
    <mat-form-field class="full-width-input">
        <input matInput placeholder="{{'label-password' | transloco}}" formControlName="password">
    </mat-form-field>
    <button type="submit" mat-stroked-button>{{'label-register-submit' | transloco}}</button>
    <div class="links">
        <a mat-raised-button color="primary" routerLink="/auth/login">{{'label-login' | transloco}}</a>
        <a mat-raised-button color="accent" routerLink="/auth/recovery">{{'label-recover' | transloco}}</a>
    </div>
</form>
```

Рисунок 3.18 – HTML форма

Елементи форми представлені у вигляді HTML тегів <input/>, які підключаються до форми за допомогою параметру formControlName.

3.2.2 Vue.js

Для побудови компонентів авторизації за допомогою Vue, було також використано патерн MVC. Структура модулю складається з сервісу даних, моделі та сторінок.

Далі йде підключення сторінок до маршрутизатору (див. рис. 3.19).

```
{
    path: "/auth", name: "Auth", redirect: "/auth/login"
},
{
    path: "/auth/login", name: "AuthLogin",
    component: () => import(/* webpackChunkName: "auth-login" */ "../modules/auth/pages/AuthLogin.vue")
},
{
    path: "/auth/register", name: "AuthRegister",
    component: () => import(/* webpackChunkName: "auth-register" */ "../modules/auth/pages/AuthRegister.vue")
},
{
    path: "/auth/recover", name: "AuthRecover",
    component: () => import(/* webpackChunkName: "auth-recover" */ "../modules/auth/pages/AuthRecovery.vue")
},
```

Рисунок 3.19 – Маршрути модуля авторизації

Об'єктом форми є змінна user типу User, яка ініціалізується у методі data, а для валідації використовуються методи бібліотеки vuelidate (див. рис. 3.20).

```
data: () => ({
    user: new User( full_name: "",  email: "",  password: ""),
}),
validations: {
    user: {
        email: {required, email},
        password: {required},
        full_name: {required}
    }
},
```

Рисунок 3.20 – Ініціалізація форми

Для зв'язку з API та відправки даних, був створений сервіс AuthService, який імпортується в компонент за допомогою інструкції import AuthService from "../services/auth.service".

Для відправки даних, використовується бібліотека axios, яка рекомендована розробниками Vue.js, та підтримує усі популярні методи для запитів: GET, POST, PUT, DELETE.

Сервіс має константу API_URL з шляхом до сервру, а також три основні методи для відправки даних (див. рис. 3.21).

```
login(params) {
    return axios.post( url: `${API_URL}/login`, params).then(response => {
        this.setSession(response.data.token, response.data.permissions);
        return response.data;
    });
}
recover(email) {
    return axios.post( url: `${API_URL}/recover`,  data: {email: email});
}
register(user) {
    return axios.post( url: `${API_URL}/register`, user);
}
```

Рисунок 3.21 – Методи сервісу авторизації

Для зберігання даних у браузері використовуються методи window.localStorage.

HTML частина складається з тегу <form></form> з атрибутом @submit.prevent для обробки натиснення кнопки відправки (див. рис. 3.22).

```
<form novalidate class="md-layout" @submit.prevent="validateUser">
    <md-field :class="getValidationClass( fieldName: 'full_name')">
        <label for="full_name">{{ $t('label-name') }}</label>
        <md-input id="full_name" v-model="user.full_name" :disabled="sending"/>
    </md-field>
    <md-field :class="getValidationClass( fieldName: 'email')">
        <label for="email">{{ $t('label-email') }}</label>
        <md-input id="email" v-model="user.email" :disabled="sending"/>
    </md-field>
    <md-field :class="getValidationClass( fieldName: 'password')">
        <label for="password">{{ $t('label-password') }}</label>
        <md-input id="password" v-model="user.password" :disabled="sending" type="password"/>
    </md-field>
    <md-button type="submit" class="md-raised md-primary" :disabled="sending">
        {{ $t('label-register-submit') }}
    </md-button>
</form>
```

Рисунок 3.22 – Форма авторизації

Елементи форми представлені у вигляді HTML тегів <input/>, які зв'язуються з формую через атрибут v-model.

3.2.3 React

Структура компонентів авторизації у React повністю співпадає зі структурою Vue та складається з сервісу даних, моделі і сторінок, які використовують JSX.

Підключення сторінок відбувається у головному модулю.

За зберігання даних форми відповідають змінні об'єкта this.state, а зміна стану цього об'єкта відбувається за допомогою методу setState().

Ініціалізація даних та методів відбувається у конструкторі класу (див. рис. 3.23).

```
constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.onChangeEmail = this.onChangeEmail.bind(this);
    this.onChangePassword = this.onChangePassword.bind(this);
    this.onChangeName = this.onChangeName.bind(this);
    this.handleClose = this.handleClose.bind(this);
    this.state = {
        email: "",
        password: "",
        full_name: "",
    };
}
```

Рисунок 3.23 – Ініціалізація форми

Сервіс даних AuthService також має константу з шляхом до серверу та три основні асинхроні методи відправки даних за допомогою fetch (див. рис. 3.24).

```
async login(params) {
    const response = await fetch( input: `${API_URL}/login`, init: {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(params)
    });
    return await response.json();
}
async recover(email) {
    const response = await fetch( input: `${API_URL}/recover`, init: {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify( value: {email: email})
    });
    return await response.json();
}
async recover(params) {
    const response = await fetch( input: `${API_URL}/register`, init: {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(params)
    });
    return await response.json();
}
```

Рисунок 3.24 – Асинхронні методи відправки даних

Для запису даних у браузер, також використовуються методи window.localStorage.

Метод render() складається з тегу <form></form> з атрибутом onSubmit для обробки натиснення кнопки відправки (див. рис. 3.25).

```
<form noValidate autoComplete="off" onSubmit={this.handleSubmit}>
    <div className="form-control">
        <TextField className="form-input" label={this.props.t('label-name')}
                   value={this.state.full_name} onChange={this.onChangeName}/>
    </div>
    <div className="form-control">
        <TextField className="form-input" label={this.props.t('label-email')}
            type="email" value={this.state.email} onChange={this.onChangeEmail}/>
    </div>
    <div className="form-control">
        <TextField className="form-input" label={this.props.t('label-password')}
                   type="password" value={this.state.password} onChange={this.onChangePassword}/>
    </div>
    <Button type="submit" className="submit" variant="contained" color="default">
        {this.props.t('label-register-submit')}
    </Button>
    <div className="buttons">
        <Button variant="contained" color="primary" href="/auth/recover">
            {this.props.t('label-recover-submit')}
        </Button>
        <Button variant="contained" color="secondary" href="/auth/login">
            {this.props.t('label-login-submit')}
        </Button>
    </div>
```

Рисунок 3.25 – JSX форма авторизації

Зв'язування форми з об'єктом state відбувається за допомогою атрибута value.

Так як у React тільки однонаправлене зв'язування даних, потрібно постійно оновлювати об'єкт state. Для цього в асинхроний атрибут onChange був доданий метод оновлення даних onChange() (див. рис. 3.26).

```
onChange = (e, param) => {
    this.props.user.info[param] = e.target.value;
}
```

Рисунок 3.26 – Оновлення даних форми

3.2.4 Результати аналізу

Результати аналізу компонентів авторизації з використанням Angular, Vue.js та React можна побачити на таблиці 3.2.

Таблиця 3.2 – Порівняння показників компонентів авторизації у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Середня	Складна
Кількість файлів компонентів	19	6	7
Розмір файлів компонентів	100 кб	44 кб	48 кб
Швидкість завантаження сторінки	10 мс	7мс	8 мс
Розмір модулю	34 кб	19 кб	22 кб

Завершуючи аналіз, можна зробити висновок, що найскладнішим та найдовшим рішенням є розробка цього компоненту на React, через те, що бібліотека не приспостосована для роботи із формами. Через це, розробникам доводиться більше витрачати часу на розробку компонентів, а однонапревлене зв'язуваня даних постійно потребує використання методів оновленя.

Vue та Angular мають вбудовані сервіси та бібліотеки для роботи з формами, що значно зменшує час на розробку логіки компонентів. Хоч Angular і має більші розміри файлів, але він дає більше інструменів для форм, а інтегрований TypeScript дає чітку валідацію змінних за рахунок строгої типізації, чого не скажеш про Vue та React.

4 РОЗРОБКА ТИПОВИХ КОМПОНЕНТІВ РОБОТИ З ДАНИМИ

4.1 Блог

Блог є невід'ємним компонентом більшості сайтів.

Основною роботою блогу є вивід постів (див. рис. 4.1) та перехід на один із них. Список має функціонал пагінації та фільтри по регуляції кількості елементів на сторінці.

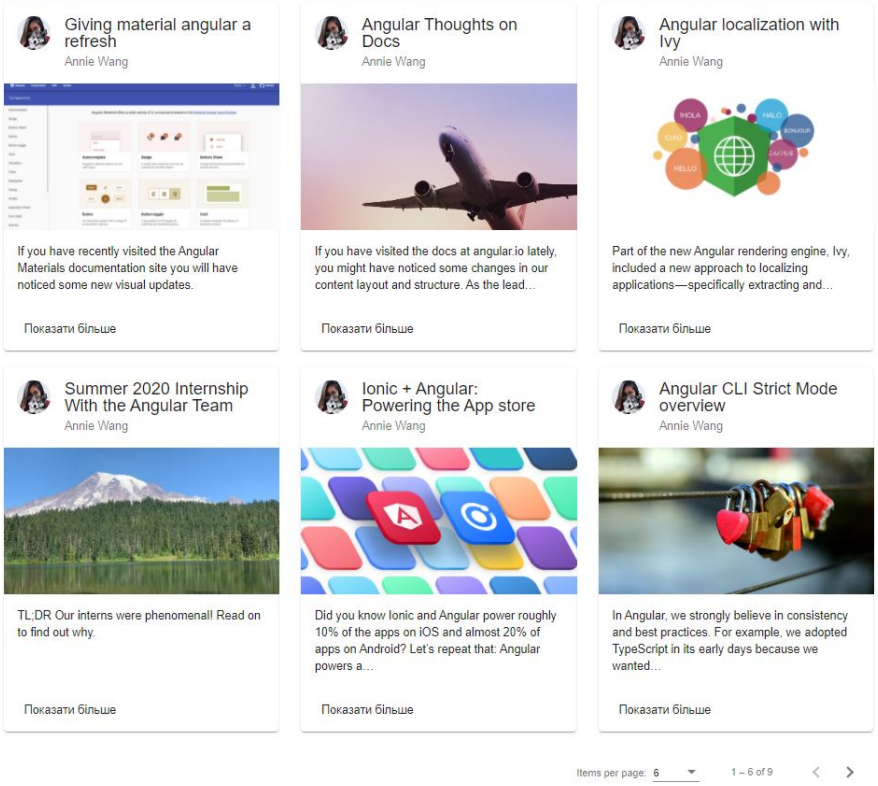

Рисунок 4.1 – Сторінка блогу

4.1.1 Angular

Структура модуля складається із одного датасету, двох компонентів, одного сервісу та головного модуля.

Підключення сторінок до головного модуля блога та задання їх маршрутів відбувається у класі BlogRoutingModule (див. рис. 4.2).

```
path: '',
component: BlogComponent,
children: [
    {path: '', component: BlogPostsComponent,},
    {path: ':id', component: BlogPostComponent}
]
```

Рисунок 4.2 – Маршрути блогу

Щоб завантажувати обраний пост, використовується параметр :id, який передається через директиву [routerLink]="'/blog/' + post.id".

Для отримання постів та конкретного посту по індентифікатору, був створений сервіс BlogService з двома методами (див. рис. 4.3).

```
getPosts() {
    return BlogPostsDataset;
}
getPostById(id: number) {
    return BlogPostsDataset.find(itm => itm.id === id);
}
```

Рисунок 4.3 – Методи сервісу BlogService

За зберігання постів відповідає масив blogPosts, який записує у себе дані з сервісу у методі ініціалізації ngOnInit.

Для реалізації пагінації на стороні клієнта, було створенно копію масива blogPosts під назвою blogPostsAll, щоб постійно мати повний список. А параметри пагінації зберігаються в об'єкті paginatorConfig, який має три параметри, а саме: perPage – відповідає за кількість постів на сторінці, total – містить у собі загальну кількість постів, currentPage – поточна сторінка.

Обробка події зміни поточної сторінки відбувається у методі onPageChange (див. рис. 4.4), а викликається вона у формі пагінації параметром (page).

```
onPageChange(event) {
    this.paginatorConfig.currentPage = event.pageIndex;
    this.paginatorConfig.perPage = event.pageSize;
    this.iterator();
}
```

Рисунок 4.4 – Метод обробки зміни сторінки

У приватному методі iterator() відбувається обробка даних та перерахунок постів під обрані параметри пагінації (див. рис. 4.5).

```
private iterator() {
    const end = (this.paginatorConfig.currentPage + 1) * this.paginatorConfig.perPage;
    const start = this.paginatorConfig.currentPage * this.paginatorConfig.perPage;
    const part = this.blogPostsAll.slice(start, end);
    this.blogPosts = part;
}
```

Рисунок 4.5 – Метод перерахунку постів

У HTML пости виводяться за допомогою директиви *ngFor (див. рис. 4.6).

```
<mat-card class="example-card" *ngFor="let post of blogPosts">
    <mat-card-header>
        <div mat-card-avatar class="example-header-image"
            [ngStyle]="{ 'background-image': 'url(' + post.author_photo + ')'}"></div>
        <mat-card-title>{{post.title}}</mat-card-title>
        <mat-card-subtitle>{{post.author_name}}</mat-card-subtitle>
    </mat-card-header>
    <img mat-card-image [src]="post.thumbnail_image">
    <mat-card-content>
        <p>{{post.description}}</p>
    </mat-card-content>
    <mat-card-actions>
        <button mat-button [routerLink]="'/blog/' + post.id">{{'blog-btn-more' | transloco}}</button>
    </mat-card-actions>
</mat-card>
```

Рисунок 4.6 – Перебір та вивід постів

Для переходу на конкретний пост, передається параметр id, по якому відбувається пошук у датасеті.

Для зберігання поточного посту було створено об'єкт post, в який записується елемент масиву BlogPostsDataset.

Ініціалізація даних вдбувається у методі getPostById, в якому береться параметр id із маршруту та по ньому записує знайдений об'єкт із сервісу BlogService в змінну post (див. рис. 4.7).

```
ngOnInit(): void {
    this.getPostById();
}
getPostById() {
    const postId = +this.route.snapshot.paramMap.get('id');
    this.post = this.blogService.getPostById(postId);
}
```

Рисунок 4.7 – Знаходження конкретного посту

4.1.2 Vue.js

Структура модуля складається із одного датасету, одного сервісу та двох компонентів.

Підключення сторінок до головного модуля проекту та задання їх маршрутів відбувається у класі VueRouter (див. рис. 4.8).

```
{
    path: "/blog", name: "Blog",
    component: () => import(/* webpackChunkName: "blog" */ "../modules/blog/Blog.vue")
},
{
    path: "/blog/:id", name: "BlogPost",
    component: () => import(/* webpackChunkName: "blog-post" */ "../modules/blog/pages/BlogPost.vue")
},
```

Рисунок 4.8 – Маршрути блогу

Щоб відобразити обраний пост, використовується параметр :id, який передається через компонент <router-link> та його параметр v-bind:to="'/blog/' + post.id".

Для отримання постів та конкретного посту по індентифікатору використовується датасет BlogPostsDataset.

За зберігання постів відповідає геттер collection, який повертає масив BlogPostsDataset у методі computed.

Для реалізації пагінації на стороні клієнта, було створенно метод paginate, який повертає перерахований масив постів (див. рис. 4.9).

```
paginate(data) {
    return _.slice(
        data,
        this.pagination.startIndex,
        end: this.pagination.endIndex + 1
    );
},
```

Рисунок 4.9 – Метод перерахування постів

Об'єкт data має три змінні, а саме: perPage – відповідає за кількість постів на сторінці, perPageList – містить у собі загальну кількість фільтрів, pagination – об'єкт пагінації.

Обробка події зміни поточної сторінки відбувається у методі setPage (див. рис. 4.10), який викликає метод paginator, для обчислення сторінок по обраним параметрам об'єкта data, а викликається він у методі ініціалізації created.

```
paginator(total, currentPage) {
    const startIndex = (currentPage - 1) * this.perPage,
        endIndex = Math.min( values: startIndex + this.perPage - 1, total - 1);

    return {
        currentPage: currentPage,
        startIndex: startIndex,
        endIndex: endIndex,
        pages: _.range( start: 1,  end: Math.ceil( x: total / this.perPage) + 1)
    };
}
```

Рисунок 4.10 – Метод обчислення пагінації

У HTML пости виводяться за допомогою директиви v-for (див. рис. 4.11).

```
<md-card class="example-card" v-for="post in collection" :key="post.id">
    <md-card-header>
        <md-card-media class="md-card-avatar">
            <img v-bind:src="post.author_photo" alt="Author"/>
        </md-card-media>
        <md-card-header-text>
            <div class="md-title">{{ post.author_name }}</div>
            <div class="md-subhead">{{ post.title }}</div>
        </md-card-header-text>
    </md-card-header>
    <md-card-media>
        <img v-bind:src="post.thumbnail_image" alt="thumbnail_image"/>
    </md-card-media>
    <md-card-content>
        {{ post.description }}
    </md-card-content>
    <md-card-actions>
        <router-link v-bind:to="'/blog/' + post.id">
            <md-button>{{ $t('blog-btn-more') }}</md-button>
        </router-link>
    </md-card-actions>
</md-card>
```

Рисунок 4.11 – Перебір та вивід постів

Для переходу на конкретний пост, передається параметр id, по якому відбувається пошук у датасеті, а для зберігання поточного посту було створено об'єкт post, в який записується елемент масиву BlogPostsDataset.

Ініціалізація даних вдбувається у методі getPostById, в якому береться параметр id із маршруту та по ньому записує знайдений об'єкт у BlogPostsDataset у змінну post (див. рис. 4.12).

```
data: () => ({
    post: {}
}),
methods: {
    getPostById() {
        this.post = BlogService.getPostById(this.$route.params.id);
    }
},
created() {
    this.getPostById();
}
```

Рисунок 4.12 – Знаходження конкретного посту

4.1.3 React

Структура модуля складається із одного датасету, одного сервісу та двох компонентів.

Підключення сторінок до головного модуля проекту та задання їх маршрутів відбувається у класі App (див. рис. 4.13).

```
<Route path="/blog/:id"
       location={props.location}
       render={({location, match}) => (
           <BlogPost match={match} />
       )}>
</Route>
<Route path="/blog">
    <Blog/>
</Route>
```

Рисунок 4.13 – Маршрути блогу

Щоб відобразити обраний пост, використовується параметр :id, який передається через компонент <Route> параметру location в об'єкт props.

Для отримання постів та конкретного посту по індентифікатору використовується сервіс BlogService з двома методами (див. рис. 4.14).

```
class BlogService {
    get allPosts() {
        return BlogPostsDataset;
    }
    getPostById(id) {
        return BlogPostsDataset.find(itm => itm.id == id);
    }
}
export default new BlogService();
```

Рисунок 4.14 – Сервіс даних BlogService

За зберігання постів відповідає об'єкт blogPosts, який є властивістю об'єкта стану this.state, а ініціалізація даних відбувається у методі componentDidMount (див. рис. 4.15).

```
componentDidMount() {
    const posts = BlogService.allPosts;
    this.setState( state: {
        blogPostsAll: posts,
        total: posts.length,
        currentPage: 0,
        perPage: 6,
    }, callback: () => {
        this.iterator();
    })
}
```

Рисунок 4.15 – Ініціалізація даних

Для реалізації пагінації на стороні клієнта, було створенно метод iterator, який перераховує масив постів (див. рис. 4.16).

```
iterator() {
    const end = (this.state.currentPage + 1) * this.state.perPage;
    const start = this.state.currentPage * this.state.perPage;
    this.setState( state: {
        blogPosts: this.state.blogPostsAll.slice(start, end)
    })
}
```

Рисунок 4.16 – Метод перерахування постів

Об'єкт state також має три змінні, а саме: perPage – відповідає за кількість постів на сторінці, total – містить у собі загальну кількість постів, currentPage – поточна сторінка.

Обробка події зміни параметру кількості відображення постів відбувається у методі onPerPageChange (див. рис. 4.17).

```
onPerPageChange = (event) => {
    this.setState( state: {
        perPage: parseInt(event.target.value, radix: 10),
    }, callback: () => {
        this.iterator();
    });
}
```

Рисунок 4.17 – Метод обробки зміни параметру кількості відображення

Обробка події зміни поточної сторінки відбувається у методі onPageChange (див. рис. 4.18), який змінює значення perPage та викликає метод iterator.

```
onPageChange = (event, newPage) => {
    this.setState( state: {
        currentPage: newPage,
    }, callback: () => {
        this.iterator();
    })
}
```

Рисунок 4.18 – Метод обробки зміни сторінки

За допомогою методу map в HTML виводяться пости (див. рис. 4.19).

```
{this.state.blogPosts.map((post :T , index :number ) => {
    return <Card className="card" key={index}>
        <CardHeader avatar={<Avatar aria-label="recipe" className="card-avatar"
                        src={post.author_photo}></Avatar>}
                    title={post.author_name} subheader={post.created_at}/>
        <img className="image" src={post.thumbnail_image} alt=""/>
        <Typography variant="body2" color="textSecondary" component="p">
            {post.description}
        </Typography>
        <Button size="small" color="primary" href={`/blog/${post.id}`}>
            {this.props.t('blog-btn-more')}
        </Button></Card>})}
```

Рисунок 4.19 – Перебір та вивід постів

Для переходу на конкретний пост, передається параметр id, по якому відбувається пошук у сервісі BlogService.

Ініціалізація даних відбувається у методі getPostById, в який передається параметр id із об'єкту props та по ньому записує знайдений об'єкт у змінну post (див. рис. 4.20).

```
componentDidMount() {
    this.setState( state: {
        post: BlogService.getPostById(this.props.match.params.id)
    })
}
```

Рисунок 4.20 – Знаходження конкретного посту

4.1.4 Результати аналізу

Результати аналізу компонентів блогу з використанням Angular, Vue.js та React можна побачити на таблиці 4.1.

Таблиця 4.1 – Порівняння показників компонентів блогу у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Середня	Легка
Кількість файлів компонентів	13	5	6
Розмір файлів компонентів	184 кб	140 кб	136 кб
Швидкість завантаження сторінки	8 мс	6 мс	6 мс
Розмір модулю	141 кб	122 кб	115 кб

Завершуючи аналіз, можна зробити висновок, що найскладнішим та найдовшим рішенням є розробка цього компоненту на Vue.js, через те, що у фреймворка відсутній модуль material пагінації. Через це, потрібно більше витрачати часу на розробку.

Angular має найбільші розміри файлів та компонентів, а також найменші показники швидкості завантаження сторінки. Але все це зумовлено тим, що фреймворк має додаткові файли для тестування компонентів та інтегрований TypeScript.

React перемагає як у швидкості та розмірах файлів, так і у простоті написання коду. Через це бібліотека від Facebook краще всіх підходить для розробки такого компонента.

4.2 Візуалізація даних

Основними компонентами сайту для відображення складних даних чи інформаціїї є графіки.

В даному випадку, графік відображає курс гривні, який береться з архівів курсів валют ПриватБанку по API (див. рис. 4.21, рис. 4.22).

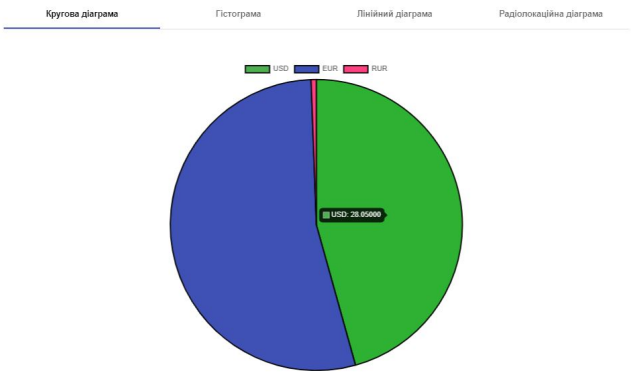

Рисунок 4.21 – Кругова діаграма курсу гривні

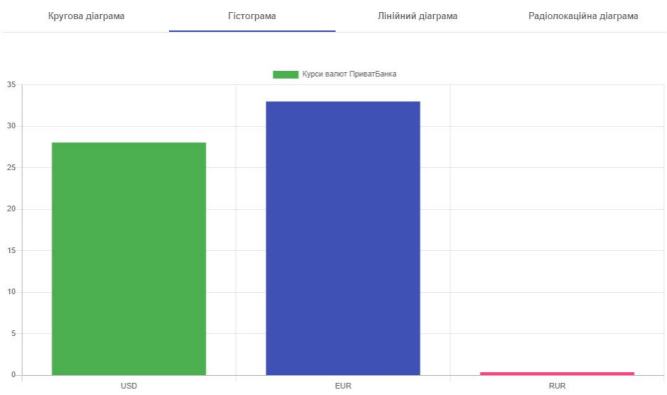

Рисунок 4.22 – Гістограма курсу гривні

4.2.1 Angular

Структура модуля складається із одного датасету, чотирьох компонентів, одного сервісу та головного модуля.

Імпорт та декларація компонентів відбувається у головному модулі visualization.module.ts.

Для роботи з графіками для Angular є обгортка бібліотеки Chart.js. Встановити її можна за допомогою пакетного менеджера npm виконав команду npm install --save ng2-charts.

Підключення бібліотеки до модулю візуалізації відбувається у класі VisualizationModule, де її модуль прописується у параметр imports.

Для отримання курсів валют був створений сервіс даних VisualizationService з одним методом getRates, змінною apiUrl та класом HttpClient (див. рис. 4.23).

```
constructor(
    private http: HttpClient,
) {
    this.apiUrl = 'https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=5';
}
getRates(): Observable<any> {
    return this.http.get(`${this.apiUrl}`);
}
```

Рисунок 4.23 – Структура сервісу VisualizationService

Відображення графіків відбувається у html-елементі canvas (див. рис. 4.24), який приймає наступні параметри:

– datasets: масив даних;
– labels: масив ярликів;
– options: об'єкт з налаштуваннями;
– chartType: тип графіку;

```
<canvas baseChart *ngIf="loaded"
        [datasets]="chartData"
        [labels]="chartLabels"
        [options]="{responsive: true}"
        [chartType]="'pie'">
</canvas>
```

Рисунок 4.24 – HTML структура графіку

Обробка даних для візуалізації відбувається у методі ngOnInit (див. рис. 4.25), який викликається при загрузкі компоненту, а ініціалізація відбувається по флагу loaded.

```
ngOnInit() {
    this.rateData.forEach(itm => {
        this.chartLabels.push(itm.ccy);
        this.chartData[0].data.push(itm.buy);
    })
    this.loaded = true;
}
```

Рисунок 4.25 – Обробка даних для візуалізації

Процес переключення типу графіків відбувається у html-компоненті <mat-tab-group></mat-tab-group> (див. рис. 4.26), який містить у собі чотири компоненти в які передається параметром @Input() отриманні дані курсів зі змінної rateData.

```
<mat-tab-group *ngIf="rateData.length">
    <mat-tab label="{{'visualization-pie' | transloco}}">
        <app-chart-pie [rateData]="rateData"></app-chart-pie>
    </mat-tab>
    <mat-tab label="{{'visualization-bar' | transloco}}">
        <app-chart-bar [rateData]="rateData"></app-chart-bar>
    </mat-tab>
    <mat-tab label="{{'visualization-line' | transloco}}">
        <app-chart-line [rateData]="rateData"></app-chart-line>
    </mat-tab>
    <mat-tab label="{{'visualization-radar' | transloco}}">
        <app-chart-radar [rateData]="rateData"></app-chart-radar>
    </mat-tab>
</mat-tab-group>
```

Рисунок 4.26 – Вкладки графіків

4.2.2 Vue.js

Структура модуля складається із чотирьох компонентів, одного сервісу та головного модуля.

Імпорт та декларація компонентів відбувається у головному модулі Visualization через директиву import.

Для роботи з графіками у Vue також використовує бібліотеку Chart.js. Встановити її можна за допомогою пакетного менеджера npm виконав команду npm install --save vue-chartjs.

Підключення бібліотеки до модулю візуалізації відбувається у файлі main.js, де її модуль прописується через метод Vue.use().

Для отримання курсів валют використовується сервіс даних VisualizationService з одним методом getRates, змінною API_URL та класом axios (див. рис. 4.27).

```
import axios from "axios";
const API_URL = "https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=5";
class VisualizationService {
    getRates() {
        return axios.get( url: `${API_URL}`);
    }
}
export default new VisualizationService();
```

Рисунок 4.27 – Структура сервісу VisualizationService

Відображення графіків відбувається у методі renderChart, який викликається при завантажені методу mounted (див. рис. 4.28), а дані зберігаються у параметрах labels та datasets.

```
this.props.rateData.forEach(itm => {
    labels.push(itm.ccy);
    data.push(itm.sale);
})
this.renderChart(
    {
        labels: labels,
        datasets: [
            {
                label: "Курси валют ПриватБанка",
                backgroundColor: "#f87979",
                data: data
            }
        ]
    },
    {responsive: true, maintainAspectRatio: false}
);
```

Рисунок 4.28 – Обробка даних для візуалізації та рендеринг сторінки

Процес переключення типу графіків відбувається у html-компоненті <md-tabs></ md-tabs>, який містить у собі чотири компоненти в які передається атрибутом :rateData отриманні дані курсів зі змінної rateData.

4.2.3 React

Структура модуля складається із чотирьох компонентів, одного сервісу та головного модуля.

Імпорт та декларація компонентів відбувається у головному компоненті Visualization через директиву import.

Для роботи з графіками у React використовується нативна версія бібліотеки Chart.js, яка не має обгортки для JSX. Встановити її можна за допомогою пакетного менеджера npm виконав команду npm install --save chart.js.

Бібліотека імпортується до компоненту візуалізації через import.

Для отримання курсів валют використовується сервіс даних VisualizationService з одним асинхронним методом getRates та змінною API_URL (див. рис. 4.29).

```
const API_URL = "https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=5";
class VisualizationService {
    async getRates() {
        const response = await fetch(API_URL);
        return await response.json();
    }
}
export default new VisualizationService();
```

Рисунок 4.29 – Структура сервісу VisualizationService

Обробка даних візуалізації відбувається у конструкторі класу, який ініціалізую дані у параметрах labels та datasets об'єкту state (див. рис. 4.30).

```
constructor(props) {
    super(props);
    const labels = [];
    const data = [];
    props.rateData.forEach(itm => {
        labels.push(itm.ccy);
        data.push(itm.sale);
    })
    this.state = {
        data: {
            labels: labels,
            datasets: [{
                data: data,
                backgroundColor: ['#FF6384', '#36A2EB', '#FFCE56'],
                hoverBackgroundColor: ['#FF6384', '#36A2EB', '#FFCE56']
            }]
        }
    }
}
```

Рисунок 4.30 – Обробка даних для візуалізації

4.2.4 Результати аналізу

Результати аналізу компонентів візуалізації з використанням Angular, Vue.js та React можна побачити на таблиці 4.2.

Таблиця 4.2 – Порівняння показників компонентів візуалізації у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Легка	Середня
Кількість файлів компонентів	22	7	8
Розмір файлів компонентів	100 кб	36 кб	36 кб
Швидкість завантаження сторінки	55 мс	48 мс	51 мс
Розмір модулю	2.6 мб	2.2 мб	2.4 мб

Завершуючи аналіз, можна зробити висновок, що найскладнішим та найдовшим рішенням є розробка цього компоненту на React, через те, що у бібліотеки відсутня обгортка для роботи із графіками. Через це, потрібно більше витрачати час на розробку такої обгортки, рендерингу її елементів та компонентів.

Angular має найбільші розміри файлів та компонентів, а також найменші показники швидкості завантаження сторінки. Але він має потужну обгортку у виді бібліотеки, що досить сильно полегшує розробку компонентів.

Vue та React мають однакові показники швидкості та розмірів файлів, але по простоті написання коду, Vue краще. Через це розробка компонентів візуалізації на цьому фреймворку дає більше переваг ніж на бібліотеці від Facebook.

4.3 Локалізація

Найпопулярнішим методом для збільшення аудиторії сайту є наявність компоненту багатомовності. Такий компонент асинхронно оновлює переклад контенту сайту на обрану мову через спеціальне меню (див. рис. 4.31) без перезавантаження сторінки.

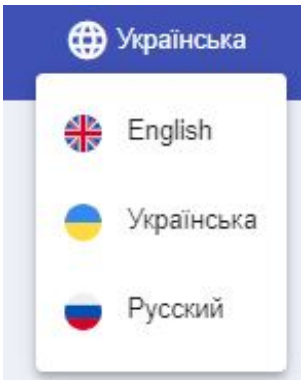

Рисунок 4.31 – Меню вибору мови сайту

4.5.1 Angular

Структура модуля складається із одного сервісу, компонента, моделі, датасету, трьох JSON файлів та головного модуля.

Для реалізації багатомовності, Angular має модуль TranslocoModule, який йде в комплекті бібліотеки ng @ngneat/transloco.

Потрібний файл переводів завантажується у методі getTranslation класу TranslocoHttpLoader (див. рис. 4.32), який бере код мови із локального сховища браузера через метод i18nInitializer (див. рис. 4.33).

```
@Injectable({providedIn: 'root'})
export class TranslocoHttpLoader implements TranslocoLoader {
    private baseUrl = environment.api.baseUrl;
    constructor(private http: HttpClient) {}
    getTranslation(lang: string) {
        return this.http.get( url: `${this.baseUrl}/app/modules/localization/translations/${lang}.json`);
    }
}
```

Рисунок 4.32 – Метод завантаження перекладу

```
export function i18nInitializer(router: Router, transloco: TranslocoService) {
    return () => {
        let lang = window.localStorage.getItem( key: 'lang') || 'en';
        transloco.setActiveLang(lang);
        return transloco.load(lang).toPromise();
    };
}
```

Рисунок 4.33 – Метод отримання коду мови

Для перемикання мови, був створений компонент LangMenuComponent, який має масив мов languages типу Language та змінну currentLang.

Ініціалізація мови відбувається у методі initLocalization, а процес вибору у setLang, який приймає параметром об'єкт масиву LanguagesDataset (див. рис. 4.34).

```
initLocalization() {
    this.languageService.detectLang();
}
setLang(language: Language) {
    this.currentLang = language;
    this.languageService.setLang(this.currentLang.code);
}
```

Рисунок 4.34 – Методи ініціалізації та вибору мови

Меню виводиться через директиву *ngFor, а по натисненю на кнопку, передається об'єкт масиву language до методу setLang (див. рис. 4.35).

```
<button mat-menu-item *ngFor="let language of languages" (click)="setLang(language)">
    <img src="assets/images/icons/{{language.code}}.svg" [alt]="language.name">
    {{language.name}}
</button>
```

Рисунок 4.35 – Вивід елементів масиву в меню

Запис до локального сховища браузеру відбувається у сервісі LanguageService через метод localStorage.setItem('lang', this.lang).

Для отриманя коду поточної мови, було створено геттер lang() рядкового типу, код якого має такий вигляд: return this.translate.getActiveLang().

Щоб виконати переклад, потрібно у шаблоні додати конструкцію {{'menu-link-1' | transloco}}, де пайп transloco приймає ключ перекладу, по якому буде підставлятися потрібне значення.

4.3.2 Vue.js

Структура модуля складається із одного сервісу, компонента, моделі, датасету, плагіну, трьох JSON файлів та головного модуля.

За багатомовність у Vue відповідає плагін i18n.js, який йде в комплекті бібліотеки vue-i18n.

Потрібний файл переводів завантажується у методі loadLocaleMessages класу VueI18n (див. рис. 4.36), який бере переклади із папки translations та перебирає їх методом forEach.

```
function loadLocaleMessages() {
    const locales = require.context(
        "./translations",
        true,
        /[A-Za-z0-9-_,\s]+\.json$/i
    );
    const messages = {};
    locales.keys().forEach(key => {
        const matched = key.match(/([A-Za-z0-9-_]+)\./i);
        if (matched && matched.length > 1) {
            const locale = matched[1];
            messages[locale] = locales(key);
        }
    });
    return messages;
}
```

Рисунок 4.36 – Метод завантаження перекладу

Щоб перемикати мови, був створений компонент LangMenu, який має масив мов languages із LanguagesDataset та змінну currentLang.

Ініціалізація мови відбувається у методі initLocalization, а процес вибору у setLang, який приймає параметром об'єкт масиву LanguagesDataset (див. рис. 4.37).

```
initLocalization() {
    LanguageService.detectLang();
},
setLang(language) {
    this.currentLang = language;
    LanguageService.setLang(this.currentLang.code);
    this.$i18n.locale = this.currentLang.code;
},
```

Рисунок 4.37 – Методи ініціалізації та вибору мови

Меню виводиться через директиву v-for, а по натисненю на кнопку через @click, передається об'єкт масиву language до методу setLang (див. рис. 4.38).

```
<md-menu-item v-for="language in languages" :key="language.code" @click="setLang(language)">
    <img v-bind:src="require('../../../assets/images/icons/' + language.code + '.svg')" alt="lang"/>
    {{ language.name }}
</md-menu-item>
```

Рисунок 4.38 – Вивід елементів масиву в меню

Для запису у локальне сховище браузеру був створений сервіс LanguageService, який містить геттер знаходження поточної мови.

Щоб виконати переклад, потрібно у шаблоні додати конструкцію {{ $t('menu-link-1') }}, де метод $t приймає ключ перекладу, по якому буде підставлятися потрібне значення.

4.3.3 React

Структура модуля складається із одного сервісу, компонента, моделі, датасету, трьох JSON файлів та головного модуля.

За багатомовність у React відповідає клас i18n, який йде в комплекті бібліотеки i18next.

Потрібний файл перекладів завантажується у константу resources, яка ініціалізується у класі i18n (див. рис. 4.39).

```
const resources = {
    en: {translation: translationEN},
    ru: {translation: translationRU},
    uk: {translation: translationUK}
};
i18n
    .use(reactI18nextModule)
    .init( options: {
        resources,
        lng: "uk",
        fallbackLng: "en",
        keySeparator: false,
        interpolation: {
            escapeValue: false
        }
    });
export default i18n;
```

Рисунок 4.39 – Метод завантаження перекладу

Щоб перемикати мови, був створений компонент LangMenu, який має менеджер стану state, де в якості параметрів задається масив мов languages із LanguagesDataset та об'єкт currentLang.

Ініціалізація мови відбувається у методі storeLanguages (див. рис. 4.40), в якому йде звернення до сервісу та класу i18n. Там же йде зміна стану масива мов та об'єкту поточної мови.

```
storeLanguages() {
    LanguageService.detectLang();
    i18n.changeLanguage(LanguageService.lang).then();
    this.setState( state: {
        languages: LanguagesDataset,
        currentLang: LanguagesDataset.find( predicate: itm => itm.code === LanguageService.lang)
    });
}
```

Рисунок 4.40 – Методи ініціалізації мови

Процес вибору мови відбувається у методі changeLanguage, який приймає параметром об'єкт масиву LanguagesDataset (див. рис. 4.41) та записує в стан об'єкту currentLang обрану мову. Клас i18n за допомогою Callback Promise змінює стан поточної мови та завантажує потрібний файл перекладу.

```
changeLanguage = (lang :null  = null) => {
    this.handleClose();
    if (lang) {
        this.setState( state: {
            currentLang: LanguagesDataset.find( predicate: itm => itm.code === lang)
        });
        i18n.changeLanguage(lang).then();
        LanguageService.setLang(lang);
    }
};
```

Рисунок 4.41 – Методи зміни мови

Для запису у локальне сховища браузеру використовується сервіс LanguageService, який містить геттер знаходження поточної мови та ініціалузацію мови браузеру.

Меню виводиться через метод масиву map, а по натисненю на кнопку через onClick, передається об'єкт масиву language до методу changeLanguage (див. рис. 4.42).

```
<Menu id="simple-menu" anchorEl={this.state.anchorEl} keepMounted
    open={Boolean(this.state.anchorEl)} onClose={this.handleClose}>
    {this.state.languages.map((lang, index) => {
        return <MenuItem key={index} onClick={() => this.changeLanguage(lang.code)}>
            <img className="icon" src={require('../../../assets/images/icons/' + lang.code + '.svg')}/>
            {lang.name}
        </MenuItem>
    })}
</Menu>
```

Рисунок 4.42 – Вивід елементів масиву в меню

Щоб виконати переклад, потрібно у шаблоні додати конструкцію {t('btn-show')}, де метод t приймає ключ перекладу, по якому буде підставлятися потрібне значення.

4.3.4 Результати аналізу

Результати аналізу компонентів локалізації з використанням Angular, Vue.js та React можна побачити на таблиці 4.3.

Таблиця 4.3 – Порівняння показників компонентів локалізації у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Легка	Легка
Кількість файлів компонентів	15	9	9
Розмір файлів компонентів	88 кб	64 кб	56 кб
Швидкість завантаження модуля	7 мс	5 мс	5 мс
Розмір модуля	42 кб	31 кб	29 кб

Завершуючи аналіз, можна зробити висновок, що розробка цього модулю на даних технологіях майже не відрізняється, а відхилення у показниках мінімальні.

Слід зазначити, що Angular має найбільші розміри файлів та найменшу швидкість завантаження.

Vue має середні розміри файлів та високу швидкість завантаження файлів модулю.

Серед усіх технологій, React має найвищі показники в швидкості та найменші розміри файлів, що робить його кращим вибором для розробки компонентів локалізації.

4.4 Чат

Популярним компонентом для для спілкування є чат. Він має функції асинхронного оновлення повідомлень та їх відправку (див. рис. 4.43).

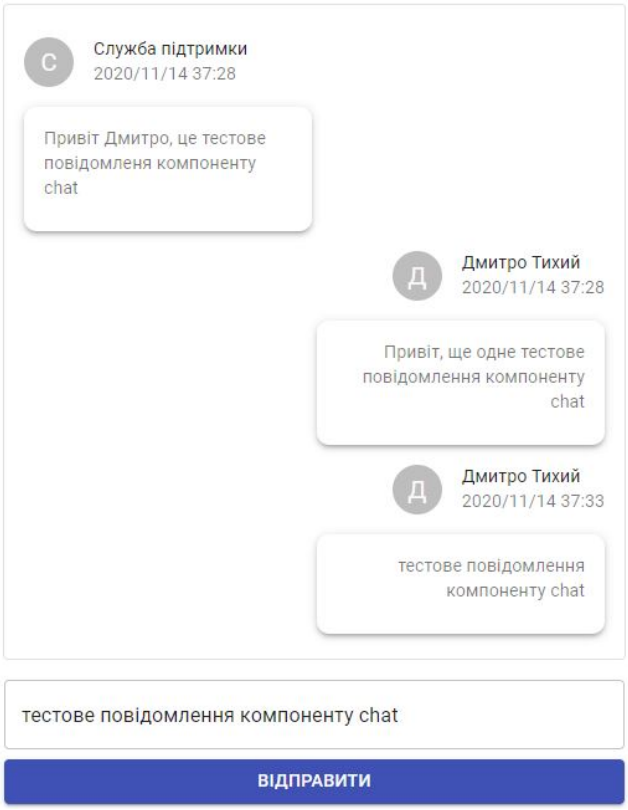

Рисунок 4.43 – Інтерфейс чату

4.4.1 Angular

Структура модуля складається із одного датасету, компоненту, моделі, сервісу та головного модуля.

Підключення сторінки чату до головного модуля відбувається у класі ChatModule через додання її у масив declarations.

За відправку та отримання повідомлень відповідає сервіс ChatService, який складається із двох Observable http методів (див. рис. 4.44).

```
send(params: {}): Observable<Message[]> {
    return <Observable<Message[]>>this.http.post(`${this.API_URL}/send`, params);
}

receive(): Observable<Message[]> {
    return <Observable<Message[]>>this.http.get(`${this.API_URL}/receive`);
}
```

Рисунок 4.44 – Методи відправки та отримання повідомлень

Клас ChatComponent має масив повідомлень messages типу Message, рядкову змінну text та змінну subscription типу Subscription.

Оновлення даних відбувається за допомогою змінної subscription, яка ініціалізується у методі ngOnInit і за допомогою Observable функції interval, робить виклик методу getMessages та відсилає запити до серверу кожні п'ять секунд (див. рис. 4.45).

```
ngOnInit(): void {
    this.subscription = interval( period: 5000)
        .subscribe( next: () => this.getMessages());
}
getMessages() {
    this.chatService.receive()
        .subscribe( next: response => this.messages = response)
}
```

Рисунок 4.45 – Ініціалізація та отримання повідомлень

За відправку даних до серверу відповідає метод addMessage, який викликає метод send сервісу ChatService та передає туди змінну text у якості параметру (див. рис. 4.46). А зв'язування змінної text з елементом вводу input відбувається за допомогою директиви [(ngModel)].

```
sendMessage() {
    this.chatService.send( params: {message: this.text})
        .subscribe( next: response => this.messages = response)
}
```

Рисунок 4.46 – Метод відправки повідомленя

Вивід повідомлень у шаблон відбувається через директиву *ngFor, яка ітерує масив messages та передає дані до компоненту <app-chat-message></app-chat-message> через декоратори @Input() (див. рис. 4.47).

```
<app-chat-message [@fade]
        *ngFor="let message of messages"
        [author]="message.author"
        [content]="message.content"
        [date]="message.date"
        [isSender]="message.isSender"
></app-chat-message>
```

Рисунок 4.47 – Ітерація масиву та передача даних до компоненту

За анімацію появи повідомлень відповідає директива [@fade], яка використовує вбудовані методи trigger, transition, style, animate, state класу Animate (див. рис. 4.48).

```
animations: [
    trigger( name: 'fade', definitions: [
        transition( stateChangeExpr: ':enter', steps: [
            style( tokens: {opacity: '0'}),
            animate( timings: 1000)
        ]),
        state( name: '*', style( tokens: {opacity: '1'})),
    ])
]
```

Рисунок 4.48 – Логіка анімації появи повідомлень

4.4.2 Vue.js

Структура модуля складається із одного датасету, двох компонентів та сервісу.

Імпорт компонентів відбувається у класі Chat за допомогою інструкції import, а відправка та отримання повідомлень відбувається у методах сервісу даних ChatService, які використовують біліотеку http запитів axios (див. рис. 4.49).

```
send(params) {
    return axios.post( url: `${API_URL}/send`, params);
}
receive() {
    return axios.get( url: `${API_URL}/receive`);
}
```

Рисунок 4.49 – Методи відправки та отримання повідомлень

Метод data компоненту Chat має масив messages датасету MessagesDataset і змінну text.

Оновлення даних відбувається за допомогою функції setInterval, яка викликається у методі getMessagesi та робить запити до серверу кожні п'ять секунд (див. рис. 4.50).

```
getMessages() {
    setInterval( handler: () => {
        ChatService.receive()
            .then(response => this.messages = response)
    }, timeout: 5000);
},
```

Рисунок 4.50 – Асинхронне отримання повідомлень

Відправка даних до серверу відбувається у методі sendMessage, який робить виклик методу send сервісу ChatService і передає в параметр message змінну text (див. рис. 4.51). А за допомогою директиви v-model, відбувається зв'язування змінної text з елементом вводу input.

```
sendMessage() {
    ChatService.send( params: {message: this.text})
        .then(response => this.messages = response)
},
```

Рисунок 4.51 – Метод відправки повідомленя

Ітерація повідомлень відбувається у шаблоні класу Chat за допомогою директиви v-for, де об'єкти масиву messages передаються до компоненту <ChatMessage/> у вигляді параметрів props (див. рис. 4.52)

```
<transition-group name="fade">
    <ChatMessage v-for="(message, index) in messages"
                 :key="index"
                 :author="message.author"
                 :content="message.content"
                 :date="message.date"
                 :isSender="message.isSender"/>
</transition-group>
```

Рисунок 4.52 – Ітерація масиву та передача параметрів props

За анімацію появи повідомлень відповідає компонент <transition-group/></transition-group>, який використовує keyframes стилі через атрибут name (див. рис. 4.53).

```
.fade-enter-active {
    animation: fade-in .5s;
}
.fade-leave-active {
    animation: fade-in .5s reverse;
}
@keyframes fade {
    0% {opacity: 0;}
    50% {opacity: 0.5;}
    100% {opacity: 1;}
}
```

Рисунок 4.53 – Стилі анімації появи повідомлень

4.4.3 React

Структура модуля складається із одного датасету, двох компонентів та сервісу.

За допомогою інструкції import відбувається декларація компонентів у класі Chat.

За відправку та отримання повідомлень відповідають асинхронні методи сервісу даних ChatService, які використовують вбудоване http api fetch (див. рис. 4.54).

```
async send(params) {
    const response = await fetch( input: `${API_URL}/send`, init: {
        method: 'POST',
        body: JSON.stringify(params)
    });
    return await response.json();
}
async receive() {
    const response = await fetch( input: `${API_URL}receive`, init: {
        method: 'GET',
    });
    return await response.json();
}
```

Рисунок 4.54 – Методи відправки та отримання повідомлень

Менеджер стану state компоненту Chat має масив messages датасету MessagesDataset і змінну text.

За допомогою функції setInterval, відбувається асинхронне оновлення даних через виклик методу getMessagesi, який робить запити до серверу кожні п'ять секунд (див. рис. 4.55).

```
getMessages = () => {
    setInterval( handler: () => {
        ChatService.receive().then(response => {
            this.setState( state: {
                messages: response
            })
        })
    }, timeout: 5000);
}
```

Рисунок 4.55 – Асинхронне отримання повідомлень

За відправку даних до серверу відповідає метод sendMessage, який викликає метод send сервісу ChatService і передає в якості параметру message змінну менеджер стану text (див. рис. 4.56). А параметр defaultValue зв'язується із змінною text у компоненті <TextField/>.

```
sendMessage = () => {
    ChatService.send( params: {message: this.state.text}).then(response => {
        this.setState( state: {
            messages: response
        })
    })
}
```

Рисунок 4.56 – Метод відправки повідомленя

За допомогою методу ітерації map, відбувається перебір об'єктів масиву message, а передача даних виконується у вигляді параметрів props (див. рис. 4.57)

```
<TransitionGroup transitionName="show" transitionEnterTimeout={500}
                 transitionLeaveTimeout={300}>
    {this.state.messages.map((message, index) => {
        return <ChatMessage
            key={index}
            author={message.author}
            content={message.content}
            date={message.date}
            isSender={message.isSender}
        />
    })}
</TransitionGroup>
```

Рисунок 4.57 – Ітерація масиву та передача параметрів props

4.4.4 Результати аналізу

Результати аналізу компонентів чату з використанням Angular, Vue.js та React можна побачити на таблиці 4.4.

Таблиця 4.4 – Порівняння показників компонентів чату у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Легка	Легка
Кількість файлів компонентів	13	4	5
Розмір файлів компонентів	72 кб	32 кб	32 кб
Швидкість завантаження модуля	6 мс	5 мс	5 мс
Розмір модуля	30.2 кб	29.1 кб	29 кб

Завершуючи аналіз, можна зробити висновок, що розробка цього модулю на Angular є найкращим рішенням, адже вбудована бібліотека rxjs та сервіси HTTP дають більш розширений функціонал, що значно полегшує розробку.

Vue має малі розміри файлів та високу швидкість завантаження модулю, але поступається у функціоналі http сервісу та асинхронних методів обробки даних.

React в свою чергу, також має високі показники параметрів, але відсутність двостороннього зв'язування ускладнює процес оновлення значень змінних.

4.5 Google карти

Основним компонентом для відображення місцезнаходження у сайтах є Google карти (див. рис. 4.58).

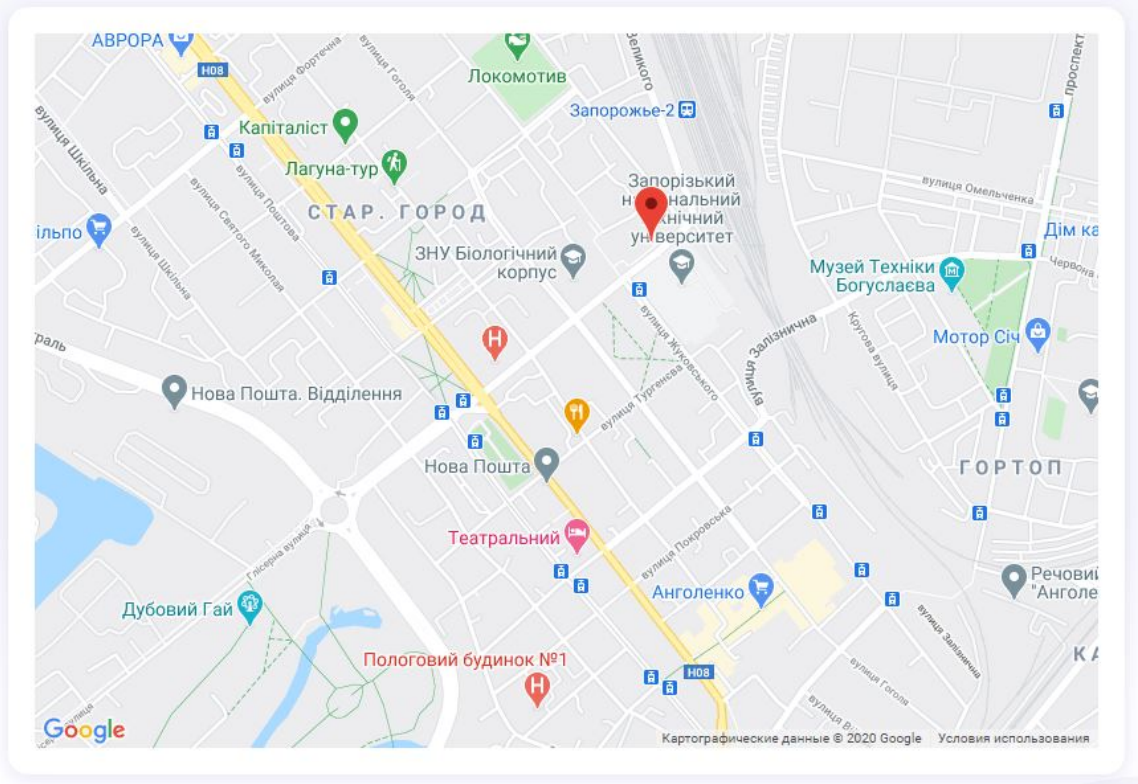

Рисунок 4.58 – Інтегрована Google карта

4.5.1 Angular

Структура модуля складається з одного головного модуля.

Для того, щоб побудувати карту, в Angular використовується вбудована бібліотека @angular/google-maps.

Декларація модуля карт відбувається у класі MapModule через додавання модуля GoogleMapsModule в параметр imports.

У класі MapComponent є об'єкт options типу google.maps.MapOptions, який відповідає за параметри налаштувань.

Ініціалізація карти відбувається у методі ngOnInit в якому викликається метод getCurrentPosition з потрібними параметрами координат об'єкту center (див. рис. 4.59).

```
ngOnInit(): void {
    navigator.geolocation.getCurrentPosition( successCallback: (position : Position ) => {
        this.center = {
            lat: position.coords.latitude,
            lng: position.coords.longitude,
        }
    })
}
```

Рисунок 4.59 – Ініціалізація карти

Для зберігання меркерів був створений масив markers, а для їх задання html-елемент button з параметром (click)="addMarker()".

Метод addMarker приймає два параметри цифрового типу: lat та lng. Вони додаються до масиву markers методом push та виводяться у шаблон ітеруємими компонентами <map-marker</map-marker> (див. рис. 4.60).

```
<google-map height="761" width="1035" [zoom]="zoom" [center]="center" [options]="options">
    <map-marker *ngFor="let marker of markers"
            [position]="marker.position" [label]="marker.label"
            [title]="marker.title" [options]="marker.options">
    </map-marker>
</google-map>
```

Рисунок 4.60 – Ініціалізація карти

4.5.2 Vue.js

Структура модуля складається з одного компонента.

Для побудуви карт, у Vue потрібно встановлювати сторонню бібліотеку google-maps-api-loader.

Імпорт бібліотеки карт відбувається у класі Map через import.

За параметри налаштувань відповідає об'єкт mapConfig, а за ключ доступу змінна apiKey.

Ініціалізація карти відбувається у асинхронному методі mounted в якому викликається метод initializeMap з потрібними параметрами координат об'єкту map (див. рис. 4.61).

```
async mounted() {
    const googleMapApi = await GoogleMapsApiLoader( params: {
        apiKey: this.apiKey
    })
    this.google = googleMapApi
    this.initializeMap()
},
methods: {
    initializeMap() {
        const mapContainer = this.$refs.googleMap
        this.map = new this.google.maps.Map(
            mapContainer, this.mapConfig
        )
    }
}
```

Рисунок 4.61 – Ініціалізація карти

У шаблоні відображається компонент GoogleMapLoader, який приймає дані, необхідні для ініціалізації карти. (див. рис. 4.62).

```
<template v-if="Boolean(this.google) && Boolean(this.map)">
    <GoogleMapLoader
        :mapConfig="mapConfig"
        apiKey="yourApiKey"
    />
</template>
```

Рисунок 4.62 – Компонент GoogleMapLoader

4.5.3 React

Структура модуля складається із одного компонента.

Побудува карт в React відбувається через сторонню бібліотеку react-google-maps.

Імпорт потрібних функцій відбувається у класі Map.

Ініціалізація карти відбувається через змінну Cmap, що приймає у себе компонент <GoogleMap></GoogleMap> (див. рис. 4.63).

```
CMap = withScriptjs(withGoogleMap( wrappedComponent: props =>
    <GoogleMap
        defaultZoom={8}
        defaultCenter={{ lat: -34.397, lng: 150.644 }}>
        {props.children}
    </GoogleMap>
));
```

Рисунок 4.63 – Ініціалізація карти

За відображення маркерів відповідає компонент <Marker/> з параметром position, а рендеринг карти відбувається у методі render (див. рис. 4.64).

```
<this.CMap
    googleMapURL={this.state.googleMapURL}
    loadingElement={<div style={{ height: `100%` }} />}
    containerElement={<div style={{ height: `700px` }} />}
    mapElement={<div style={{ height: `100%` }} />}
    center= {{ lat: 47.8186214, lng: 35.1809629 }}>
    <Marker position={{ lat: 47.8186214, lng: 35.1809629 }}/>
</this.CMap>
```

Рисунок 4.64 – Рендеринг карти

4.5.4 Результати аналізу

Результати аналізу компонентів Google карт з використанням Angular, Vue.js та React можна побачити на таблиці 4.5.

Таблиця 4.5 – Порівняння показників компонентів Google карт у Angular, Vue.js та React

Назва показника	Angular	Vue.js	React
Складність реалізації	Легка	Легка	Легка
Кількість файлів компонентів	5	1	1
Розмір файлів компонентів	20 кб	8 кб	8 кб
Швидкість завантаження модуля	4 мс	2 мс	2 мс
Розмір модуля	51.3 кб	32.5 кб	32.5 кб

Завершуючи аналіз, можна зробити висновок, що розробка цього модулю на Angular є найкращим рішенням, адже фреймворк від Google має вбудовану бібліотеку для роботи з картами та її повну підтримку.

Vue та React мають однакові розміри та схожі показники швидкості, але для роботи з картами, потрібно встановлювати сторонню бібліотеку.

ВИСНОВКИ

У першому розділі даної роботи були зібрані теоретичні відомості про JavaScript фреймворк та структуру проектів.

У другому розділі були розглянуті і проаналізовані всі використані технології.

У третьому розділі були проаналізовані реалізації компонентів авторизації та навігації на сайті.

У четвертому розділі роботи були проаналізовані реалізації типових компонентів роботи з даними.

У ході виконання роботи були досліджені та проаналізовані методи створення типових компонентів сайту. Проведено дослідження ефективності використання різних технологій для реалізації функціонального і готового до експлуатації web-компоненту. Для цього було проаналізовано переваги бібліотек та фреймворків, а також, їх засоби розробки.

Результати свідчать про те, що найкращим рішенням для розробки компонентів сайту є фреймворк Vue.js. Він має високі показники швидкості завантаження модулів та малі розміри файлів. Також підтримує інтеграцію мови TypeScript, що робить його універсальною технологією для розробки як малих, так і великих web-компонентів та додатків.

React в свою чергу не поступається у швидкості та розмірах файлів, але вбудований JSX та відсутність двостороннього зв'язування ускладнює розробку компонентів.

Angular має найгірші показники швидкості та розмірів файлів, що зумовлено інтегрованим TypeScript та великою кількістю залежностей, а також відсутністю віртуального DOM. Все це, робить фреймворк від Google непристосованим до розробки малих проектів та компонентів, але дає потужні інструменти для розробки та підтримки великих сайтів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wikipedia JavaScript — URL : https://wikipedia.org/wiki/JavaScript (дата звернення: 30.06.2020).

2. Metanit JavaScript — URL : https://metanit.com/web/javascript (дата звернення: 20.07.2020).

3. Habr JavaScript — URL : https://habr.com/ru/hub/javascript (дата звернення: 10.08.2020).

4. Офіційний сайт Angular — URL : https://angular.io (дата звернення: 20.09.2020).

5. Wikipedia Angular — URL : https://wikipedia.org/wiki/Angular (дата звернення: 20.09.2020).

6. Офіційний сайт TypeScript — URL : https://typescriptlang.org/(дата звернення: 20.09.2020).

7. Офіційний сайт Vue.js — URL : https://vuejs.org (дата звернення: 21.09.2020).

8. Wikipedia Vue.js — URL : https://wikipedia.org/wiki/Vue.js (дата звернення: 22.09.2020).

9. Офіційний сайт VueX — URL : https://vuex.vuejs.org (дата звернення: 24.09.2020).

10. Офіційний сайт React — URL : https://reactjs.org (дата звернення: 28.09.2020).

11. Wikipedia React — URL : https://wikipedia.org/wiki/React (дата звернення: 22.09.2020).

12. Офіційний сайт Redux — URL : https://redux.js.org (дата звернення: 28.09.2020).