МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ПЕРСОНАЛЬНОГО ПОМІЧНИКА ЗАСОБАМИ REACT ТА NODE.JS»**

Виконала: студентка 4 курсу, групи 6.1260
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та технології
(назва освітньої програми)
О.О. Пипко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(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 – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та технології

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Пипко Олені Олександрівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка персонального помічника засобами React та Node.js

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 01.06.2024

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Розробка проєкту з використанням зазначених засобів

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка персонального помічника засобами React та Node.js

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023

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.	13.02.2024	
2.	Збір вихідних даних.	21.02.2024	
3.	Обробка методичних та теоретичних	06.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	20.03.2024	
5.	Розробка третього розділу.	10.04.2024	
6.	Оформлення та нормоконтроль	24.05.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	22.06.2024	

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

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка персонального помічника засобами React та Node.js»: 56 с., 25 рис., 4 табл., 14 джерел, 1 додаток.

API, АРХІТЕКТУРА БАЗИ ДАНИХ, БІБЛІОТЕКА, ЗАСТОСУНОК, ІНТЕРФЕЙС КОРИСТУВАЧА, JAVASCRIPT, КЛІЄНТСЬКА ЧАСТИНА, КОМПОНЕНТИ, NODE.JS, ОБРОБКА ДАНИХ, ПЕРСОНАЛЬНИЙ ПОМІЧНИК, ПЛАТФОРМА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, REACT, СЕРВЕРНА ЧАСТИНА, СУБД.

Об’єкт дослідження – процес розробки персонального помічника засобами React та Node.js.

Предмет дослідження – архітектура та ключові компоненти цифрового помічника, що базується на платформах React та Node.js із використанням мови програмування JavaScript для клієнтської та серверної частин.

Мета роботи: розробка персонального помічника за допомогою платформ React та Node.js.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проектування, конструювання та ручного тестування програмного забезпечення.

При розробці додатку було проведено аналіз предметної області, обрано та спроектовано архітектуру за допомогою UML у вигляді діаграм варіантів використання, послідовностей, компонентів, розгортання та роботи системи.

Реалізовано серверну частину з використанням Node.js, клієнтська частина розроблена засобами React та його бібліотек.

В результаті роботи отримано програмне забезпеченням для допомоги гравцям в розумінні ігрового контенту.

SUMMARY

Bachelor’s Qualifying Theses «Development of a Personal Assistant Using React and Node.js»: 56 pages, 25 figures, 4 tables, 14 references, 1 supplement.

API, APPLICATION, ARCHITECTURE OF DATABASES, CLIENT-SIDE, COMPONENTS, DATA PROCESSING, JAVASCRIPT, LIBRARY, NODE.JA, PERSONAL ASSISTANT, PLATFORM, REACT, SERVER-SIDE, SOFRTWARE, SQL, USER INTERFACE.

The object of the study is the process of development the personal assistant using React and Node.js.

The aim of the study is development of the personal assistant using React and Node.js platforms.

The methods of research are methods for collecting and analyzing software requirements, methods, for modeling, designing, constructing and manual testing software .

During the development of the application, a domain analysis was conducted, and the architecture was selected and designed using UML in the form of the use case diagrams, sequence diagrams, component diagrams, deployment diagrams and system workflow. The server-side was implemented using Node.js, and the client-side was developed using React and its libraries.

As a result, the software was created to assist players in understanding game content.

ЗМІСТ

ВСТУП

У сучасному світі люди намагаються спростити або автоматизувати багато процесів. Такий спосіб спрощення можна використати у різних сферах життя, де за допомогою автоматизації та різних помічників, планів можна полегшити роботу та підвищити якість продукту. Прикладом є звичайний календарний план, який має багато аспектів і є незамінним помічником для кожного.

Подібного типу помічники можна побачити в звичайній роботі та у незвичних місцях, наприклад, ігрова інформаційна система. В даній роботі розглянуто гру «Genshin Impact», в якій без сучасного помічника тяжко пройти сюжет або завдання у грі. «Genshin Impact» – це захоплююча екшн-RPG з відкритим світом, яка є популярною завдяки своїй графіці, захоплюючому геймплею та загальній доступності на різних платформах, також гра має як одиночний так і мультиплеєрний режими.

Тому, як бачимо, актуальність кваліфікаційної роботи обумовлена широким застосуванням таких помічників та технологій JavaScript на серверній (Node.js) та клієнтській (React) сторонах у розробці сучасних веб-додатків.

Метою даної роботи є розробка цифрового помічника, що демонструє можливості сучасних веб-технологій на прикладі Node.js та React. Дослідження зосереджується на аналізі даних технологій, виборі найкращих практик та підходів до розробки, а також на імплементації конкретного проєкту, який міг би слугувати ефективним помічником у повсякденному житті.

Структурно робота складається зі вступу, трьох розділів та додатків. В першому розділі наведено теоретичні відомості про обрані платформи, а саме React та Node.js, їх переваги та недоліки, ключові аспекти кожної з платформ. В другому розділі детально розглянуто процес розробки помічника, наведено дизайн макет, логічну схему даних інформаційної системи. В третьому розділі розглянуто програмну реалізацію, наведено фрагменти коду та опис роботи.

1 ТЕОРЕТИЧНІ АСПЕКТИ РОБОТИ ВЕБ-ДОДАТКІВ

1.1 Аналіз сучасних засобів створення онлайн сервісів

На сьогодні є досить багато сервісів та перспектив у сфері ІТ, особливо в сфері веб-розробки. Веб-додатки є популярними в наш час, кожен, навіть маленький бізнес, має власний онлайн ресурс для реалізації своєї продукції чи регулювання роботи свого бізнесу. Високий попит на веб-сервіси посприяв швидкому розвитку та створенню різноманітних застосунків, додатків, технологій для легкого створення та експлуатації майбутнього веб-сервісу.

В даній роботі буде розглянуто один із прикладів таких сервісів та платформ, наведено аспекти роботи з такими ресурсами та програмним забезпеченням, яке буде використано.

Будь-який веб-сервіс складається із двох важливих компонентів, які і є його головними складовими: клієнтська та серверна частини. Кожна із них відповідає за свою частину роботи веб-застосунку, які можуть працювати окремо, але некоректно та непередбачувано. Серверна або бекенд частина містить в собі код, який структурує всю інформацію, яку він отримує від бази даних (якщо вона є). Клієнтська або фронтова частина – зрозуміло та зручно демонструє інформацію на сайті, а також надсилає запити до серверної частини, які йдуть від користувача.

Основою задачею серверної частини є пошук, завантаження в базу даних інформації, якщо така база даних реалізована розробником і надано всі дані для доступу до цієї бази даних, а також обробка запитів із клієнтської частини, які безпосередньо пов’язані із самим користувачем та інтерфейсом сервісу.

За допомогою різних бібліотек та запитів, між собою можуть комунікувати різні компоненти веб-додатку та виконувати різні завдання від користувача, якщо, звичайно, є доступна для обробки інформація, інакше користувач або не отримає жодної відповіді на свій запит, або станеться помилка роботи веб-додатка.

Далі буде детально розглянуто програмні засоби для створення веб додатків, які було обрано для проекту, а також виділено їх переваги і недоліки.

1.2 Огляд основних переваг Node.js

Node.js – це потужне середовище виконання, засноване на двигуні V8 від Google Chrome, яке дозволяє виконувати JavaScript на стороні сервера. На даний час платформа досить активно оновлюється та активно використовується в проектах, але як і кожен застосунок в нього є недоліки та переваги [9].

Далі наведемо основні переваги даної платформи, які зумовили її вибір для розробки інформаційної системи. Швидкість Node.js робить її ідеальним вибором для динамічних або інформаційно насичених застосунків. Крім того, Node.js забезпечує високу продуктивність, допомагаючи збільшити швидкість розробки та впровадження проектів. Його активна спільнота забезпечує підтримку та вирішення проблем, що можуть виникнути під час розробки. Node.js дозволяє кешувати окремі модулі, що сприяє покращенню часу відповіді додатка, а також забезпечує вартісну ефективність, допомагаючи зекономити на витратах на ліцензування програмного забезпечення та обслуговування [5, 6].

До недоліків використання платформи Node.js можна віднести: обчислювальні обмеження, нестабільність API, відсутність підтримки бібліотек, оскільки вибір бібліотеки може бути обмеженим, що може створити проблеми [8].

На етапі вивчення технічних засобів для реалізації проєкту було також проаналізовано й інші платформи, такі як Rust, Ruby, Python. Кожну з них проаналізуємо окремо.

Першим додатком, який має схожу сферу роботи є Rust. Переваги цієї платформи полягають у забезпеченні безпеки пам’яті без збирання сміття, що робить Rust відмінним вибором для розробки надійних і безпечних програм. Rust має потужну систему статичних типів, що дозволяє уникнути появи нульових або зайвих вказівників, що сприяє уникненню багатьох типових помилок. Rust

розроблено для продуктивності та безпеки, особливо ця платформа є безпечною для паралелізму. Недоліком цієї платформи є вимогливість компілятора, який перевіряє кожен компонент та всю програму на безпеку та відповідність високим стандартам безпеки, а також Rust має недостатньо зрозумілу документацію, яку буде складно зрозуміти новачкам. Також через нетипові ситуації, які можуть виникнути при компіляції, є велика ймовірність не знайти потрібної інформації в документації, через її складний опис від самих розробників. Отже, через компілятор та недосконалу документацію, безпековий паралелізм (який досить непогано реалізовано і в Node.js) цей технічний засіб і не був взятий для основи серверної частини[14].

Далі коротко розглянемо Python, який є дуже популярним та досить легким для серверної та, частково, клієнтської частини. Цей засіб також має переваги, через які його було розглянуто, а також недоліки, які зумовили відмову від цієї мови програмування. Отже, до переваг Python можна віднести простий синтаксис та велику кількість документації, наявність великої бібліотеки, яка покриває певну кількість завдань і полегшує розробку різноманітних програм під різні задачі. Python можна використовувати не тільки для створення веб додатків, а і для різних типів мобільних чи десктопних додатків. Також він підтримує кросплатформність. До недоліків, які було виявлено при дослідженні даної платформи, можна віднести невелику швидкість обробки даних та їх завантаження в базу даних чи навпаки. Python не завжди ефективно використовує ресурси пам’яті та часто перевантажує її, що є ключовим недоліком, оскільки вимагатиме багато часу на вивантаження даних для веб-застосунку[13].

Наступна платформа, яку було розглянуто це Ruby. Вона є подібною до платформи Python, має схожі недоліки, наприклад, неправильний розподіл ресурсів пам’яті чи обробка даних. Перевагою даної платформи є легкість навчання та інтуїтивність, швидке створення веб-додатків, гнучкість та швидке розгортання проектів. Також Ruby фокусується на простоті та продуктивності. Також, недоліком є хаотична типізація, через яку може бути складно знайти помилки у втраті даних, є також велика вірогідність великої кількості запитів на

один із компонентів веб-сторінки, через що може бути спричинені помилки і не коректне відображення у веб-застосунку[11].

Розглянувши вищезазначені платформи можна зробити висновок, що у кожної з них наявна технічна особливість, яка не відповідає вимогам до проекту. Наприклад, неправильне використання пам’яті, перевантаженість окремих компонентів, в той час як інші не матимуть достатньо ресурсів, вимогливість компілятора та швидкість розгортання проекту, велика кількість навантажень даних, особливо вивантаження до бази даних із декількох репозиторіїв або вивантаження до клієнтської частини даних із бази даних.

Отже, вирішальними факторами, що обумовили вибір Node.js є робота в одному потоці, використовуючи неблокуючі виклики введення/виведення, що дозволяє йому підтримувати десятки тисяч одночасних з'єднань без витрат на перемикання контексту потоку. Дизайн спільного використання одного потоку для всіх запитів робить Node.js високомасштабованим та ефективним, особливо для задач з інтенсивним введенням/виведенням на відміну від Rust, Python та Ruby.

1.3 Огляд основних переваг React

В цьому пункті наведено основні переваги використання React в контексті розробки онлайн помічника для гри «Genshin Impact»[12].

React – це JavaScript бібліотека для створення користувацьких інтерфейсів. Вона надає можливість розробникам створювати веб-додатки з окремих частин, які називаються компонентами, які можна використовувати повторно, в різних частинах додатку. React в основному фокусується на створенні шару представлення (View Layer) додатку [1, 4, 5].

React було обрано через його численні переваги, що роблять його ідеальним вибором для розробки додатків. Цей фреймворк є відносно легким у вивченні та використанні, що дозволяє розробникам швидко опанувати його можливості. Однією з ключових характеристик React є можливість створення

багаторазових компонентів інтерфейсу, що спрощує та прискорює процес розробки. Завдяки використанню віртуального DOM, React значно покращує продуктивність додатків, забезпечуючи швидше відновлення та ефективнішу взаємодію з користувачем [1]. Велика екосистема бібліотек та інструментів, що підтримують React, надає розробникам різноманітні можливості для реалізації їх проектів. Крім того, React дружній до пошукових оптимізацій, що забезпечує кращу видимість додатків у пошукових системах. Його масштабованість дозволяє використовувати його як для малих, так і для великих проектів, роблячи його універсальним рішенням для будь-якого бізнесу [2, 6].

Також в процесі вибору технічного засобу для створення серверної частини було розглянуто такі фреймворки:

а) Angular – потужний фреймворк для створення складних додатків;

б) Vue.js, який є популярним через свою простоту та продуктивність;

в) Svelte: Новіший фреймворк, який компілює код у компактний та зрозумілий JS без фреймворків;

г) Preact, який є меншою та швидшою альтернативою React.

React краще, оскільки виділяється завдяки ефективному Virtual DOM, компонентній архітектурі та потужній підтримці спільноти. Його декларативний синтаксис та одностороннє зв'язування даних спрощують розробку. Крім того, завдяки React Native, він дозволяє розробляти кросплатформні мобільні додатки. Зосередженість React на користувацькому досвіді та простоті розробки також робить його кращим вибором серед розробників.

Тому на основі цього короткого розгляду, використання React для створення проекту було обрано через великі компоненти, які максимально подібні один до одного. При обиранні інших додатків, була би потреба кожного разу створювати, та, можливо, вручну вводити нові компоненти, які окремо довелось б оптимізувати під дизайн інтерфейсу і постійно тестувати кожен елемент для більш ідеальної реалізації та кінцевого результату таким, який він є в дизайн форматі. Тому як бачимо обраний фреймворк дозволяє частково автоматизувати розташування однакових компонентів на сторінці без додаткового втручання.

1.4 Огляд СУБД MySQL

Даний проект буде реалізовано для гри «Genshin Impact», яка має багато різних складових частин, які в свою чергу будуть між собою пов’язані. Щоб уникнути втрати великої кількості частково хаотично пов'язаних елементів під час подальшої роботи та вдосконалення проекту, було розглянуто створення власної бази даних. Також було обрано СУБД, яка найбільше відповідає необхідним критеріям для виконання поставлених завдань.

Наведемо перелік особливостей СУБД MySQL, які зумовили її вибір:

а) простота використання та адміністрування. MySQL відносно проста у вивченні, налаштуванні та підтримці. Вона працює швидко і може бути завантажена та встановлена протягом декількох хвилин [3];

б) виняткова продуктивність: MySQL оптимізована для продуктивності, здатна ефективно обробляти великі обсяги транзакцій і великі набори даних;

в) масштабованість та гнучкість: MySQL може масштабуватися як вертикально, так і горизонтально, щоб впоратися зі зростаючими обсягами даних і навантаженням на користувачів;

г) безпека: MySQL надає надійні засоби безпеки для захисту даних, включаючи контроль доступу, шифрування та можливості аудиту;

д) сумісність: MySQL широко підтримується багатьма мовами програмування, фреймворками та інструментами [2, 7].

Ці особливості роблять MySQL популярним вибором для веб-сервісів, забезпечуючи ефективну обробку даних, масштабованість, безпеку даних. Також наявна оптимізація під різні типи проектів та кількості інформації, яку необхідно зберігати в таблицях.

2 ПРОЕКТУВАННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ

Даний розділ присвячено проєктуванню персонального помічника за мотивами комп’ютерної гри «Genshin Impact» для допомоги гравцям в накопиченні ресурсів для покращення своїх персонажів та орієнтації в ігровому світі під назвою Тейват. Для реалізації проекту буде використано дизайн-макет, який буде створено за допомогою застосунку Figma. Буде сформульовано основні вимоги до системи та наведено аналіз предметної області.

2.1 Аналіз вимог

В даному розділі буде наведено перелік основних функціональних вимог яким має задовольняти помічник.

Основні вимоги до інформаційної системи:

а) наявність головної сторінки з описом;
б) можливість швидкого та простого пересування між різними вкладками (меню сторінок);
в) зрозуміле та приємне відображення контенту у вигляді таблиць у вкладках Character, Weapon, Enemies, Artifacts;
г) наявність можливості динамічного переміщення до різних персонажів безпосередньо в карточці персонажа;
д) можливість перегляду повної інформації відповідно до рівня персонажу, короткий опис та його таланти;
е) використання фільтрів на сторінках Character, Weapon, Enemies, Artifacts за допомогою різних критеріїв (легендарність персонажа, елемент персонажа, тип артефакту, тип зброї і тощо);
ж) Реалізувати можливість пошуку на всіх вкладках, таких як: Character, Weapon, Enemies, Artifacts.

Ці критерії стосуються користувацької частини, реалізацію яких згодом буде продемонстровано в дизайн макеті. Далі буде наведено вимоги до серверної частини та платформи, яку було обрано для використання – Node.js. В основному ці вимоги стосуватимуться використанню бази даних та зв'язку між базою даних та серверною частиною, тому нижче наведено основні та найголовніші з них. Через великий обсяг ігрового контенту та великої кількості інформації, яка пов’язана між собою, є такі вимоги:

а) швидке завантаження інформації до бази даних;
б) створення зв’язку між кожним компонентом та присвоювання імені кожному компоненту відповідно до користувацького шаблону;
в) при перемиканні між вкладками швидке реагування на відображення інформації, та, якщо потрібно, частково її кешувати;
г) обробка великої кількості інформації та її структурування, сортування.

Таким чином, сформульовано функціональні та технічні вимоги до застосунку. Розробка клієнтської частини починається із створення дизайн-макету. Для такого завдання було використано платформу Figma, яка дозволяє швидко створити макет для будь-якого веб-застосунку або мобільного додатку.

В наступному розділі буде представлено декілька макетів веб-сервісу.

2.2 Створення дизайн макету

Головна сторінка веб-сервісу має показати новому користувачеві тип та вміст вебресурсу. Також було додано позиції та кнопки переходу на соціальні сторінки такі як Instagram, YouTube та Facebook. Серед іншого заплановано можливість додавання актуальних новин, проте в даній версії проекту це не буде реалізовано, лише для майбутнього удосконалення продукту. Вигляд головної сторінки наведено на рисунку 2.1.

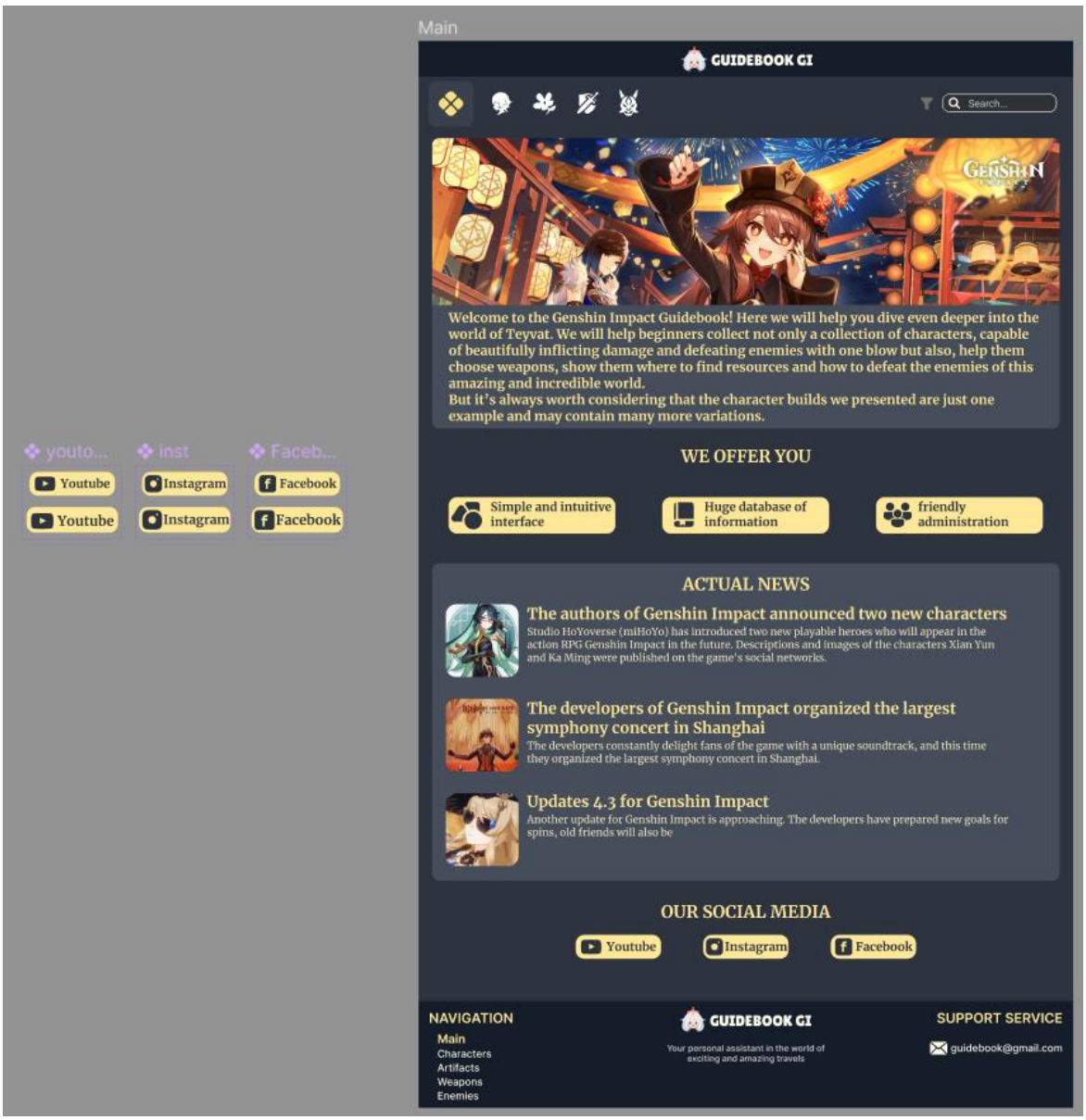

Рисунок 2.1 – Головна сторінка помічника

Як бачимо, реалізовано всі вимоги, які було поставлено вище.

Наступним етапом є розробка дизайну сторінок Character, Weapon, Enemies, які будуть мати схожий інтерфейс, тому тут наведемо тільки один макет з названих вкладок, а саме Character (див. рис. 2.2).

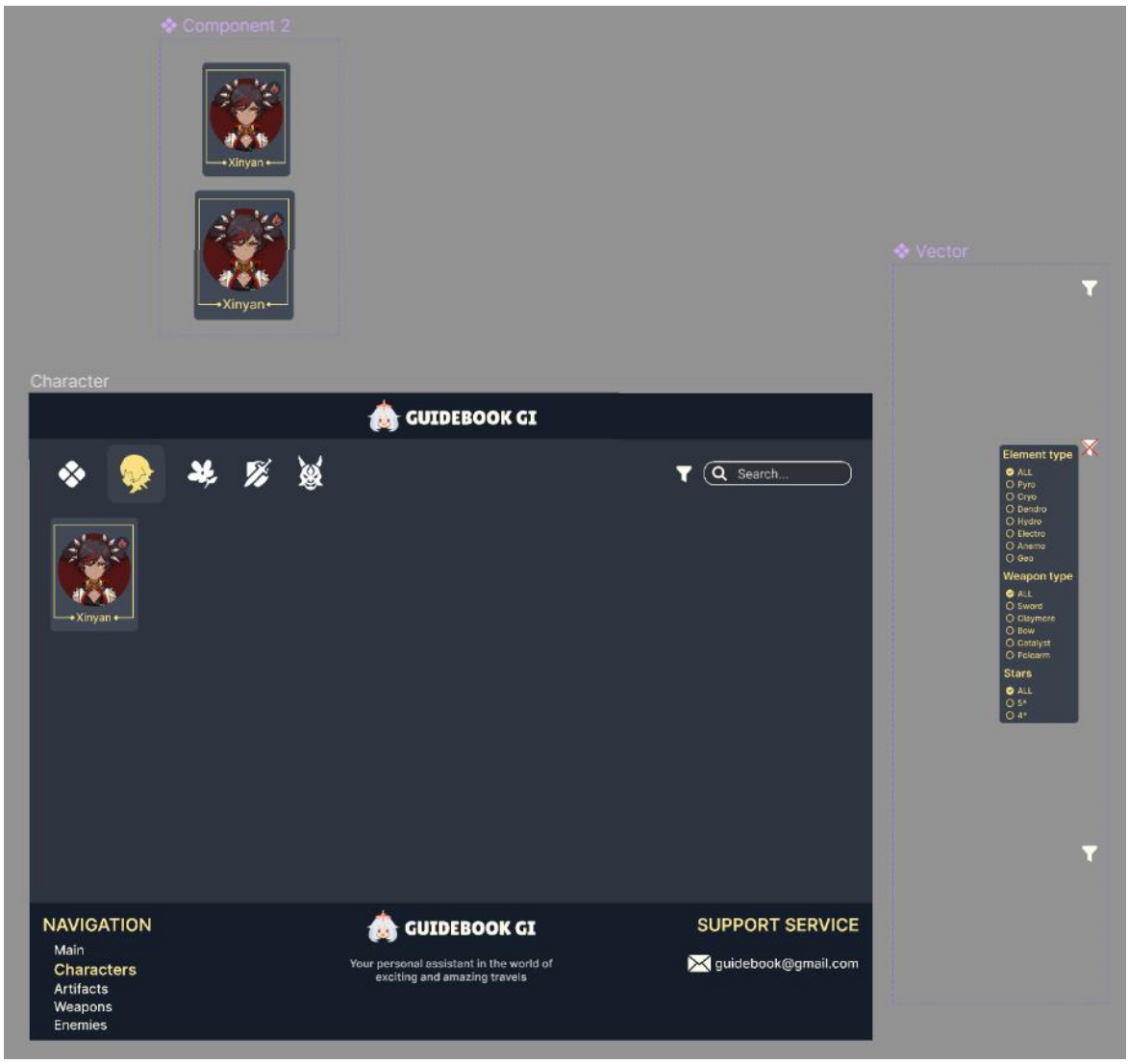

Рисунок 2.2 – Сторінка персонажів

Як бачимо виконано умови щодо зручного та легкого розуміння, таким чином буде влаштовано сторінки Character, Weapon, Enemies у вигляді таблиці з досить крупним візерунком персонажа чи ворога у грі. Також бачимо реалізацію фільтрування та пошуку.

Далі перейдемо до картки самого персонажу та його особливостей (див. рис. 2.3).

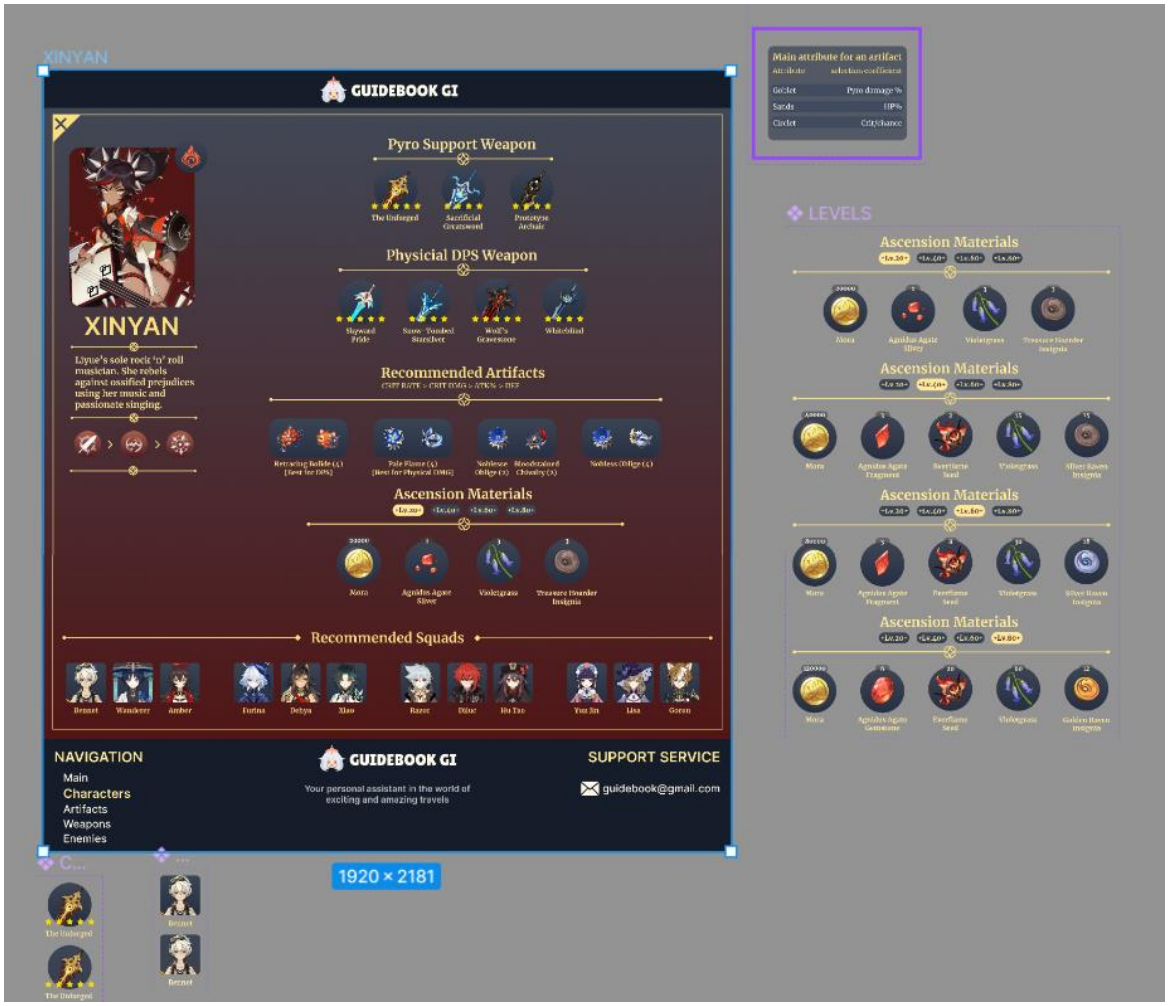

Рисунок 2.3 – Картка персонажу

Як бачимо тут реалізовано сторінку, на якій міститься повна інформація, що стосується конкретного персонажа в грі. При створенні такого макету, було додано такі критерії:

а) показ можливих командних партнерів для даного персонажа;

б) показ відповідних артефактів з додатковою інформацією, щодо конкретизації підібраних артефактів;

в) зброя, яка найкраще підходить даному персонажу.

Останнім компонентом, який є досить важливим у грі, є артефакти для персонажа. Вони є основним фактором поліпшення персонажу, мають свої особливості, кожен такий артефакт є унікальною колекцією із 5 елементів, тому постає задача максимально просто і при цьому легкодоступно відображати всю потрібну інформацію для досить великого компонента.

Цю проблему можна вирішити показавши лише один елемент із колекції, що порушує головний критерій в зручному візуальному пошуку компонентів для користувача. Тому, було вирішено показати всі типи елементів однієї колекції під одним атрибутом у таблиці.

Реалізацію вкладки Artifacts представлено на рисунку 2.4.

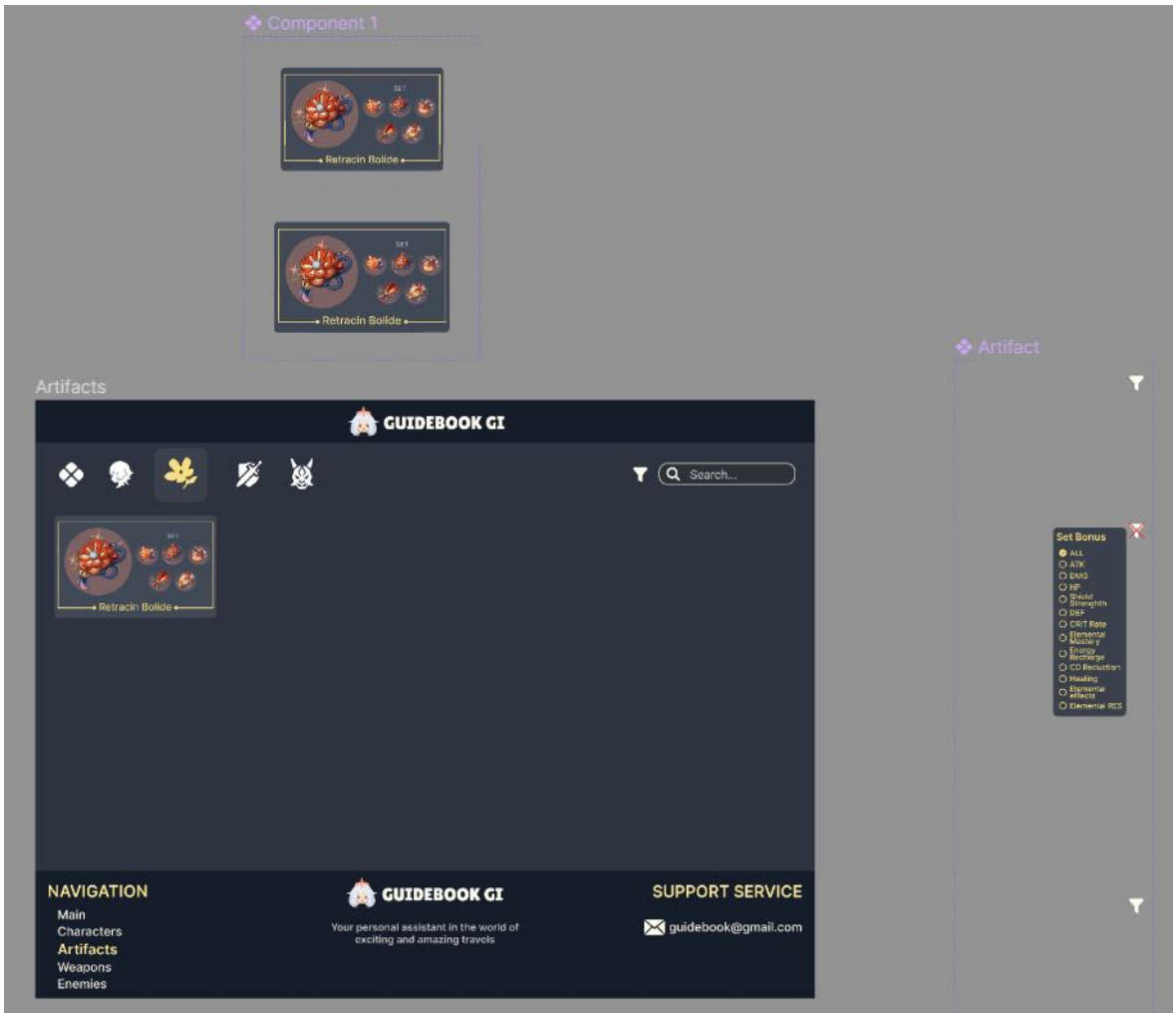

Рисунок 2.4 – Сторінка артефактів із 5 елементами в одній колекції

Як бачимо, на сторінці буде показано всю колекцію. Кожна колекція містить 5 позицій, їх всі видно на сторінці одразу, що дозволить реалізувати швидкий візуальний пошук для користувача.

Далі буде показано картку артефактів, для якої також важливо показати бонуси, які дає колекція, наприклад, у випадку, коли на одному персонажі буде 2 з 5 або 4 з 5 елементів (див. рис. 2.5).

Рисунок 2.5 – Сторінка однієї колекції артефактів

Як бачимо передбачено короткий опис кожного елемента з колекції та його характеристик. Також було додано поле про назви персонажів, для яких дана колекція артефактів найкраще підходить.

2.3 Діаграма прецедентів

Далі буде представлено діаграму прецедентів, завдяки якій буде продемонстровано можливі варіанти використання системи. Маємо два актори: користувач та помічник.

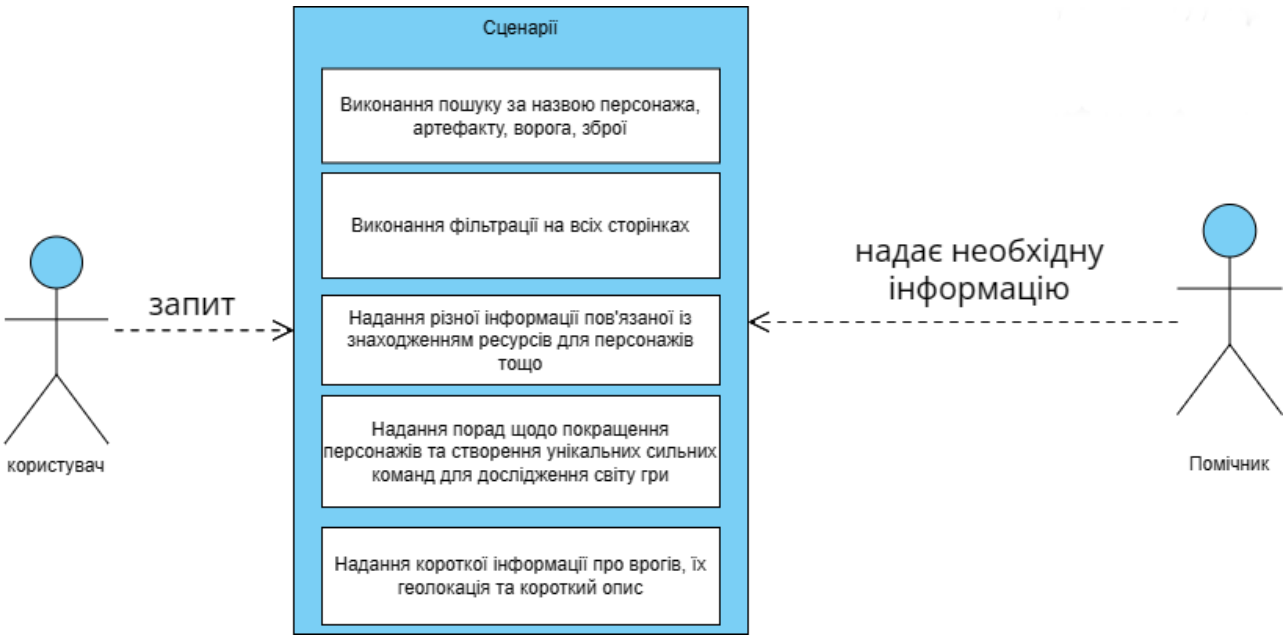

Рисунок 2.5 – Діаграма прецедентів

Як видно з рисунку 2.5 від користувача надходить запит на отримання необхідної інформації з боку помічника.

Отже, розглянемо більш детально варіанти використання системи.

а) пошук інформації про персонажів та предмети:

Актори: Користувачі, помічник.

Сценарій:

– користувач може виконати пошук персонажів за ім’ям або властивостями;
– користувач може знайти інформацію про предмети, такі як зброя, артефакти, рецепти тощо.

б) отримання інформації щодо персонажів-ворогів:

Актори: Користувачі, помічник.

Сценарій:

– користувач може знайти потрібного персонажа-ворога та побачити точне його розташування завдяки інтерактивній карті;
– користувач може знайти розташування у картці ворога та за назвою регіону переміститись на потрібне місце.

в) отримання інформації щодо персонажа та його можливого поліпшення:

Актори: Користувачі, помічник.

Сценарій:

– користувач може за допомогою картки персонажу дізнатись всю необхідну інформацію щодо покращення персонажа;
– користувач може дізнатись, які саме персонажі найкраще підходитимуть для персонажа користувача.

г) здійснення фільтрації на вкладках:

Актори: Користувачі, помічник.

Сценарій:

– користувач відфільтровує персонажів, ворогів, артефакти чи зброю відповідно своїх вподобань;
– користувач може використати додатковий фільтр для більш точного пошуку інформації.

Отже в інформаційній системі можливі 2 види сценаріїв, які повністю вирішують головне завдання проекту: знайти та надати потрібну інформацію максимально швидко та зрозуміло.

2.4 Діаграма компонентів

Розглянемо діаграму компонентів інформаційної системи (див. рис. 2.6).

Наведемо опис представленої діаграми компонентів.

а) компоненти системи:
   1) гра «Genshin Impact»: основний компонент, який містить в собі всю логіку гри, графіку, механіку бою, завдання, персонажів тощо;
   2) сервер: відповідає за обробку запитів від клієнтів, зберігання даних гри, авторизацію користувачів;
   3) база даних: зберігає інформацію про гравців, персонажів, предмети, завдання, досягнення тощо;

Рисунок 2.6 – Діаграма компонентів

4) клієнтська частина (веб-сервіс): веб-інтерфейс, через який користувачі можуть взаємодіяти з грою. Тут вони можуть переглядати статистику, керувати персонажами, виконувати завдання тощо;

б) взаємодія між компонентами:

1) клієнтська частина взаємодіє з сервером через API-запити;
2) сервер обробляє запити, взаємодіє з базою даних для отримання/збереження інформації;
3) база даних зберігає дані про гравців, персонажів, предмети тощо.

На цьому етапі проєктування інформаційної системи, зокрема формулювання вимог, створення дизайну веб-сторінок та побудова діаграм завершена. Наступним етапом є створення бази даних.

2.5 Проєктування бази даних

У першому розділі в пункті 1.3 було наведено опис переваг обраної СУБД, які є ключовими для даного проекту.

Розглянемо структуру бази даних більш детально.

Задача даної бази даних зберігати інформацію таким чином, щоб при надходженні запиту від клієнтської частини, наприклад відкриття картки персонажа, надати структуровано всю важливу інформацію, наприклад картинки самого персонажа, опис його особливостей, список найкращої зброї саме для цього персонажа або перелік матеріалів необхідних для підвищення рівня. Ці всі дані не можуть бути в хаотичному порядку завантажені без конкретного ключа чи помітки, щодо розташування цієї інформації, тому створення бази даних вирішує ці дві проблеми.

Також розглянемо саму ідею створення бази даних до цього проекту.

Сервер отримує запит від клієнтської частини на отримання інформації, задача бази даних в цей час мінімізувати відповідь сервера та максимально структуровано і поступово надавати інформацію із мінімальним часом на обробку. Дана база даних одразу, отримуючи інформацію від сервера, ставить на неї ключі та поєднує одразу до інших компонентів відповідно до архітектури. Задача бази даних надавати інформацію частково на запити від серверу.

Далі на рисунку 2.7 наведено логічну схему даних інформаційної системи, на якій зазначено зв’язки між таблицями, назви таблиць та яка інформація між собою пов’язана.

Кожна вкладка в базі даних (див. рис. 2.7) не просто має свій відділ, але й відповідає за певний аспект інформації, яку планується відобразити для гравців «Genshin Impact». Це може бути все, від даних про персонажів, їхніх здібностей, до інформації про ворогів та ресурси в грі. Важливо, щоб ці вкладки були правильно структуровані та взаємопов’язані, для того щоб забезпечити швидке та ефективне відображення інформації. Крім того, вони є гнучкими, адаптовними до будь-яких змін або оновлень, які реалізовуватимуться у майбутньому в грі «Genshin Impact». Завдяки цьому веб-сервіс завжди буде актуальним та корисним для гравців, оскільки міститиме останні новини серед персонажів та багато іншої інформації.

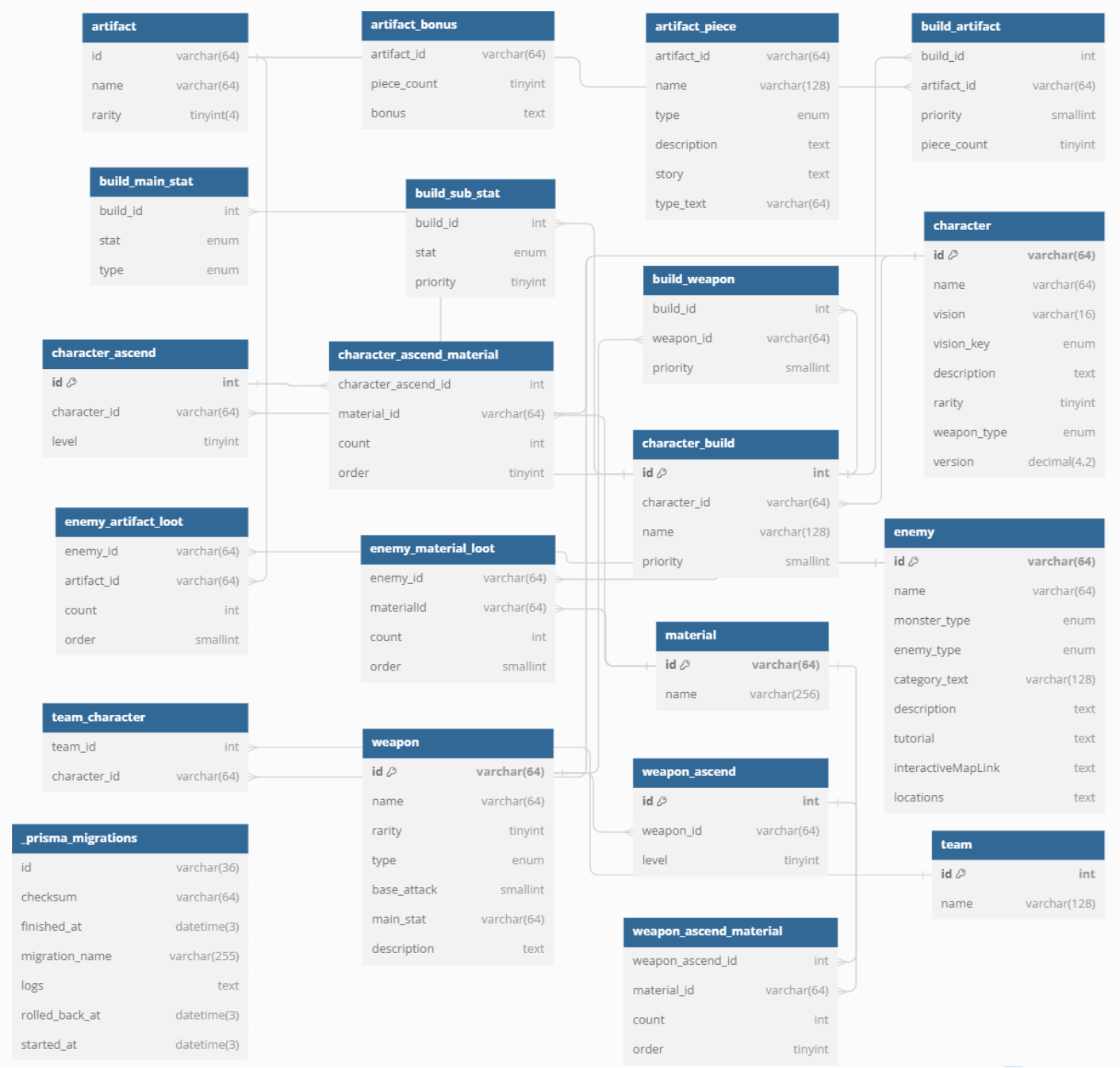

Рисунок 2.7 – Схема бази даних

Розглянемо більш детально сутності представленої схеми бази даних, її атрибути та головні зв’язки.

Головні сутності даної бази даних є: Character, Weapon, Artifact, Enemy. Кожна сутність це окрема сторінка в даному проекті, сторінка також має додаткові сутності для того, щоб відобразити максимальну кількість інформації до кожного персонажу та пов’язати між собою деякі елементи, наприклад

сутність Character має зв’язок із сутністю TeamCharacter, а ця сутність вже з іншою – Team сутністю. Таким чином побудовано майже всю базу даних. Атрибути даних сутностей є схожими і зазвичай складаються з таких типів String, Decimal, Int або назву пов’язаної сутності.

Зв’язки, які використовуються в даній базі даних і показані на схемі, переважно «один-до-одного» та «один-до-багатьох». Наприклад, розглянемо сутність Enemy, де використано два типи зв’язків до різних атрибутів. Атрибути monsterType та enemyType мають зв’язок «один-до-одного» з атрибутами MONSTER_BOSS та BOSS в сутностях MonsterType EnemyType відповідно та атрибути lootMaterials та lootArtifacts мають зв’язок із атрибутами enemy в сутності EnemyMaterialLoot та enemy в сутності EnemyArtifactsLoot.

За допомогою такого легкого вибору меншої кількості зв’язків, можна уникнути втрати даних чи, навпаки, неправильного відображення в конкретній сутності. Також було максимально спрощено більшість сутностей та розділено на декілька сутностей, наприклад Artifact та ArtifactBonus, які на початковому етапі проєктування структури бази даних містились в одній таблиці, але це не дозволяло реалізувати повноцінно інтерфейс та, здебільшого, призводило до втрати даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПЕРСОНАЛЬНОГО ПОМІЧНИКА

На початку роботи, було зроблено перевірку програмного забезпечення, яке буде використано для реалізації додатку.

Проект реалізовуватиметься таким чином, щоб згодом його можна було вивести на конкретний домен та реалізувати через пошукову систему, наприклад Google. Також в подальшому планується створення серії прикладів поліпшення персонажів, надання додаткової інформації щодо потрібних матеріалів, їх місцезнаходження та динамічне оновлення актуальних новин.

3.1 Створення бази даних

Базу даних буде створено за допомогою СУБД MySQL. Перед тим як почати створювати базу даних необхідно перевірити програмне забезпечення, яке має відповідати останній або передостанній версії для коректної роботи. Якщо не встановлено програмне забезпечення, його можна завантажити та встановити через офіційний сайт (всі джерела буде наведено у відповідному розділі) та пройти стандартну покрокову установку програмного забезпечення із створенням шляху до своєї бази даних.

Після того як програма буде встановлена, її треба запустити, якщо вона не запустилась автоматично. Після запуску самої програми можна створити свою базу даних та встановити зв’язок, через який серверна частина матиме доступ до бази даних. Для того щоб створити новий зв’язок, необхідно в головному меню натиснути «+» (див. рис. 3.1), далі відкриється діалогове меню для налаштування нового зв’язку (див. рис. 3.2).

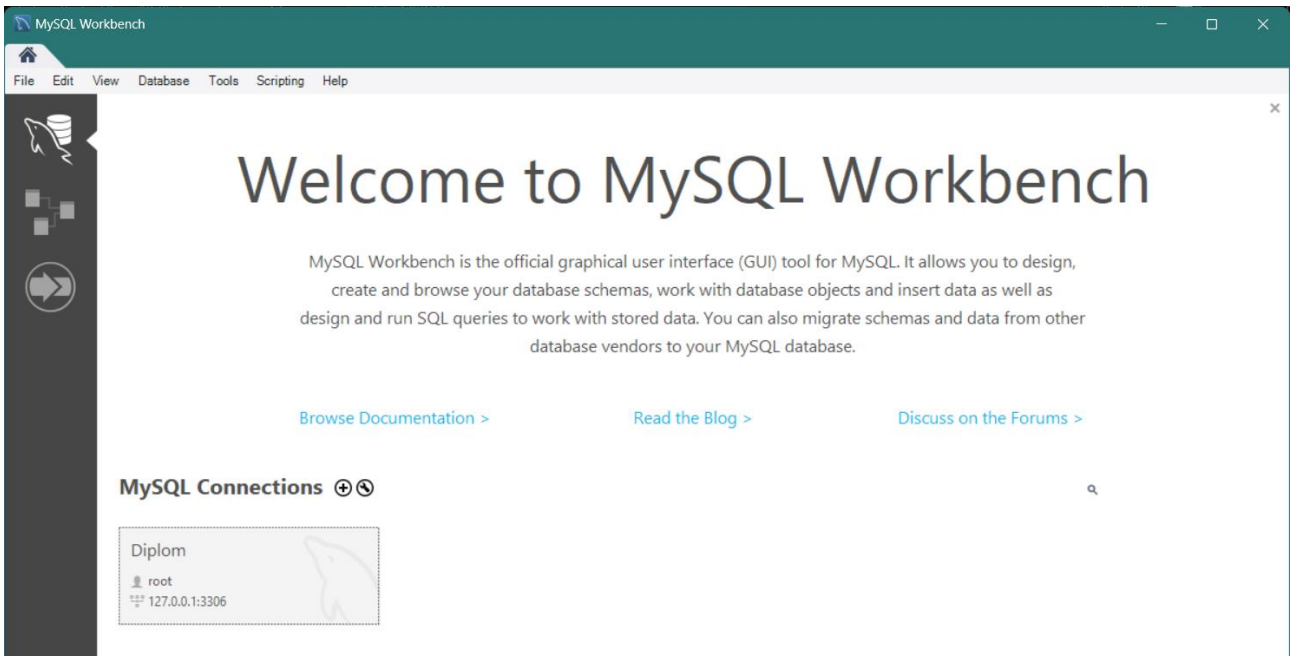

Рисунок 3.1 – Головне меню СУБД MySQL

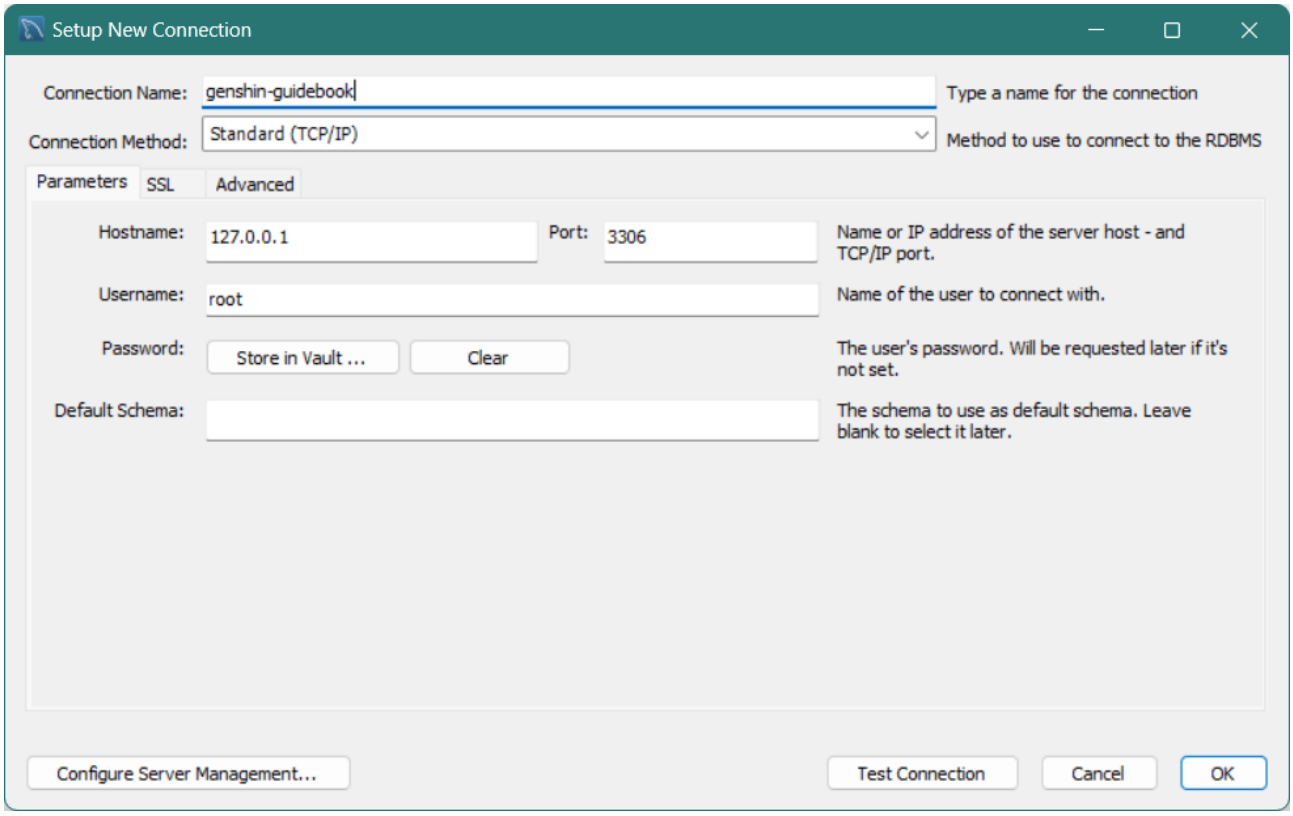

Рисунок 3.2 – Меню створення нового підключення

На рисунку 3.2 можемо побачити налаштування нового зв’язку з базою даних. Далі задаємо своє ім’я для зв’язку, параметри входу, створюємо власний пароль для входу, який потім обо’язково надамо до нашого проекту. Також є

можливість тестового підключення, для того щоб, у разі виявлення якихось невідповідностей між даними, які використовує сервер, та даними для входу, які містяться на сервері бази даних, можна було б редагувати чи змінити дані для входу або інші параметри самого доступу.

В даному проекті буде використано звичайний стандартний тип зв’язку з паролем на назвою зв’язку.

Через те що, проект знаходиться на локальному сервері не обов’язково використовувати додаткові захисні можливості даної СУБД, які вбудовані в неї.

Далі після створення такого зв’язку можна відкрити і переглянути подробиці роботи цієї бази даних, кількість інформації, яку завантажено в базу даних. Ця інформація (див. рис. 3.3) допомагає відслідковувати кількість того, що завантажується в базу даних та вивантажується, також інші додаткові графіки відслідковування роботи бази даних.

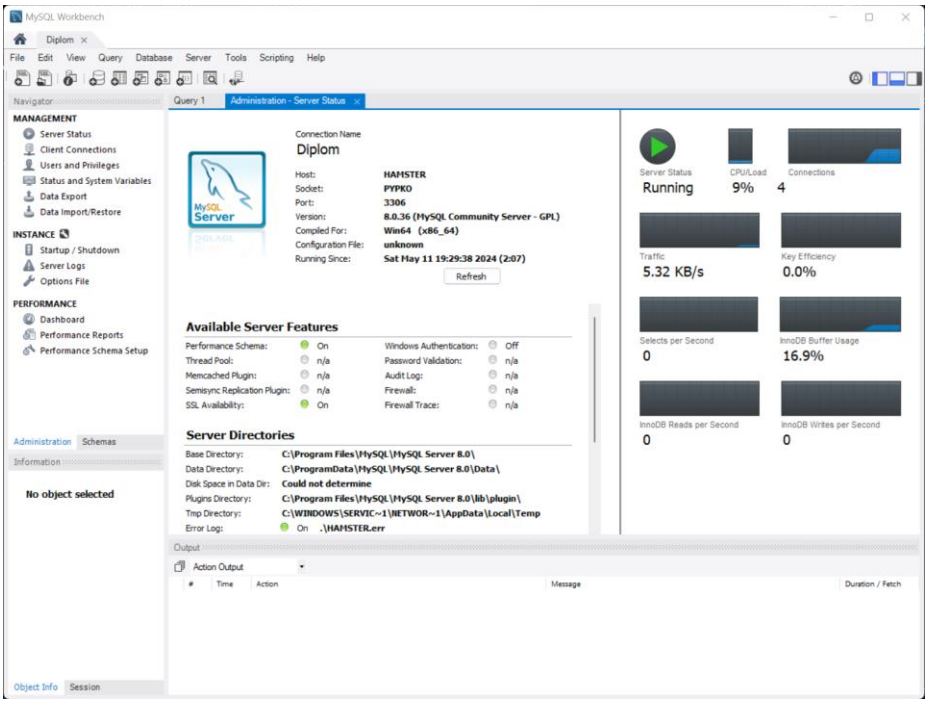

Рисунок 3.3 – Меню сервер статус

Також можна створити базу даних через написання програмного коду, а саме створення таблиць, зв’язки між ними та поєднання інформації, сортування та її вивантаження в разі потреби від клієнтської частини (див. рис. 3.4).

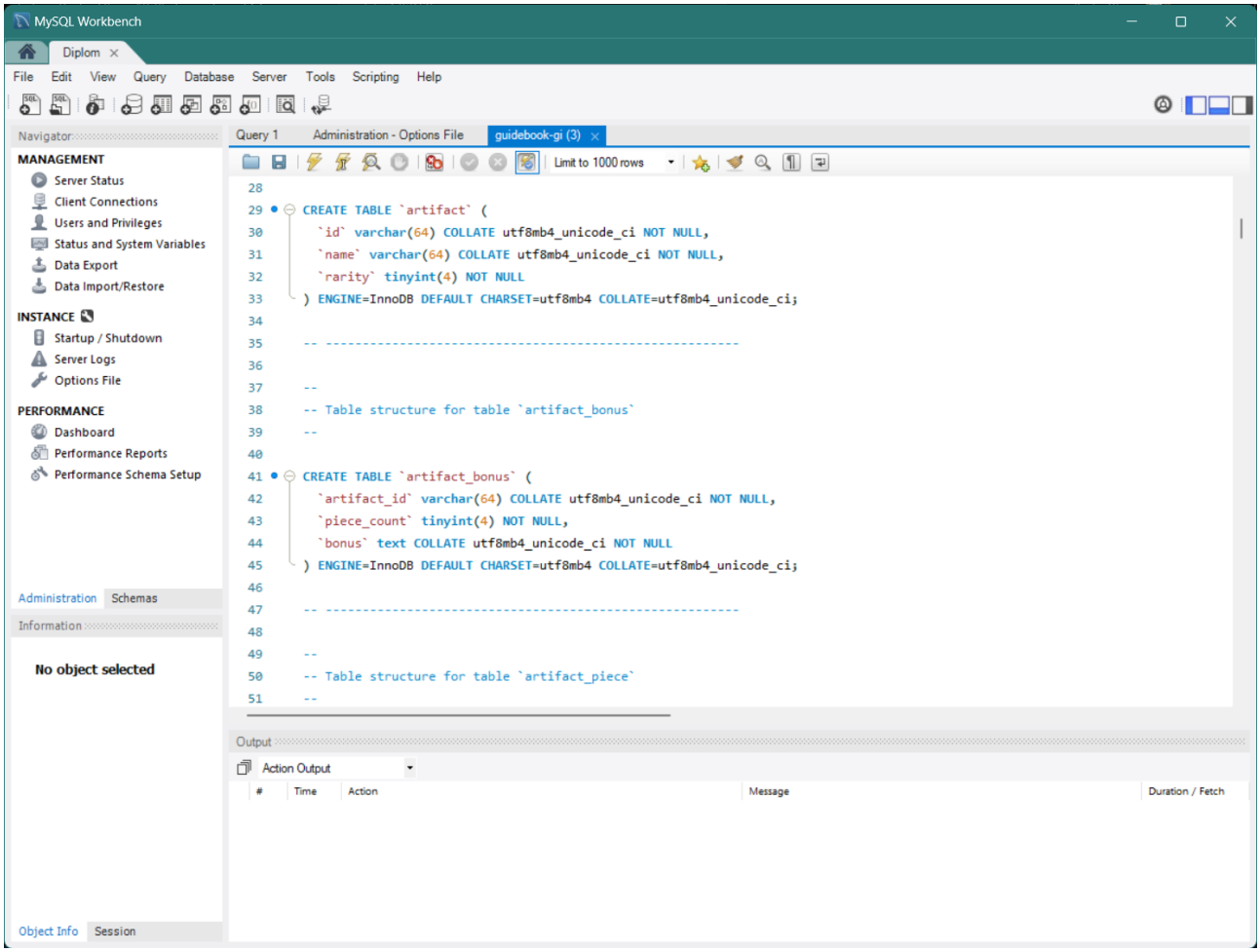

Рисунок 3.4 – Приклад створення таблиць та атрибутів в базі даних

Таким чином будуть створенні всі сутності, які було представлено в другому розділі на рисунку 2.7. База даних є найголовнішою частиною даного проекту, через велику кількість інформації та її структуруванні. Загалом, при розробці схеми, база даних є пріоритетною частиною, яка значно спростить роботу проекта, пришвидшить його роботу та відповіді зі сторони серверної частини на подання конкретної інформації для клієнтської частини, оскільки серверна частина вже не відповідатиме за збереження інформації, за це вже відповідатиме створена база даних.

3.2 Програмування серверної частини персонального помічника

Водночас із створенням бази даних було розпочато створення серверної частини персонального помічника. Головним завданням серверної частини є завантаження із конкретного репозиторію всієї інформації та передавання її у базу даних. Окрім цього серверна частина є посередником між базою даних та клієнтською частиною. Це все було реалізовано за допомогою Node.js.

Далі створюємо проєкт за допомогою таких команд: mkdir «назва_проекту», cd «назва_проекту» (див. рис. 3.5).

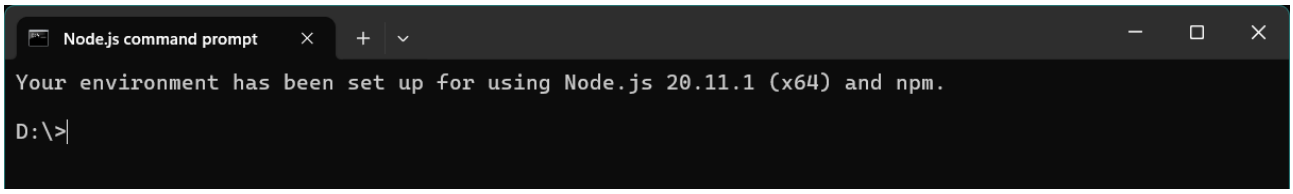

Рисунок 3.5 – Приклад вигляду програми Node.js

За допомогою команд входимо в папку із майбутнім проектом та створюємо в цій папці нашу головну папку, де зберігатиметься ключова інформація про серверну та клієнтську частини нашого помічника і його роботу.

Створюємо папку за допомогою команди «mkdir «назва_проекту»» задаємо назву папки після чого натискаємо «Enter», папка буде створена. Далі за допомогою команди «cd «назва_проекту»» переходимо всередину нашого проекту і тут вже будемо виконувати всі команди для створення клієнтської та серверної частин.

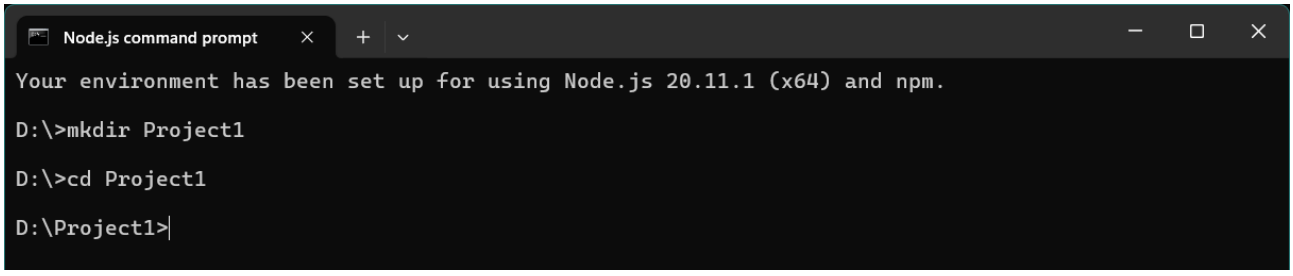

Рисунок 3.6 – Створення проектної папки та вхід через командний рядок

Далі використовуватимемо команди Node.js для створення локального сервера.

Після використання команди «nmp init» буде створено папку «package.json», яка є кореневою частиною серверної частини, далі необхідно визначити декілька параметрів, а саме:

а) **package name**: ім’я проекту (за замовчуванням це назва папки, в якій ініціалізується проект);
б) **version**: версія проекту (починається з 1.0.0);
в) **description**: короткий опис проекту;
г) **entry point**: головний файл проекту, який запускатиметься за замовчуванням (зазвичай **index.js**);
д) **test command**: команда для запуску тестів;
е) **git repository**: URL вашого Git репозиторію, якщо такий є;
ж) **keywords**: ключові слова для опису проекту, які полегшують його пошук;
з) **author**: ім’я автора проекту;
и) **license**: ліцензія проекту (зазвичай **ISC** або **MIT**).

Деякі питання не є обов’язковими і їх можна пропустити або не відповідати, також можна пришвидшити створення проекту скориставшись командою «npm init -y», в результаті опрацювання якої буде автоматично створено проект із стандартними назвами, що допоможе швидко відкрити його та пришвидшить його розгортання.

Після того як проект було створено, можна зайти в головну папку проекту і відкрити файл із назвою «package.json» і побачити всі подробиці проекта, які попередньо було введено або автоматичні дані, в залежності від того, як саме створювалась головна папка. Такий спосіб створення допомагає швидко розгорнути проект та почати роботу з ним, через те що Node.js вже частково сам налаштував роботу локального сервера, отже все, що залишається розробнику це

додавати свої критерії щодо бази даних та клієнтської частини, а саме зв’язок та доступ.

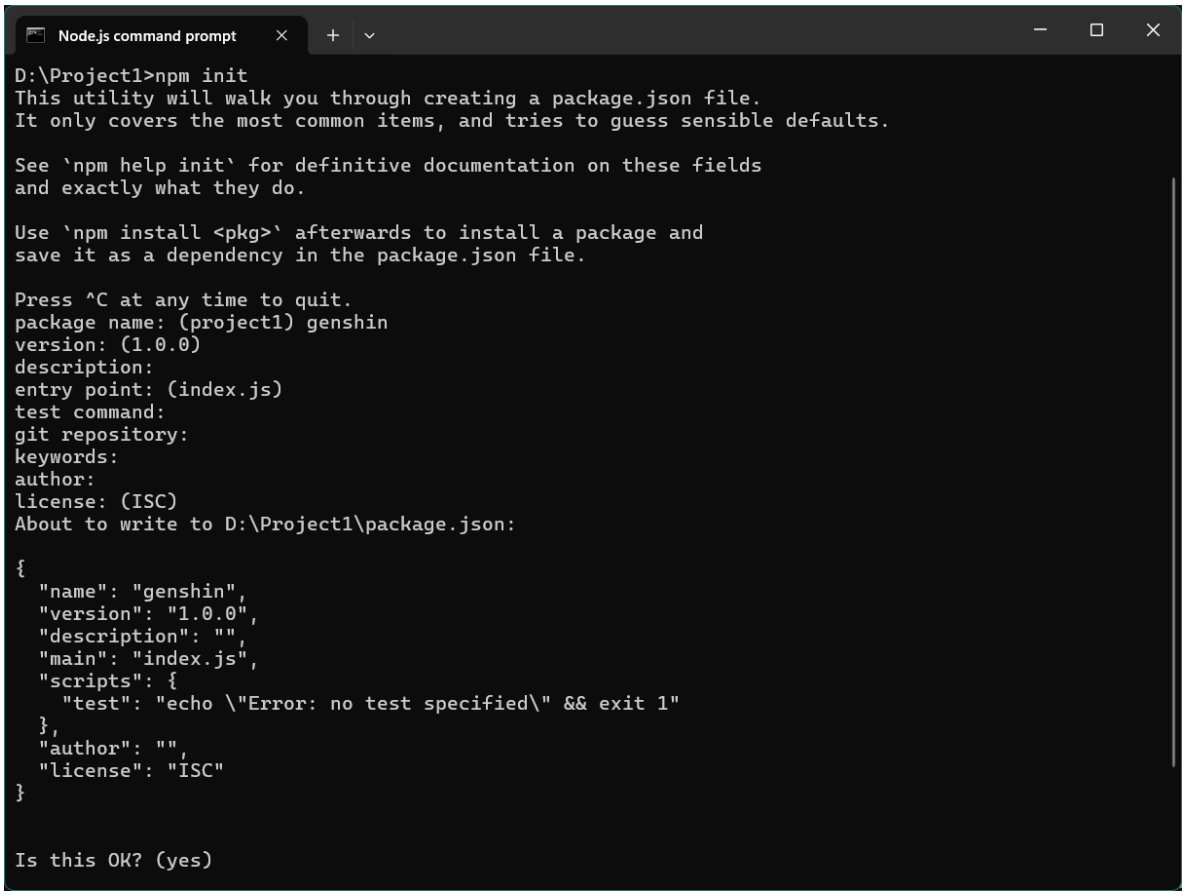

Рисунок 3.7 – Приклад роботи команди «nmp init»

Таким чином було створено серверну частину проекту та вже надано декілька його особливостей. Далі стандартне створення базового опису сервера, представлення доступу до БД та пов’язування всіх трьох компонентів, а саме клієнтської частини, серверної частини та БД, в результаті чого буде створено веб-застосунок персонального помічника.

Далі треба створити вже робочу частину сервера, стандартно це буде виглядати таким чином для нашого проекту (див. рис. 3.8).

В наведеному фрагменті коду (див. рис. 3.8) продемонстровано налаштування серверної частини нашого проекту, в даному коді проводяться такі операції: імпорт необхідних модулів, створення та налаштування сервера Express, обробка помилок та запуск самого серверу. Як бачимо цей код є

головною реалізацією серверної частини, а також є основною частиною роботи всього проекту.

```
Import { handler } from «@api/handler»;
import { endpoints } from «@api/routers»;
import * as configure from «@api/configure»;
import express from «express»;

const server = express();
configure.serverBefore?.(server);
const { PORT = 3000, PUBLIC_DIR = «import.meta.env.PUBLIC_DIR» } =
process.env;
server.use(«import.meta.env.BASE», express.static(PUBLIC_DIR));
server.use(«import.meta.env.BASE_API», handler);
configure.serverAfter?.(server);
server.on(«error», (error) => {
  console.error(`Error at http://localhost:${PORT}import.meta.env.BASE`, error);
  configure.serverError?.(server, error);
});
server.listen(PORT, () => {
  console.log(`Ready at http://localhost:${PORT}import.meta.env.BASE`);
  configure.serverListening?.(server, endpoints);
});
```

Рисунок 3.8 – Приклад коду для серверної частини помічника

Далі створюються допоміжні компоненти для завантаження даних у БД та вивантаження їх, підключення до клієнтської частини, перевірка на вивантаженість всієї інформації, додатково вручну додавання необхідних частин, якщо їх не вистачає з тих чи інших причин в репозиторії, який підключено для вивантаження інформації в базу даних.

3.3 Створення клієнтської частини за допомогою React

Для початкового етапу створення проекту на основі фреймворку React необхідно розуміти структуру проекту, який розробляється. Кожна частина веб-сторінки має відповідати всім елементам на сайті таким чином, щоб дизайн та сайт збігались максимально.

В цій роботі клієнтська частина відіграє найважливішу роль, оскільки це є головним інтерфейсом даного помічника, задача якого полягає в тому, щоб доступно та максимально зрозуміло показати користувачу всю інформацію, яка йому потрібна в той чи іншій вкладці.

Головним є розбиття сторінки сайту на деякі компоненти, які будуть працювати таким чином, що всі компоненти між собою пов’язані та створюють єдиний інтерфейс, який відповідає запланованому дизайну. Отже, після розуміння того, що сайт буде поділений на компоненти, можна сказати частини, треба проаналізувати, які саме частини будуть компонентами.

Переглянувши весь дизайн та його опис, можна одразу побачити декілька головних компонентів на Main web і інших сторінках, які є ідентичними і не змінюються, тому такі компоненти будуть загальними для усього сервісу і матимуть мінімальну кількість коду. Сторінка поділена на секції, а отже, в React відомо створення різних компонентів, які за допомогою самих бібліотек даного React середовища можна імплементувати. Кожна частина сайту складається перш за все з нижньої та верхньої частин сторінки. Якщо уважно переглянути ці два вже визначених компоненти, то буде помітним те, що саме ці два компоненти є на кожній сторінці помічника в однаковому стилі та вигляді, з мінімальними візуальними змінами, такими як позначення, на якій саме сторінці знаходиться користувач, в частині нижня частина сторінки.

Отже розробка клієнтської частини починається із створення файлів та папок і імплементації середовища React. Почнемо з того, що створимо HTML сторінку, в яку вже буде вбудовано наше середовище React та створимо точку

входу для подальшого рендеру в середовище DOM. Код буде виглядати саме таким чином (див. рис. 3.9).

```
<!doctype html>
<html lang="en" data-bs-theme="dark">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Guidebook GI</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Рисунок 3.9 – Фрагмент коду з частини формування веб-сторінки

В даному коді видно, що ми підключаємо кореневу папку, де міститимуться компоненти із використанням React системи. Далі буде перейдено до папки, в якій і відбувається вся робота зі створення клієнтської частини.

Кожен компонент не розглядатиметься окремо, а лише найбільш використані або важливі. Для цих компонентів буде представлено програмну реалізацію та їх роботу в інтерфейсі даного помічника. Раніше було виявлено два важливих компонента, які в дизайн макеті є максимально на всіх сторінках. Найлегшою частиною даних компонентів є їх простота та легкість реалізації згідно вимогам до проєкту зазначеним у другому розділі.

Для нижньої частини сторінки, яка знаходиться в нижній частині сайту та містить інформацію щодо навігаційних посилань по сайту, авторські права, саму назву помічника та текст відповідного характеру (наприклад текст із

підказками), та, за наявності, зв’язок із службою підтримки даного сервісу, наприклад, пошта чи номер телефона, тощо. Нижня частина сторінки має бути в однаковому стилі із однаковою інформацію на всіх вкладках сайту, включає навіть вкладки карток персонажів, артефактів, ворогів.

Нижче буде наведено код (рис. 3.10), який виконує цю роль та створює однакову нижня частина сторінки частину для всіх вкладок.

```
const Footer = ({ tabs, activeTab, handleSelect }) => {
 return <footer className={'d-flex justify-content-between bg-dark'}>
  <div className={'left-col'}>
   <span className={'title-text'}>Navigation</span>
   <ul className={'list list-unstyled'}>
    {tabs.map((tab, index) => {
     const classNames = ['list-item'];
     if (activeTab === tab.name) {
      classNames.push('active');
     }
     return <li
      className={classNames.join(' ')}
      key={index}
      onClick={() => handleSelect(tab.name)}
     >
      {tab.name}
     </li>;
    })}
   </ul>
  </div>
  <div className={'middle-col'}>
```

Рисунок 3.10 – Фрагмент коду створення нижньої частини веб-застосунку, аркуш 1

```
    <div>
      <Image
        alt=''
        src='/logo.svg'
        width={78}
        height={73}
      />
      GUIDEBOOK GI
    </div>
    <span>Your personal assistant in the world of exciting and amazing
travels</span>
  </div>
  <div className={'right-col'}>
    <span className={'title-text'}>SUPPORT SERVICE</span>
    <div className={'d-flex align-items-center justify-content-center'}>
      <Image src={MailIcon} alt={'mail'} width={51} height={38}/>
      guidebook@gmail.com
    </div>
  </div>
 </footer>
}
```

Рисунок 3.10, аркуш 2

Як бачимо цей фрагмент коду демонструє роботу середовища React, а саме те, що можна написати один компонент, із зазначенням невеликих додаткових команд, заздалегідь продуманих змін, наприклад зміна кольору та розміру навігаційної частини в залежності від того, яка сторінка наразі використовується (див. рис. 3.11).

На рисунку 3.11 бачимо, що текст має однакову форму, окрім заголовків та назви помічника і короткий опис мети даного помічника, також бачимо навігацію, що на даний момент користувач знаходиться на сторінці Character через зміну кольору та збільшення розміру тексту. На даному етапі нижня

частина сторінки є готовим, далі буде ручне тестування нижня частина сторінки з метою перевірки його наявності на всіх сторінках даного помічника.

Рисунок 3.11 – Навігаційна система в секції Нижня частина сторінки

Після створення нижньої частини сторінки на сайті, було створено верхню частину сторінки, яка містить в собі небагато компонентів, а саме назву даного помічника та незмінну властивість.

Далі буде наведено код (рис. 3.12), який створює верхня частина сторінки однаковим на всіх вкладках на сайті.

```
import { Navbar } from "react-bootstrap";
const Header = () => {
 return <Navbar className={'bg-dark p-0 justify-content-center'}>
   <Navbar.Brand href="/" className={'text-white'}>
     <img
       alt=''
       src='/logo.svg'
       width={78}
       height={73}
     />
     GUIDEBOOK GI
   </Navbar.Brand>
 </Navbar>
}
export default Header;
```

Рисунок 3.12 – Фрагмент коду створення верхньої частини веб-застосунку

Розглянемо код детальніше. В даному коді легко побачити використання бібліотек React, а саме бібліотеки react-bootstrap, яка імпортується та використовується, в даному випадку, для імпортування елементу Navbar, щоб надалі в цей компонент передати всю необхідну інформацію, таку як логотип даного помічника та його назву. Також цей компонент, як і нижня частина сторінки, експортується для всіх вкладок даного веб сервісу.

Результат можна побачити на рисунку 3.13

Рисунок 3.13 – Верхня частина сторінки на головній сторінці меню

Далі буде розглянуто навігаційну частину даного помічника (див. рис. 3.13). В частині навігації за вимогами та дизайн макетом має бути: кнопки для різних типів вкладок, а саме Main, Character, Artifacts,Weapons, Enemies, фільтрація та пошук, які доступні лише на сторінках Characters, Artifacts, Weapons і Enemies. На Main сторінці зазначені функції будуть недоступні, але видимі.

Для реалізації даної частини сайту необхідно додати такі аспекти:

а) зміна кольору та розміру при переході на іншу сторінку;

б) зміна розміру при наведені курсору;

в) відкриття додаткового вікна для фільтрації;

г) пошукове поле.

Щоб реалізувати кожну із цих частин, було вирішено розділити на окремі функціонали до різних файлів для зрозумілості читання коду, для меншої кількості можливих помилок при рендеруванні кнопок та фільтруванні та

зручності використання, при зменшенні часових витрат при можливих візуальних чи функціональних помилках під час тестування.

Роботу було розпочато із створення фільтрації даних, через велику кількість фільтрованих даних та можливу велику кількість помилок, яку було передбачено заздалегідь при формуванні фільтрацій на етапі дизайну.

Отже, початок був з кнопки, яка мала відкривати додаткове вікно із зазначеними критеріями, код виглядає наступним чином:

```
import { Image } from "react-bootstrap";
import CheckboxCheckedImage from "../../../assets/img/checkbox-checked.svg";
import CheckboxImage from "../../../assets/img/checkbox.svg";
import PropTypes from "prop-types";
export  const FilterButton = ({ filterKey, value, children, applyFilter, isFilterApplied
}) => {

 return <button onClick={() => applyFilter(filterKey, value)}>
     <Image src={isFilterApplied(filterKey, value) ? CheckboxCheckedImage :
CheckboxImage} width={18} height={18}/>
   <span>{children}</span>
 </button>
}

FilterButton.propTypes = { //типи властивостей
 filterKey: PropTypes.string.isRequired,
 value: PropTypes.any,
 children: PropTypes.node.isRequired,
 applyFilter: PropTypes.func.isRequired,
 isFilterApplied: PropTypes.func.isRequired
}
```

Рисунок 3.14 – Фрагмент коду створення кнопки фільтру

Дана частина коду використовує бібліотеки React для створення навігаційної панелі за допомогою бібліотеки react-bootstrap для відображення зображень та бібліотеку prop-types для перевірки властивостей, які передаються в конкретний компонент.

Далі формуються вимоги, які саме дії дана кнопка має робити за допомогою вже зазначених властивостей, а саме ключ для ідентифікації фільтра, значення фільтра, дочірні елементи тощо. Також наявна частина коду для перевірки, що фільтр було використано.

Ці всі елементи також мають свої типи, які будуть в подальшому визначати саму фільтрацію при програмуванні функціональності самого фільтру як функції.

Наступним завданням є створення кнопок фільтрів із деякими властивостями і відображення доступних фільтрів, код має такий вигляд як на рис. 3.15.

```
const EntityFilter = ({ availableFilters, isFilterApplied, applyFilter }) => {
 const sortByLabel = (a, b) => {
  if (typeof a.value === 'number' && typeof b.value === 'number') {
   return b.value - a.value;
  }
  let x = a.label.toLowerCase();
  let y = b.label.toLowerCase();
  return x > y ? 1 : (x < y ? -1 : 0);
 }

 return <div className={'filter'}>
  {availableFilters.map((filter, index) => {
   return <div key={index} className={'d-flex flex-column filter-content'}>
```

Рисунок 3.15 – Фрагмент коду створення функції фільтрування, аркуш 1

```
      <span className={'filter-type'}>{filter.title}</span>
      <FilterButton
        filterKey={filter.key}
        value={null}
        applyFilter={applyFilter}
        isFilterApplied={isFilterApplied}
      >
        All
      </FilterButton>
      {filter.values && filter.values
        .sort(sortByLabel)
        .map((filterValue, index2) => {
          return <FilterButton
            key={index2}
            filterKey={filter.key}
            value={filterValue.value}
            applyFilter={applyFilter}
            isFilterApplied={isFilterApplied}
          >
            {filterValue.label}
          </FilterButton>
        })}
    </div>
  })}
 </div>
}
```

Рисунок 3.15, аркуш 2

Ключовими елементами даного фрагменту програми є імпортування перевірки типів властивостей та компоненти кнопок фільтра, оголошення та

надання властивостей в компоненті EntitiFilter, функція сортування sortByLabel, де виконується сортуваннями за мітками (label) в залежності від типу даних, також обов’язково рендеринг компоненту та перевірка типів властивостей, в останньому описано типи та структура властивостей, які повинен приймати виществорений компонент.

На даному етапі фільтрацію вже реалізовано та виглядає вона таким чином (див. рис. 3.16). Потрібно звернути увагу також на те, що через створений компонент та експортування даної частини кодування на всі можливі сторінки, то вигляд фільтрування є стандартним, а саме таким, як визначено на дизайн макеті, але інформація відповідає саме тій сторінці, атрибути яких були отримані при завантаженні всієї інформації в базу даних та фільтруванні з неї. Таким чином інформація є актуальною саме для даної сторінки, яку використовує користувач.

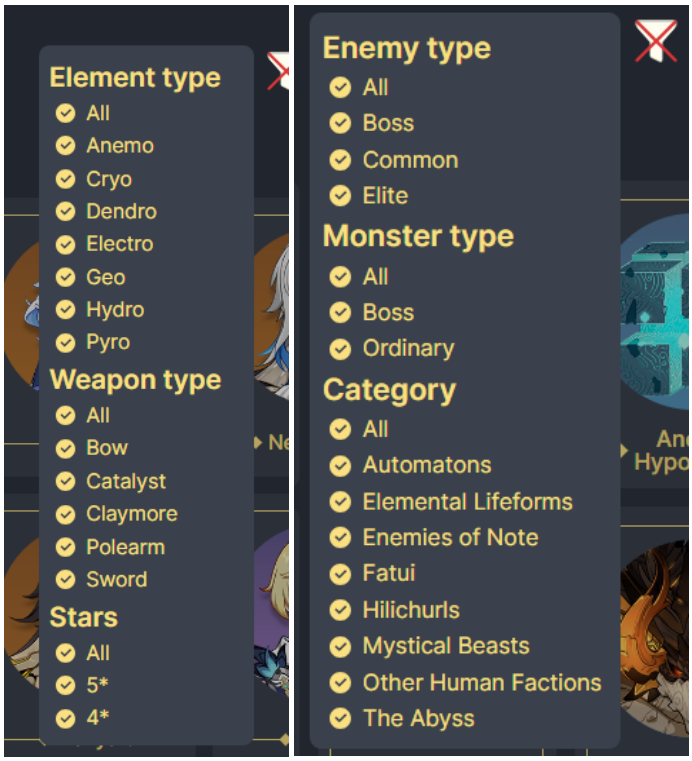

Рисунок 3.16 – Фільтр на сторінці Character та Enemies

Після реалізації фільтрації та відповідної йому кнопки було окремо реалізовано кожну сторінку, за допомогою компонентів, завдяки чому було

уникнуто проблему написання коду для кожного персонажу як окремого атрибуту на сторінці для відображення кожного персонажа та його імені. Схожі критерії відображень стосувались і інших вкладок, окрім вкладки Artifacts, на цій сторінці дизайн виглядає дещо інакше, а саме кількість іконок в одному блоці має бути 5, тому для цієї сторінки було створено окремі компоненти.

За допомогою технологій React можна створити в одному файлі програму на мові програмування JavaScript та уникнути написання коду для кожного аналогічного елементу на одній сторінці.

Таким чином, реалізовуючи ці кроки та поступово використовуючи бібліотеки та компоненти, було створено і інші сторінки, які передбачав дизайн та вимоги, які було сформульовано у другому розділі.

3.4 Тестування вебсайту

Наступним кроком після завершення програмної реалізації та перевірки належної роботи основних компонентів, а також зв’язку між базою даних та сервером, зокрема відповідей серверної частини на запити від клієнтської частини, було розпочато фінальну частину розробки – тестування.

При плануванні макет дизайну було прийнято рішення тестувати мануально, через декілька причин. Це і відносно невелика кількість коду серверної частини та роботи над базою даних. За допомогою ручного тестування можна знайти невідповідності у візуальній частині (наприклад втрата зображень або інформації, тощо). Такий підхід забезпечить швидке виявлення проблем та їх усунення.

Ручне тестування є більш повільним ніж автоматичне, але в даному проекті не передбачалось автоматичне тестування до моменту поліпшень. Наприклад, додавання опції створення особистих критеріїв поліпшення персонажів, в такому випадку тестування треба робити автоматизованим для швидкого аналізу різних аспектів даної функції та її критеріїв виконання. Також в даній версії проекту не

було передбачено автоматичне оновлення бази даних на основі нових актуальних персонажів, артефактів тощо, які з’явились на поточний момент у грі.

Далі буде наведено приклади тест-кейсів, які було створено для тестування дизайн макету і часткової функціональності.

На початку тестування було поділено тест-кейси на такі частини: візуальна відповідність всіх вкладок, карток розробленому макету; пошук за різними критеріями; фільтрація за різними властивостями; картки всіх компонентів запланованих на етапі проєктування; перехід між персонажами; окрема демонстрація додаткової інформації у картках персонажів та у картках персонажів-ворогів, а також коректне відображення всієї інформації, зокрема і зображень, відповідно очікуванням та дизайну на всіх сторінках веб-сервісу.

Результати тестування було поділено на позитивні – відповідає очікуванням і дизайн макету та негативні – відповідно має невідповідності із макет дизайном та очікуваннями.

Тест-кейси на перевірку коректності відображення всіх зображень на сторінках: Main (див. табл. 3.1), Character, Artifacts, Weapons і Enemies (див. табл. 3.2) відповідно до вимог та макету.

Таблиця 3.1 – Тест-кейс на відповідність зображень на Main сторінці

№	Крок	Очікуваний результат
1	Зайти на сайт	Сайт завантажується у браузері та відображається головна сторінка
2	Завантаження головної сторінки	Головна сторінка повністю відповідає дизайн макету та всі картинки коректно відображаються

Далі наведено приклад тестування фільтра та пошукової системи, відповідні тест-кейси наведено у таблицях 3.3 та 3.4 відповідно.

Таблиця 3.2 – Тест-кейс перевірки зображень на сторінках Character, Artifacts, Weapons і Enemies

№	Крок	Очікуваний результат
1	Зайти на сайт	Завантажується головна сторінка
2	Перейти до вкладки Characters	Завантажується інформація щодо всіх персонажів, які наявні у грі та зберігаються у базі даних
3	На вкладці Characters	Згідно із дизайн макетом, відображено таблицю, де кожен атрибут відповідає зображенню та назві персонажа.
4	Перейти до інших вкладок Weapons, Enemies, Artifacts	Очікуваний результат збігається із дизайн макетом та попереднім кроком

Таблиця 3.3 – Тест-кейс пошукової системи

№	Крок	Очікуваний результат
1	Відкрити сайт	Відкриється головна сторінка
2	Перейти на вкладку Characters	Автоматично фільтруються дані та буде показано всіх персонажів, відповідно до макету
3	В пошуковій системі ввести символи, наприклад ! чи ? чи не латинські літери або слово, наприклад Вода	Список має бути порожнім, оскільки не існує персонажа із такою назвою
4	Видалити попередні спроби пошуку та ввести літеру “h”	Із обраним символом буде відображений цілий список персонажів, які мають цю літеру в назві

Продовження таблиці 3.3

№	Крок	Очікуваний результат
5	Видалити попередні спроби пошуку та ввести ім’я Neuvillette	Буде відображений лише один персонаж, який має це ім’я
6	Повторити кроки 3-5 із наступними вкладками: Artifacts, Weapons та Enemies	Результат має бути відповідним до кроків 3-5 і відповідати очікуванням

Таблиця 3.4 – Тест-кейс перевірки відповідності до фільтрів у різних вкладках

№	Крок	Очікуваний результат
1	Відкрити сайт	Відкриється головна сторінка
2	Перейти на вкладку Characters	Автоматично фільтруються дані та буде показано всіх персонажів, відповідно до макету
3	Відкрити фільтр	Відкривається вікно із критеріями доступними для цієї вкладки
4	Змінити фільтри, наприклад вимкнути всі All та обрати Hydro та 5* в категоріях Element Type та Stars відповідно	Список має змінитись відповідно до обраних критеріїв
5	Кроки 2-4 повторити із вкладками Artifacts, Weapons та Enemies	Результат має бути відповідним до кроків 2-4 і відповідати очікуванням

Дані тест-кейси було створено для перевірки на коректність відображення сервісу та наявності невідповідностей. Наприклад, при тестуванні кейсу із

таблиці 3.2, було помічено, що один із атрибутів не має відображення порівняно з іншими (див. рис. 3.17), також подібну помилку було виявлено і на інших вкладках. Дану проблему було знайдено та виправлено, а повторне тестування надало можливість переконатись в її усуненні.

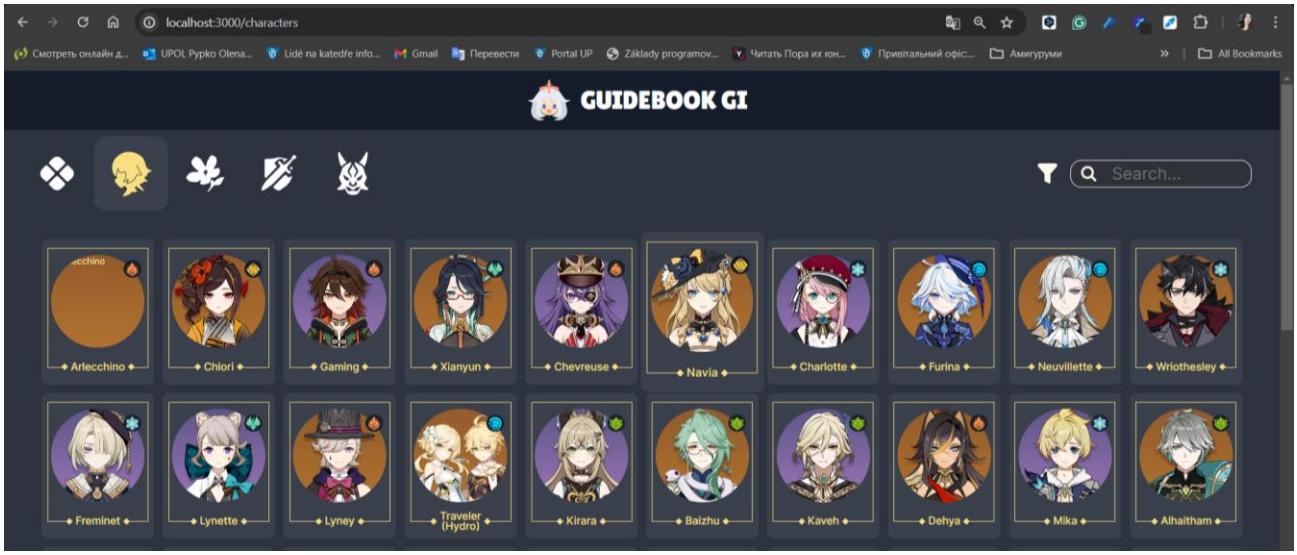

Рисунок 3.17 – Помилка, що виявлена при тестуванні на вкладці Characters

На даному етапі розробка була завершена, тестування було пройдено двічі для вирішення проблем, які могли виникнути в різні проміжки часу.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено персональний помічник, який використовує можливості сучасних технологій React та Node.js. Процес розробки складався з таких основних етапів: проектування архітектури, формулювання вимог, створення бази даних, створення серверної та клієнтської частин помічника, проведення тестування.

В першому розділі кваліфікаційної роботи наведено опис переваг та недоліків сучасних платформ для створення вебдодатків. Зокрема, наведено основні характеристики React та Node.js. Як систему управління базою даних обрано MySQL.

В другому розділі сформульовано вимоги до помічника, а також створено дизайн-макет майбутнього додатку. Варіанти використання системи наведено на діаграмі прецедентів. Також присутні діаграма компонентів та логічна схема даних.

В третьому розділі продемонстровано створення бази даних, наведено фрагменти програмної реалізації певних модулів системи, які охоплюють важливу частину створення елементів серверної чи клієнтської частини. Під час тестування помічника було виявлено досить високу продуктивність та швидку відповідь на запити користувача. Також було виявлено критичні помилки, які вдалось усунути. Розроблено систему тестів для перевірки візуальних компонентів вебсайту.

У подальшому планується розширення функціоналу помічника в бік додавання актуальних новин із автоматичним оновленням, авторизація користувача, додавання персональних методів поліпшення окремих персонажів, автоматичне оновлення даних, тощо.

ПЕРЕЛІК ПОСИЛАНЬ

1. Banks A., Porcello E. Learning React : Modern Patterns for Developing React Apps. 2nd ed. Beijing; Boston; Farnham; Sebastopol; Tokyo: O'Reilly, 2020. 310 p.

2. Nixon R. Learning PHP, MySQL & JavaScript : With jQuery, CSS & HTML5. 5th ed. Sebastopol, CA: O'Reilly, 2018.

3. Boronczyk T. Jump Start MySQL. Česky, překlad : Milan Daněk. Brno: Computer Press, 2016.

4. Facebook Open Source. Getting Started | Create React App. URL : https://create-react-app.dev/docs/getting-started/ (дата звернення: 29.05.2024).

5. Barger R. How to Create a React App with a Node Backend: The Complete Guide. URL : https://www.freecodecamp.org/news/how-to-create-a-react-app-with-a-node-backend-the-complete-guide/ (дата звернення: 29.05.2024).

6. Inamdar M. Building Your Personal AI Assistant with OpenAI, NodeJS, and ReactJS. 10.03.2024. URL : https://medium.com/@mosininamdar/building-your-personal-assistant-website-with-openai-nodejs-and-reactjs-d0c527555389 (дата звернення: 29.05.2024).

7. Tahaghoghi S.M.M., Williams H. Learning MySQL. Sebastopol, CA: O'Reilly, 2006.

8. Casciaro M., Mammino L. Node.js Design Patterns: Design and Implement Production-Grade Node.js Applications Using Proven Patterns and Techniques. 3rd ed. 2020. 660 p.

9. Herron D. Node.js Web Development : Server-Side Development with Node 10 Made Easy. 4th ed. Birmingham; Mumbai: Packt, 2018.

10. Herron D. Node.js Web Development. 4th ed. May 30, 2018. 492 p.

11. Ruby Documentation. URL : https://ruby-doc.org/ (дата звернення: 29.05.2024).

12. Офіційний веб-сайт гри «Genshin Impact». URL : https://genshin.hoyoverse.com/en/ (дата звернення: 29.05.2024).

13. Chollet F. Deep Learning with Python. Česky První vydání. Praha : Grada Publishing, 2019. 328 stran.

14. Vykydal J. Vývoj webových aplikací s využitím frameworku [рукопис]. 2014.

ДОДАТОК А

Фрагмент коду картки персонажа

```
export const CharacterModal = () => {
  const { id } = useParams();
  const [entity, setEntity] = useState();
  const [ascendTab, setAscendTab] = useState(1);

  useEffect(() => {
    const abortController = new AbortController();
    fetch(`/api/v2/characters/${id}`, { signal: abortController.signal })
      .then(res => res.json())
      .then((data) => {
        setEntity(data)

      })
      .catch(e => !abortController.signal.aborted ? console.error(e) : null)

    setAscendTab(1)
    return () => {
      abortController.abort();
    }
  }, [id]);

  return <Modal show={true} fullscreen={true} className={'entity-modal'}>
    <Modal.Header>
      <Header/>
    </Modal.Header>
```

```
          <Modal.Body>
            <div className={'entity-details'}>
              {entity ?
                <div className={'entity'}>
                  <Link to={'..'} className={'btn btn-close'}/>
                  <div className={'entity-wrapper'}>
                    <Row style={{ justifyContent: "space-between" }}>
                      <Col xl={3} xs={12} className={'position-relative'} style={{
minWidth: 386 }}>
                        <div className={'d-flex align-items-center justify-content-center
justify-content-xl-start'}>
                          <div className={'image'}>
                            <div>
                              <Image src={getCharacterImageUrl(id, 'portrait')} alt={'portrait
image'}/>
                            </div>
                            <div className={'element'}>
                              <Image
src={`/images/elements/${entity.vision.toLowerCase()}.webp`}/>
                            </div>
                          </div>
                        </div>
                        <div className={'w-100 d-flex align-items-center justify-content-
center'}>
                          <div className={'entity-name'}>
                            <DynamicSizedText defaultFontSize={70} maxSymbols={15}

maxWidth={345}>{entity.name}</DynamicSizedText>
                          </div>
                        </div>
```

```
                    <Splitter small/>
                    <div className={'entity-description'}>{entity.description}</div>
                    <Splitter small/>
                    <div className={'entity-description justify-content-evenly'}>
                     <Image src={getCharacterImageUrl(id, 'talent-na')} width={75}
height={75} alt={'talent na'}/>
                     <Image src={ArrowImage} alt={'arrow'}/>
                     <Image src={getCharacterImageUrl(id, 'talent-skill')} width={75}
height={75} alt={'talent skill'}/>
                     <Image src={ArrowImage} alt={'arrow'}/>
                     <Image src={getCharacterImageUrl(id, 'talent-burst')} width={75}
height={75} alt={'talent burst'}/>
                    </div>
                    <Splitter small/>
                   </Col>
                   <Col xl={7} xxl={8} xs={12} className={'justify-content-center'}>
                    {entity.builds.map((build, index) => {
                     return <div key={index} className={'entity-build me-5'}>
                      <div className={'d-flex flex-column gap-3'}>
                       <span className={'entity-title'}>{build.name}</span>
                       <Splitter/>
                       <div className={'d-flex mb-5 gap-3 flex-wrap justify-content-
center '}>
                        {build.weapons
                         .sort(sortByPriority)
                         .map((weaponData, jindex) => {
                          return <Link
                           to={`/weapons/${weaponData.weapon.id}`}
                           key={jindex}
```

```
                    >
                      <MaterialImage
                        src={`/api/v2/weapons/${weaponData.weapon.id}/icon`}
                  hoverSrc={`/api/v2/weapons/${weaponData.weapon.id}/awakenicon`}
                        rarity={weaponData.weapon.rarity}
                        title={weaponData.weapon.name}
                      />
                    </Link>
                  })}
                </div>
              </div>
              <div className={'d-flex flex-column gap-3'}>
                <div className={'d-flex flex-column align-self-center'}>
                  <span                    className={'entity-title'}>Recommended
Artifacts</span>
                  <span className={'entity-subtitle'}>
                  {
                    build.subStats
                      .sort(sortByPriority)
                      .map(item => item.priority)
                      .filter(filterUnique)
                      .map(priority => (
                        build.subStats
                          .filter(stats => stats.priority === priority)
                          .map(stats => convertStat(stats.stat, true))
                          .join(' / ')
                      )).join(' > ')
                  }
                  </span>
                </div>
```