МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ЗАСОБАМИ REACT»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.С. Киндюк
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Киндюку Олегу Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку засобами React

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Основні теоретичні відомості.
2. Проєктування вебзастосунку.
3. Програмна реалізація.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	14.01.2025	
2.	Збір вихідних даних.	14.02.2025	
3.	Обробка методичних та теоретичних	10.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2025	

Студент _____ О.С. Киндюк
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебзастосунку засобами React»: 71 с., 31 рис., 1 табл., 7 джерел, 1 додаток.

ВЕБЗАСТОСУНОК, ВЕБІНТЕРФЕЙС, ІНТЕРФЕЙС, КОМПОНЕНТ, КОРИСТУВАЧ, МАРШРУТИЗАЦІЯ, РОЗРОБКА, СТАН КОМПОНЕНТА, ТЕСТУВАННЯ, ХУК, JAVASCRIPT, MARVEL API, REACT, REST API, UI.

Об’єкт дослідження – процес розробки клієнтських вебзастосунків з використанням сучасних технологій JavaScript та бібліотеки React.

Мета роботи – створення інформаційного вебзастосунку для перегляду даних про персонажів та комікси Marvel на основі офіційного API, з використанням компонентного підходу та клієнтської маршрутизації.

Метод дослідження – структурно-функціональний аналіз, моделювання, прототипування, тестування вебзастосунків, метод компонентного проєктування інтерфейсів.

У роботі розглянуто етапи створення односторінкового вебзастосунку з використанням бібліотеки React. Реалізовано інтерфейс, що дозволяє користувачам переглядати інформацію про персонажів та комікси всесвіту Marvel, здійснювати пошук персонажів та переходити між сторінками без перезавантаження. Проєкт взаємодіє з Marvel API для отримання актуальної інформації. У роботі описано архітектуру застосунку, структуру компонентів, логіку запитів, маршрутизацію та обробку помилок. Результати реалізації підтверджують доцільність використання React у розробці вебзастосунків.

SUMMARY

Bachelor’s qualifying paper “Development of a Web Application Using React”: 71 pages, 31 figures, 1 table, 7 references, 1 supplement.

COMPONENT, DEVELOPMENT, HOOK, INTERFACE, JAVASCRIPT, MARVEL API, REACT, REST API, ROUTING, STATE MANAGEMENT, TESTING, UI, USER, WEB APPLICATION.

The object of the study is the process of developing client-side web applications using modern JavaScript technologies and the React library.

The aim of the study is to create an informational web application for browsing data about Marvel characters and comics using the official Marvel API and a component-based approach with client-side routing.

The methods of research are structural-functional analysis, modeling, prototyping, web application testing, and component-based UI design.

The paper outlines the development process of a single-page web application built with the React library. A user interface was implemented to allow users to browse information about Marvel characters and comics, perform search operations, and navigate between views without page reloads. The application interacts with the Marvel API to retrieve real-time data. The study describes the application architecture, component structure, request logic, routing system, and error handling. The implementation results confirm the effectiveness of React for developing web applications.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Основні поняття та технології, що використовуються для розробки вебзастосунку 10
1.1 Вебзастосунки: визначення, види, особливості 10
1.1.1 Класифікація вебзастосунків 10
1.1.2 Переваги вебзастосунків 11
1.1.3 Недоліки та виклики у розробці вебзастосунків 11
1.2 Сучасні інструменти та технології розробки вебзастосунків..... 12
1.2.1 React як основа для створення SPA 12
1.2.2 React Router як інструмент маршрутизації..... 13
1.2.3 Стилізація та розгортання застосунку 14
1.3 Робота з API та реалізація запитів у вебзастосунку 14
1.3.1 REST API та його структура..... 15
1.3.2 Кастомний хук та його переваги 15
1.3.3 Переваги використання кастомного хука..... 16
1.4 Висновки до розділу 1 16
2 Проєктування вебзастосунку на основі React з використанням Marvel API 18
2.1 Архітектура застосунку 18
2.1.1 Структура застосунку..... 18
2.1.2 Планування маршрутизації..... 19
2.1.3 Архітектурне проєктування 20
2.1.4 Використання API Marvel 20
2.2 Таблиця атрибутів компонентів 21
2.3 ER-діаграма сутностей..... 22
2.4 Висновки до розділу 2 23

3 Реалізація вебзастосунку “Marvel info” засобами react 25
3.1 Використання Marvel API та MarvelService 25
3.2 Реалізація основних функцій 29
3.2.1 Відображення списку персонажів (CharList) 29
3.2.2 Відображення випадкового персонажа (RandomChar) 31
3.2.3 Пошук персонажів(Search) 33
3.2.4 Відображення інформації про обраного персонажа(CharInfo) 35
3.2.5 Відображення розширеної інформації про персонажа або коміксу (CommonPage) 38
3.2.6 Відображення списку коміксів (ComicsList) 40
3.2.7 Головна сторінка (MainPage) 41
3.2.8 Головний компонент (App) 43
3.3 Робота вебзастосунку та взаємодія з користувачем 44
3.4 Висновки до розділу 3 49
Висновки 50
Перелік посилань 52
Додаток А Код основних компонентів вебззастосунку “Marvel Info” 53
A.1 ComicsList.js 53
A.2 CommonPage.js 55
A.3 CharList.js 59
A.4 CharInfo.js 61
A.5 RandomChar.js 64
A.6 Search.js 66
A.7 http.hook.js 68
A.8 App.js 69
A.9 MainPage.js 70

ВСТУП

Інформаційні технології доволі швидко розвиваються в усіх сферах людского життя. Один із розвинутих напрямків є веброзробка – створення вебзастосунків для збреження, обробки та представлення потрібної інформації. Про це свідчать постійні оновлення фреймворків, бібліотек та інструментів.

За останні роки струмко розвивається популярний фреймворк React, який розробила компанія Meta. Основні переваги цього фреймворку є архітектура, ефективне оновлення інтерфейсу, адаптація коду для вилучення його повторень. Завдяки цим перевагам React і зацікавив усіх розробників для створення односторінкових вебзастосунків [1].

Marvel API – сучасний інтерфейс, який має відкритий доступ до бази даних персонажів та коміксів всесвіту Marvel [3]. Завдяки цьому розробники можут безкоштовно створювати свої проєкти з інтеграцією Marvel API.

Актуальність теми полягає в необхідності створення інтуітивно зрозумілого, інформативного вебзастосунку, який дозволяє користувачу дізнатись деталі про необхідного персонажа або комікса всесвіту Marvel. Цей сайт втішить не тільки шанувальників, а й новачків, які хочуть внурити у всесвіт Marvel, а також стане демонстрацією можливостей React при роботі без баз даних.

Метою даної кваліфікаційної роботи є розробка вебзастосунку засобами React, що дозволяє переглядати інформацію про персонажів та комікси у зручному форматі.

У процесі роботи були поставлені такі основні завдання:

- проаналізувати особливості архітектури React та принципи роботи з REST API;
- розробити компоненти інтерфейсу користувача;
- реалізувати отримання та обробку даних з Marvel API;

– організувати навігацію по сайту;
– забезпечити розгортання сайту за допомогою GitHub Pages.

Результатом роботи є повністю функціональний сайт, створений на базі React, що демострує правильну взаємодію з Marvel API та відображає інформацію у інтуітивно зрозумілому вигляді.

1 ОСНОВНІ ПОНЯТТЯ ТА ТЕХНОЛОГІЇ, ЩО ВИКОРИСТОВУЮТЬСЯ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ

1.1 Вебзастосунки: визначення, види, особливості

Вебзастосунок – програмний продукт, який функціонує у веббраузері та забезпечує користувачу доступ до його функціоналу, сервісів, інтерфейсу без потреби встановлювати окремі програмні забезпечення. Цей продукт є безумовно ключовим елементом сучасної цифрової індустрії, який дозволяє автоматизувати бізнес-процеси та полегшити отримання, передавання і обробку інформації між користувачами.

З часом технології розвиваються, тому і архітектура вебзастосунків ускладнюється, перетворюючи вебсайт із статичним тексом у динамічні односторінкові застосунки, що дозволяє підвищити продуктивність передачі інформації, плавність взаємодії користувача із інтерфейсом застосунку та зменшити навантаження на сервер.

1.1.1 Класифікація вебзастосунків

Вебзастосунки в основному можна класифікувати за наступними критеріями:

– за архітектурою;
– за типом взаємодії з користувачем;
– за сферою застосування.

За архітектурою:

– мультисторінкові застосунки (MPA);
– односторінкові застосунки (SPA);
– прогресивні вебзастосунки (PWA).

За типом взаємодії з користувачем:

– статичні: містять фіксований контент;

– динамічні: змінюють вміст у залежності від дій користувача.

За сферою застосування:

– бізнес-застосунки;

– освітні;

– комерційні;

– соціальні мережі.

1.1.2 Переваги вебзастосунків

Основними перевагами вебзастосунків є:

– **кросплатформеність** – застосунок доступний на будь-якому пристрої з браузером;

– **централізоване оновлення** – не потрібно оновлювати застосунок вручну на кожному пристрої;

– **масштабованість** – легко адаптується під зростаючі потреби бізнесу;

– **простота розгортання та підтримки** – відсутність необхідності встановлення клієнтського ПЗ.

Ці переваги роблять вебзастосунки надзвичайно популярними серед розробників і компаній, що прагнуть автоматизувати процеси та покращити взаємодію з користувачем.

1.1.3 Недоліки та виклики у розробці вебзастосунків

Попри численні переваги, вебзастосунки мають низку недоліків:

– **залежність від підключення до інтернету** – у разі відсутності мережі функціональність може бути обмеженою;

– **безпека** – відкритий характер взаємодії з браузером вимагає ретельного захисту даних;
– **обмеження продуктивності** – у складних обчисленнях вебзастосунки поступаються нативним додаткам;
– **сумісність між браузерами** – можливі проблеми візуалізації або роботи функцій у різних браузерах.

Під час розробки слід враховувати ці обмеження, щоб досягти стабільної та безпечної роботи застосунку.

1.2 Сучасні інструменти та технології розробки вебзастосунків

У сучасній веброзробці активно використовуються різноманітні фреймворки, бібліотеки та методи, що дозволяють значно пришвидшити процес створення функціональних, масштабованих і зручних застосунків. Правильний вибір інструментів впливає на швидкість розробки, якість коду та подальшу підтримку проєкту.

Особливе значення мають фреймворки, що дозволяють створювати клієнтську частину застосунку, яка взаємодіє з сервером через API. Крім того, використовуються засоби маршрутизації, стилізації та розгортання, які забезпечують повний цикл життєдіяльності вебзастосунку.

1.2.1 React як основа для створення SPA

React – це популярна бібліотека JavaScript, створена компанією Meta, яка дозволяє створювати інтерактивні інтерфейси користувача. Основна ідея полягає в побудові застосунку з окремих компонентів, що мають власну логіку та інтерфейс.

Ключові особливості React:

- **компонентний підхід** – застосунок складається з багаторазово використовуваних блоків;
- **Virtual DOM** – забезпечує високу продуктивність за рахунок ефективного оновлення інтерфейсу;
- **JSX** – спеціальний синтаксис, що поєднує HTML і JavaScript, полегшує створення компонентів;
- **стан (State) і Пропси (Props)** – основні засоби для зберігання даних у компонентах та передачі інформації між ними.

React підтримується великим співтовариством, має безліч додаткових бібліотек (наприклад, React Router, Redux) і є одним із найпопулярніших інструментів для побудови SPA.

1.2.2 React Router як інструмент маршрутизації

У вебзастосунках важливою складовою є маршрутизація – механізм, що відповідає за перемикання між сторінками без повного перезавантаження. Для цієї мети у React-застосунках зазвичай використовується бібліотека **React Router**.

Основні можливості React Router:

- **маршрутизація без перезавантаження сторінки**;
- **параметри в URL** – динамічні шляхи та пошукові параметри;
- **підтримка вкладених маршрутів** – дозволяє створювати структуру сторінок будь-якої складності;
- **навігація за допомогою компонентів Link та NavLink** – забезпечує зручність використання та контроль стилів активних сторінок.

Завдяки цій бібліотеці вебзастосунок функціонує як повноцінний односторінковий сайт, що покращує зручність користування і сприйняття інтерфейсу [2].

1.2.3 Стилізація та розгортання застосунку

Щоб вебзастосунок мав привабливий вигляд і відповідав сучасним стандартам UX/UI, використовуються технології стилізації.

У рамках цього проєкту застосовується **Sass** – препроцесор CSS, який дозволяє створювати більш структуровані стилі з використанням змінних, вкладеності та міксинів.

Переваги використання Sass [6]:

- **краща організація коду**;
- **менше дублювання стилів**;
- **можливість повторного використання стилів у різних компонентах**.

Для публікації вебзастосунку застосовується **GitHub Pages** – безкоштовний сервіс для хостингу статичних сайтів. Розгортання здійснюється за допомогою скриптів, які автоматизують процес компіляції проєкту та його завантаження у віддалений репозиторій.

1.3 Робота з API та реалізація запитів у вебзастосунку

Одним з основних джерел даних для сучасних вебзастосунків є **API** (Application Programming Interface). У рамках проєкту використовується публічний API від Marvel, який надає доступ до інформації про персонажів, комікси тощо [3].

Завдяки API застосунок може динамічно завантажувати та відображати актуальні дані, що дозволяє створювати інтерактивні інтерфейси з оновленням інформації в режимі реального часу.

1.3.1 REST API та його структура

Marvel API є прикладом **RESTful API**, який базується на принципах HTTP-протоколу.

Marvel API підтримує наступні основні методи:

– **POST**, **PUT**, **DELETE** – не використовуються у цьому проєкті, оскільки API лише для читання;

– **GET** – для отримання інформації (наприклад, дані про персонажа).

Запит до Marvel API містить:

– **базову адресу**;

– **параметри запиту** (limit, offset, name, apikey, hash, ts);

– **ключ доступу (API key)** – персональний ідентифікатор для авторизації запитів.

Використання цих параметрів дозволяє виконувати гнучкі фільтрації, посторінкову навігацію та пошук.

1.3.2 Кастомний хук та його переваги

Для зручної роботи з API у проєкті реалізовано **кастомний хук** “useRequest”, який інкапсулює логіку запитів до Marvel API. Це дозволяє уникнути дублювання коду і підвищує його читабельність.

Основні особливості “useRequest”:

– базується на вбудованому API fetch;

– виконує запити з параметрами до API;

– обробляє стани завантаження, успішного отримання та помилки;

– повертає функції для отримання окремих сутностей (персонажів, коміксів тощо).

Хук має наступну структуру: const { loading, error, getCharacter, getComics } = useRequest().

Цей підхід відповідає концепції розділення логіки та представлення, де компоненти відповідають лише за відображення, а хук – за взаємодію з сервером [4].

1.3.3 Переваги використання кастомного хука

Використання кастомного хука має низку переваг:

- **уніфікація логіки запитів** – єдиний спосіб взаємодії з API у всьому застосунку;
- **повторне використання** – один і той самий хук використовується у різних компонентах;
- **простота тестування** – оскільки логіка винесена в окремий блок, її легко перевірити;
- **підвищення масштабованості** – додавання нових запитів відбувається централізовано.

Таким чином, хук “useRequest” є ключовим компонентом проєкту, що забезпечує ефективну інтеграцію з Marvel API та сприяє побудові надійного архітектурного рішення.

1.4 Висновки до розділу 1

Отже, було розглянуто основні поняття, архітектурні підходи та технології, що використовуються під час розробки сучасних вебзастосунків. Вебзастосунки відіграють важливу роль у цифровій трансформації бізнесу, забезпечуючи зручний, швидкий та універсальний доступ до сервісів через веббраузер. Класифікація застосунків за типом взаємодії, архітектурою та сферою використання дозволяє обирати оптимальні рішення відповідно до цілей проєкту.

Основну увагу приділено бібліотеці React як основі для створення односторінкових застосунків, що забезпечує високу продуктивність та ручну роботу з данними. React Router, як інструмент маршрутизації, дозволяє ефективно керувати навігацією без перезавантаження сторінок, а використання препроцесора Sass – створювати підтримувані та гнучкі стилі.

Окремо розглянуто роботу з Marvel API – приклад RESTful сервісу, з яким взаємодія здійснюється за допомогою хука. Такий підхід дозволяє керувати запитами до API, спрощує розробку, забезпечує повторне використання логіки та підвищує масштабованість застосунку.

Таким чином, описані технології та інструменти становлять основу реалізації сучасного вебзастосунку.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ НА ОСНОВІ REACT З ВИКОРИСТАННЯМ MARVEL API

Цей розділ присвячений етапам проєктування вебзастосунку для отримання та відображення даних із Marvel API. Розглядаються основні аспекти структури та архітектури застосунку, а також планування розробки.

2.1 Архітектура застосунку

Архітектура вебзастосунку побудована з використанням компонентного підходу React. Застосунок складається з кількох основних компонентів: для відображення списку персонажів, окремої сторінки з детальною інформацією про персонажа, а також компонента для обробки запитів до Marvel API.

2.1.1 Структура застосунку

Застосунок має декілька компонентів, що виконують різні компоненти:

- **App** – основний компонент, що керує маршрутизацією і взаємодією з іншими компонентами;
- **MainPage** – головна сторінка, що поділяється на декілька компонентів;
- **CharList** – компонент, що відображає список персонажів;
- **CharInfo** – компонент, що відображає деталі вибраних персонажів;
- **RandomChar** – компонент, що відображає випадкових персонажів;
- **ComicsList** – сторінка для перегляду списку коміксів;
- **CommonPage** – сторінка для детальної інформації про обраного персонажа або комікс;
- **NotFound** – сторінка помилки для неіснуючих маршрутів.

Усі компоненти в додатку взаємодіють між собою через стейти і пропси. Головний компонент, App, містить маршрутизацію та відображає різні сторінки залежно від запиту користувача. Структура компонентів зображена на рисунку 2.1.

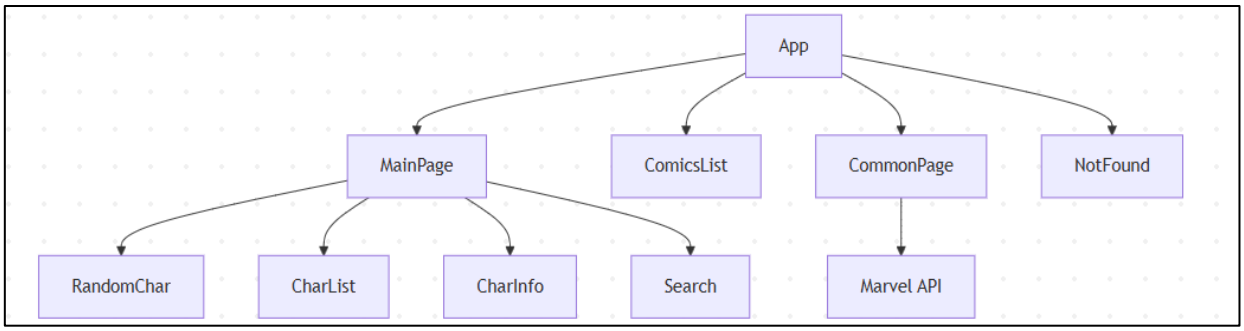


Рисунок 2.1 – Схема структури компонентів

2.1.2 Планування маршрутизації

Маршрутизація в застосунку здійснюється через React Router, який відповідає за управління переходами між сторінками. Схема маршрутизації зображена на рисунку 2.2.

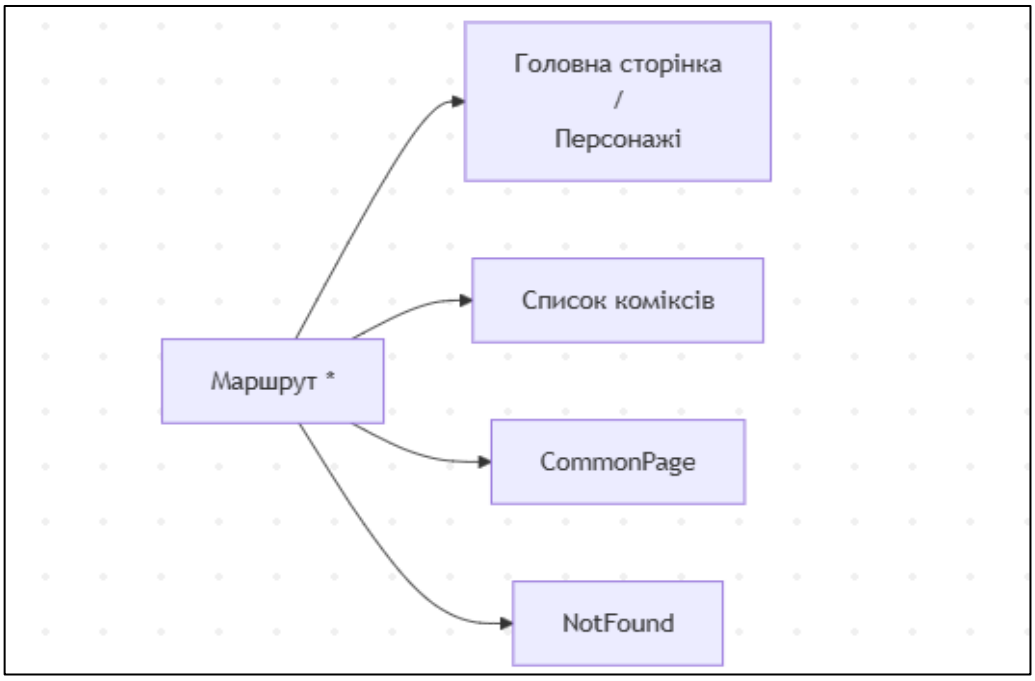


Рисунок 2.2 – Схема маршрутизації застосунку

Основні маршрути:

– **/marvel_app** – головна сторінка з випадковим персонажем;
– **/comics** – сторінка зі списком коміксів;
– **/:type/:id** – сторінка з детальною інформацією про персонажа чи комікс;
– ***/ (NotFound)*** – сторінка помилки для невірно введених шляхів.

2.1.3 Архітектурне проєктування

Проєктування застосунку базується на принципах компонентного підходу, що дає змогу легко змінювати й оновлювати різні частини інтерфейсу без впливу на інші частини. Кожен компонент виконує певну роль і має чітко визначену функціональність та дозволяє полегшити тестування, а також модульне оновлення та підтримку коду.

2.1.4 Використання API Marvel

Застосунок використовує Marvel API для отримання даних про персонажів та комікси. Схема запитів застосунку зображено на рисунку 2.3.

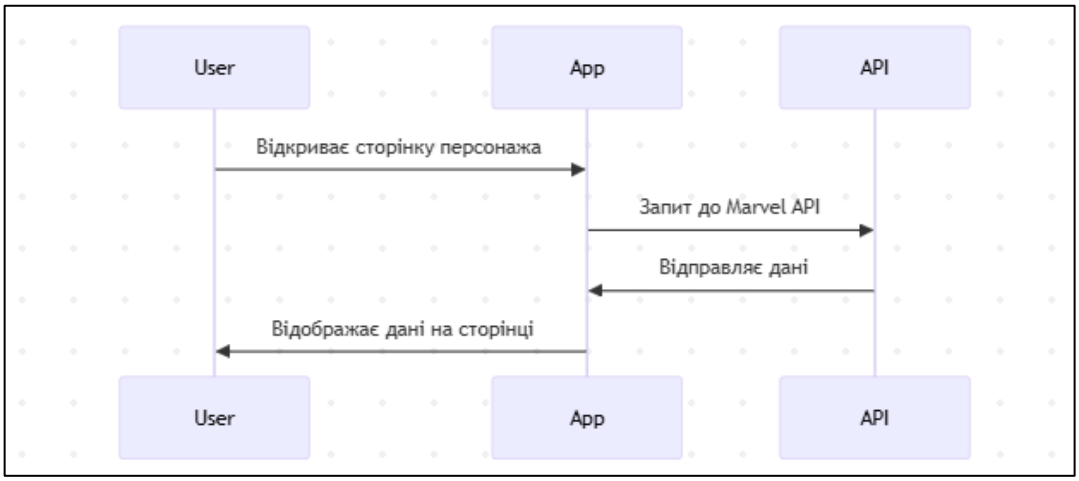


Рисунок 2.3 – Схема запитів застосунку

Здійснюється запит до API для отримання необхідної інформації. Всі запити здійснюються асинхронно через кастомний хук useRequest, що використовує метод fetch для отримання даних.

2.2 Таблиця атрибутів компонентів

Основні атрибути компонентів і їх типи, що використовуються для налаштування та взаємодії між різними частинами вебзастосунку (див. табл. 2.1).

Таблиця 2.1 – Опис атрибутів компонентів та їх типів

Компонент	Атрибут	Тип	Опис
App	selectForCommonPage	Функція	Функція для передачі даних обраного елемента на сторінку CommonPage (ідентифікатор та тип елемента).
AppHeader	none	--	Компонент заголовка без атрибутів, відповідає за відображення заголовку додатку.
MainPage	selectForCommonPage	Функція	Функція для передачі даних про обраного персонажа або комікс на CommonPage.
MainPage	selectedChar	Число або null	Ідентифікатор вибраного персонажа, використовується для відображення детальної інформації про персонажа.
RandomChar	selectForInfo	Функція	Функція, що передає ідентифікатор вибраного випадкового персонажа на MainPage.
RandomChar	selectForCommonPage	Функція	Функція для передачі даних про випадковий персонаж на сторінку CommonPage.
CharList	selectForInfo	Функція	Функція для вибору персонажа та передачі його ідентифікатора на компонент CharInfo.
CharList	selectedChar	Число або null	Ідентифікатор вибраного персонажа для відображення його детальної інформації.
CharInfo	selectedId	Число або null	Ідентифікатор вибраного персонажа, використовується для отримання та відображення його інформації.

Продовження таблиці 2.1

Компонент	Атрибут	Тип	Опис
CharInfo	selectForInfo	Функція	Функція для оновлення ідентифікатора вибраного персонажа на компоненті CharInfo.
Search	selectForCommonPage	Функція	Функція для передачі даних про вибраний елемент через функцію selectForCommonPage.
ComicsList	selectForCommonPage	Функція	Функція для передачі даних про вибраний комікс на сторінку CommonPage.
NotFound	none	--	Компонент для відображення сторінки з помилкою 404 при невірному маршруті.
CommonPage	selectedElem	Об’єкт або null	Об’єкт, що містить id та тип елемента (персонаж або комікс), який вибрано для відображення на сторінці.
CommonPage	selectForCommonPage	Функція	Функція для передачі даних вибраного елемента для відображення на сторінці CommonPage.

2.3 ER-діаграма сутностей

ER-діаграма, яка ілюструє структуру базових компонентів вебзастосунку “Marvel Info”. Діаграма демонструє, як основні елементи взаємодіють один з одним у контексті додатка. Ключові компоненти включають персонажів, комікси, головну сторінку, сторінки з детальною інформацією та пошук:

- **MainPage:** відповідає за відображення списку персонажів і коміксів, надаючи користувачам можливість взаємодії;
- **CommonPage:** надає детальну інформацію про вибраного персонажа чи комікс;
- **Search:** дозволяє здійснювати пошук серед персонажів та коміксів;
- **AppHeader:** відповідає за відображення заголовку застосунку, а також навігацію.

Зв’язки між компонентами показують логіку взаємодії та передавання даних між різними частинами застосунку (див. рис. 2.4).

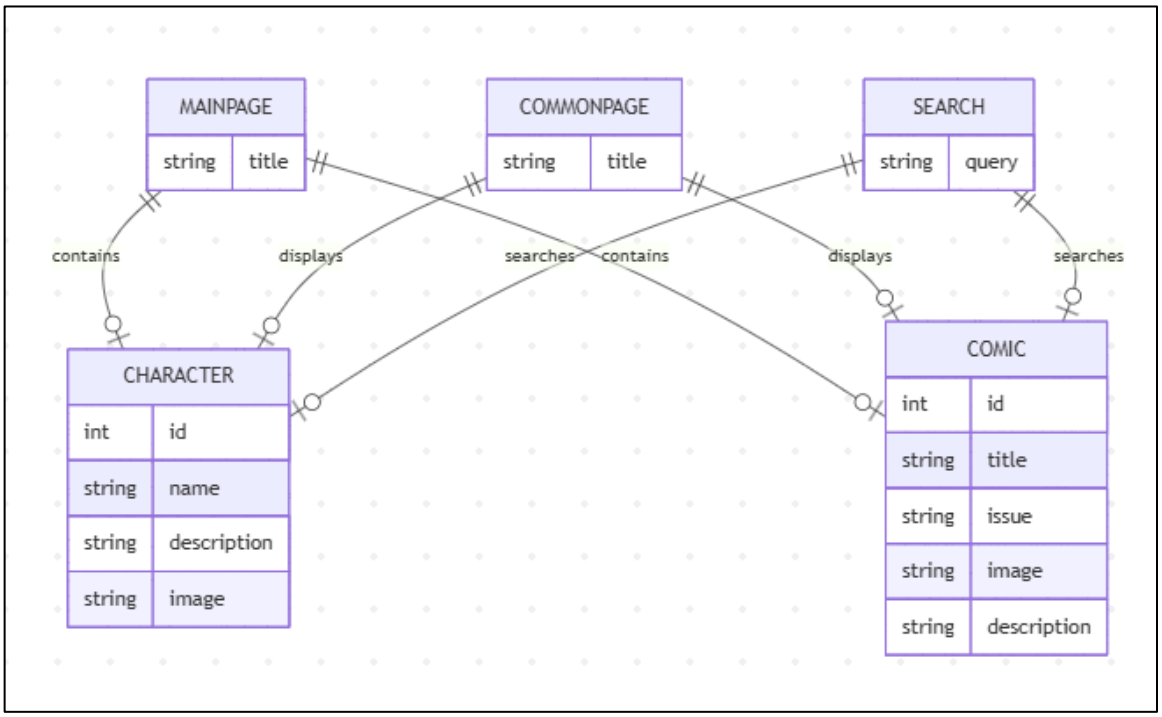


Рисунок 2.4 – ER-діаграма сутностей

2.4 Висновки до розділу 2

Отже, було здійснено проєктування вебзастосунку “Marvel Info”, що включає визначення основних компонентів, їх взаємодії та структуру. Ми розглянули ключові елементи застосунку, такі як сторінки для відображення списків персонажів та коміксів, а також детальні сторінки з інформацією про вибрані елементи. Процес проєктування охоплює не лише визначення функціональних компонентів, але й організацію їх взаємозв’язків для забезпечення безперебійної та зручної роботи застосунку.

ER-діаграма допомогла уявити взаємодію між різними частинами системи та визначити структуру компонентів. Крім того, було наведено таблицю атрибутів, що дає чітке уявлення про властивості кожного елемента застосунку.

У результаті проєктування було створено чітке розуміння структури вебзастосунку, що дозволить ефективно реалізувати подальші етапи розробки та забезпечити інтеграцію усіх компонентів у єдину систему.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ “MARVEL INFO” ЗАСОБАМИ REACT

У третьому розділі описується процес реалізації проєкту вебзастосунку на основі технології React, з використанням Marvel API для отримання даних про персонажів та комікси. Основна увага приділяється опису структури коду, компонентів, реалізації основних функціональних можливостей, а також опису технічних аспектів застосунку.

3.1 Використання Marvel API та MarvelService

Marvel API є основним джерелом даних для нашого застосунку. З його допомогою здійснюються запити для отримання інформації про персонажів, комікси та інші елементи всесвіту Marvel. API надає різні кінцеві точки для роботи з даними, що дозволяє отримувати як загальну інформацію, так і деталі окремих персонажів чи коміксів.

Для реалізації взаємодії з API в проєкті використовується розроблений сервіс MarvelService, який об’єднує всі необхідні функції для отримання даних. Всі запити до API здійснюються через кастомний хук **useRequest**, що забезпечує обробку станів завантаження, помилок та результатів запитів.

Основні функції MarvelService: отримання всіх персонажів, отримання персонажа за ID, отримання персонажа за іменем, отримання усіх коміксів, отримання комікса за назвою та номером випуску, трансформація даних, обробка помилок та станів завантаження.

Розглянемо детальніше основні функції MarvelService.

Отримання всіх персонажів. Функція getAllCharacters отримує список персонажів з API з параметрами limit і offset (див. рис. 3.1). Вона використовує метод request для отримання даних з кінцевої точки API. Параметр offset дозволяє визначати, з якого елемента потрібно починати завантаження даних.

```
const getAllCharacters = async (offSet = _baseOffSet) => {
    const res = await request(`${_apiBase}/characters?limit=9&offset=${offSet}&${_apiKey}`);
    return res.data.results.map(_transformChar);
};
```

Рисунок 3.1 – Отримання усіх персонажів

Отримання персонажа за ID. Функція getCharactersById дозволяє отримати детальну інформацію про персонажа за його унікальним ID (див. рис. 3.2). Функція обробляє тип елемента (characters або comics), щоб в залежності від цього правильно обробити отримані дані.

```
const getCharactersById = async (id, type='characters') =>{
        const res = await request(`${_apiBase}/${type}/${id}?${_apiKey}`)
        if(type==='characters'){
            return _transformChar(res)
        } else if(type==='comics'){
            console.log(res)
            return _transformComics(res)
        } else {
            return
        }
    }
```

Рисунок 3.2 – Отримання персонажів за айді

Отримання персонажа за іменем. Функція getCharactersByName здійснює пошук персонажа за його іменем (див. рис. 3.3). У разі наявності результатів пошуку повертаються трансформовані дані про персонажа. У випадку відсутності персонажа за зазначеним іменем, функція повертає порожній список.

```
const getCharactersByName = async (name) => {
    const res = await request(`${_apiBase}/characters?name=${encodeURIComponent(name)}&${_apiKey}`);
    if (res.data.results.length) {
        return _transformChar(res.data.results[0]);
    }
    return [];
};
```

Рисунок 3.3 – Отримання персонажу за іменем

Отримання всіх коміксів. Функція getAllComics (див. рис. 3.4) отримує список коміксів через запит до кінцевої точки "/comics". Додатково є параметри limit і offset, що дозволяють контролювати кількість та початковий номер ID завантажуваних коміксів.

```
const getAllComics = async (offSet = _baseOffSet) => {
    const res = await request(`${_apiBase}/comics?limit=8&offset=${offSet}&${_apiKey}`);
    return res.data.results.map(_transformComics);
};
```

Рисунок 3.4 – Отримання усіх коміксів

Отримання комікса за назвою та номером випуску. Функція getComicsByName дозволяє здійснювати пошук коміксів за назвою та номером випуску (див. рис. 3.5). Для цього використовується параметр title для пошуку коміксів, а також параметри issueNumber та dateDescriptor для фільтрації за роком випуску.

```
const getComicsByName = async (name, issueNumber = _issueNumber) => {
    const res = await request(`${_apiBase}/comics?title=${encodeURIComponent(name)}&issueNumber=${issueNumber}&${_apiKey}`);
    if (res.data.results.length) {
        return _transformComics(res.data.results[0]);
    }
    return [];
};
```

Рисунок 3.5 – Отримання коміксу за ім’ям

Трансформація даних. Для кожного типу даних (персонажі, комікси) перед поверненням результатів з API, дані трансформуються в зручний для відображення формат, за це відповідають функції _transformChar та _transformComic.

_transformChar: трансформує дані про персонажа, форматує дату змін і відображення зображення, а також надає стислий опис персонажа (див. рис. 3.6).

```
const _transformChar= (char) =>{
    console.log(char)
    const date = new Date(char.modified);
    const formattedModified = date.toLocaleDateString('en-CA');
    let objStyle = { objectFit: "cover" };
    if (char.thumbnail.path === "http://i.annihil.us/u/prod/marvel/i/mg/b/40/image_not_available" || char.thumbnail.path === "http://i.annihil.us/u/prod/marvel/i/mg/f/60/4c002e0305708") {
        objStyle = { objectFit: "fill" };
    }

        return{
            id: char.id,
            name: char.name,
            description: char.description.length > 0 ? (char.description.length > 150 ? char.description.substring(0, 170) + "..." : char.description) : "The character has no description",
            fullDescription: char.description.length > 0 ? char.description : "The character has no description",
            thumbnail: `${char.thumbnail.path}.${char.thumbnail.extension}`,
            homepage: char.urls[0].url,
            wiki: char.urls[1].url,
            comics: char.comics.items,
            modified: formattedModified,
            objStyle: objStyle
        }
    }
```

Рисунок 3.6 – Нормалізація даних персонажів

_transformComics: перетворює дані про комікс, форматуючи дату змін, додаткові дані про комікс, авторів та інші параметри (див. рис. 3.7).

```
const _transformComics = (comics) => {
    const formattedModified = new Date(comics.modified).toLocaleDateString('en-CA');
    return {
        id: comics.id,
        name: comics.title,
        series: comics.series.name,
        description: comics.description?.length ? comics.description : "The comic has no description",
        fullDescription: comics.description?.length ? comics.description : "The comic has no description",
        modified: formattedModified,
        saleDate: comics.dates,
        issueNumber: comics.issueNumber,
        characters: comics.characters.items,
        author: comics.creators.items,
        thumbnail: `${comics.thumbnail.path}.${comics.thumbnail.extension}`,
        printPrice: typeof comics.prices[0]?.price === 'number' ? `${comics.prices[0].price} $` : "N/A",
        digitalPrice: typeof comics.prices[1]?.price === 'number' ? `${comics.prices[1].price} $` : "Not available",
        pageCount: comics.pageCount,
        detail: comics.urls[0]?.url || "#"
    };
};
```

Рисунок 3.7 – Нормалізація даних коміксів

Обробка помилок та станів завантаження. За допомогою хука **useRequest** забезпечується обробка помилок та станів завантаження під час взаємодії з API (див. рис. 3.8).

```
import { useState, useCallback } from 'react';  4.7k (gzipped: 1.9k)    You, 5 days ago • Ready …

const useRequest = () =>{
    const [loading, setLoading] = useState(false)
    const [error, setError] = useState(false)

const request = useCallback(async (url, method = "GET", headers = { "Content-Type": "application/json" }, data = undefined) => {
    setLoading(true)
    setError(false)
    try{

        let res = await fetch(url, {
            method: method,
            headers,
            body: JSON.stringify(data)
        });

        if (!res.ok) {
            setError(true)
            const errorText = await res.text();
            throw new Error(`HTTP ${res.status}: ${errorText}`);
        }

        return await res.json()

    } catch (error) {
        setError(true)
        throw new Error(error.message);
    } finally {
        setLoading(false)
    }}, [])

    return {request, loading, error}
}

export default useRequest;
```

Рисунок 3.8 – Кастомний хук із обробкою станів та помилок

Якщо виникають помилки або затримки в процесі запитів, відповідні стани (loading, error) сповіщають користувача про поточний стан операції.

3.2 Реалізація основних функцій

3.2.1 Відображення списку персонажів (CharList)

Компонент **CharList** відповідає за відображення списку персонажів з Marvel API (див. рис. 3.9). Компонент реалізує прокручування для завантаження нових персонажів при натисканні кнопки “Load more”. Реалізовано наступним чином: завантаження персонажів, керування станом, рендеринг, завантаження додаткових персонажів, обробка помилок і індикатор завантаження.

Завантаження персонажів. При першому рендерингу використовуються функції хука useEffect та сервісу MarvelService, щоб завантажити персонажів з Marvel API за допомогою методу getAllCharacters. Перше завантаження виконується автоматично під час ініціалізації компонента, підвантаження нових даних виконується після натискання кнопки “Load more”. Наступні дані завантажуються по частинах за допомогою параметра offset, що дозволяє отримувати наступні персонажі без повторень.

Керування станом:

– char – масив персонажів, який зберігає дані про всіх завантажених персонажів;
– firstLoading – стан, який вказує, чи відбувається перше завантаження даних;
– offSet – значення, яке визначає, скільки персонажів було вже завантажено, і дозволяє завантажувати нові персонажі.

Завантаження додаткових персонажів:

– кнопка “Load more” дозволяє завантажити наступні персонажі, при цьому дані додаються до вже існуючого списку.

```
useEffect(() => {
    currentRef.current = [];
    updateChar(offSet, true)
}, [])

const updateChar = (offset, initialRender = false) => {
    if (initialRender) {
        setFirstLoading(true);
    }
    getAllCharacters(offset)
        .then(res => {onCharLoaded(res)
        setFirstLoading(false)})
        .catch(() => {
            console.log("Error on getAllCharacters")
            setFirstLoading(false);
        })
}

const onCharLoaded = (newChar) => {
    if (newChar.length === 0) return;
    setFirstLoading(false)
    setOffSet(prev => prev + 9);
    setChar(prevChar => [...prevChar, ...newChar])
}

const onClickChar = (charId, index) => {
    selectForInfo(charId)
    currentRef.current[index].focus()
    currentRef.current[index].scrollIntoView({
        behavior: "smooth",
        block: "start",
    });
}

const myRef = (e, index) => {
    if (e && !currentRef.current[index]) {
        currentRef.current[index] = e
    }
}

const View = memo(({ char }) => {
    return char.map((char, index) => {
        const animation = currentRef.current[index] ? "" : "char__item_animate"
        return (
            <li className={selectedChar === char.id ? "char__item char__item_selected" : `char__item ${animation}`}
                tabIndex={index}
                ref={(e) => myRef(e, index)}
                onClick={() => onClickChar(char.id, index)}
                id={char.id}
                key={char.id}>
                <img src={char.thumbnail} alt={char.name} style={char.objStyle} />
                <div className="char__name">{char.name}</div>
            </li>
        );
    });
}, [])

const content = !firstLoading ? <View char={char} /> : null

return (
    <div className="char__list">
        <ul className="char__grid">
            {content}
        </ul>
        {error ? <ErrorMessage /> : null}
        {(loading && !error) ? <div style={{ "gridColumn": "2" }}><Spinner /></div> : null}
        <button onClick={() => updateChar(offSet)} className="button button__main button__long" disabled={loading}>
            <div className="inner">{loading ? "Loading..." : "Load more"}</div>
        </button>
    </div>
)
```

Рисунок 3.9 – Компонент CharList

Рендеринг:

– використовуються useRef і memo для оптимізації рендерингу та збереження фокусу на вибраному персонажі;
– при кліку на персонажа, він виділяється за допомогою CSS класу char__item_selected, а також фокусується на елементі, який відображає інформацію про персонажа.

Обробка помилок і індикатор завантаження:

– при виниканні помилки при запиті, відображається компонент ErrorMessage;
– індикатор завантаження Spinner показується, коли дані ще не завантажені або виконуються додаткові запити.

3.2.2 Відображення випадкового персонажа (RandomChar)

Компонент RandomChar випадковим чином завантажує одного персонажа Marvel та відображає його коротку інформацію (див. рис. 3.10).

Користувачу дізнатись про нового персонажа, обраного випадковим чином, вдається за допомогою:

– **стани (useState):** зберігає інформацію про поточного випадково вибраного персонажа (або null, якщо ще не обрано);
– **getCharactersById(id):** функція з MarvelService, яка отримує інформацію про персонажа за його ID;
– **onCharLoaded(char)**: зберігає отриманого персонажа в стан state;
– **updateChar(id)**: викликає API з конкретним ID і обробляє результат;
– **generatedId()**: генерує випадкове число у зафіксованих межах, яке використовується як ID персонажа для завантаження;
– **AltView**: відображається, якщо ще не вибрано жодного персонажа: заглушка з аватаром StubAvatar і підписом “Select a character”**;**
– **ErrorMessage**: показується, якщо виникла помилка;
– **Spinner**: показується під час завантаження;

```
    const onCharLoaded = (char) => {
        setState(char);
    };
    const updateChar = (id) => {
        getCharactersById(id)
            .then(onCharLoaded)
            .catch(() => {
                console.log('Error');
            });
    };
    const generatedId = () => {
        const id = Math.floor(Math.random() * (1011400 - 1011000 + 1) + 1011000);
        updateChar(id);
    };
    const altContent = !(state || loading || error) ? <AltView /> : null;
    const errorMessage = error ? <ErrorMessage /> : null;
    const spin = loading && !error ? <Spinner /> : null;
    const content = !(loading || error) && state ? <View char={state} select={props} /> : null;
    return (
        <div className="randomchar">
            <h2 className="randomchar__title">Discover a Random Character!</h2>
            <p className="randomchar__subtitle">Click below to learn more about a randomly selected character</p>
            <div className="randomchar__block">
                {errorMessage}
                {altContent}
                {content}
                {spin}
                <div className="randomchar__block-logo">
                    <ChooseBanner />
                </div>
                <div className="randomchar__block-static">
                    <div className="randomchar__block-name">Want Another?</div>
                    <div className="randomchar__block-img-static">
                        <svg width="200" height="200" viewBox="0 0 200 200" fill="none" xmlns="http://www.w3.org/2000/svg">
                            <circle cx="100" cy="80" r="30" fill="#222" stroke="#fff" strokeWidth="3" />
                            <path d="M60 160 Q100 85, 140 160" fill="#222" stroke="#fff" strokeWidth="5" />
                        </svg>
                    </div>
                    <div className="randomchar__block-static-descr">Click "Try it" to preview a random character</div>
                    <button onClick={generatedId} className="button button__main button__glow">
                        <div className="inner">Try it</div>
                    </button>
                </div>
            </div>
        </div>
    );
};
const AltView = () => {
    return (
        <div className="randomchar__block-char">
            <div className="randomchar__block-name">Select a character</div>
            <div className="randomchar__block-img">
                <StubAvatar />
            </div>
            <div className="randomchar__block-descr">Character information will appear here</div>
        </div>
    );
};
const View = ({ char: { id, name, description, thumbnail, objStyle }, select: { selectForInfo } }) => {
    let borderRad = { borderRadius: "50%" };
    if (objStyle?.objectFit === 'fill') {
        borderRad = { borderRadius: 0 };
    }       You, 2 months ago • Progress 95% …
    return (
        <div className="randomchar__block-char">
            <div className="randomchar__block-name">{name}</div>
            <div className="randomchar__block-img">
                <img
                    className="randomchar__block-img"
                    style={{ ...borderRad, ...objStyle }}
                    src={thumbnail}
                    alt={name}
                />
            </div>
            <div className="randomchar__block-descr">{description}</div>
            <button onClick={() => selectForInfo(id)} className="button button__main">
                <div className="inner">Select</div>
            </button>
        </div>
    );
```

Рисунок 3.10 – Компонент RandomChar

– **View**: рендерить інформацію про знайденого персонажа (ім’я, зображення, опис, кнопка Select для передачі ID у верхній компонент). Перед цим отримує char (персонаж) та select (проп з функцією selectForInfo), визначає стиль зображення, показує ім’я, зображення, опис та кнопку Select, яка викликає selectForInfo(id) для завантаження цього персонажа в CharInfo;
– кнопка **“Try it”**: завантажує нового випадкового персонажа;
– блок із статичним SVG-банером у стилі Marvel;
– кнопка **Select**: передає ID в інший компонент для показу розширеної інформації.

3.2.3 Пошук персонажів(Search)

Компонент Search дозволяє користувачу знайти персонажа Marvel за ім’ям (див. рис. 3.11). Якщо персонажа знайдено – показується посилання на його детальну сторінку. Якщо ім’я введене некоректно або персонажа не існує – з’являється відповідне повідомлення.

Основні стани (useState):

– **char** – об’єкт з інформацією про знайденого персонажа, або null, або { name: null }, якщо нічого не знайдено;
– **required** – булевий прапорець, що сигналізує, чи було порожнє поле пошуку (потрібно ввести хоча б 1 символ).

Marvel API:

– **getCharactersByName** – функція з MarvelService, яка викликається для отримання персонажа за ім’ям.

Інтеграція:

– компонент приймає проп selectForCommonPage – функцію для передачі вибраного ID у вищий компонент (потрібна для переходу на CommonPage), працює в парі з CharInfo у компоненті MainPage.

```
const startSearch = (name) =>{
    setChar(null)
    getCharactersByName(name)
        .then(completeSearch)
        .catch(() => setChar(null));
}
const completeSearch = (res) => {
    if (res && res.name) {
        setChar(res);
    } else {
        setChar({ name: null });
    }
};
const onSubmitName = (e) => {
    e.preventDefault();
    setRequired(false);
    const searchVal = e.target.name.value;
    if (searchVal.length >= 1){
        startSearch(searchVal);
        e.target.name.value = '';
    } else{
        setRequired(true)
    }
};
const Done = ({name, id}) =>{
    console.log(id)
    if (name) {
    return (
        <div className="char__search-result-done">
            "There is! Visit "{name}" page?"
            <Link to={`/${type}/${id}`} onClick={()=>{selectForCommonPage(id)}} className="button button__secondary">
                <div className="inner">To page</div>
            </Link>
        </div>
    )
} else {
    return (
        <div className='char__search-result-bad'>
            The character was not found. Check the name and try again<br/>
        </div>
    )
}
}
const Empty = () =>{
    return (
    <div className='char__search-result-bad'>
        This field is required
    </div>
    )
}
const content = !loading && char ? <Done name={char.name} id={char.id} /> : required ? <Empty /> : null;
return (
    <div className="char__search">
        <form action="" className="char__search-form" onSubmit={e=>onSubmitName(e)}>
            <label htmlFor="char__name-input" className="char__name-label">
                Or find a character by name:
            </label>
            <div>
                <input
                    type="text"
                    id="char__name-input"
                    name="name"
                    className="char__name-input"
                    placeholder="Enter name"
                />
                <button type="submit" className="button button__main">
                    <div className="inner">Find</div>
                </button>
            </div>
        </form>
        {content}
        {loading?<Spinner/>:null}
        {error?<Done/>:null}
        </div>
);
```

Рисунок 3.11 – Компонент Search

Основні функції:

- **startSearch(name)**: ініціює пошук, обнуляє char, робить запит через API, обробляє результат;
- **completeSearch(res)**: зберігає результат у стан char, або записує { name: null }, якщо персонажа не знайдено;
- **onSubmitName(e)**: обробляє форму: перевіряє наявність введеного тексту, запускає пошук або виводить повідомлення про обов'язкове поле.

Внутрішні компоненти для відображення результату:

- **Done**: відображає результат пошуку;
- якщо персонажа знайдено – показується його ім'я та кнопка "To page"**,** яка веде на /characters/:id;
- якщо персонажа не знайдено – повідомлення: "The character was not found...";
- **Empty** – повідомлення "This field is required"**,** якщо спроба пошуку була з порожнім полем.

Візуальні елементи:

- форма з інпутом (input) для введення імені та кнопкою Find;
- повідомлення про результат (успіх / не знайдено / поле обов'язкове);
- у процесі пошуку показується Spinner;
- у разі помилки з API – компонент Done без імені (відображає, що не знайдено).

3.2.4 Відображення інформації про обраного персонажа(CharInfo)

Компонент CharInfo відображає детальну інформацію про вибраного персонажа Marvel, включаючи його зображення, опис, посилання на Wiki, та список коміксів, у яких він з'являється. Дозволяє переходити до детальної сторінки персонажа або до сторінки з коміксом (див. рис. 3.12).

```
useEffect(()=>{
    if(selectedId !== null && selectedId !== undefined && selectedId !== ""){
        onCharLoad(selectedId)
        setSelected(true)
    }
}, [selectedId])
const onCharLoad = (selected) =>{
    getCharactersById(selected)
        .then(onCharLoaded)
        .catch(()=>{
        })
}

const onCharLoaded = (res) =>{
    setChar(res)
}
const onClickComics = (url) => {
    const searchId= /(\d+)$/
    const myId = searchId.exec(url)
    const comicsId = myId === null ? 0 : myId[1]
    selectForCommonPage(comicsId, 'comics')
};
const handleClose = () => {
    setSelected(false)
    setChar()
    selectForInfo()
};
const InfoChar = ({char}) => {
    const {id,name,thumbnail,description,wiki, homepage, comics, objStyle } = char
    return(
        <>
            <div className="close-icon-wrapper" onClick={handleClose}>
                <CloseIcon />
            </div>
            <div className="char__basics">
                <img src={thumbnail} alt={name} style={objStyle}/>
                <div>
                    <div className="char__info-name">{name}</div>
                    <div className="char__btns">
                        <Link
                        to={`/${type}/${id}`}
                        onClick={()=>selectForCommonPage(id)}
                         className="button button__secondary">
                            <div className="inner">Details</div>
                        </Link>
                    </div>
                </div>
            </div>
            <div className="char__descr">
                {description}
            </div>
            <div className="char__comics">Comics:</div>
            <ul className="char__comics-list">
                {comics && comics.length > 0 ? comics.slice(0,10).map((comic,index) => (
                <li key={index} onClick={()=>{onClickComics(comic.resourceURI)}} className="char__comics-item">
                    <Link to={`/comics/${comic.resourceURI.match(/(\d+)$/)?.[1]}`}>
                        {comic.name}
                    </Link>
                </li>
                )) : <li>No comics available</li>}
            </ul>
        </>
    )
}
const err = error ? <ErrorMessage/> : null
const load = loading ? <Spinner/> : null;
const skelet = selected ? null : skeleton()
const content = (!loading && selected && !error) ? <InfoChar char={char}/> : null

return(
    <div className="char__info">
        {err}
        {load}
        {skelet}
        {content}
    </div>
)
```

Рисунок 3.12 – Компонент CharInfo

Основні стани (useState):

– **char**: об’єкт з детальною інформацією про персонажа, отриманий з API;
– **selected**: булеве значення, що сигналізує про наявність вибраного персонажа;
– **comicId**: ID коміксу (використовується при кліку на комікс, хоча фактично лише передається).

Отримання даних (useEffect):

– коли змінюється selectedId, викликається функція onCharLoad, яка за допомогою getCharactersById з сервісу MarvelService отримує дані персонажа і оновлює стан char.

Функції:

– **onCharLoad(selectedId)**: отримує персонажа за ID з Marvel API;
– **onCharLoaded(res)**: встановлює отримані дані персонажа в стан char;
– **onClickComics(url)**: парсить ID коміксу з resourceURI та викликає selectForCommonPage для переходу на сторінку цього коміксу;
– **handleClose()**: очищає стан компонента, ховає вікно персонажа, скидає ID.

Візуальна частина (вкладений компонент InfoChar):

– зображення персонажа (thumbnail);
– ім’я персонажа (name);
– кнопка **Details**, яка веде до сторінки з повною інформацією (/characters/:id);
– опис (description);
– список коміксів, у яких персонаж фігурує (максимум 10), кожен з яких можна клікнути, щоб перейти до окремої сторінки коміксу.

Індикація станів:

– якщо персонаж ще не вибраний – відображається **skeleton**;
– якщо дані завантажуються – показується **Spinner**;
– у разі помилки – **ErrorMessage**;
– якщо все завантажено – відображається InfoChar.

Додаткова логіка:

– кнопка закриття (іконка CloseIcon) дозволяє згорнути панель з інформацією про персонажа.

3.2.5 Відображення розширеної інформації про персонажа або коміксу (CommonPage)

Компонент CommonPage відповідає за відображення детальної інформації про конкретного персонажа або комікс. Він є універсальним і підтримує два типи контенту (characters та comics), динамічно змінюючи структуру вмісту в залежності від переданого selectedElem (див. рис. 3.13).

Основні функції:

– відображення докладної інформації про один комікс або персонажа;
– динамічна побудова структури вмісту в залежності від типу;
– можливість переходу між пов’язаними об’єктами (наприклад, із персонажа на комікс, де він зустрічається, і навпаки);
– обробка завантаження, помилок і кліків поза елементами списку.

Ключові особливості реалізації:

– завантаження даних: через кастомний сервіс MarvelService, функція getCharactersById витягує потрібний об’єкт за ID, переданий через проп selectedElem, тип (type) визначає, чи йдеться про персонажа чи комікс.

Універсальний підхід до вмісту**:**

– компонент View визначає, як рендерити дані для characters або comics.

Модульні підкомпоненти**:**

– CharList відповідає за побудову списку пов’язаних елементів. Кожен пункт – це посилання на інший об’єкт (Link до react-router-dom), а при кліку викликається selectForCommonPage, який оновлює поточний стан.

Інтерактивна взаємодія:

– при натисканні кнопки “Click” відкривається список пов’язаних елементів. Подія mousedown поза елементом автоматично закриває цей список, завдяки слухачу подій у useEffect;

– обробка помилок: якщо запит не вдався – показується компонент ErrorMessage, а також посилання “Back to all” для повернення назад.

```
useEffect(() => {
    console.log("selectedElem:", selectedElem);
    setListStat(false);
    setElem(null);
    if (id && type) {
        onNewId(id, type);
    }
}, [id, type]);

useEffect(() => {
    const handleClickOutside = (event) => {
        if (listStat && !event.target.closest('.common-page__metadata-first-char')) {
            setListStat(false);
        }
    };

    document.addEventListener('mousedown', handleClickOutside);

    return () => {
        document.removeEventListener('mousedown', handleClickOutside);
    };
}, [listStat]);

const onNewId = (id, type = 'characters') => {
        getCharactersById(id, type)
            .then((res)=>{
                updateElem(res)})
            .catch((error) => {
            });
}

const updateElem = (elem) =>{

    console.log("Текущий елемент:", elem)
    setElem(elem);
}

const onClickElementName = (charId, type) => {
    setElem(null)
    selectForCommonPage(charId, type);
}

const CharList = ({ elementId, type }) => {
    console.log(elementId)
    if (!elementId || !Array.isArray(elementId) || elementId.length === 0) {
        return <li className="common-page__metadata-first-char-item">Not found</li>;
    }

    return elementId.map(item => {
        const baseCharId = /\/(\d+)/;
        const foundBaseCharId = baseCharId.exec(item.resourceURI);
        const elemId = foundBaseCharId ? foundBaseCharId[1] : null;
        console.log(elemId)

        return (
            <li key={item.id || elemId} onClick={() => onClickElementName(elemId, type)} className="common-page__metadata-first-char-item">
                {elemId &&
                    <Link to={`/${type}/${elemId}`}>
                        {item.name || 'Unknown Character'}
                    </Link>
                }
            </li>
        );
    });
};
```

Рисунок 3.13 – Компонент CommonPage

3.2.6 Відображення списку коміксів (ComicsList)

Компонент ComicsList відповідає за відображення списку коміксів, отриманих через Marvel API. Забезпечує динамічне завантаження нових елементів при натисканні на кнопку “load more”, а також передає обраний комікс до CommonPage для відображення розширеної інформації (див. рис. 3.14).

Основні функції:

- **useState:** зберігає список коміксів (comics), поточне значення offset (offSet) для пагінації, а також стан першого завантаження;
- **useEffect:** при монтуванні компонента викликається функція getComics, що завантажує початковий список коміксів;
- **getComics(newOffSet):** викликає функцію getAllComics із сервісу MarvelService, передаючи offset для пагінації. Отримані дані передаються в comicsLoaded;
- **comicsLoaded(newComics):** додає нові комікси до вже існуючого списку та оновлює offset для наступного запиту;
- **myRef(e, index):** зберігає посилання на DOM-елементи, щоб уникнути повторної анімації при повторному рендерінгу;
- **View:** відображає список коміксів (кожен елемент містить: зображення коміксу, назву коміксу, ціну, дату останнього оновлення);
- **selectForCommonPage**: викликається при кліку на комікс із передачею ID та типу, після чого користувач переходить на детальну сторінку;
- **selectForCommonPage**: функція для передачі ID коміксу та типу “comics” до CommonPage з подальшим відображенням детальної інформації.

Обробка завантаження та помилок:

- Spinner відображається під час завантаження;
- ErrorMessage показується у разі помилки.

```jsx
const ComicsList = ({ selectForCommonPage }) => {
    const [comics, setComics] = useState([]);
    const [firstLoading, setFirstLoading] = useState(true);
    const { loading, error, getAllComics } = MarvelService();
    const [offSet, setOffSet] = useState(0);
    const type = 'comics';

    const renderedItem = useRef([]);

    useEffect(() => {
        renderedItem.current = [];
        getComics(offSet);
    }, []);

    const getComics = (newOffSet = offSet) => {
        getAllComics(newOffSet)
            .then(comicsLoaded);
    }

    const comicsLoaded = (newComics) => {
        setComics(comics => [...comics, ...newComics]);
        setOffSet(prevState => prevState + 8);
    }

    const onComicsNew = (newOffSet) => {
        getComics(newOffSet);
    }

    const myRef = (e, index) => {
        if (e && !renderedItem.current[index]) {
            renderedItem.current[index] = e;
        }
    }

    const View = useCallback(({ comics }) => {
        return comics.map((comic, index) => {
            const firstRenderedItem = renderedItem.current[index] ? '' : " fade-in";
            return (
                <li key={comic.id} ref={(e) => myRef(e, index)} className={`comics__item${firstRenderedItem}`}>
                    <Link to={`/${type}/${comic.id}`} onClick={() => { selectForCommonPage(comic.id, type); }}>
                        <div className="comics__item-own">
                            <img src={comic.thumbnail} alt={comic.name} className="comics__item-img" />
                            <div className="comics__item-name">{comic.name}</div>
                        </div>
                        <div className="comics__item-sub">
                            <div className="comics__item-sub-price">{comic.printPrice}</div>
                            <data className="comics__item-sub-data">{comic.modified}</data>
                        </div>
                    </Link>
                </li>
            );
        });
    }, []);

    const content = comics.length ? <View comics={comics} /> : null;
    const load = loading ? <Spinner /> : null;
    const err = error ? <ErrorMessage /> : null;

    return (
        <div className="comics__list">
            <ul className="comics__grid">
                {content}
            </ul>
            {load}
            {err}
            <button onClick={() => onComicsNew(offSet)} disabled={loading} className="button button__main button__long">
                <div className="inner">load more</div>
            </button>
        </div>
    );
}

export default ComicsList;
```

Рисунок 3.14 – Компонент ComicsList

3.2.7 Головна сторінка (MainPage)

Компонент MainPage складається з основних функцій та складових компонентів, це головна сторінка застосунку, яка показує список персонажів

Marvel, їх короткий опис, випадкового героя, а також дозволяє переглядати розширену інформацію та здійснювати пошук (див. рис. 3.15).

Основні функції:

– **useState:** зберігає ID обраного персонажа (selectedChar), який передається в компонент CharInfo для відображення повної інформації;
– **selectForInfo(id):** встановлює ID вибраного персонажа зі списку (CharList), який потім передається до CharInfo.

Складові:

– **RandomChar:** відображає випадкового персонажа з можливістю оновлення та переходу до його детальної інформації;
– **CharList:** список персонажів, з якого можна вибрати одного для перегляду детальної інформації;
– **CharInfo:** відображає детальну інформацію про вибраного персонажа, має кнопку для переходу на загальну сторінку з деталями (CommonPage);
– **Search:** компонент для пошуку персонажів за ім’ям;
– **ErrorBoundary:** використовується для обгортання компонентів, які можуть викликати помилку під час рендеру.

```
const MainPage = ({selectForCommonPage}) =>{
    const [selectedChar, setSelectedChar] = useState(null)

    const selectForInfo = (id) =>{
        setSelectedChar(id)
    }

    return (
        <>
          <RandomChar selectForInfo={selectForInfo} selectForCommonPage={selectForCommonPage}/>
            <div className="char__content">
                <CharList selectForInfo={selectForInfo} selectedChar={selectedChar}/>
                <ErrorBoundary>
                    <div className="char__info__search">
                    <CharInfo selectedId={selectedChar} selectForInfo={selectForInfo} selectForCommonPage={selectForCommonPage}/>
                    <Search selectForCommonPage={selectForCommonPage}/>
                    </div>
                </ErrorBoundary>
            </div>
        </>
    )
}

export default MainPage;
```

Рисунок 3.15 – Компонент MainPage

3.2.8 Головний компонент (App)

Компонент App це головний компонент застосунку, що відповідає за структуру сторінок, маршрутизацію між ними та загальне оформлення (див. рис. 3.16) Саме тут реалізується логіка переходу між головною, коміксами, сторінкою з детальною інформацією та сторінкою 404.

```
function App() {
  const [selectedElem, setSelectedElem] = useState(null);

  const selectForCommonPage = (id, type = 'characters') => {
    setSelectedElem({ id, type });
  };

  return (
    <div className="app">
      <AppHeader />
      <main>
        <Routes>
          <Route path="/marvel_app" element={<MainPage selectForCommonPage={selectForCommonPage} />} />
          <Route path="/comics" element={<ComicsList selectForCommonPage={selectForCommonPage} />} />
          <Route path="/:type/:id" element={<CommonPage selectedElem={selectedElem} selectForCommonPage={selectForCommonPage}/>} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </main>
      <img className="bg-decoration-own" src={back} alt="vision"/>
    </div>
  );
}

export default App;
```

Рисунок 3.16 – Компонент App

Основні функції:

– **useState:** зберігає стан selectedElem, який містить ID обраного персонажа або коміксу та його тип (characters або comics). Цей стан передається у компонент CommonPage;

– **selectForCommonPage(id, type):** функція для встановлення вибраного елемента. Використовується при переході на сторінку з розширеною інформацією (CommonPage).

Маршрутизація (react-router-dom):

– **/marvel_app** – головна сторінка з персонажами (MainPage);

– **/comics** – сторінка з коміксами (ComicsList);

– **/:type/:id** – сторінка з деталями про конкретного персонажа чи комікс (CommonPage);

– ***** – сторінка 404 (NotFound).

3.3 Робота вебзастосунку та взаємодія з користувачем

Після запуску вебзастосунку користувач потрапляє на головну сторінку, яка містить логотип, навігаційне меню, список персонажей та компонент із випадковим персонажем (див. рис. 3.17).

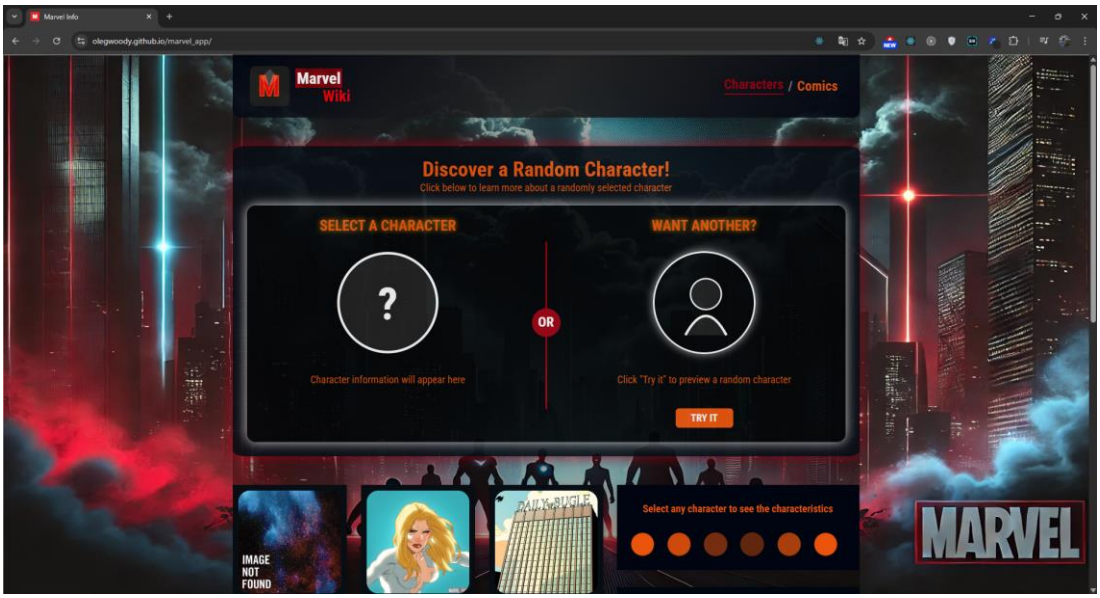


Рисунок 3.16 – Головна сторінка застосунку “Marvel Info”

При натисканні кнопки “Try it” генерується новий випадковий персонаж (див. рис. 3.17). Дані отримуються через запит до Marvel API.

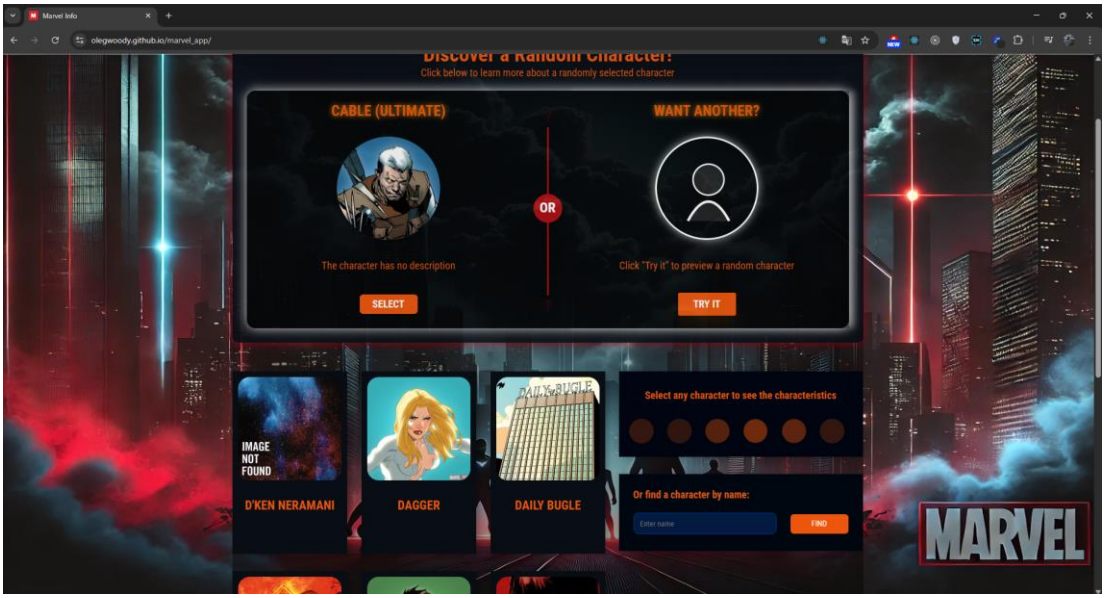


Рисунок 3.17 – Результат генерування випадкового персонажа

Нижче зліва розташований список доступних персонажів (див. рис. 3.18). Користувач може натиснути на будь-якого персонажа, після чого праворуч з’явиться коротка інформація про цього персонажа (див. рис. 3.19).

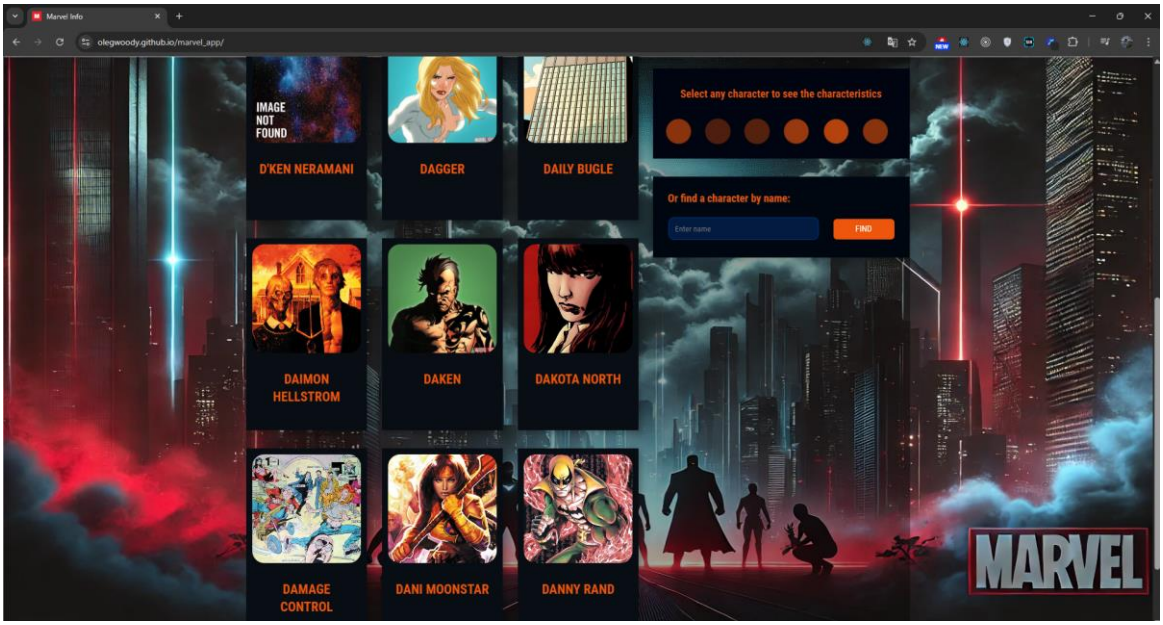


Рисунок 3.18 – Список доступних персонажів

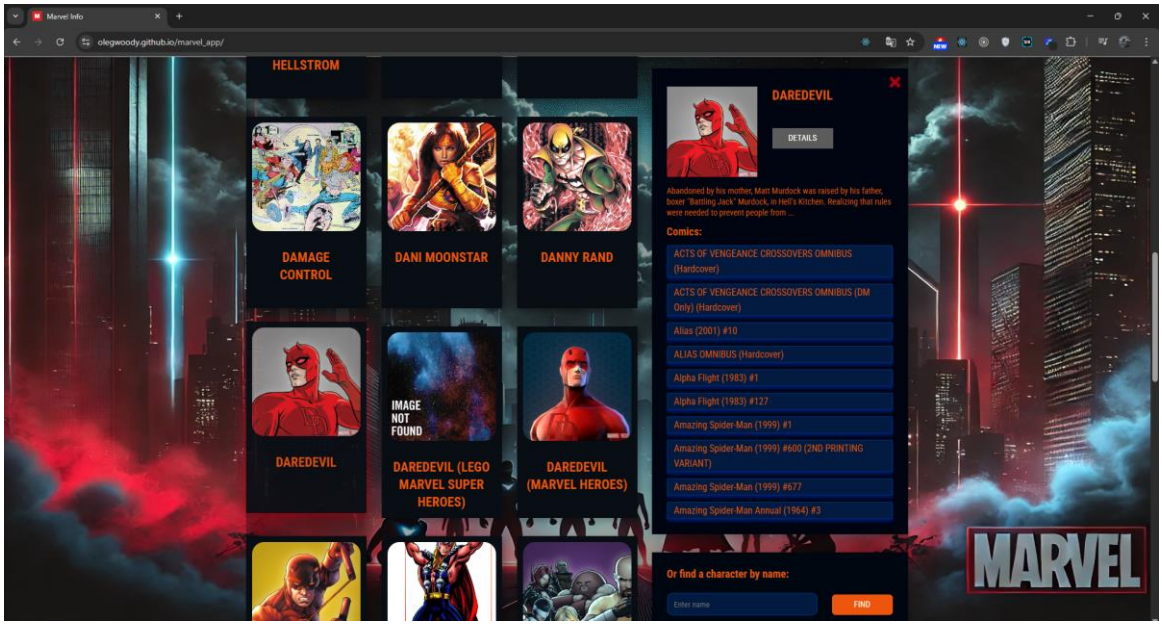


Рисунок 3.19 – Коротка інформація про обраного персонажа

Під інформацією персонажа знаходиться компонент пошуку персонажів за ім’ям. Якщо персонаж знайдений – виводиться посилання на його сторінку (див. рис. 3.20), інакше з’являється повідомлення про помилку (див. рис. 3.21).

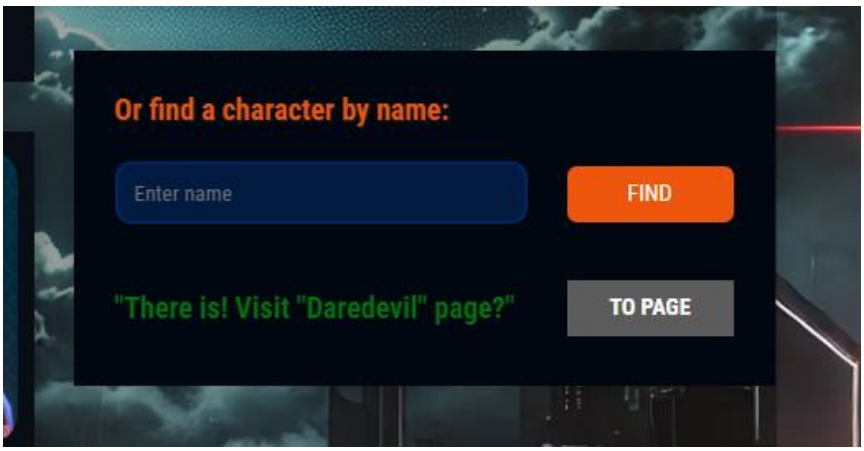


Рисунок 3.20 – Компонент пошуку персонажів із вдалим пошуком

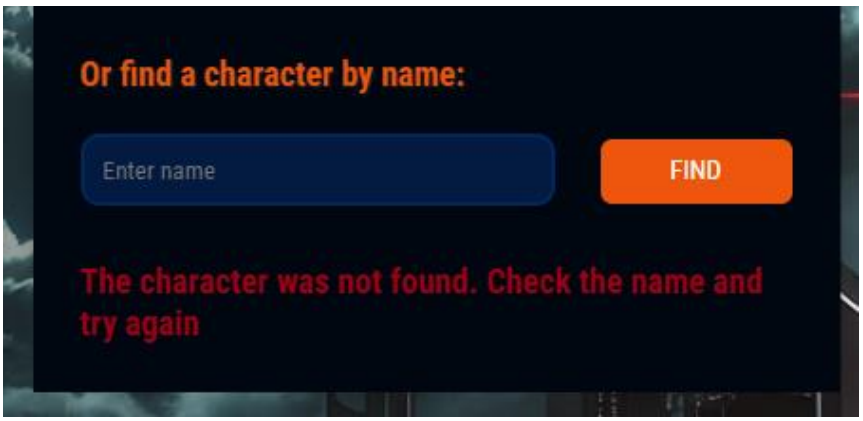


Рисунок 3.21 – Компонент пошуку персонажів із невдалим пошуком

При переході за посиланням у обраного посилання(натисканням на кнопку Details) відкривається окрема сторінка персонажа (див. рис. 3.22).

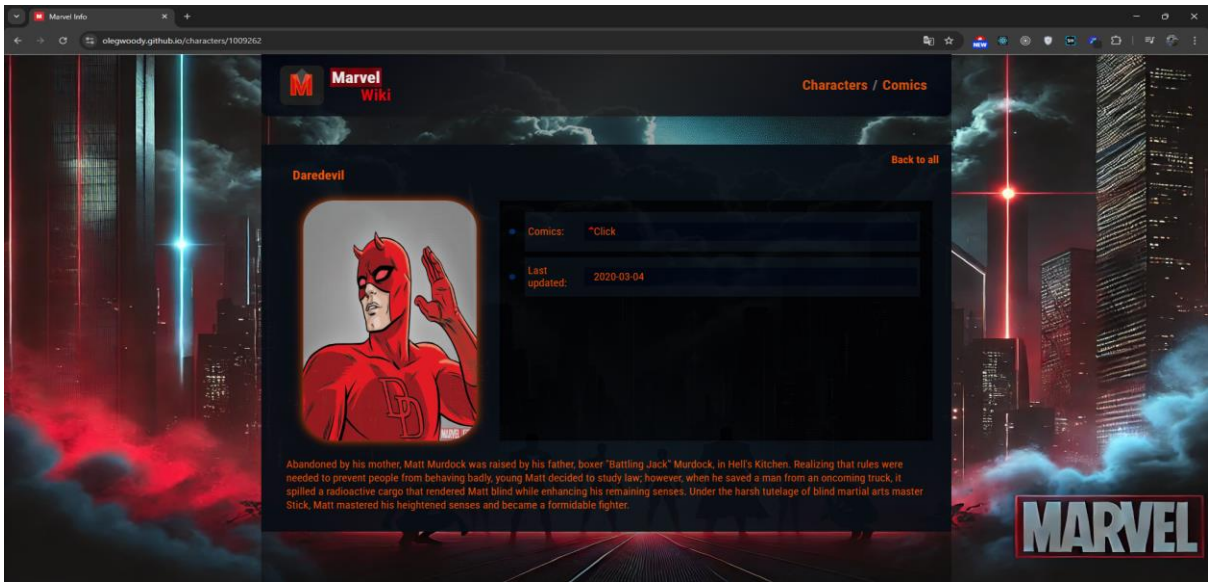


Рисунок 3.22 – Сторінка персонажа

Після натискання кнопки “Click” у компоненті відображається динамічно згенерований список коміксів, в яких з’являється відповідний персонаж (див. рис. 3.23). Користувач має можливість обрати будь-який із запропонованих коміксів, після чого система перенаправляє його на сторінку з детальною інформацією, де розміщено повну інформацію: обкладинка, опис, дата виходу, а також інші супутні дані про обраний комікс (див. рис. 3.24). На цій сторінці користувач також може натиснути кнопку “Click” і згенеровувати список персонажів, які з’являлись у цьому коміксі (див. рис. 3.25).

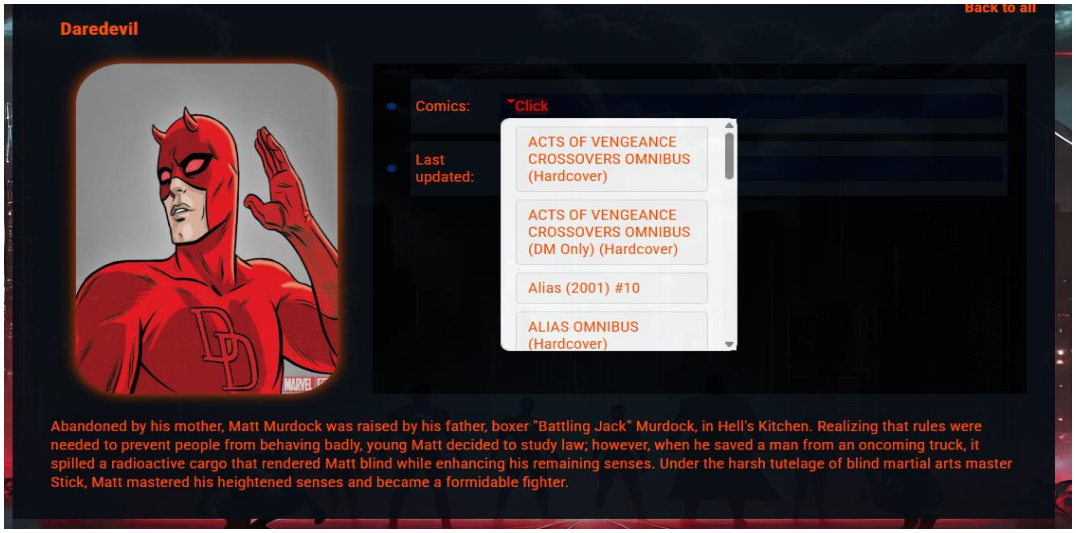


Рисунок 3.23 – Список коміксів з обраним персонажем

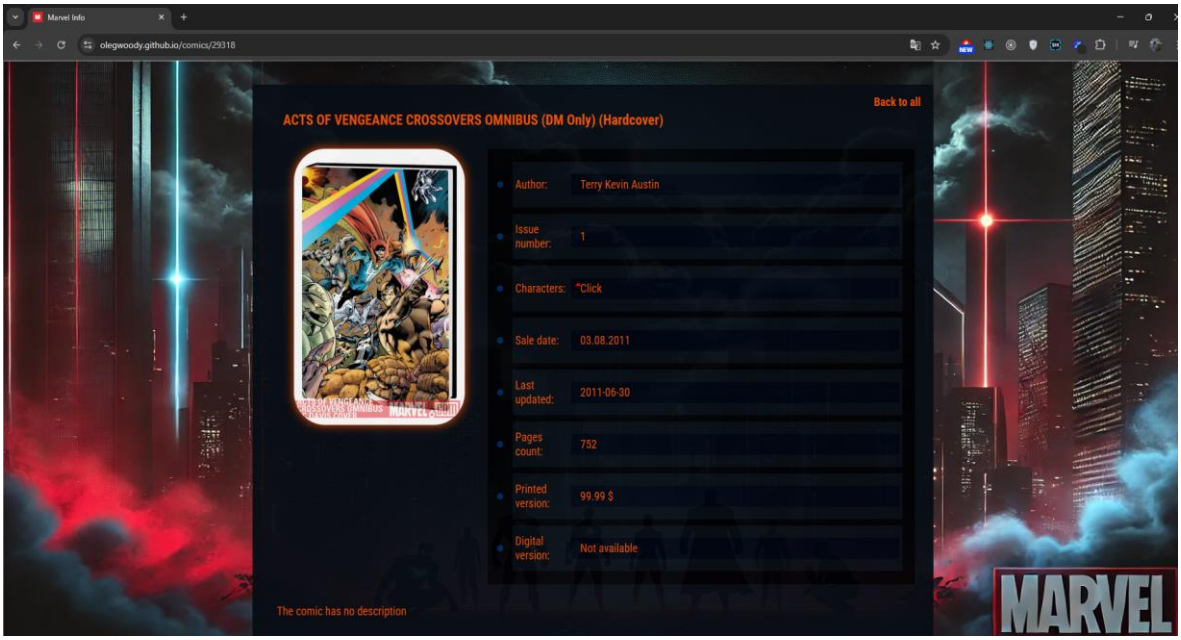


Рисунок 3.24 – Сторінка коміксу

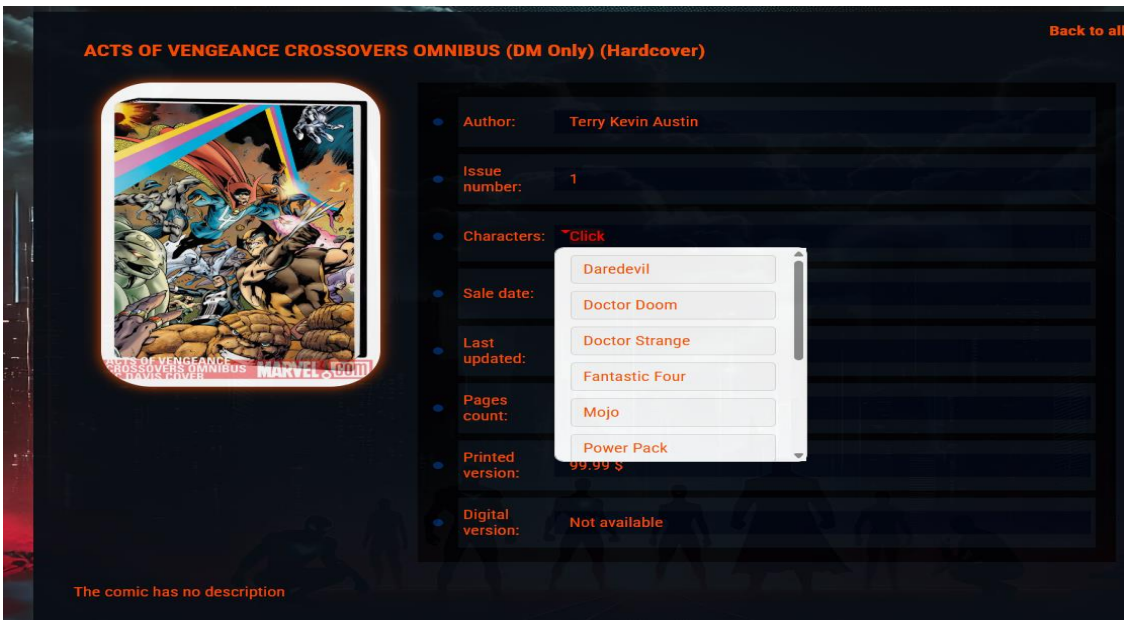


Рисунок 3.25 – Список персонажів у обраному коміксі

У верхній частині інтерфейсу присутня навігаційна панель, що дозволяє швидко переходити між основними розділами застосунку. Зокрема, кнопка “Comics” відкриває окрему сторінку, на якій представлено перелік доступних коміксів (див. рис. 3.26).

Кожен комікс у списку відображається у вигляді картки з назвою, зображенням та коротким описом. При натисканні на будь-який з коміксів користувач перенаправляється на детальну сторінку цього коміксу.

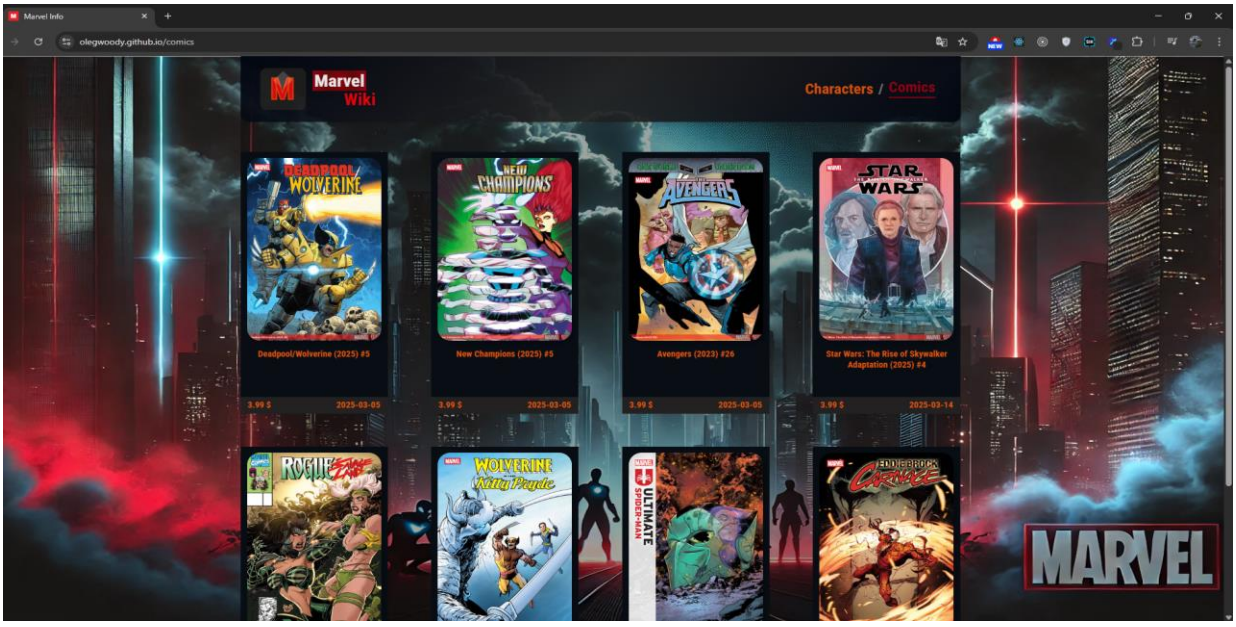


Рисунок 3.26 – Сторінка із списком коміксів

3.4 Висновки до розділу 3

Отже, було реалізовано вебзастосунок для пошуку та перегляду інформації про персонажів і комікси всесвіту Marvel на основі технології **React**. У процесі розробки були створені ключові компоненти, які формують структуру клієнтської частини застосунку:

- **головна сторінка** – містить блок випадкового персонажа, список персонажів, короткий опис вибраного героя та поле для пошуку за іменем;
- **CharList** – відповідає за динамічне завантаження списку персонажів з Marvel API;
- **CharInfo** – відображає детальну інформацію про обраного персонажа: опис, зображення та список коміксів (із можливістю перейти на сторінку обраного коміксу зі списку) у яких він з'являвся;
- **Search** – дозволяє здійснювати пошук персонажа за повним ім'ям із валідацією введених даних [7];
- **RandomChar** – відображає випадкового персонажа з API при натисканні кнопки;
- **ComicsList** – сторінка, що містить перелік коміксів з назвою, банером та можливістю перейти на сторінку обраного коміксу;
- **CommonPage** – універсальна сторінка для відображення розширеної інформації про персонажів та про комікси, залежно від типу вибраного об'єкта.

Також було додано навігацію між сторінками за допомогою бібліотеки React Router, що дозволило реалізувати маршрут переходу між компонентами та головною сторінкою.

Для обробки асинхронних запитів використовувався кастомний хук useRequest, що дозволяє ефективно отримувати дані з API Marvel без використання сторонніх бібліотек.

Додатково реалізовані компоненти Spinner, ErrorMessage та Skeleton для відображення стану запиту – завантаження або помилки.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи на тему «Розробка вебзастосунку засобами React» було досягнуто поставлену мету – створення функціонального вебзастосунку засобами React з використанням Marvel API-сервісу. Реалізований застосунок дозволяє користувачам переглядати детальну інформацію про персонажів та комікси, здійснювати пошук, отримувати випадкових героїв, а також переходити до сторінок з розширеним описом обраних елементів.

В процесі реалізації були вирішені прикладні і технічні завдання:

– вивчено особливості роботи з API-комунікацією у React-додатках через fetch і розроблено кастомний хук useRequest;
– опрацьовано концепцію компонентного підходу та створено багаторазові компоненти: RandomChar, CharList, CharInfo, ComicsList, Search, які забезпечують гнучкість додатку;
– реалізовано маршрутизацію з допомогою React Router v7.2.0, що забезпечує зручну навігацію між сторінками без перезавантаження;
– додан механізм звітування користувача який статус має запит, завантаження чи помилка, за допомогою компонентів Spinner, ErrorMessage, Skeleton;
– додано валідацію введення для пошуку персонажів та адаптацію до різних варіантів відповіді API;
– застосовано SCSS для створення стилізованого, уніфікованого та візуально привабливого інтерфейсу.

У результаті виконаної роботи отримано повноцінний клієнтський вебзастосунок, що відповідає вимогам поставленої задачі. Він поєднує ефективну взаємодію з API та чітку структуру коду, що дозволяє легко підтримувати та розширювати проєкт у майбутньому. Вебзастосунок було розгорнуто за допомогою GitHub Pages [5].

Під час написання роботи було вдосконалено практичні навички роботи з бібліотекою React, опрацьовано принципи розробки односторінкових додатків, написання кастомних хуків, управління станом, а також створення інтуїтивно зрозумілого інтерфейсу.

Отже, результати виконаної кваліфікаційної роботи підтверджують доцільність застосування бібліотеки React у поєднанні з відкритими API для створення сучасних вебзастосунків.

ПЕРЕЛІК ПОСИЛАНЬ

1. React – офіційна документація. URL : https://reactjs.org (дата звернення: 12.04.2025).

2. React Router – офіційна документація. URL : https://reactrouter.com/en/main (дата звернення: 12.04.2025).

3. Marvel Developer API – офіційна документація. URL : https://developer.marvel.com/documentation/getting_started (дата звернення: 12.04.2025).

4. MDN Web Docs – ресурси з HTML, CSS, JavaScript. URL : https://developer.mozilla.org/ (дата звернення: 12.04.2025).

5. GitHub Pages – публікація вебзастосунків. URL : https://pages.github.com/ (дата звернення: 12.04.2025).

6. Sass – офіційна документація з препроцесора стилів. URL : https://sass-lang.com (дата звернення: 12.04.2025).

7. ECMAScript Language Specification (JavaScript стандарт). URL : https://tc39.es/ecma262 (дата звернення: 12.04.2025).

ДОДАТОК А

Код основних компонентів вебззастосунку “Marvel Info”

A.1 ComicsList.js

```
import './comicsList.scss';
import { useEffect, useState, useCallback, useRef } from 'react';
import MarvelService from '../../services/MarvelServices';
import Spinner from '../spinner/Spinner';
import ErrorMessage from '../errorMessage/ErrorMessage';
import { Link } from 'react-router-dom';

const ComicsList = ({ selectForCommonPage }) => {
    const [comics, setComics] = useState([]);
    const [firstLoading, setFirstLoading] = useState(true);
    const { loading, error, getAllComics } = MarvelService();
    const [offSet, setOffSet] = useState(210);
    const type = 'comics';

    const renderedItem = useRef([]);

    useEffect(() => {
        renderedItem.current = [];
        getComics(offSet);
    }, []);

    const getComics = (newOffSet = offSet) => {
        getAllComics(newOffSet)
            .then(comicsLoaded);
    }

    const comicsLoaded = (newComics) => {
        setComics(comics => [...comics, ...newComics]);
        setOffSet(prevState => prevState + 8);
    }

    const onComicsNew = (newOffSet) => {
        getComics(newOffSet);
    }

    const myRef = (e, index) => {
        if (e && !renderedItem.current[index]) {
            renderedItem.current[index] = e;
```

```
            }
        }

        const View = useCallback(({ comics }) => {
            return comics.map((comic, index) => {
                const firstRenderedItem = renderedItem.current[index] ? " : " fade-in";
                return (
                                    <li key={comic.id} ref={(e) => myRef(e, index)}
classNameclassName={`comics__item${firstRenderedItem}`}>
                                     <Link to={`/${type}/${comic.id}`} onClick={() => {
selectForCommonPage(comic.id, type); }}>
                        <div classNameclassName="comics__item-own">
                                                <img src={comic.thumbnail} alt={comic.name}
classNameclassName="comics__item-img" />
                            <div classNameclassName="comics__item-name">{comic.name}</div>
                        </div>
                        <div classNameclassName="comics__item-sub">
                            <div classNameclassName="comics__item-sub-price">{comic.printPrice}</div>
                            <data classNameclassName="comics__item-sub-data">{comic.modified}</data>
                        </div>
                    </Link>
                </li>
                );
            });
        }, []);

        const content = comics.length ? <View comics={comics} /> : null;
        const load = loading ? <Spinner /> : null;
        const err = error ? <ErrorMessage /> : null;

        return (
            <div classNameclassName="comics__list">
                <ul classNameclassName="comics__grid">
                    {content}
                </ul>
                {load}
                {err}
                        <button onClick={() => onComicsNew(offSet)} disabled={loading}
classNameclassName="button button__main button__long">
                    <div classNameclassName="inner">load more</div>
                </button>
            </div>
        );
    }

    export default ComicsList;
```

A.2 CommonPage.js

```
import './commonPage.scss'
import { useEffect, useCallback, useState } from 'react';
import MarvelService from '../../services/MarvelServices';
import ErrorMessage from '../errorMessage/ErrorMessage';
import Spinner from '../spinner/Spinner';
import xMen from '../../resources/img/x-men.png';
import hero_banner from '../../resources/img/hero_banner.png'
import { Link } from 'react-router-dom';

const CommonPage = ({selectedElem, selectForCommonPage}) => {
    const {loading, error, getCharactersById, getCharactersByName} = MarvelService()
    const [elem, setElem] = useState(null)
    const [listStat, setListStat] = useState(false)
    const { type, id } = selectedElem || {};
    const [charId, setChar] = useState()

    useEffect(() => {
        console.log("selectedElem:", selectedElem);
        setListStat(false);
        setElem(null);
        if (id && type) {
            onNewId(id, type);
        }
    }, [id]);

    useEffect(() => {
        const handleClickOutside = (event) => {
            if (listStat && !event.target.closest('.common-page__metadata-first-char')) {
                setListStat(false);
            }
        };

        document.addEventListener('mousedown', handleClickOutside);

        return () => {
            document.removeEventListener('mousedown', handleClickOutside);
        };
    }, [listStat]);

    const onNewId = (id, type = 'characters') => {
            getCharactersById(id, type)
                .then((res)=>{
                    updateElem(res)})
                .catch((error) => {
```

```
                console.error("Error:", error);
            });
    }

    const updateElem = (elem) =>{

        setElem(elem);
    }

    const onClickElementName = (charId, type) => {
        selectForCommonPage(charId, type);
    }

    const CharList = ({ elementId, type }) => {
        if (!elementId || !Array.isArray(elementId) || elementId.length === 0) {
            return <li className="common-page__metadata-first-char-item">Not found</li>;
        }

        return elementId.map(item => {
            const baseCharId = /\/(\d+)/;
            const foundBaseCharId = baseCharId.exec(item.resourceURI);
            const elemId = foundBaseCharId ? foundBaseCharId[1] : null;

            return (
                <li key={item.id || elemId} onClick={() => onClickElementName(elemId, type)}
className="common-page__metadata-first-char-item">
                    {elemId && <Link to={`/${type}/${elemId}`}>{item.name || 'Unknown
Character'}</Link>}
                </li>
            );
        });
    };

    const View = ({elem, type}) =>{
        // if (!elem) return null;
        if (type === 'comics') {
            console.log(type)
            console.log(elem)

            let dateSale = elem.saleDate?.find(item => item.type === "onsaleDate")?.date ||
null;
            if (dateSale) {
                dateSale = new Date(dateSale);
            }

            return (
                <>
                    <h2 className="common-page__name">{elem.name || 'Unknown
```

```
Name'}</h2>
                    <Link to={`/${type}`} className="common-page__back">Back to all</Link>
                     <div className='common-page__wrapper'>
                        <img
                           src={elem.thumbnail || 'placeholder.jpg'}
                           alt={elem.name || 'No image'}
                           className="common-page__img"
                        />
                        <ul className='common-page__metadata'>
                           <li className='common-page__metadata-first'>
                                       Author: <span>{elem.author?.find(item => item.role ===
"writer")?.name || 'Unknown'}</span>
                           </li>
                           <li className='common-page__metadata-first'>
                              Issue number: <span>{elem.issueNumber ?? 'Not available'}</span>
                           </li>
                           <li className='common-page__metadata-first'>
                              Characters: <span>
                                 <a
                                  className={`common-page-button ${listStat ? 'open' : 'closed'}`}
                                   onClick={() => setListStat(listStat => !listStat)}
                                 >
                                    Click
                                 </a>
                              </span>
                              {elem.characters?.length > 0 ? (
                                    <ul className={`common-page__metadata-first-char ${listStat ?
'open' : ''}`}>
                                     <CharList elementId={elem.characters} type='characters' />
                                 </ul>
                              ):(
                                    <ul className={`common-page__metadata-first-char ${listStat ?
'open' : ''}`}>
                                     <li className="common-page__metadata-first-char-item">
                                        N/A
                                     </li>
                                 </ul>
                              )}
                           </li>
                           <li className='common-page__metadata-first'>
                              Sale date: <span>{dateSale ? dateSale.toLocaleDateString("ua-UA") :
"Not found"}</span>
                           </li>
                           <li className='common-page__metadata-first'>
                              Last updated: <span>{elem.modified || 'Unknown'}</span>
                           </li>
                           <li className="common-page__metadata-first">
                              Pages count: <span>{elem.pageCount ?? 'Unknown'}</span>
```

```
                        </li>
                        <li className='common-page__metadata-first'>
                            Printed version: <span>{elem.printPrice ?? 'N/A'}</span>
                        </li>
                        <li className='common-page__metadata-first'>
                            Digital version: <span>{elem.digitalPrice ?? 'N/A'}</span>
                        </li>
                    </ul>
                </div>
                <div className="common-page__info">
                        <p className="common-page__descr">{elem.fullDescription || 'No description available.'}</p>
                </div>
            </>
        );
    } else if(type==='characters'){
        console.log(type)
        console.log(elem)
        return(
            <>
            <h2 className="common-page__name">{elem.name}</h2>
                <Link to={`/marvel_app`} className="common-page__back">Back to all</Link>
            <div className='common-page__wrapper'>
                    <img src={elem.thumbnail} alt={elem.name} className="common-page__img"/>
                <ul className='common-page__metadata'>
                                {/* <li className='common-page__metadata-first'>Author: <span>{elem.author.find(item=>item.role==="writer")?.name}</span></li> */}
                            {/* <li className='common-page__metadata-first'>Issue number: <span>{elem.issueNumber}</span></li> */}
                    <li className='common-page__metadata-first'>
                    Comics: <span><a className={`common-page-button ${listStat ? 'open' : 'closed'}`} onClick={()=>setListStat(listStat => !listStat)}>Click</a></span>
                        <ul className={`common-page__metadata-first-char ${listStat ? 'open' : ''}`}>
                            <CharList elementId={elem.comics} type='comics'/>
                        </ul>
                    </li>
                    <li className='common-page__metadata-first'>
                        Last updated: <span>{elem.modified || 'Unknown'}</span>
                    </li>
                    {/* <li className='common-page__metadata-first'>
                        Sale date: <span>{dateSale ? dateSale.toLocaleDateString("ua-UA") : "Not found"}</span>
                    </li> */}
                            {/* <li className='common-page__metadata-first'>Last updated: <span>{elem.modified}</span></li> */}
```

```
                            {/* <li className="common-page__metadata-first">Pages count:
<span>{elem.pageCount}</span></li> */}
                            {/* <li className='common-page__metadata-first'>Printed version:
<span>{elem.printPrice}</span></li> */}
                            {/* <li className='common-page__metadata-first'>Digital version:
<span>{elem.digitalPrice}</span></li> */}
                        </ul>
                        </div>
                        <div className="common-page__info">
                           <p className="common-page__descr">{elem.fullDescription}</p>
                        </div>
                     </>
                  )
               }
            }
               const err = error ? <><Link to={`/marvel_app`} className="common-
page__back">Back to all</Link><ErrorMessage/></>:null
            const content = elem && !(loading || error) ? <View elem={elem} type={type}/> : err
            return (
               <div>
                  <div className="common-page">
                     {content}
                     {err}
                     {loading?<Spinner/>:null}
                  </div>
               </div>
            )
         }

         export default CommonPage;
```

A.3 CharList.js

```
         import './charList.scss'
         import MarvelService from '../../services/MarvelServices';
         import { useState, useEffect, memo, useRef } from 'react';
         import Spinner from '../spinner/Spinner';
         import ErrorMessage from '../errorMessage/ErrorMessage'

         const CharList = ({selectForInfo, selectedChar}) => {
            const [char, setChar] = useState([])
            const [firstLoading, setFirstLoading] = useState(false)
            const [offSet, setOffSet] = useState(290)
            // const [activeChar, setActiveChar] = useState(null)
```

```
const { getAllCharacters, loading, error } = MarvelService()

const currentRef = useRef([])

useEffect(() => {
   currentRef.current = [];
   updateChar(offSet, true)
}, [])

const updateChar = (offset, initialRender = false) => {
   if (initialRender) {
      setFirstLoading(true);
   }
   getAllCharacters(offset)
      .then(res => {onCharLoaded(res)
      setFirstLoading(false)})
      .catch(() => {
         console.log("Error on getAllCharacters")
         setFirstLoading(false);
      })
}

const onCharLoaded = (newChar) => {
   if (newChar.length === 0) return;
   setFirstLoading(false)
   setOffSet(prev => prev + 9);
   setChar(prevChar => [...prevChar, ...newChar])
}

const onClickChar = (charId, index) => {
   selectForInfo(charId)
   currentRef.current[index].focus()
   currentRef.current[index].scrollIntoView({
      behavior: "smooth",
      block: "start",
   });
   // setActiveChar(charId)
}

const myRef = (e, index) => {
   if (e && !currentRef.current[index]) {
      currentRef.current[index] = e
   }
}

const View = memo(({ char }) => {
   return char.map((char, index) => {
      const animation = currentRef.current[index] ? "" : "char__item_animate"
```

```
                return (
                    <li className={selectedChar === char.id ? "char__item char__item_selected" :
`char__item ${animation}`}
                        tabIndex={index}
                        ref={(e) => myRef(e, index)}
                        onClick={() => onClickChar(char.id, index)}
                        id={char.id}
                        key={char.id}>
                        <img src={char.thumbnail} alt={char.name} style={char.objStyle} />
                        <div className="char__name">{char.name}</div>
                    </li>
                );
            });
        }, [])

        const content = !firstLoading ? <View char={char} /> : null

        return (
            <div className="char__list">
                <ul className="char__grid">
                    {content}
                </ul>
                {error ? <ErrorMessage /> : null}
                 {(loading && !error) ? <div style={{ "gridColumn": "2" }}><Spinner /></div> :
null}
                  <button onClick={() => updateChar(offSet)} className="button button__main
button__long" disabled={loading}>
                    <div className="inner">{loading ? "Loading..." : "Load more"}</div>
                </button>
            </div>
        )
    }

    export default CharList;
```

A.4 CharInfo.js

```
    import './charInfo.scss'
    import skeleton from '../skeleton/skeleton'
    import { useState, useCallback, useEffect } from 'react'
    import MarvelService from '../../services/MarvelServices'
    import Spinner from '../spinner/Spinner'
    import ErrorMessage from '../errorMessage/ErrorMessage'
    import CloseIcon from '../closeIcon/CloseIcon'
```

```
import { Link } from 'react-router-dom';

const CharInfo = ({selectForCommonPage, selectedId, selectForInfo}) =>{
    const [char, setChar] = useState({})
    const [selected, setSelected] = useState(false)
    const [comicId, setComicId] = useState(null)
    const {getComicsByName, getCharactersById, loading, error} = MarvelService()
    const type='characters'

    useEffect(()=>{
        if(selectedId !== null && selectedId !== undefined && selectedId !== ""){
            onCharLoad(selectedId)
            setSelected(true)
        }
    }, [selectedId])

    const onCharLoad = (selected) =>{
        // setLoading(true)
        getCharactersById(selected)
            .then(onCharLoaded)
            .catch(()=>{
                // setError(true)
                // setLoading(false)
            })
    }

    const onCharLoaded = (res) =>{
        setChar(res)
        // setLoading(false)
    }

    const onClickComics = (url) => {
        const searchId= /(\d+)$/
        const myId = searchId.exec(url)
        const comicsId = myId === null ? 0 : myId[1]
        selectForCommonPage(comicsId, 'comics')
    };

    const handleClose = () => {
        setSelected(false)
        setChar()
        selectForInfo()
    };


    const InfoChar = ({char}) => {
        const {id,name,thumbnail,description,wiki, homepage, comics, objStyle } = char
        return(
```

```
                <>
                    <div className="close-icon-wrapper" onClick={handleClose}>
                        <CloseIcon />
                    </div>
                    <div className="char__basics">
                        <img src={thumbnail} alt={name} style={objStyle}/>
                        <div>
                            <div className="char__info-name">{name}</div>
                            <div className="char__btns">
                                <Link
                                to={`/${type}/${id}`}
                                // href={wiki}
                                onClick={()=>selectForCommonPage(id)}
                                 className="button button__secondary">
                                    <div className="inner">Details</div>
                                </Link>
                            </div>
                        </div>
                    </div>
                    <div className="char__descr">
                        {description}
                    </div>
                    <div className="char__comics">Comics:</div>
                    <ul className="char__comics-list">
                        {comics && comics.length > 0 ? comics.slice(0,10).map((comic,index) => (
                            <li  key={index}  onClick={()=>{onClickComics(comic.resourceURI)}}
classNameName="char__comics-item">
                                <Link to={`/comics/${comic.resourceURI.match(/(\d+)$/)?.[1]}`}>
                                    {comic.name}
                                </Link>
                            </li>
                        )) : <li>No comics available</li>}
                    </ul>
                </>
            )
        }

        const err = error ? <ErrorMessage/> : null
        const load = loading ? <Spinner/> : null;
        const skelet = selected ? null : skeleton()
        const content = (!loading && selected && !error) ? <InfoChar char={char}/> : null

        return(
            <div className="char__info">
                {err}
                {load}
                {skelet}
                {content}
```

```
            </div>
        )
    }

    export default CharInfo;
```

A.5 RandomChar.js

```
    import "./randomChar.scss";
    import { useState, useEffect, useCallback } from "react";
    import MarvelService from "../../services/MarvelServices";
    import Spinner from "../spinner/Spinner";
    import ChooseBanner from "../chooseBanner/ChooseBanner";
    import ErrorMessage from "../errorMessage/ErrorMessage";
    import StubAvatar from "../stubAvatar/StubAvatar";

    const RandomChar = (props) => {
        const [state, setState] = useState(null);
        const { getCharactersById, loading, error } = MarvelService();

        const onCharLoaded = (char) => {
            setState(char);
        };

        const updateChar = (id) => {
            getCharactersById(id)
                .then(onCharLoaded)
                .catch(() => {
                    console.log('Error');
                });
        };

        const generatedId = () => {
            const id = Math.floor(Math.random() * (1011800 - 1011000 + 1) + 1011000);
            updateChar(id);
        };

        const altContent = !(state || loading || error) ? <AltView /> : null;
        const errorMessage = error ? <ErrorMessage /> : null;
        const spin = loading && !error ? <Spinner /> : null;
        const content = !(loading || error) && state ? <View char={state} select={props} /> :
null;

        return (
```

```
            <div className="randomchar">
              <h2 className="randomchar__title">Discover a Random Character!</h2>
                <p className="randomchar__subtitle">Click below to learn more about a randomly selected character</p>
              <div className="randomchar__block">
                {errorMessage}
                {altContent}
                {content}
                {spin}
                <div className="randomchar__block-logo">
                  <ChooseBanner />
                </div>
                <div className="randomchar__block-static">
                  <div className="randomchar__block-name">Want Another?</div>
                  <div className="randomchar__block-img-static">
                        <svg width="200" height="200" viewBox="0 0 200 200" fill="none" xmlns="http://www.w3.org/2000/svg">
                              <circle cx="100" cy="80" r="30" fill="#222" stroke="#fff" strokeWidth="3" />
                            <path d="M60 160 Q100 85, 140 160" fill="#222" stroke="#fff" strokeWidth="5" />
                        </svg>
                  </div>
                  <div className="randomchar__block-static-descr">Click "Try it" to preview a random character</div>
                          <button onClick={generatedId} className="button button__main button__glow">
                            <div className="inner">Try it</div>
                          </button>
                </div>
              </div>
            </div>
          );
        };

        const AltView = () => {
          return (
            <div className="randomchar__block-char">
              <div className="randomchar__block-name">Select a character</div>
              <div className="randomchar__block-img">
                <StubAvatar />
              </div>
               <div className="randomchar__block-descr">Character information will appear here</div>
            </div>
          );
        };
```

```
    const View = ({ char: { id, name, description, thumbnail, objStyle }, select: { selectForInfo
} }) => {
        let borderRad = { borderRadius: "50%" };
        if (objStyle?.objectFit === 'fill') {
            borderRad = { borderRadius: 0 };
        }

        return (
            <div className="randomchar__block-char">
                <div className="randomchar__block-name">{name}</div>
                <div className="randomchar__block-img">
                    <img
                        className="randomchar__block-img"
                        style={{ ...borderRad, ...objStyle }}
                        src={thumbnail}
                        alt={name}
                    />
                </div>
                <div className="randomchar__block-descr">{description}</div>
                <button onClick={() => selectForInfo(id)} className="button button__main">
                    <div className="inner">Select</div>
                </button>
            </div>
        );
    };

    export default RandomChar;
```

A.6 Search.js

```
import './search.scss';
import { useState, useCallback } from 'react';
import MarvelService from '../../services/MarvelServices';
import Spinner from '../spinner/Spinner';
import ErrorMessage from '../errorMessage/ErrorMessage';
import { Link } from 'react-router-dom';

const Search = ({selectForCommonPage}) => {
    const [char, setChar]=useState(null)
    const [required, setRequired]=useState(false)
    const {loading, error, getCharactersByName} = MarvelService()
    const type='characters'

    const startSearch = (name) =>{
        setChar(null)
        getCharactersByName(name)
```

```
            .then(completeSearch)
            .catch(() => setChar(null));
        }

        const completeSearch = (res) => {
            if (res && res.name) {
                setChar(res);
            } else {
                setChar({ name: null });
            }
        };


        const onSubmitName = (e) => {
            e.preventDefault();
            setRequired(false);
            const searchVal = e.target.name.value;
            if (searchVal.length >= 1){
                startSearch(searchVal);
                e.target.name.value = '';
            } else{
                setRequired(true)
            }
        };


        const Done = ({name, id}) =>{
            console.log(id)
            if (name) {
            return (
                <div className="char__search-result-done">
                    "There is! Visit "{name}" page?"
                        <Link to={`/${type}/${id}`} onClick={()=>{selectForCommonPage(id)}}
className="button button__secondary">
                        <div className="inner">To page</div>
                    </Link>
                </div>
            )
        } else {
            return (
                <div className='char__search-result-bad'>
                    The character was not found. Check the name and try again<br/>
                </div>
            )
        }
        }

        const Empty = () =>{
            return (
            <div className='char__search-result-bad'>
                This field is required
            </div>
```

```
            )
        }

        const content = !loading && char ? <Done name={char.name} id={char.id} /> : required
? <Empty /> : null;

        return (
            <div className="char__search">
                                        <form      action=""      className="char__search-form"
onSubmit={e=>onSubmitName(e)}>
                    <label htmlFor="char__name-input" className="char__name-label">
                        Or find a character by name:
                    </label>
                    <div>
                        <input
                            type="text"
                            id="char__name-input"
                            name="name"
                            className="char__name-input"
                            placeholder="Enter name"
                        />
                        <button type="submit" className="button button__main">
                            <div className="inner">Find</div>
                        </button>
                    </div>
                </form>
                {content}
                {loading?<Spinner/>:null}
                {error?<Done/>:null}

                </div>
        );
    }

    export default Search;
```

A.7 http.hook.js

```
    import { useState, useCallback } from "react";

    const useRequest = () =>{
        const [loading, setLoading] = useState(false)
        const [error, setError] = useState(false)

    const request = useCallback(async (url, method = "GET", headers = { "Content-Type":
"application/json" }, data = undefined) => {
        setLoading(true)
```

```
    setError(false)
    try{

        let res = await fetch(url, {
            method: method,
            headers,
            body: JSON.stringify(data)
        });

        if (!res.ok) {
            setError(true)
            const errorText = await res.text();
            throw new Error(`HTTP ${res.status}: ${errorText}`);
        }

        return await res.json()

    } catch (error) {
        setError(true)
        throw new Error(error.message);
    } finally {
        setLoading(false)
    }}, [])

    return {request, loading, error}
}

export default useRequest;
```

A.8 App.js

```
import back from '../../resources/img/own_bg.webp';
import AppHeader from '../appHeader/AppHeader';
import ComicsList from '../comicsList/ComicsList';
import NotFound from '../notFound/NotFound';
import { useState } from "react";
import CommonPage from '../commonPage/CommonPage';
import { Route, Routes } from 'react-router-dom';
import MainPage from '../pages/MainPage';

function App() {
  const [selectedElem, setSelectedElem] = useState(null);

  const selectForCommonPage = (id, type = 'characters') => {
```

```
        // setSelectedElem(null);
        setSelectedElem({ id, type });
      };

      return (
        <div className="app">
          <AppHeader />
          <main>
            <Routes>
                                    <Route path="/marvel_app" element={<MainPage
selectForCommonPage={selectForCommonPage} />} />
                                      <Route path="/comics" element={<ComicsList
selectForCommonPage={selectForCommonPage} />} />
              <Route path="/:type/:id" element={<CommonPage selectedElem={selectedElem}
selectForCommonPage={selectForCommonPage}/>} />
              <Route path="*" element={<NotFound />} />
            </Routes>
          </main>
          <img className="bg-decoration-own" src={back} alt="vision"/>
        </div>
      );
    }

    export default App;
```

A.9 MainPage.js

```
    import CharInfo from "../charInfo/CharInfo";
    import CharList from "../charList/CharList";
    import RandomChar from "../randomChar/RandomChar";
    import ErrorBoundary from '../errorBoundary/ErrorBoundary';
    import Search from "../search/Search";
    import './mainPage.scss'
    import { useState } from "react";

    const MainPage = ({selectForCommonPage}) =>{
      const [selectedChar, setSelectedChar] = useState(null)

      const selectForInfo = (id) =>{
        setSelectedChar(id)
      }

      return (
        <>
```

```
                                        <RandomChar selectForInfo={selectForInfo}
selectForCommonPage={selectForCommonPage}/>
                <div className="char__content">
                  <CharList selectForInfo={selectForInfo} selectedChar={selectedChar}/>
                  <ErrorBoundary>
                    <div className="char__info__search">
                            <CharInfo selectedId={selectedChar} selectForInfo={selectForInfo}
selectForCommonPage={selectForCommonPage}/>
                    <Search selectForCommonPage={selectForCommonPage}/>
                    </div>
                  </ErrorBoundary>
                </div>
              </>
            )
          }

          export default MainPage;
```