Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка генератора зовнішності ігрових персонажів для гри D&D з використанням технології ReactJS

Виконав: студент IV курсу, групи СП-43
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Фаль О.П. (прізвище та ініціали)

Керівник (підпис) Стоянов Ю.М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Фаль Олександр Петрович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка генератора зовнішності ігрових персонажів для гри D&D з використанням технології ReactJS

Керівник роботи к.т.н., доц. каф. ПІ Стоянов Ю. М
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області, огляд популярних сервісів та вибір технології розробки. Розділ 2. Проєктування та розробка генератора зовнішності ігрових персонажів. Розділ 3 Конструювання та тестування генератора зовнішності. Розділ 4. Безпека життєдіяльності та основи охорони праці. Висновки. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми, знімки екрану з проробленою роботою, ілюстративні зображення, інформативні зображення для доповнення тексту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	к.т.н, доц. каф. МТ Лазарюк В. В.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи		
2.	Підбір джерел по темі кваліфікаційної роботи		
3.	Опрацювання джерел інформації по темі кваліфікаційної роботи		
4.	Розробка архітектури програмного рішення		
5.	Ознайомлення з новими технологіями		
6.	Розробка генератора зовнішності ігрових персонажів на основі опрацьованої інформації		
7.	Виконання звіту по проробленій роботі		
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»		
9.	Виконання завдання до підрозділу «Основи охорони праці»		
10.	Оформлення кваліфікаційної роботи		
11.	Нормоконтроль		
12.	Перевірка на плагіат		
13.	Попередній захист кваліфікаційної роботи		
14.	Захист кваліфікаційної роботи		

Студент _____ Фаль О.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка генератора зовнішності ігрових персонажів для гри D&D з використанням технології ReactJS // Кваліфікаційна робота освітнього рівня «Бакалавр» // Фаль Олександр Петрович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-43 // Тернопіль, 2024 // С. – 87, рис. – 66, табл. – 0, кресл. – 0, додат. – 3, бібліогр. – 20.

Ключові слова: розробка генератора, ігрові персонажі, D&D, ReactJS, фронтенд, JavaSript, TypeScript, CSS, C#, PostgreSQL, Visual Studio Code, API

Першочеговою метою цієї кваліфікаційної роботи є дослідження процесів аналізу, проєктування, розробки та тестування генератора зовнішності ігрових персонажів для гри D&D із використанням сучасних технологій та інструментів фронтенд-розробки, зокрема ReactJS.

В першому розділу приділено увагу дослідженню предметної області та аналізу популярних додатків до неї. Розглянуто вибір технологій розробки та проведено аналіз вимог.

Другий розділ присвячено опису проєктування генератора зовнішності персонажів та вибору архітектурного підходу.

Третій розділ зосереджено на розробці та автоматизованому тестуванні генератора ігрових персонажів.

ABSTRACT

Development of a game character appearance generator for the D&D game using ReactJS technology // Qualification work for the educational level ‘Bachelor’ / Fal Oleksandr Petrovych // Ternopil National Technical University named after Ivan Puluj, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-43 // Ternopil, 2024 // P. - 87, Fig. - 66, Table - 0, Drawings - 0, Supplement - 3, Bibliog - 20.

Keywords: generator development, game characters, D&D, ReactJS, frontend, JavaScript, TypeScript, CSS, C#, PostgreSQL, Visual Studio Code, API

The primary goal of this qualification work is to study the processes of analysing, designing, developing and testing the appearance generator of game characters for the D&D game using modern technologies and front-end development tools, in particular ReactJS.

The first section focuses on the study of the subject area and the analysis of popular applications to it. The choice of development technologies is considered and the requirements analysis is conducted.

The second section is devoted to the description of the design of the character appearance generator and the choice of the architectural approach.

The third section focuses on the development and automated testing of the game character generator.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Генератор зовнішності – це застосунок для створення зовнішності ігрового персонажа для гри в D&D. Включає в себе вибір раси персонажа, статі, класу та вибір кількох елементів одягу відповідно до вибраних раніше параметрів.

Ігровий персонаж – вигаданий гравцем персонаж, за якого він грає у настільно-рольовій грі D&D. Для персонажа обираються характеристики, передісторія, здібності, та зовнішній вигляд, в чому допомагатиме генератор зовнішності.

D&D (Dungeons&Dragons) – настільно-рольова гра, у якій гравці створюють персонажів, часто заснованих на культових світах, та йдуть по інтерактивній історії, через яку їх проводить Майстер Підземелля (DM).

Фронтенд – це інтерфейс веб-додатків (веб-сайтів). Він є публічним і користувач має можливість безпосередньо взаємодіяти з ним.

ReactJS – публічна бібліотека мови програмування JavaScript для створення користувацьких інтерфейсів.

JavaScript – об’єктно-орієнтовна, динамічна мова програмування.

TypeScript – мова програмування, яка розширює можливості JavaScript у створенні веб-застосунків.

CSS – спеціальна мова, яка використовується для опису зовнішнього вигляду сторінок

C# – об’єктно-орієнтовна мова програмування.

PostgreSQL – об'єктно-реляційна система баз даних з відкритим кодом

Visual Studio Code – безкоштовний легкий редактор коду створений компанією Microsoft.

API – це набір інструкцій які дозволяють програмам спілкуватись одна з одною.

ЗМІСТ

ВСТУП

Настільні ігри стали важливим культурним пластом. Мабуть ви часто бачили як люди грають «настілки» в кіно, серіалах, і вже точно самі грали хоча б раз у якусь настільну гру. Також існує такий підвид, як настільно-рольові ігри, де передбачається створення ігрового персонажа. Основою та, можна сказати що, матір'ю всіх настільно-рольових ігор (НРІ), є Dungeons&Dragons, або ж D&D.

Метою даної дипломної роботи, є розробка генератора зовнішності ігрових персонажів для гри D&D з використанням технології ReactJS. Ця технологія допоможе створити зрозумілий та естетично привабливий веб-застосунок який допоможе створювати дизайни для ігрових персонажів.

Генератор зовнішності включатиме в себе такі основні функціональні можливості, як вибір раси персонажа, статі, категорії класів та елементів одягу. Серед другорядних - можливість введення імені персонажа та завантаження зображення під цим іменем. Також генератор буде наповнений авторськими художніми малюнками.

Планується глибоке дослідження та застосування основної технології ReactJS та суміжних інструментів, що дозволить розробити сучасний та зручний інтерфейс. Очікується, що створений в ході бакалаврської роботи веб-застосунок, допоможе новачкам та досвідченим гравцям в D&D та інші НРІ краще оформити дизайн своїх персонажів, який буде не лише привабливим, а й підкреслюватиме унікальні риси характеру та здібності.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ОГЛЯД ПОПУЛЯРНИХ СЕРВІСІВ ТА ВИБІР ТЕХНОЛОГІЇ РОЗРОБКИ

1.1 Історія та суть гри Dungeons&Dragons

«Підземелля та Дракони» була розроблена в 1974 році Гері Гайгексом, засновником клубу настільних ігор International Federation of Wargamers [1]. Це гра з вигаданим середовищем найчастіше в сеттингу Середньовіччя. Гравці ж керують власноруч створеними персонажами різних класів починаючи від воїна та варвара, закінчуючи магом та монахом. Правила гри засновані на стратегічних настільних іграх 1950-60-х років на військову тематику де за успіх атаки чи іншої дії відповідали кубики.

В D&D кубики відіграють ключову роль. Під час гри користуються комплектом з семи багатогранних кубиків. Основний, 20-гранний кубик К20, з його допомогою визначають успіх дій, атак, перевірок, рятунку, тощо. Також в наборі є кубики К4, К6, К8, К10, К12, відсотковий кубик К100, які використовуються для розрахування пошкоджень, лікування чи магічних ефектів (рис. 1.1).

Проводить гру Майстер Підземелля або ж DM. Він описує гравцям антураж, відіграє сюжетних персонажів, просить виконувати кидки та оголошує результати.

Основні правила розписані у «Книзі Гравця», примірник якої є перекладений українською мовою [2]. Там наведені інструкції по створенню персонажа, основні правила гри з описом усіх механік, починаючи битвами і закінчуючи заклинаннями.

Рисунок 1.1 – Гравці грають на полі підземелля використовуючи гральні кубики

1.2 Аналіз популярних додатків для гри

Для кращого інтерактивного сприйняття під час гри або створення персонажів чи сюжетів, компанії чи соло-розробники, створюють додатки,

застосунки, веб-сайти та багато іншого, що допомагає краще зрозуміти та взаємодіяти з грою. Ці інструменти дозволяють користувачам ігрових продуктів глибше відчувати іммерсію, персоналізувати свої досвіди та знаходити нові можливості для творчості та вдосконалення геймплею. Декілька таких рішень розберемо:

1. TaleSpire

TaleSpire це цифровий продукт у крамниці Steam, який дозволяє проводити онлайн партії у настільно-рольові ігри. Платформа надає можливості створення власник карт, будівель на них, налаштування кольорової гамми та музичного супроводу [3].

Також TaleSpire включає в себе 3D фігурки персонажів і монстрів, та крім цього, має в собі розширення для публічного контенту, куди можна завантажувати свої об’єкти і додавати в свою кампанію об’єкти інших людей.

Платформа має зручні функції як набір гральних кубиків з автоматичним підрахунком результатів, режим ініціативи, режим кінематографу, для запису кат-сцен, інструменти для вимірів та фізики об’єктів. Крім того, є можливість мультиплеєру, де кілька гравців можуть грати в одну кампанію.

TaleSpire має цінник у 329₴ на платформі Steam та часто отримує сезонні знижки (рис. 1.2). Також доступна внутрішньоігрова покупка «стільця», для гравців які зможуть приєднуватись до різних кампаній, але не зможуть використовувати весь функціонал створення додатку. Платформа перебуває в дочасному доступі, але вже завоювала велику прихильність, та здобула схвальні оцінки користувачів та критиків.

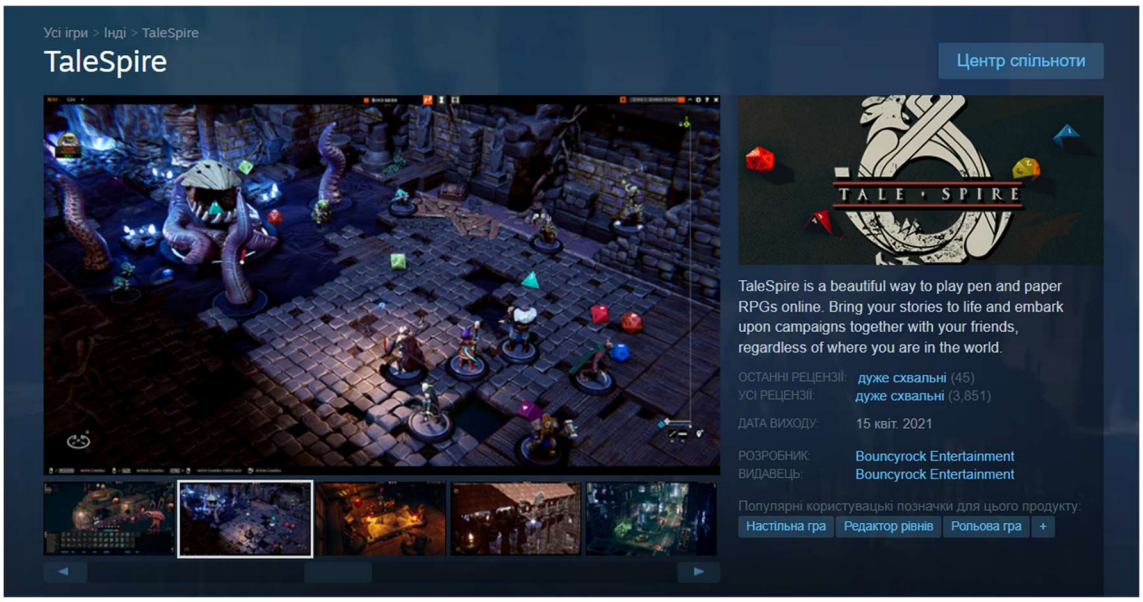

Рисунок 1.2 – платформа TaleSpire

2. Owlbear Rodeo

Owlbear Rodeo – це 2D веб-застосунок, для проведення D&D кампаній. У цього сайту зручний та зрозумілий інтерфейс, та функціонал серед якого набір кубиків, лінійка, та стандартні набори фігурок персонажів та монстрів [4].

Ресурс також дозволяє не тільки користуватись стандартними елементами гри, а й завантажувати власні мапи та створювати кастомні сцени, що дає можливість налаштувати кампанію до ваших потреб.

Важливою перевагою також є підтримка сайтом мультиплеєрної гри, де за посиланням гравці можуть заходити на спільну сцену та приєднувати до гри, керуючи своїми фішками персонажів у режимі реального часу.

Також вагомими плюсами, є наявність візуальних ефектів, як туман війни, можливість малювання на мапі, для поміток радіусу дій ефектів чи заклинань, що дозволяє покращити імерсивність кампанії (рис. 1.3).

У висновку Owlbear Rodeo є чудовим варіантом як для початківців так і для досвідчених гравців у настільно-рольові ігри. Простота інтерфейсу застосунку робить його доступним. Крім того, Owlbear Rodeo є абсолютно безкоштовним, що робить його прекрасним вибором для всіх любителів D&D та інших HPI, які хочуть насолоджуватися грою без зайвих витрат.

Рисунок 1.3 – ігровий інтерфейс сайту Owlbear Rodeo

3. D&D Wiki

D&D Wiki – аналог онлайн-енциклопедії Wikipedia, але створений для D&D. На D&D Wiki зібраний увесь контент по грі в текстовому варіанті [5].

Ресурс здобув популярність серед гравців за можливість додавання власного контенту на сайт, серед якого велика кількість користувацьких кастомних рас, класів, зброї, магічних предметів тощо. Все це отримало назву Homebrew (рис. 1.4).

Кастомні елементи стали невід'ємною частиною D&D. Вони значно розширюють стандартний мануал правил та всіх частин, включно з расами, класами, предметами, новими світами та змінами механік. За допомогою них, можна втілити буквально будь-який ваш концепт персонажів, наприклад створити персонажа за мотивами поп-культурних франшиз як «Володар Перснів», «Гра Престолів», «Зоряні Війни», готичні фільми про вампірів, ковбойські вестерни, кіберпанк тощо.

Таким чином, D&D Wiki є чудовим посібником, для того щоб дізнатись як стандартну інформацію з правил, так і для того щоб знайти нові цікаві елементи та створення унікальних персонажів чи сюжетів.

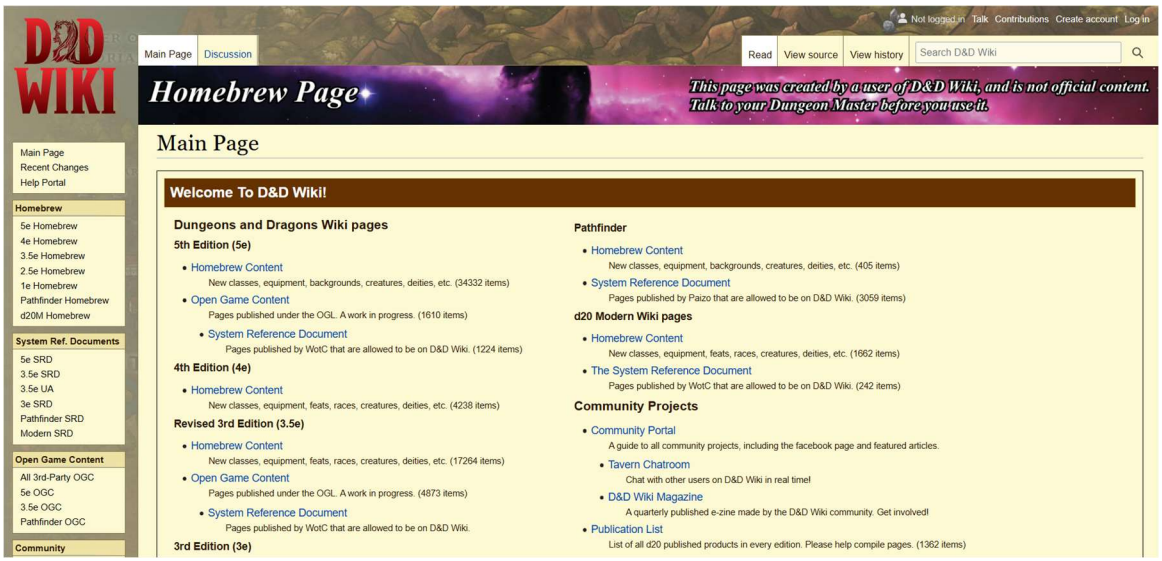

Рисунок 1.4 – Головна сторінка сайту D&D Wiki

1.3 Вибір технологій розробки

1.3.1 ReactJS

Для розробки генератора зовнішності ігрових персонажів для гри D&D було обрано технологію ReactJS. Ось кілька переваг, які вплинули на рішення використовувати саме цю технологію:

1. Продуктивність. Оскільки технологія ReactJS використовує віртуальний DOM, вона в рази зменшує кількість дій з реальним DOM, що в свою чергу оптимізовує оновлення інтерфейсу та позитивно впливає на продуктивність і зміни в ньому відбуваються швидше. Це дозволить створити швидкий, динамічний та легкий у використанні веб-застосунок з привабливим інтерфейсом [7].
2. Компонентний спосіб будування проекту. ReactJS дозволяє створювати окремі повторювані компоненти для інтерфесу, та підключати їх до основного файлу, що значно спрощує роботу з кодом. Можна створити один компонент та повторно використовувати його в коді безліч разів.

Крім того, це значно покращує масштабованість та вигляд коду і допомагає не заплутатись у ньому.

3. Зв’язок з іншими технологіями. ReactJS добре інтегрується з різними бібліотеками та фреймворками. Це дозволяє використовувати цю технологію у найрізноманітніших проектах, як простих сайтах з однією функцією, так і у великих веб-застосунках за широким спектром використання.
4. Односторонній потік даних. React використовує односторонній потік даних, що робить його простим та передбачуваним для відстеження змін та налагодження.
5. Екосистема. React має велику базу програмістів які користуються ним та активно досліджують його. Це дозволяє легко знайти відповіді та допомогу у будь-яких незрозумілих ситуаціях.
6. Простота навчання. У технології React дуже простий синтаксис, а його механіки та властивості, легко зрозуміти навіть новачкам програмістам, які лише починають свій шлях у фронтенд-розробці. Компонентна складова тут особливо виділяється, тому що як було підмічено раніше, це допомагає розібратись в коді та не навантажувати його непотрібними елементами.
7. Серверний рендеринг. React рендерить контент прямо на сервері, це в свою чергу покращує оптимізацію і прискорює час завантаження веб-сторінки.

Підсумовуючи, технологія ReactJS є ідеальним кандидатом, для розробки генератора зовнішності ігрових персонажів для гри D&D. Його широкі можливості дозволять втілити функціональний та естетично привабливий дизайн для веб-застосунку. А характеристики продуктивності та рендерингу, зроблять сайт динамічним та дозволить швидко завантажувати контент на сторінках. Широка спільнота програмістів які користуються цією технологією, допоможе у виникненні складних ситуацій, та підкаже які ще функціональні рішення можна імплементувати в систему та зробити її ще більш різноманітною.

1.3.2 TypeScript

TypeScript, представлений Microsoft восени 2012 року, є надмножиною JavaScript, яка розширює можливості цієї мови для розробки вебзастосунків, забезпечуючи статичну типізацію та запобігання помилкам у коді [8].

TypeScript має декілька вагомих переваг:

По-перше, типізація допомагає покращити підтримку та масштабованість коду, та допомагає виявити помилки в коді на ранньому етапі.

По-друге, TypeScript підтримує сучасні можливості JavaScript та стандарти ECMAScript, забезпечуючи сумісність з усіма бібліотеками та фреймворками JavaScript.

Також, інструменти для роботи з TypeScript, наприклад розширена підтримка в IDE, що включає інтелектуальне автодоповнення та інструменти для рефакторингу. Ці інструменти в рази спрощують роботу розробників і підвищують продуктивність.

Нарешті, TypeScript полегшує документацію коду та співпрацю в команді, сприяючи більш структурованому підходу до розробки програмного забезпечення. Ці можливості роблять TypeScript потужним вибором для створення складних і масштабованих вебзастосунків.

1.3.3 Бекенд частина. C# та PostgreSQL

Для розробки бекенд-частини генератора зовнішності ігрових персонажів для гри D&D було обрано мову програмування C#.

C# – це мова програмування, яка посідає визначне місце у світі розробки програмного забезпечення. Вона привертає увагу розробників з усього світу завдяки своїй потужності, універсальності та широкому спектру застосування. У

цій статті ми розглянемо, що таке середовище програмування сі шарп, та його ключові особливості й переваги в розробці [9].

Основною функцією бекенду на мові C# стане генерування хеш-коду готового зображення. Ось кілька переваг, цієї мови програмування:

1. Продуктивність: C# є швидкою мовою програмування, та дозволяє ефективно працювати з великими обсягами даних, що потрібно для генерації хеш-кодів зображень. Завдяки компіляції в машинний код, програми на C# працюють швидко і знижують затримки при обробці запитів.
2. Інтеграція з іншими технологіями: C# ідеально інтегрується з іншими інструментами та технологіями, наприклад, з базами даних через Entity Framework або з зовнішніми API, що може знадобитися для обробки зображень чи збереження даних.
3. Масштабованість: C# і платформа .NET дозволяють створювати високонавантажені додатки, які можуть ефективно працювати з великими обсягами даних і підтримувати велику кількість одночасних користувачів, що робить C# чудовим вибором для масштабування генератора ігрових персонажів.

Таким чином, мова програмування C#, стала очевидним кандидатом для написання бекенду для веб-застосунку. Простий синтаксис та легка інтеграція з фронтедом допоможуть під’єднати базу даних.

Для налаштування самої бази даних, було обрано систему PostgreSQL. PostgreSQL — це потужна об'єктно-реляційна система баз даних з відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, що безпечно зберігають та масштабують найскладніші робочі навантаження даних. Витоки PostgreSQL сягають 1986 року як частина проекту POSTGRES в Каліфорнійському університеті в Берклі та має понад 35 років активної розробки на основній платформі [10].

PostgreSQL має кілька вагомих переваг, які роблять його найкращим варіантом серед додатків для роботи з базами даних.

1. Підтримка складних запитів: PostgreSQL може підтримувати складні SQL-запити, підзапити, агрегації, обчислення, приєднання та інші операції з даними.
2. Підтримка JSON та JSONB: PostgreSQL підтримує JSON, що дозволяє зберігати та обробляти неструктуровані дані. Формат JSONB забезпечує додаткову ефективність для зберігання та пошуку даних у форматі JSON.
3. Безпека: PostgreSQL має вбудовані функції безпеки: шифрування, контроль доступу на основі ролей, можливість інтеграції з різними системами аутентифікації.

1.4 Визначання та аналіз вимог

Для ефективної розробки генератора зовнішності ігрових персонажів для гри D&D, необхідно встановити чіткі та зрозумілі вимоги. Вони формуються на основі аналізу потреб потенційних користувачів та особливостей предметної області, аналіз якої був проведений раніше.

Вимоги діляться на функціональні та нефункціональні. Перші визначають головні можливості розроблюваного продукту, а другі технічні характеристики, такі як продуктивність, безпека, юзабіліті та інші фактори.

Встановлення вимог для генератора зовнішності ігрових персонажів для гри D&D допоможе забезпечити його функціональність, комфорт використання, надійність та безпеку, а також відповідатиме очікуванням користувачів.

Функціональні вимоги:

1. Вибір атрибутів персонажа:

– можливість вибору раси, статі та категорію класів персонажа з реалізацією відображення зображень у форматі «каруселі»

– можливість вибору елементів одягу на такі частини тіла: голова, тулуб, ноги. Елементи накладаються поверх шаблону, підібраного на основі раси та статі, та прокручуються за допомогою кнопок стрілок

2. Зміна мови:

– додавання спеціального перемикача для зміни мови на англійську

3. Введення імені персонажа та збереження:

– поле для введення імені персонажа, та збереження зображення готового персонажа під цим іменем

4. Генерування хеш-коду зображення:

– Згенерований хеш-код утворюватиметься після натискання на кнопку “Готово” та відображатиметься під полем для введення імені персонажа.

Нефункціональні вимоги:

1. Продуктивність:

– проєкт повинен без затримок прогортати зображення в каруселі, та миттєво завантажувати готове зображення.

– можливість обробки великої кількості запитів без відчутного зниження швидкості.

2. Юзабіліті

– кроссплатформенний інтерфейс для різних пристроїв та розмірів екранів

3. Надійність

– Підтримка постійної роботи генератора без великих перерв на технічне обслуговування

4. Масштабованість

– можливість додавання нових функцій та покращувати можливість збільшеного навантаження

5. Сумісність

– сумісність із різними веб-браузерами (Google Chrome, Firefox, Safari) та підтримка популярних операційних систем (Windows, macOS, Linux).

Визначення функціональних і нефункціональних вимог є дуже важливим етапом у розробці генератора зовнішності ігрових персонажів для гри D&D. Визначення вимог охарактеризує головні цілі та критерії у розробці проєкту.

Функціональні вимоги гарантують користувачам можливість легко та точно налаштовувати зовнішній вигляд персонажа на свій вибір, включаючи вибір раси, класу, статі та одягу.

Нефункціональні вимоги, в свою чергу, забезпечують зручність, масштабованість та надійність системи. Це дозволить без проблем та з комфортом користуватись генератором. Слідування цим вимогам, дасть змогу розробити якнайкращий веб-застосунок для генерування зовнішності ігрових персонажів для гри D&D.

Основна проблема яку вирішує моя дипломна робота, це розробка генератора ігрових персонажів на українську аудиторію. В українському ком'юніті Dungeons&Dragons немає аналогу зарубіжним мейкерам та галереям.

Генератор ігрових персонажів "Character maker" дозволить зручно та швидко підібрати саме такий стиль одягу, який задумав гравець. Режим перегляду каруселі, в сукупності з віртуальним DOM від ReactJS, забезпечать швидку зміну малюнків та цим прискорять створення зовнішності ігрового персонажа. Саме наповнення одягу матиме стиль який естетично підходить до класичних сетингів гри D&D, таких як Forgotten Realms, Ebberon та інші.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ГЕНЕРАТОРА ЗОВНІШНОСТІ ІГРОВИХ ПЕРСОНАЖІВ

2.1 Архітектурне проєктування

Почати проєктування слід з аналізу та складання загальної картини архітектури, за якою функціонуватиме генератор зовнішності ігрових персонажів.

Повний процес генерування зовнішності персонажа відбуватиметься в 5 етапів:

1. Вибір раси персонажа
2. Вибір статі (чоловіча та жіноча)
3. Вибір категорії ігрового класу (бойові, спритні, та магічні)
4. Вибір елементів зовнішності з трьох доступних категорій: голова, тулуб, ноги
5. Введення імені персонажа та завантаження зображення під цим іменем. Генерування хеш-коду готового зображення після натискання на кнопку “Готово”

Перехід між етапами відбуватиметься миттєво та без затримок, завдяки серверному рендерингу ReactJS. Генерування HTML сторінок прямо на сервері покращить швидкість завантаження контенту, в тому числі і динамічного. На додачу це підвищить оптимізацію веб-застосунку для пошукових систем як Google, Bing чи Yahoo.

Передбачено реалізувати вибір усіх атрибутів (окрім статі) у форматі каруселі, з прокруткою доступних зображень за допомогою кнопок зі стрілками.

Також серед функціональних вимог: перемикання мови, за допомогою кнопки в правому верхньому куті. Серед доступних мов – українська та англійська.

Генератор зовнішності ігрового персонажа буде наповнено авторськими малюнками від автора LemonFrog [11], з чітким візуальним стилем, підходящим естетиці гри Dungeons&Dragons. Естетичне наповнення також покращить шрифт CyrillicGoth [12]. Він має характерні готичні форми та виводи літер, що чудово

підходить до загального стилю, а також крім латиниці, він повністю адаптований під кирилицю та особливі літери українського алфавіту «ї», «є» та «ґ».

Генерація хеш-коду відбуватиметься на віддаленому сервері, через інтеграцію з бекендом. Хеш-код буде створюватись лише по готовому зображенню, але може створюватись без заповненого імені. Також база даних відкликається на post-запити, що означає що вона повністю функціональна та готова до впровадження майбутніх функцій.

В сукупності, розроблена архітектура та естетичні дизайнерські рішення дадуть змогу відчути приємний досвід у створенні унікальних персонажів для гри Dungeons&Dragons.

2.2 Проєктування інтерфейсу користувача

UI («User Interface») — «призначений для користувача інтерфейс», а UX («user experience») – це «досвід користувача» [13]. UI-дизайнер відповідає за візуалізацію додатку, роблячи його зручним і функціональним. Щоб продукт з комфортом візуально сприймався користувачем, фахівець UI відповідає за підбір форм, кольорів та інших параметрів.

Головна мета під час проєктування інтерфейсу користувача – зробити його максимально інтуїтивно зрозумілим. Структура генератора зовнішності ігрових персонажів для гри D&D, передбачає єдиний послідовний сценарій, з вибором різних аспектів на кожному етапі. Основні елементи, зображення та кнопки, будуть розміщуватись посередині та займатимуть основну роль UI.

Позаду буде розміщуватись фонове зображення, також стилізоване під малюнки рас, класів та елементів одягу. Було обрано елемент вітражу, щоб відобразити яскравість та спрощено показати особливості усіх аспектів.

На основі проєктування, та розробленої раніше архітектури, було створено діаграму варіантів використання (рис. 2.1), де показано усі елементи UI з якими

може взаємодіяти користувач, а також відображено що усі елементи є послідовними.

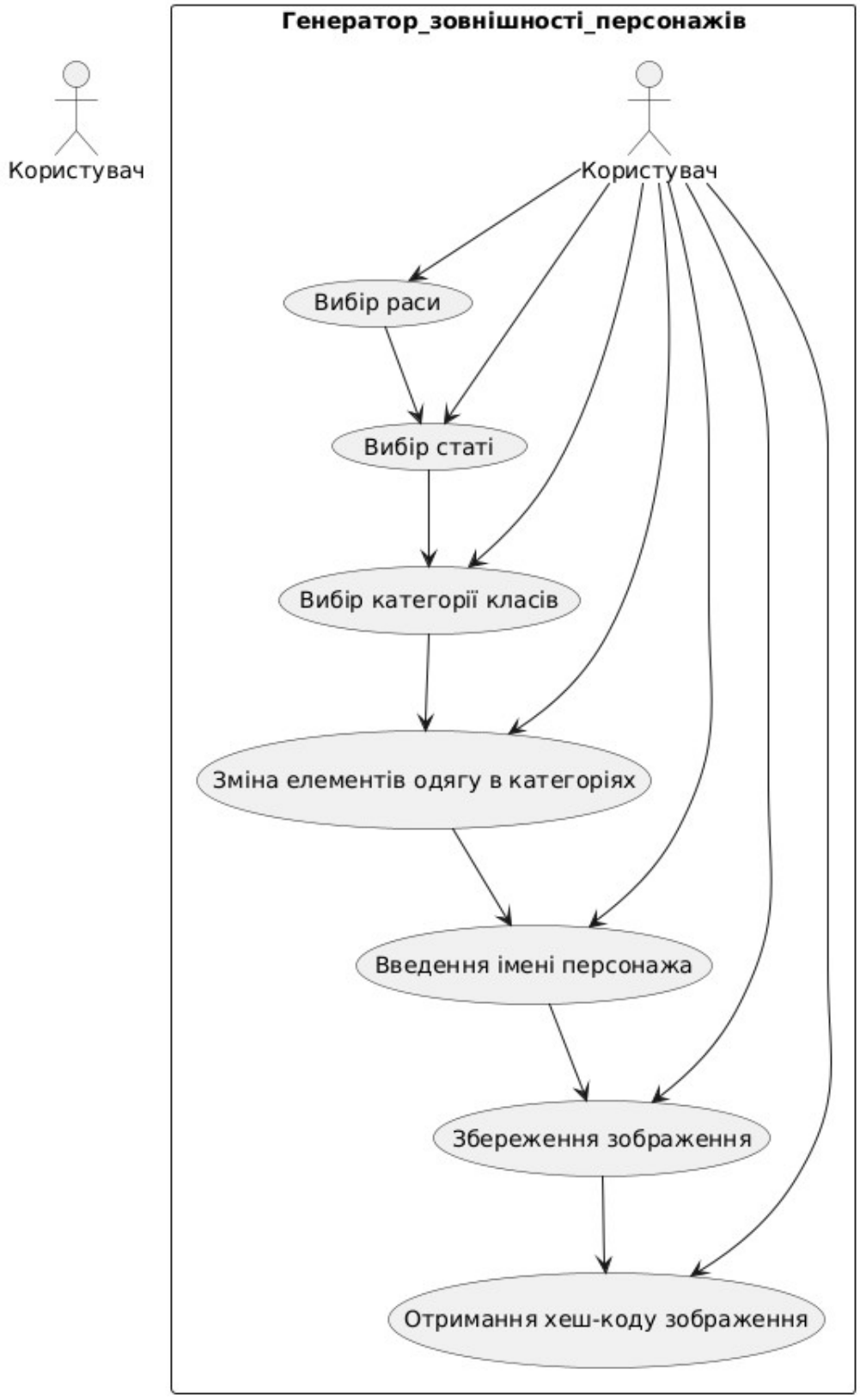

Рисунок 2.1 – Діаграма варіантів використання користувача

Далі було проведено детально проаналізовано послідовність. Початковою було задумано створити окремий компонент на кожен ігровий клас, та це мало декілька недоліків:

По-перше, це значно збільшувало обсяг роботи та розтягувало код однаковими елементами.

По-друге, і що найважливіше, це зменшувало варіативність та креатив у підході створення дизайну. До прикладу, канонічно клас Чаклун – клас, персонаж якого заключає угоду з темним покровителем, від того в дизайні чаклунів переважають темні елементи, плащі, балахони та окультні елементи. Та це лише канонічність і можна сказати що, стереотипи. В реальності, гравці можуть комбінувати всі можливі дизайни, та описувати їх як завгодно, навіть якщо вони не сумісні з типовим уявленням про клас. Тому було прийняте рішення, об’єднати класи у три групи: бойові, спритні та магічні. Це дозволить гравцям комбінувати дизайни, та створювати унікальні стилі для їхніх персонажів. Варіативність це завжди плюс для такого роду веб-застосунків.

Решти етапів не були об’єднані в більші угруповання. Раси та об’єднані класи будуть мати вибір формату «каруселі». Статі матимуть вибір між двома малюнками, де при натисканні обирається конкретний. І останній етап зміни одягу, включатиме формат «каруселі» для кожного типу: голова, тіло, ноги. Змінюватись вони будуть прямо на модельці раси та статі, які були обрані раніше.

У висновку, лінійна послідовність це найкращий варіант для такого типу проєкту, де фінальне рішення буде базуватись на попередніх виборах. За цими даними була сформована діаграма послідовностей (рис. 2.2). Тут було згадано як кожен етап створення зображення з дизайном. Так і окремі функції як завантаження зображення та генерація хеш-коду.

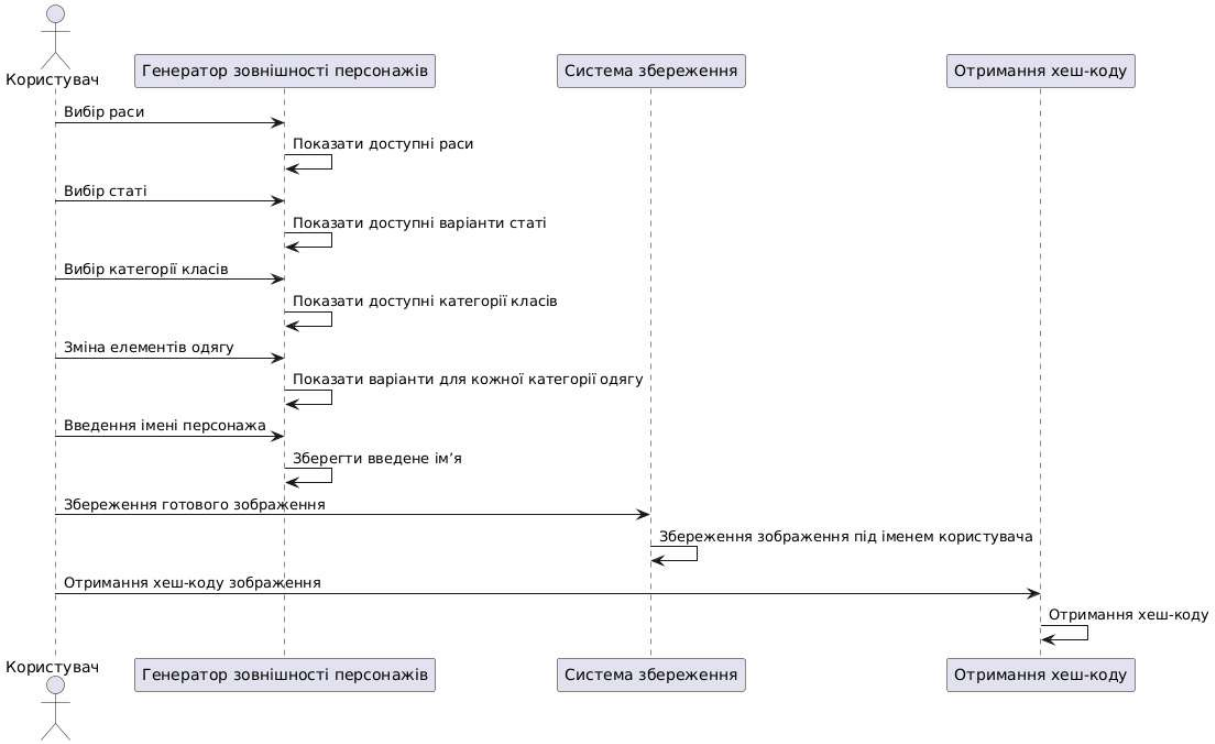

Рисунок 2.2 – Діаграма варіантів використання користувача

В підсумку, така система варіантів використання, даватиме найшвидший і найпростіший результат по створенню дизайну для ігрових персонажів

2.3 Робота з бекенд частиною

Для підготовки, було створено базу даних через PostgreSQL (рис. 2.3), та проведено ряд міграцій для перевірки роботи. На даному етапі буде розроблено лише функцію для генерації хеш-коду.

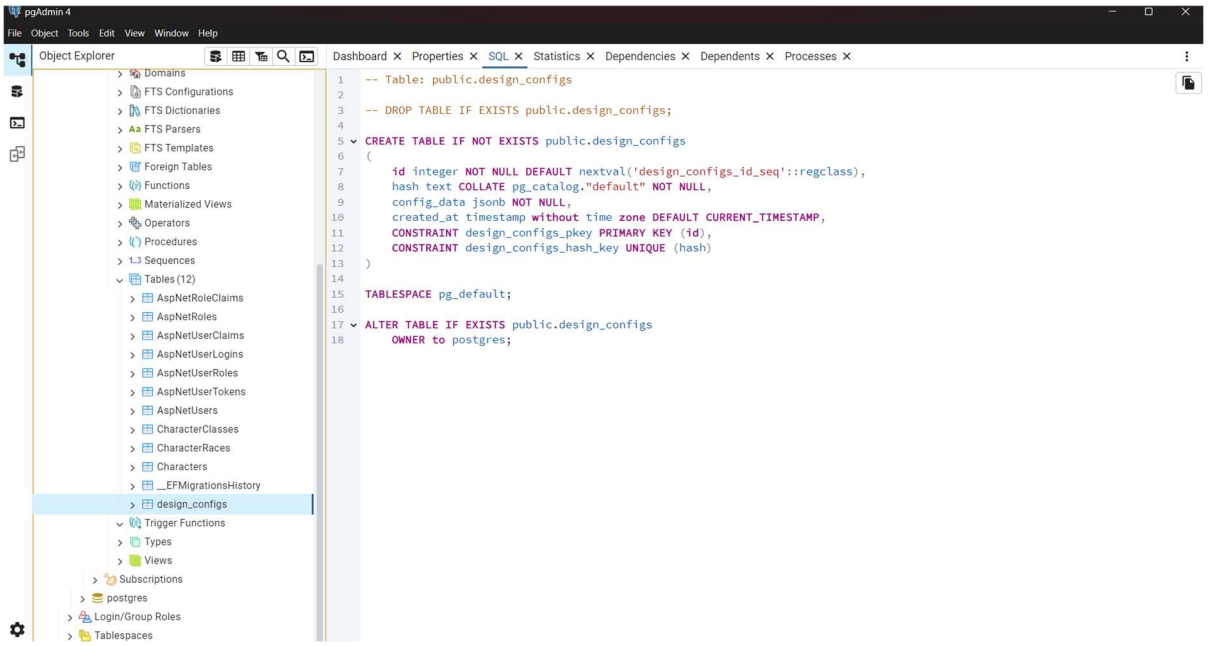

Рисунок 2.3 – інтерфейс роботи з таблицями в pgAdmin 4

Для планування архітектури та моделювання структурою даних, було створено діаграму класів (рис. 2.4). Нижче описано коротко кожен клас.

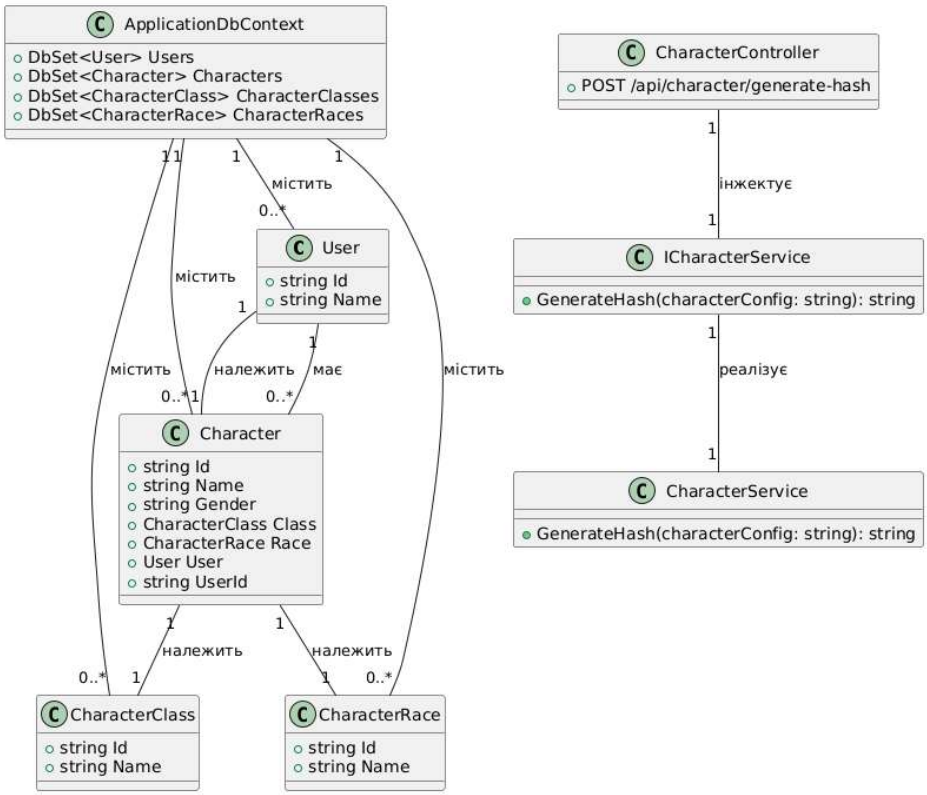

Рисунок 2.4 – Діаграма класів

У цій діаграмі класів:

1. User — клас користувача, має властивості Id та Name, зв'язаний з класом Character.
2. Character — клас для зберігання інформації про персонажа, містить посилання на клас та расу, а також на користувача.
3. CharacterClass — клас для опису класу персонажа.
4. CharacterRace — клас для опису раси персонажа.
5. ApplicationDbContext — контекст бази даних для роботи з таблицями користувачів, персонажів, класів та рас.
6. ICharacterService — інтерфейс для сервісу з роботою з персонажами, описує метод для генерації хешу.
7. CharacterService — реалізація сервісу для роботи з персонажами.
8. CharacterController — контролер API для генерації хешу через endpoint.

User.cs – Модель користувача, яка наслідує стандартний клас IdentityUser для інтеграції з ASP.NET Identity (рис. 2.5). Його принцип роботи полягає в тому, що він додає додаткові властивості (наприклад, Name) та встановлює зв’язок "користувач — персонажі".

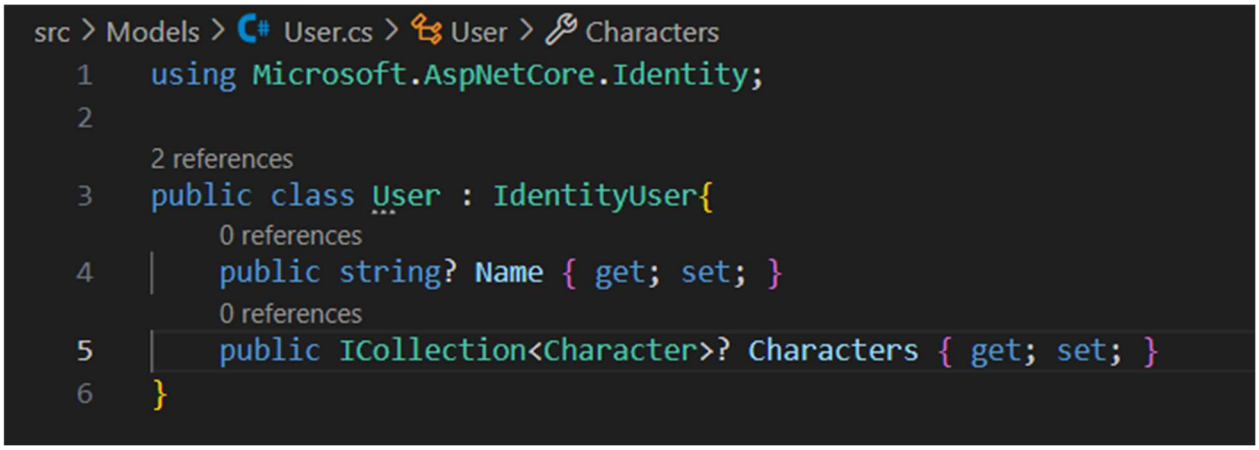

Рисунок 2.5 – Клас User

Character.cs – Модель для зберігання інформації про персонажа (рис. 2.6). Принцип роботи: Містить властивості для ідентифікатора, імені, статі, класу, раси

та користувача. Визначає зовнішні ключі та навігаційні властивості для зв’язку з іншими таблицями.

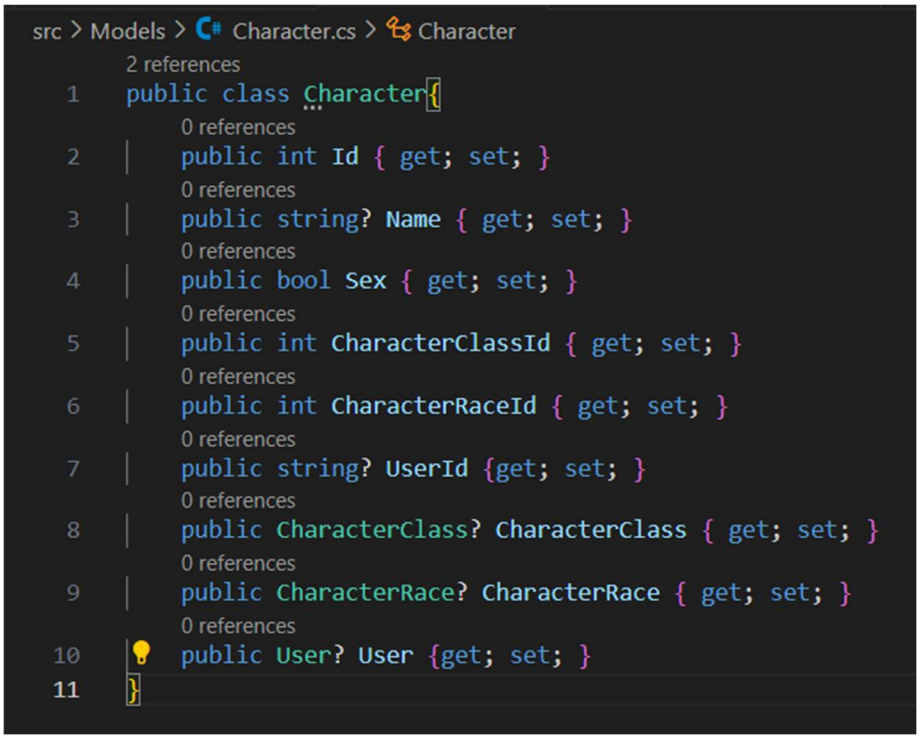

Рисунок 2.6 – Клас Character

CharacterClass.cs – Модель для класу персонажа. Містить властивості для ідентифікатора та назви класу (рис. 2.7).

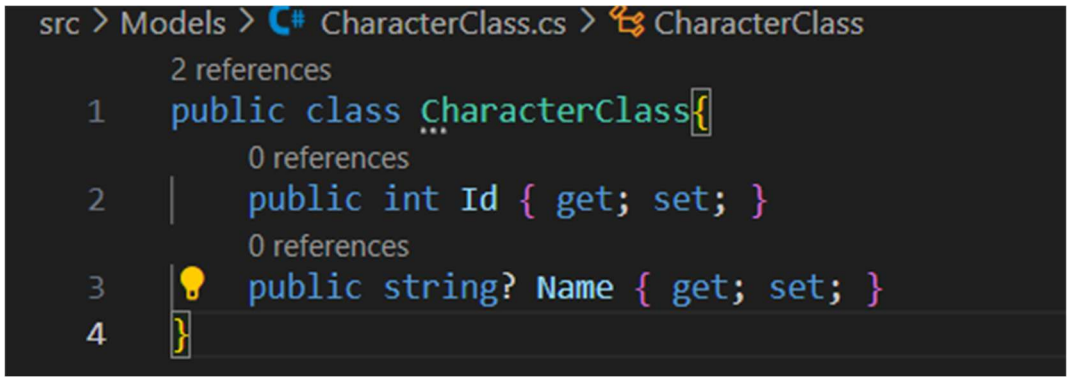

Рисунок 2.7 – Клас CharacterClass

CharacterRace.cs – Модель для раси персонажа. Містить властивості для ідентифікатора та назви раси (рис. 2.8).

```
src > Models > CharacterRace.cs > CharacterRace
    2 references
1   public class CharacterRace{
        0 references
2       public int Id { get; set; }
        0 references
3       public string? Name { get; set; }
4   }
```

Рисунок 2.8 – Клас CharacterRace

ApplicationDbContext – цей файл створюється для налаштування контексту бази даних Entity Framework Core (рис. 2.9). Він наслідує IdentityDbContext<User>, що дозволяє використовувати ASP.NET Identity для роботи з користувачами. Цей файл визначає таблиці (DbSet) для персонажів, класів і рас, та дозволяє виконувати CRUD-операції з цими сутностями через EF Core.

```
src > ApplicationDbContext.cs > ApplicationDbContext
1   using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
2   using Microsoft.EntityFrameworkCore;
3
    5 references
4   public class ApplicationDbContext : IdentityDbContext<User>
5   {
        0 references
6       public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }
7
        0 references
8       public DbSet<Character> Characters { get; set; }
        0 references
9       public DbSet<CharacterClass> CharacterClasses { get; set; }
        0 references
10      public DbSet<CharacterRace> CharacterRaces { get; set; }
11  }
```

Рисунок 2.9 – Файл контексту бази даних

ICharacterService – Інтерфейс для сервісу роботи з персонажами (рис. 2.10). Описує метод для генерації хешу на основі конфігурації персонажа.

```
src > Interfaces > ICharacterService.cs > ICharacterService > GenerateHash
    4 references
1   public interface ICharacterService
2   {
        2 references
3       string GenerateHash(object configData);
4   }
5
```

Рисунок 2.10 – інтерфейс для сервісу з роботою з персонажами

CharacterService – Реалізація сервісу для роботи з персонажами (рис. 2.11). Принцип полягає в тому, що Метод GenerateHash приймає об'єкт конфігурації, серіалізує його в JSON, обчислює SHA256-хеш і повертає його у вигляді рядка. Також CharacterService використовується для унікальної ідентифікації конфігурації персонажа.

```
src > Services > CharacterService.cs > ...
1   using System.Security.Cryptography;
2   using System.Text;
3   using Newtonsoft.Json;
4
    1 reference
5   public class CharacterService : ICharacterService
6   {
        2 references
7       public string GenerateHash(object configData)
8       {
9           using (SHA256 sha256Hash = SHA256.Create())
10          {
11              string jsonConfig = JsonConvert.SerializeObject(configData);
12              byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(jsonConfig));
13
14              StringBuilder builder = new StringBuilder();
15              foreach (byte b in bytes)
16              {
17                  builder.Append(b.ToString("x2"));
18              }
19              return builder.ToString();
20          }
21      }
22  }
23
```

Рисунок 2.11 – Сервіс CharacterService

CharacterController – контроллер для роботи з API персонажів (рис. 2.12). Має endpoint POST /api/character/generate-hash, який приймає конфігурацію персонажа, викликає сервіс для генерації хешу та повертає його клієнту. Інжектує сервіс через конструктор.

src > Controllers > CharacterController.cs > CharacterController

```
using Microsoft.AspNetCore.Mvc;

namespace CharacterMakerApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class CharacterController : ControllerBase
    {
        2 references
        private readonly ICharacterService _characterService;

        0 references
        public CharacterController(ICharacterService characterService)
        {
            _characterService = characterService;
        }

        [HttpGet("test")]
        0 references
        public IActionResult Test() => Ok("Works");

        [HttpPost("generate-hash")]
        0 references
        public IActionResult GenerateCharacterHash([FromBody] object configData)
        {
            string hash = _characterService.GenerateHash(configData);

            return Ok(new { hash });
        }
    }
}
```

Рисунок 2.11 – Контроллер CharacterController

Також присутній Файл appsettings.json, який містить основні налаштування для ASP.NET Core застосунку. Цей файл використовується для централізованого зберігання конфігурації, яку застосунок зчитує під час запуску.

У висновку, всі вищеописані класи, разом реалізують базову архітектуру веб-застосунку для зберігання, обробки та ідентифікації персонажів.

РОЗДІЛ 3. КОНСТРУЮВАННЯ ТА ТЕСТУВАННЯ ГЕНЕРАТОРА ЗОВНІШНОСТІ

3.1 Визначення основних компонентів проєкту

Завдяки компонентній структурі ReactJS, можна зручно розбити усі потрібні функції для виклику їх у основному файлі CharacterCreator. В першу чергу це було зроблено для спрощення основного файлу. Також компонентна складова дозволяє створити один компонент та викликати його безліч разів.

Також для зручності, компоненти розширення .tsx, з другорядними функціями, як зміна мови чи введення імені персонажа, було винесено в окрему папку AdditionalFunctions. Також там знаходиться файл translation.ts, де зібрано варіанти слів в українському та англійському перекладі. Вони потрібні для реалізації функції перекладу.

У папці assets знаходиться папка для шрифтів зі шрифтом CyrillicGoth та додаткові зображення які не відносяться до зображень персонажів, як от фонове зображення та зображення для стрілок на кнопки.

У папці helpers\jsons знаходяться JSON-файли з даними для застосунку:

- race.json — містить масив об'єктів з посиланнями на зображення чоловічих та жіночих персонажів різних рас.
- classes.json — містить масив об'єктів з інформацією про класи персонажів, включаючи назву класу та шляхи до зображень елементів одягу (голова, тіло, ноги).

У папці types знаходяться TypeScript-інтерфейси, які описують структуру основних даних для застосунку Ці файли використовуються для підвантаження даних про раси та класи у головному компоненті.

Решта файлів це основні компоненти які будуть описані далі, файл main.tsx, який є точкою входу для React-додатку, імпортує необхідні модулі та рендерить головний компонент додатку. І в кінці файл index.css, який містить глобальні стилі

для всього застосунку, а саме полів, кнопок, контейнерів та іншого. Ці стилі забезпечують сучасний вигляд інтерфейсу та адаптацію під тематику застосунку.

CSS (англ. Cascading Style Sheets, укр. Каскадні таблиці стилів) — спеціальна мова, яка використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних [14].

Найбільш часто CSS використовують для візуальної презентації сторінок, написаних на HTML та XHTML, але формат CSS може застосовуватись і до інших видів XML-документів.

Специфікації CSS були створені і розвиваються Консорціумом Всесвітньої павутини - W3C.

Список усіх компонентів генератора зовнішності показано на рисунку 3.1.

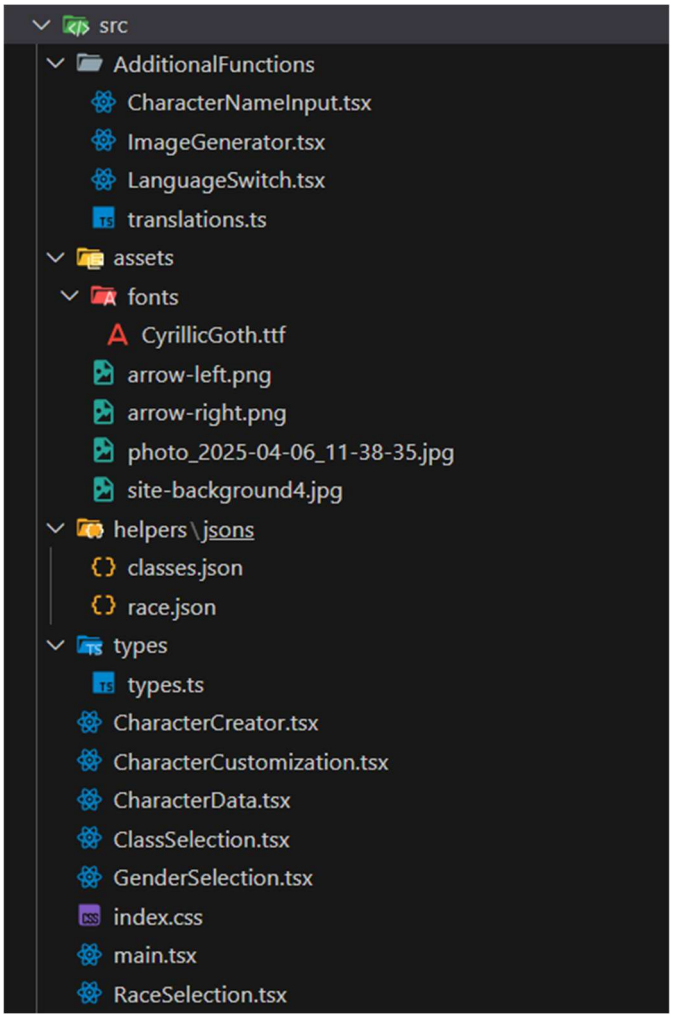

Рисунок 3.1 – список усіх компонентів генератора

Основні компоненти: RaceSelection, GenderSelection, ClassSelection, CharacterCustomization, CharacterData та CharacterCreator, відповідають за стадії самого проекту, вибір рішень які робляться протягом нього, та власне головний компонент CharacterCreator.

3.2 Розробка компонентів вибору

Ми вже визначили нашу послідовність дій у створенні персонажа, тож перейдемо до створення генератора

Першим етапом йде вибір раси. Створюємо компонент вибору раси RaceSelection. Його основні функції це відображення поточної раси, за допомогою малюнку та назви, можливість перемикання рас стрілками вліво/вправо, та перехід до наступного етапу по кнопці «Далі». Приймає пропс races – масив об’єктів рас, де знаходяться зображення чоловічих і жіночих моделей тіла. Лістинг коду цього компоненту показано на рисунках 3.2-3.3. Вигляд стадії вибору раси на сайті, зображено на рисунку 3.4.

```tsx
import React from "react";
import arrowRight from "./assets/arrow-right.png";
import arrowLeft from "./assets/arrow-left.png";
import { Race } from "./types/types";
import {  Translation } from "./types/types";

interface RaceSelectionProps {
  races: Race[];
  selectedRaceIndex: number;
  handlePrevRace: () => void;
  handleNextRace: () => void;
  handleNext: () => void;
  t: Translation;
}

const RaceSelection: React.FC<RaceSelectionProps> = ({
  races,
  selectedRaceIndex,
  handlePrevRace,
  handleNextRace,
  handleNext,
  t,
}) => {
  return (
    <div>
      <div
```

Рисунок 3.2 – Код компоненту RaceSelection

```
        style={{
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
        <button onClick={handlePrevRace} className="arrow-button">
          <img src={arrowLeft} alt="Next" />
        </button>
        <div>
          <img
            src={races[selectedRaceIndex].maleImage}
            alt={t.nameRace[selectedRaceIndex]}
            style={{ width: "200px" }}
          />
          <p>{t.nameRace[selectedRaceIndex]}</p>
        </div>
        <button onClick={handleNextRace} className="arrow-button">
          <img src={arrowRight} alt="Next" />
        </button>
      </div>
      <button onClick={handleNext} style={{ marginTop: "10px" }}>
        {t.next}
      </button>
    </div>
  );
};

export default RaceSelection;
```

Рисунок 3.3 – Код компоненту RaceSelection

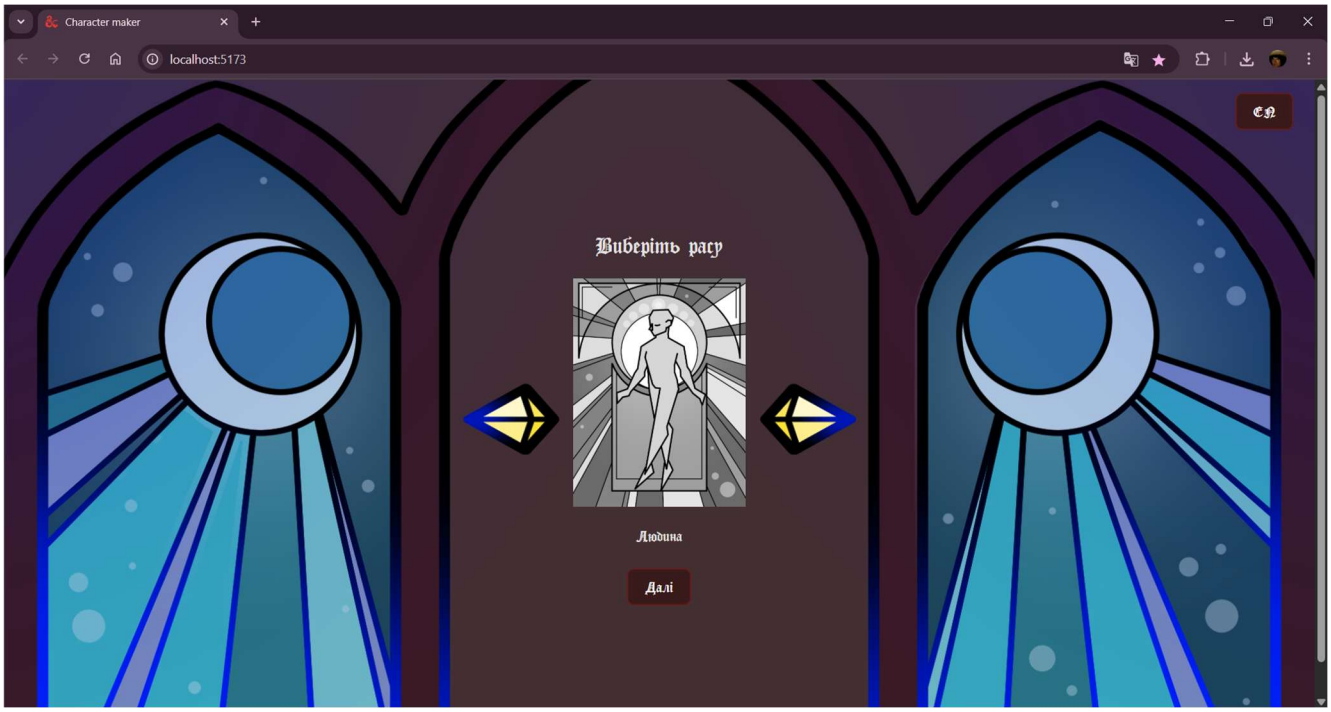

Рисунок 3.4 – Етап вибору раси

Далі йде стадія вибору статі. Для неї створено компонент GenderSelection. Він працює простіше, просто відображаючи дві моделі тіла, для раніше вибраної раси, і дає змогу обрати одну з них. По кліку на зображення відбувається перехід на наступну стадію. Компонент так само приймає пропс races де розташовані зображення чоловічих та жіночих тіл. Лістинг коду компоненту GenderSelection

показано на рисунках 3.5-3.6. Вигляд етапу вибору статі на сайті, зображено на рисунку 3.7.

```tsx
import React from "react";
import { Race } from "./types/types";
import { Translation } from "./types/types";
interface GenderSelectionProps {
  races: Race[];
  selectedRaceIndex: number;
  handleGenderSelect: (gender: true | false) => void;
  t: Translation;
}

const GenderSelection: React.FC<GenderSelectionProps> = ({
  races,
  selectedRaceIndex,
  handleGenderSelect,
  t,
}) => {
  return (
    <div>
      <div
        style={{
          display: "flex",
          justifyContent: "center",
          gap: "20px",
          marginTop: "20px",
        }}
      >
```

Рисунок 3.5 – Код компоненту GenderSelection

```tsx
        <div
          onClick={() => handleGenderSelect(true)}
          style={{ cursor: "pointer" }}
        >
          <img
            src={races[selectedRaceIndex].maleImage}
            alt={t.male}
            style={{ width: "150px" }}
          />
          <p>{t.male}</p>
        </div>
        <div
          onClick={() => handleGenderSelect(false)}
          style={{ cursor: "pointer" }}
        >
          <img
            src={races[selectedRaceIndex].femaleImage}
            alt={t.female}
            style={{ width: "150px" }}
          />
          <p>{t.female}</p>
        </div>
      </div>
    </div>
  );
};

export default GenderSelection;
```

Рисунок 3.6 – Код компоненту GenderSelection

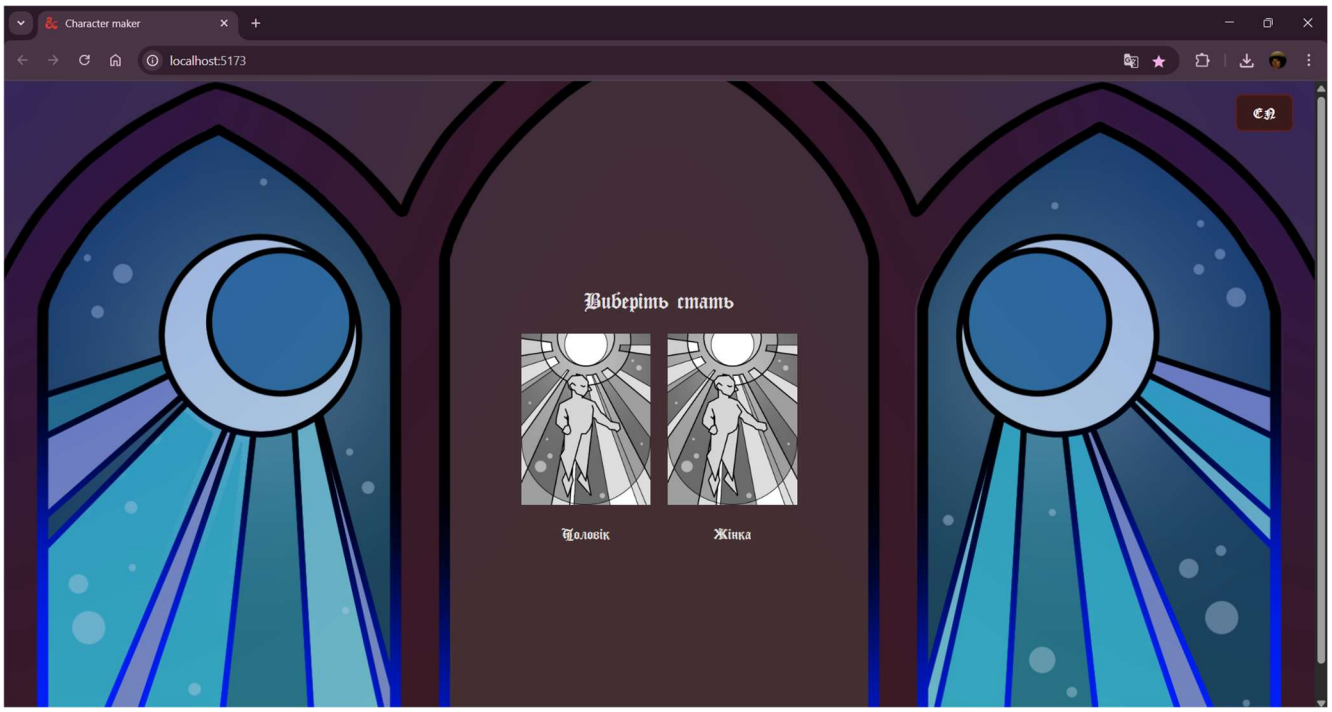

Рисунок 3.7 – Етап вибору статі

Наступним розглянемо компонент ClassSelection, який відповідає за вибір однієї з категорій класів – бойові, спритні та магічні. Він працює так само як і компонент вибору раси – по стрілках вліво/вправо можна переглядати зображення з категоріями класів, та по кнопці «Далі» перейти до наступного етапу. У пропсах компонент приймає масив об’єктів classes. Об’єкти на даному етапі не використовуються, проте він потрібен для індексації. Код компоненту ClassSelection показано на рисунках 3.8-3.9. Вигляд етапу вибору категорії класів на сайті, зображено на рисунках 3.10-3.12.

```
src > ClassSelection.tsx > ClassSelection
import React from "react";
import arrowRight from "./assets/arrow-right.png";
import arrowLeft from "./assets/arrow-left.png";
import { CharacterClass, Translation } from "./types/types";

interface ClassSelectionProps {
  classes: CharacterClass[];
  selectedClassIndex: number;
  handlePrevClass: () => void;
  handleNextClass: () => void;
  setStep: React.Dispatch<React.SetStateAction<number>>;
  t: Translation;
}

const ClassSelection: React.FC<ClassSelectionProps> = ({
  classes,
  selectedClassIndex,
  handlePrevClass,
  handleNextClass,
  setStep,
  t,
}) => {
  return (
    <div>
      <div
        style={{
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
```

Рисунок 3.8 – Код компоненту ClassSelection

```
        <button onClick={handlePrevClass} className="arrow-button">
          <img src={arrowLeft} alt="Prev" />
        </button>
        <div>
          <img
            src={classes[selectedClassIndex].icon}
            alt="Class Icon"
            className="icon-img"
          />
          <p>{t.nameClass[selectedClassIndex]}</p>
        </div>
        <button onClick={handleNextClass} className="arrow-button">
          <img src={arrowRight} alt="Next" />
        </button>
      </div>
      <button onClick={() => setStep(4)} style={{ marginTop: "10px" }}>
        {t.next}
      </button>
    </div>
  );
};

export default ClassSelection;
```

Рисунок 3.9 – Код компоненту ClassSelection

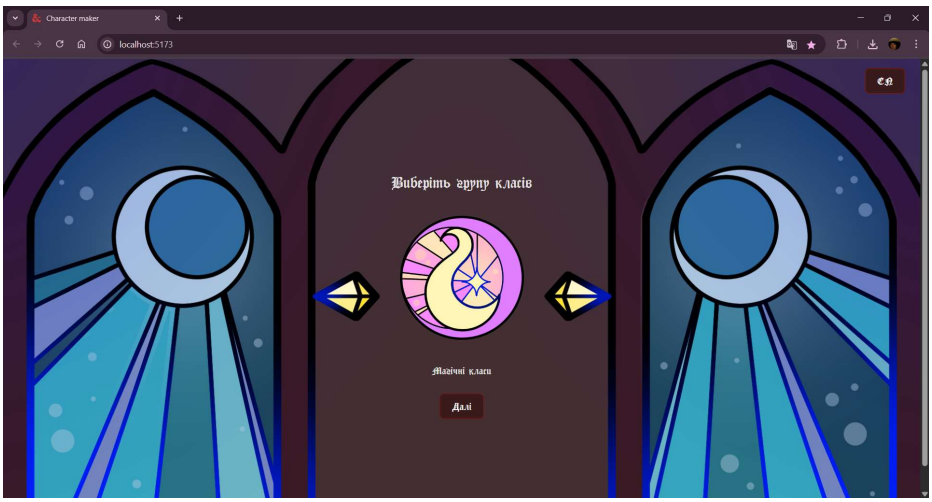

Рисунок 3.10 – Етап вибору категорії класів

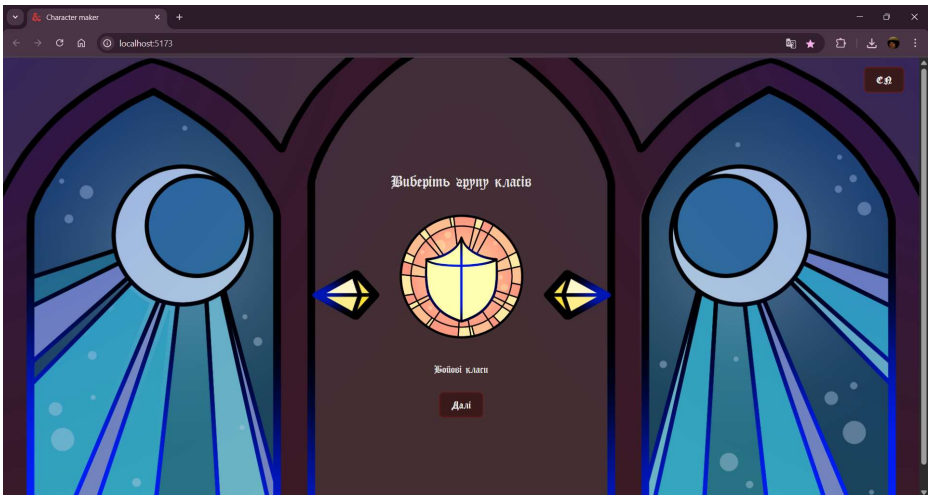

Рисунок 3.11 – Етап вибору категорії класів

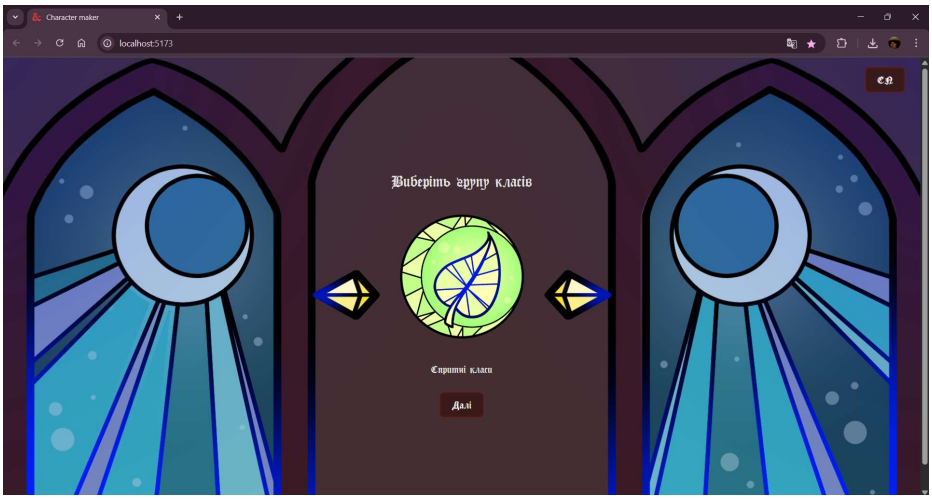

Рисунок 3.12 – Етап вибору категорії класів

Фінальним компонентом є CharacterCustomization, у якому реалізовано підбір елементів одягу для персонажа. Він відображатиме зображення моделі тіла, з урахуванням раніше вибраних раси, статі та категорії класів. По боках розташовано по 3 власно намальованих стрілки, по дві на кожну частину тіла – голова, тіло, ноги. По кліку зображення елементів одягу змінюються, підлаштовуючись під модельку тіла. Підбір елементів одягу до вибраних раніше, в попередніх компонентах, аспектів персонажу, допомагають реалізувати json-файли. Ось як це працює на прикладі виведення основного зображення, на основі попередньо вибраних аспектів:

- classes[selectedClassIndex] — вибирається об'єкт класу персонажа за індексом.
- outfits.find(outfit => outfit.race === raceName) — знаходиться набір одягу для обраної раси.
- sex[selectedGender ? "male" : "female"] — вибирається набір для обраної статі (чоловік/жінка).
- classGroup — містить шлях до основного зображення для цього класу, раси та статі.

Так само це працює і з підбором одягу. Ось приклад з підбором елементу одягу для голови:

- head[headIndex] — вибирає зображення голови за індексом.
- body[bodyIndex] — вибирає зображення тіла за індексом.
- legs[legsIndex] — вибирає зображення ніг за індексом.

Логіка компоненту показана на рисунках 3.13-3.18 та загальний вигляд етапу підбору одягу на сайті показано на рисунку 3.19

```
src > CharacterCustomization.tsx > CharacterCustomization
1   import React from "react";
2   import arrowRight from "./assets/arrow-right.png";
3   import arrowLeft from "./assets/arrow-left.png";
4   import { CharacterClass, Race } from "./types/types";
5
6   interface CharacterCustomizationProps {
7     races: Race[];
8     selectedRaceIndex: number;
9     selectedGender: boolean;
10    classes: CharacterClass[];
11    selectedClassIndex: number;
12    headIndex: number;
13    bodyIndex: number;
14    legsIndex: number;
15    handlePrevItem: (
16      setIndex: React.Dispatch<React.SetStateAction<number>>,
17      itemArray: string[]
18    ) => void;
19    handleNextItem: (
20      setIndex: React.Dispatch<React.SetStateAction<number>>,
21      itemArray: string[]
22    ) => void;
23    setHeadIndex: React.Dispatch<React.SetStateAction<number>>;
24    setBodyIndex: React.Dispatch<React.SetStateAction<number>>;
25    setLegsIndex: React.Dispatch<React.SetStateAction<number>>;
26  }
27
28  const CharacterCustomization: React.FC<CharacterCustomizationProps> = ({
29    races,
30    selectedRaceIndex,
31    selectedGender,
32    classes,
33    selectedClassIndex,
34    headIndex,
35    bodyIndex,
36    legsIndex,
37    handlePrevItem,
```

Рисунок 3.13 – Код компоненту CharacterCustomization

```
38    handleNextItem,
39    setHeadIndex,
40    setBodyIndex,
41    setLegsIndex,
42  }) => {
43    const raceName = races[selectedRaceIndex].race;
44    const genderKey = selectedGender ? "male" : "female";
45    const selectedOutfit = classes[selectedClassIndex].outfits.find(
46
47      (outfit) => outfit.race === raceName
48
49    );
50
51
52    if (!selectedOutfit) {
53      return (
54        <div style={{ color: "red" }}>
55          ❌ Outfit not found for race <strong>{raceName}</strong>
56        </div>
57      );
58    }
59
60    const sexOutfit = selectedOutfit.sex[genderKey];
```

Рисунок 3.14 – Код компоненту CharacterCustomization

```
return (
  <div
    style={{
      display: "flex",
      justifyContent: "center",
      alignItems: "center",
    }}
  >
    <div
      style={{
        display: "flex",
        flexDirection: "column",
        justifyContent: "space-between",
        height: "400px",
      }}
    >
      <button onClick={() => handlePrevItem(setHeadIndex, sexOutfit.head)} className="arrow-button">
        <img src={arrowLeft} alt="Prev Head" />
      </button>
      <button onClick={() => handlePrevItem(setBodyIndex, sexOutfit.body)} className="arrow-button">
        <img src={arrowLeft} alt="Prev Body" />
      </button>
      <button onClick={() => handlePrevItem(setLegsIndex, sexOutfit.legs)} className="arrow-button">
        <img src={arrowLeft} alt="Prev Legs" />
      </button>
    </div>

    <div
      id="character-container"
      style={{
        position: "relative",
        width: "375px",
        height: "500px",
        overflow: "hidden",
      }}
    >
```

Рисунок 3.15 – Код компоненту CharacterCustomization

```
        <img
          src={sexOutfit.classGroup}
          alt="Class Group"
          className="character-image"
          style={{
            width: "100%",
            height: "100%",
            objectFit: "cover",
            position: "absolute",
            top: "0",
            left: "0",
          }}
        />
        <img
          src={sexOutfit.head[headIndex]}
          alt="Head"
          className="character-image"
          style={{
            position: "absolute",
            top: "0",
            left: "0",
            width: "100%",
            height: "100%",
            objectFit: "cover",
          }}
        />
        <img
          src={sexOutfit.body[bodyIndex]}
          alt="Body"
          className="character-image"
          style={{
            position: "absolute",
            top: "0",
            left: "0",
            width: "100%",
            height: "100%",
```

Рисунок 3.16 – Код компоненту CharacterCustomization

```
            left: "0",
            width: "100%",
            height: "100%",
            objectFit: "cover",
          }}
        />
        <img
          src={sexOutfit.legs[legsIndex]}
          alt="Legs"
          className="character-image"
          style={{
            position: "absolute",
            top: "0",
            left: "0",
            width: "100%",
            height: "100%",
            objectFit: "cover",
          }}
        />
      </div>

      <div
        style={{
          display: "flex",
          flexDirection: "column",
          justifyContent: "space-between",
          height: "400px",
        }}
```

Рисунок 3.17 – Код компоненту CharacterCustomization

```
        >
          <button onClick={() => handleNextItem(setHeadIndex, sexOutfit.head)} className="arrow-button">
            <img src={arrowRight} alt="Next Head" />
          </button>
          <button onClick={() => handleNextItem(setBodyIndex, sexOutfit.body)} className="arrow-button">
            <img src={arrowRight} alt="Next Body" />
          </button>
          <button onClick={() => handleNextItem(setLegsIndex, sexOutfit.legs)} className="arrow-button">
            <img src={arrowRight} alt="Next Legs" />
          </button>
        </div>
      </div>
    );
};

export default CharacterCustomization;

```

Рисунок 3.18 – Код компоненту CharacterCustomization

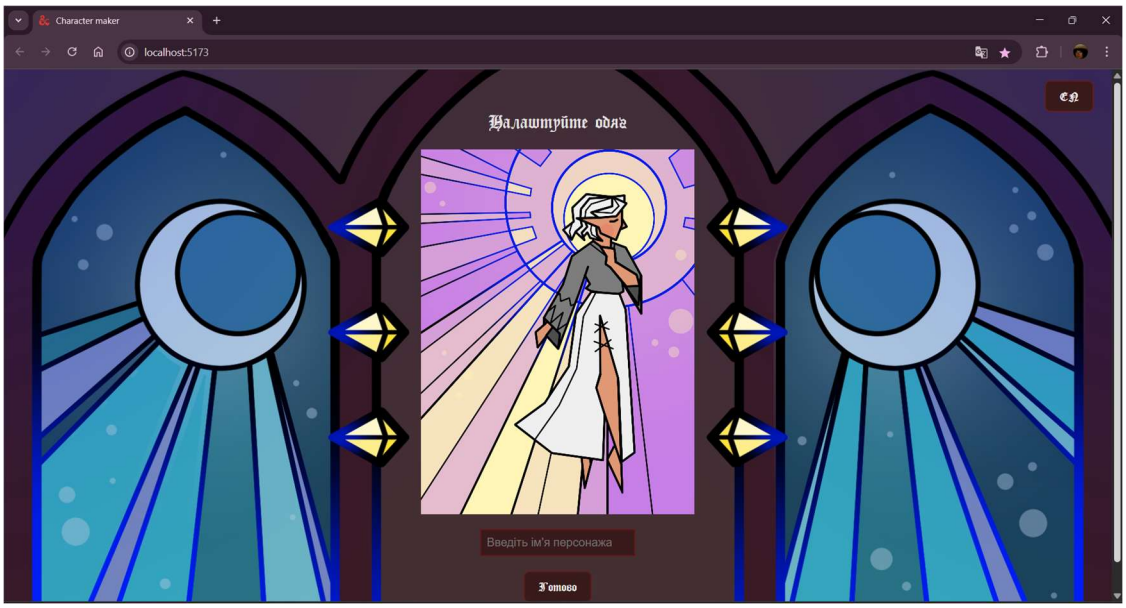

Рисунок 3.19 – Етап вибору елементів одягу

3.3 Розробка додаткових компонентів

Після завершення розробки основних компонентів для роботи над зовнішністю ігрових персонажів, можна перейти до компонентів які забезпечать додаткові функції для покращення генератора.

Першим таким компонентом став LanguageSwitch. Він дозволяє перемикати мову інтерфейсу між українською ("ua") та англійською ("en"). Приймає в себе пропси language що є поточною мовою (українська або англійська) та функцію зміни мови setLanguage.

В релізації, відображає функціональну кнопку, при натискання на яку, змінюється мова інтерфейсу. Також на кнопці показується код мови, на яку буде перемикання (UA/EN).

Компонент взаємодіє з файлом translations.ts, що є словником з усіх зібраних термінів які відображаються на сайті.

Код з логікою компоненту показано на рисунку 3.20. Відображення кнопки перемикання на сайті та її функціонал на рисунках 3.21-3.23. Вміст файлу translations.ts – на рисунку 3.24.

```tsx
import React from "react";

interface LanguageSwitchProps {
  language: "ua" | "en";
  setLanguage: React.Dispatch<React.SetStateAction<"ua" | "en">>;
}

const LanguageSwitch: React.FC<LanguageSwitchProps> = ({ language, setLanguage }) => {
  return (
    <div style={{ position: "fixed", top: 10, right: 20 }}>
      <button onClick={() => setLanguage((prev) => (prev === "ua" ? "en" : "ua"))}>
        {language === "ua" ? "EN" : "UA"}
      </button>
    </div>
  );
};

export default LanguageSwitch;
```

Рисунок 3.20 – Код компоненту LanguageSwitch

Рисунок 3.21 – Кнопка перемикання мови

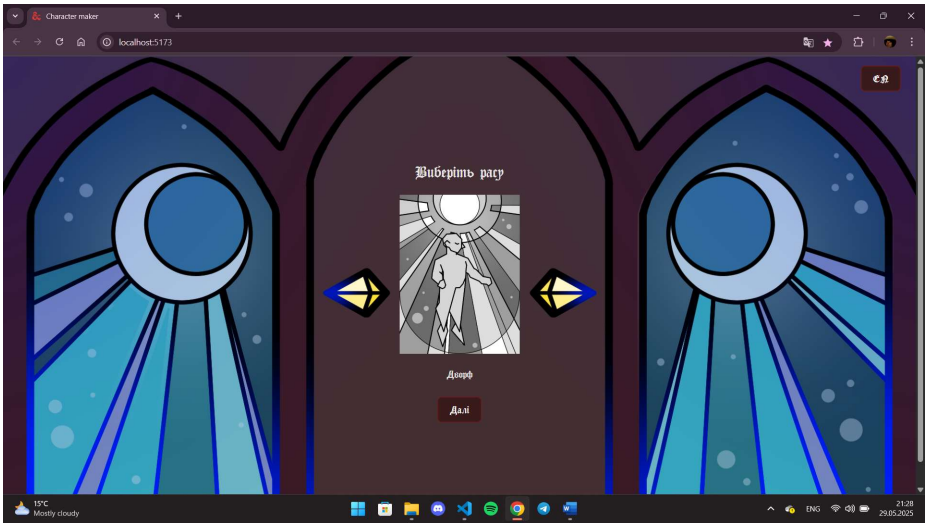

Рисунок 3.22 – Інтерфейс українською мовою

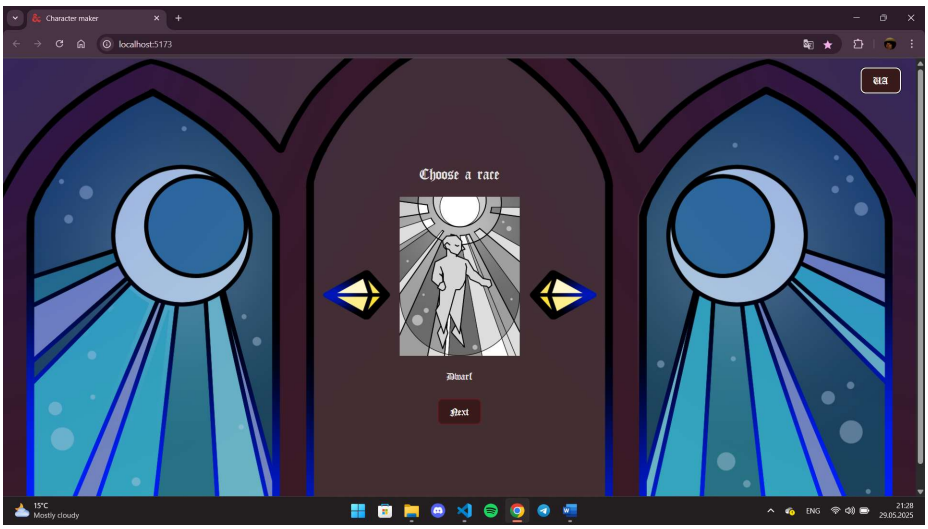

Рисунок 3.23 – Інтерфейс англійською мовою, після перемикання по кнопці

```ts
const translations = {
  ua: {
    chooseRace: "Виберіть расу",
    nameRace: ["Дворф", "Ельф", "Напівросли к", "Людина", "Драконороджений", "Напів-орк", "Гоблін", "Тифлінг"],
    chooseGender: "Виберіть стать",
    male: "Чоловік",
    female: "Жінка",
    chooseClass: "Виберіть групу класів",
    nameClass: ["Магічні класи", " Спритні класи", "Бойові класи",],
    customizeOutfit: "Налаштуйте одяг",
    next: "Далі",
    done: "Готово",
    enterName: "Введіть ім'я персонажа",
  },
  en: {
    chooseRace: "Choose a race",
    nameRace: ["Dwarf", "Elf", "Halfling", "Human", "Dragonborn", "Half-Orc", "Goblin", "Tiefling"],
    chooseGender: "Choose gender",
    male: "Male",
    female: "Female",
    chooseClass: "Choose a class group",
    nameClass: ["Magic classes", " Agile classes", "Magic classes"],
    customizeOutfit: "Customize outfit",
    next: "Next",
    done: "Done",
    enterName: "Enter character name",
  },
};

export type Language = keyof typeof translations;
export default translations;
```

Рисунок 3.24 – Файл translations.ts

Наступним на черзі компонент для створення поля для введення імені готового персонажа. Він оперує пропсами characterName, що є поточним іменем персонажа, та setCharacterName – функцією для зміни цього імені. А також з t, як об’єктом перекладу для локалізації підказки на текстовому полі. Все це наведено у коді компоненту (рис. 3.25)

Сам компонент вміщає в себе просте текстове поле, та placeholder з підказкою яка відображається мовою, яка зараз встановлена на сайті.

При завантаженні зображення готового персонажу, файл завантажиться саме під іменем яке було введено у текстове поле. Якщо такого імені не було введено, зображення буде завантажено під простим іменем character.

Вигляд поля для введення імені персонажа показано на рисунку 3.26. Приклад завантаження зображення під введеним іменем та без нього – на рисунках 3.27-3.28

```tsx
import React from "react";

interface CharacterNameInputProps {
  characterName: string;
  setCharacterName: React.Dispatch<React.SetStateAction<string>>;
  t: any;
}

const CharacterNameInput: React.FC<CharacterNameInputProps> = ({ characterName, setCharacterName, t }) => {
  return (
    <div style={{ marginTop: "20px" }}>
      <input
        type="text"
        value={characterName}
        onChange={(e) => setCharacterName(e.target.value)}
        placeholder={t.enterName}
        style={{ padding: "0.5em", fontSize: "1em" }}
      />
    </div>
  );
};

export default CharacterNameInput;
```

Рисунок 3.25 – Код компоненту CharacterNameInput

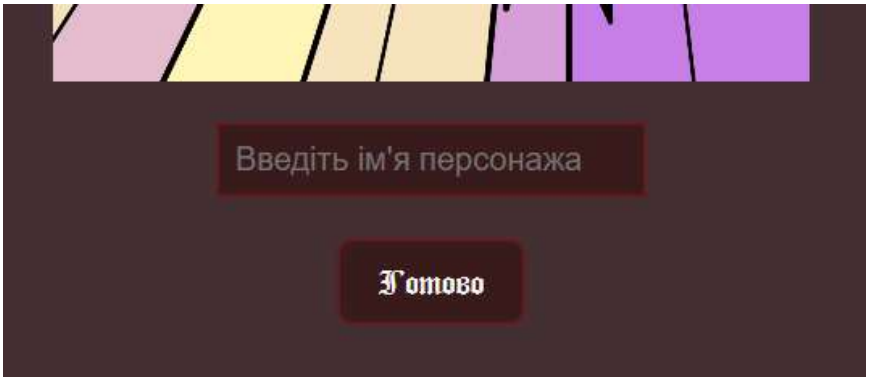

Рисунок 3.26 – Вигляд поля для введення імені персонажу на сайті

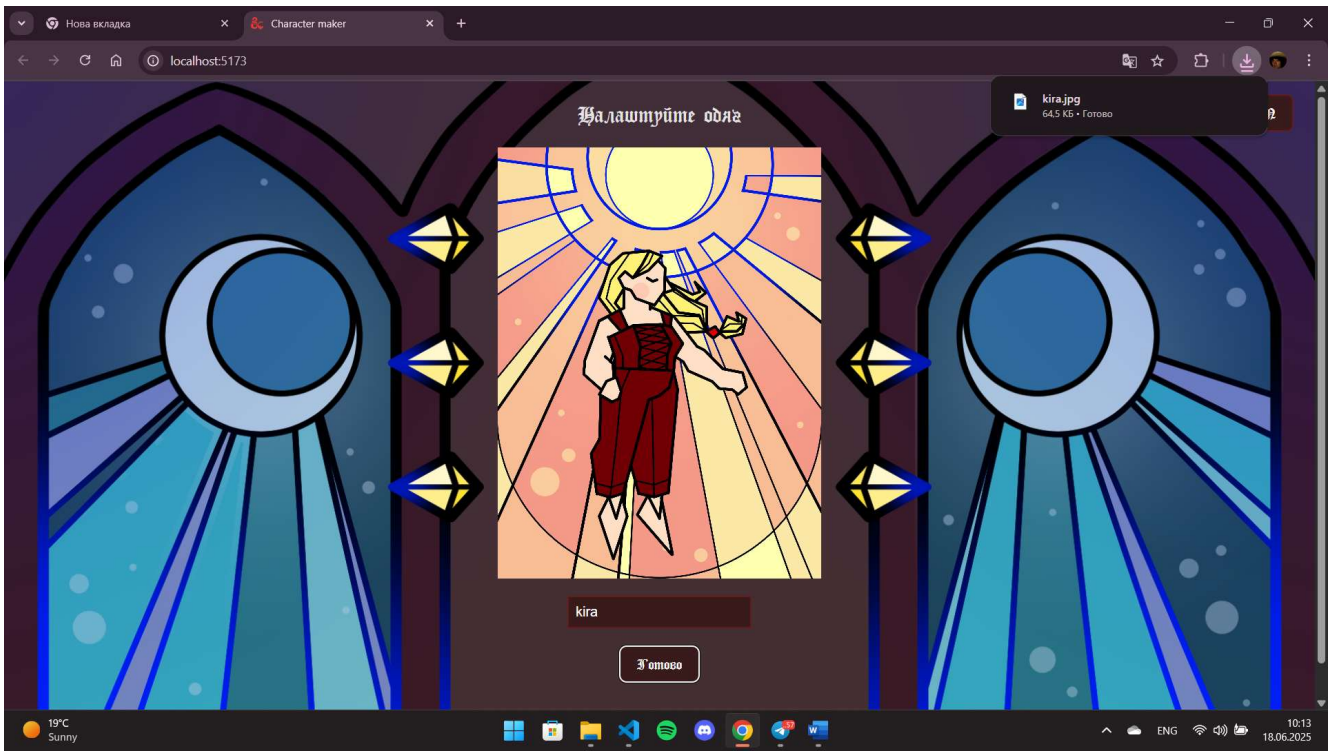

Рисунок 3.27 – Приклад завантаження готового зображення з вписаним іменем

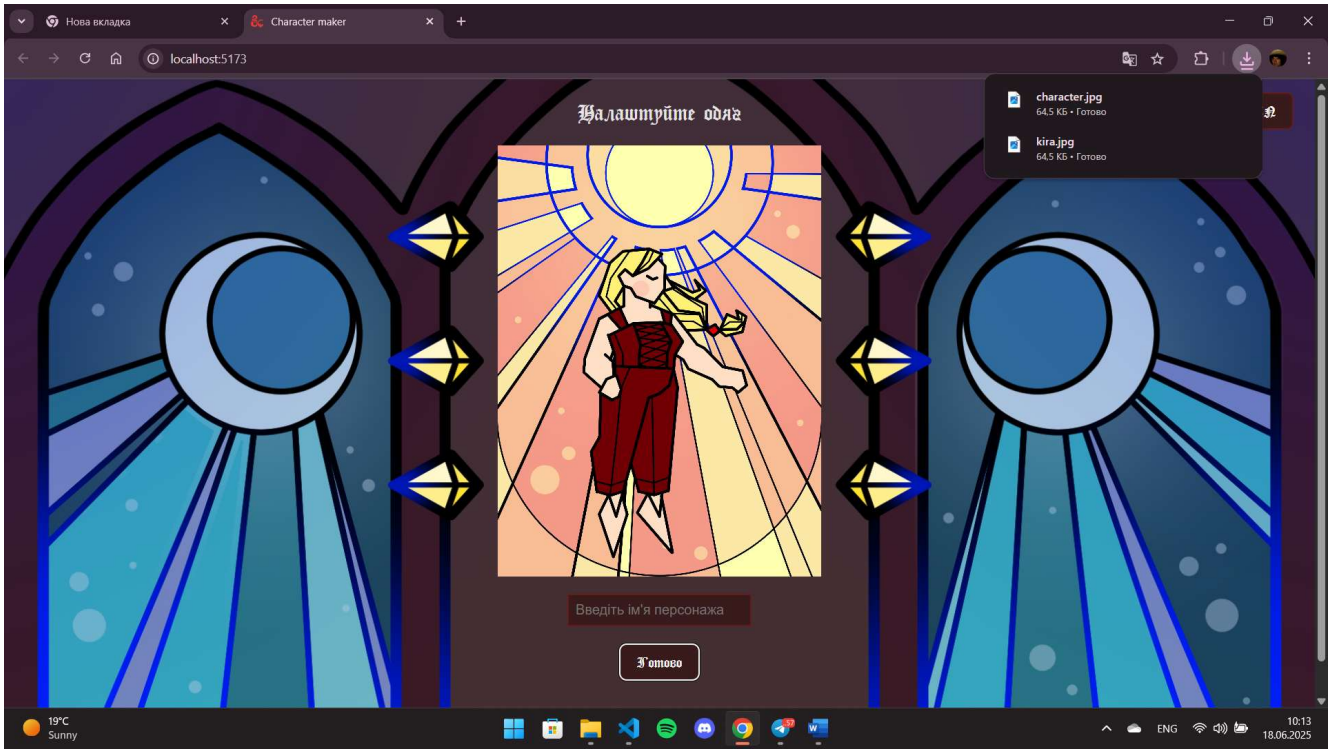

Рисунок 3.28 – Приклад завантаження готового зображення без вписаного імені

Останнім додатковим компонентом є ImageGenerator, який дозволяє зберегти готове зображення як файл. Він приймає пропси characterName, з іменем персонажу

яке використовується в імені файлу. Та generateHash – функцію яка викликається перед генерацією зображення та відповідно генерує готовий хеш.

Компонент реалізовано у кнопці готово, яка знаходиться під полем для введення імені. При натисканні на цю кнопку викликається функція generateHash, знаходиться елемент з id "character-container", що є контейнером зображення з персонажем та елементами одягу які співставляються до нього. За допомогою html2canvas робиться скріншот цього зображення, яке автоматично завантажується у форматі JPEG, з раніше написаним іменем персонажа, або "character.jpg" за замовчуванням.

Логіку компоненту наведено у коді (рис. 3.29-3.30). Приклад завантаженого зображення показано на рисунку 3.31. Приклад згенерованого хешу – рисунок 3.32

```tsx
import React from 'react';
import html2canvas from 'html2canvas';

interface ImageGeneratorProps {
  characterName: string;
  t: any;
  generateHash: () => Promise<void>;
}

const ImageGenerator: React.FC<ImageGeneratorProps> = ({ characterName, t, generateHash }) => {
  const handleGenerateImage = async () => {
    await generateHash();

    const characterContainer = document.getElementById("character-container");

    if (characterContainer) {
      html2canvas(characterContainer, {
        width: 375,
        height: 500,
        scale: 1,
        useCORS: true,
        allowTaint: true,
        backgroundColor: "#5a5e6b"
      }).then((canvas) => {
        const imageURL = canvas.toDataURL("image/jpeg");
        const link = document.createElement("a");
        const fileName = characterName ? `${characterName}.jpg` : "character.jpg";
        link.href = imageURL;
        link.download = fileName;
        link.click();
      });
    }
  };
```

Рисунок 3.29 – Код компоненту ImageGenerator

```
  return (
    <button onClick={handleGenerateImage} style={{ marginTop: "20px" }}>
      {t.done}
    </button>
  );
};

export default ImageGenerator;
```

Рисунок 3.30 – Код компоненту ImageGenerator

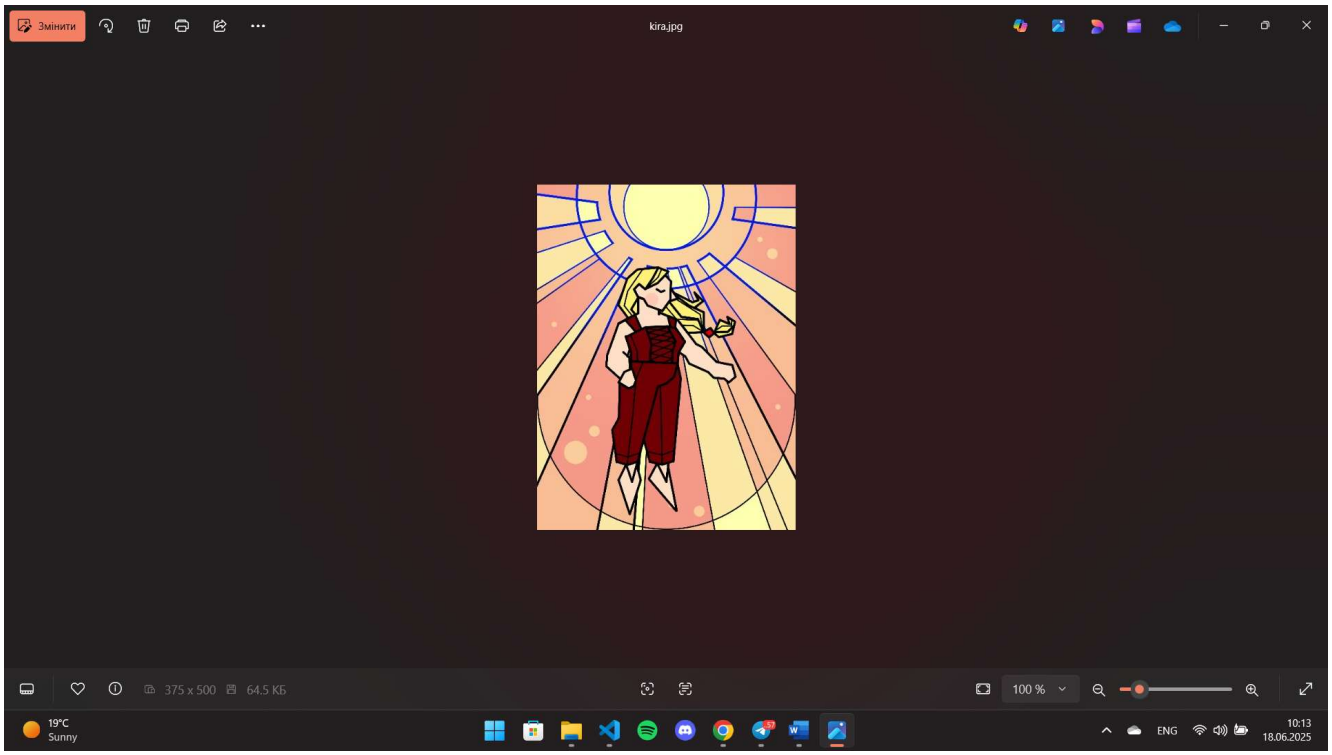

Рисунок 3.31 – Приклад завантаженого зображення

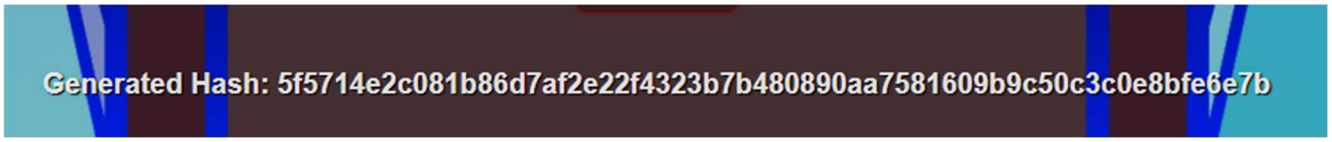

Рисунок 3.32 – Приклад згенерованого хешу

Наостанок розглянемо файл index.css, який відповідає за усі стилі генератора зовнішності, має прописані кольори, шрифти, відмінні компоненти та інше. Ось короткий опис його компонентів:

- body: Центрує увесь контент на сторінці, задає кастомний фон, використовує шрифт "CyrillicGoth".
- :root: Встановлює глобальні стилі для шрифтів, кольорів, згладжування тексту.
- @font-face: Підключає власний шрифт "CyrillicGoth" із локальної папки.
- h2: Використовує кастомний шрифт для заголовків другого рівня.
- button: Оформлює кнопки з заокругленими кутами, темним фоном та плавною анімацією при наведенні.
- **.**arrow-button: Стилізує кастомні кнопки зі стрілками, прибирає стандартне оформлення, робить їх прозорими, додає відступи.
- .arrow-button img: Задає розміри стрілкам у кнопках.
- .icon-img: Встановлює розміри для іконок персонажів, зберігає пропорції.
- input: Оформлює поля вводу у темних тонах із білим текстом.
- .character-image: Додає анімацію для зображення персонажа, фіксує розміри та позицію.
- .hash-display: Оформлює блок для відображення хешу з тінню для кращої читабельності.

В сукупності цей компонент допомагає створити естетично привабливий дизайн, який підійде атмосфері гри D&D. Код файлу зображено на рисунках 3.33-3.35.

src > index.css > ...

```css
body {
  font-family: "CyrillicGoth", sans-serif;
  margin: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  min-width: 320px;
  min-height: 100vh;
  background-image: url("assets/site-background4.jpg");
  background-size: cover; /* Зображення буде покривати весь фон */
  background-position: center center; /* Центрування зображення */
  background-repeat: no-repeat; /* Заборона на повторення зображення */
}

:root {
  font-family: "Cinzel", "Creepster", "Garamond", "Georgia", serif;
  line-height: 1.5;
  font-weight: 400;
  color-scheme: light dark;
  color: rgba(255, 255, 255, 0.87);
  background-color: #242424;
  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

@font-face {
  font-family: "CyrillicGoth";
  src: url("./assets/fonts/CyrillicGoth.ttf") format("truetype");
  font-style: normal;
}

h2 {
  font-family: "CyrillicGoth", sans-serif;
}
```

Рисунок 3.33 – Файл index.css

```css
button {
  border-radius: 8px;
  border: 2px solid #641c1c;
  padding: 0.6em 1.2em;
  margin: 5px;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  background-color: #3a1b1b;
  cursor: pointer;
  transition: border-color 0.25s;
}

button:hover {
  border-color: #c80000;
}

.arrow-button {
  all: unset;
  display: inline-block;
  padding: 0;
  margin-left: 1em;
  margin-right: 1em;
  background: transparent;
  border: none;
  cursor: pointer;
}

.arrow-button img {
  display: block;
  width: 7em;
  height: auto;
}

.icon-img {
  width: 15em;
  height: 15em;
```

Рисунок 3.34 – Файл index.css

```css
.icon-img {
  width: 15em;
  height: 15em;
  object-fit: contain;
}


input {
  color: #ffffff;
  background-color: #3a1b1b;
  border: 2px solid #641c1c;
}

.character-image {
  transition: transform 0.3s ease-in-out, opacity 0.3s ease-in-out;
  position: absolute;
  left: 0%;
  width: 375px;
  height: 500px;
}

.hash-display{
  font-family: Arial, Helvetica, sans-serif;
  text-shadow: #000000 1px 1px 1px;
}
```

Рисунок 3.35 – Файл index.css

Нарешті, після розробки усіх необхідних компонентів та функцій, можна скласти усе в один файл CharacterCreator (рис 3.36-3.41), що є головним файлом з викликом усіх компонентів у генераторі зовнішності ігрових персонажів. Основний файл включає в себе такі функції:

1. Керування етапами створення персонажа – вибір раси, статі, категорії класів, елементів одягу для створення зовнішності персонажа, викликаючи відповідні компоненти.
2. Зберігання стану – зберігає обрані раніше стани раси, статі, класу, індексів одягу, імені персонажа та хешу, для оперування ними в подальшому.
3. Локалізація – додає перемикання мови через компонент LanguageSwitch.
4. Введення імені – введення бажаного імені для готового зображення.
5. Генерація зображення та хешу – завантаження готового jpeg-зображення під обраним раніше іменем, та відображення хешу внизу.

```
src > CharacterCreator.tsx > CharacterCreator
import React, { useState } from "react";
import LanguageSwitch from "./AdditionalFunctions/LanguageSwitch";
import translations, { Language } from "./AdditionalFunctions/translations";
import RaceSelection from "./RaceSelection";
import GenderSelection from "./GenderSelection";
import ClassSelection from "./ClassSelection";
import CharacterCustomization from "./CharacterCustomization";
import CharacterNameInput from "./AdditionalFunctions/CharacterNameInput";
import ImageGenerator from "./AdditionalFunctions/ImageGenerator";
import classesJson from "./helpers/jsons/classes.json";
import raceJson from "./helpers/jsons/race.json";
import { Race, CharacterClass } from "./types/types";


const races: Race[] = raceJson;
const classes: CharacterClass[] = classesJson;

const CharacterCreator: React.FC = () => {
  const [language, setLanguage] = useState<Language>("ua");
  const t = translations[language];

  const [step, setStep] = useState<number>(1);
  const [selectedRaceIndex, setSelectedRaceIndex] = useState<number>(0);
  const [selectedGender, setSelectedGender] = useState<boolean | null>(null);
  const [selectedClassIndex, setSelectedClassIndex] = useState<number>(0);
  const [headIndex, setHeadIndex] = useState<number>(0);
  const [bodyIndex, setBodyIndex] = useState<number>(0);
  const [legsIndex, setLegsIndex] = useState<number>(0);
  const [characterName, setCharacterName] = useState<string>("");
  const [hash, setHash] = useState<string>("");

  const handlePrevRace = () => {
    setSelectedRaceIndex((prev) => (prev === 0 ? races.length - 1 : prev - 1));
  };
```

Рисунок 3.36 – Компонент CharacterCreator

```
const handleNextRace = () => {
  setSelectedRaceIndex((prev) => (prev === races.length - 1 ? 0 : prev + 1));
};

const handleNextClass = () =>
  setSelectedClassIndex((prev) => (prev + 1) % classes.length);

const handlePrevClass = () =>
  setSelectedClassIndex((prev) => (prev === 0 ? classes.length - 1 : prev - 1));

const handleGenderSelect = (gender: boolean) => {
  setSelectedGender(gender);
  setStep(3);
};

const handleNext = () => setStep(2);

const handlePrevItem = (
  setState: React.Dispatch<React.SetStateAction<number>>,
  items: string[]
) => {
  setState((prev) => (prev === 0 ? items.length - 1 : prev - 1));
};

const handleNextItem = (
  setState: React.Dispatch<React.SetStateAction<number>>,
  items: string[]
) => {
  setState((prev) => (prev + 1) % items.length);
};
```

Рисунок 3.37 – Компонент CharacterCreator

```
  const generateHash = async () => {
  const configData = {
    race: races[selectedRaceIndex].maleImage,
    gender: selectedGender,
    class: classes[selectedClassIndex].nameClass,
    head: classes[selectedClassIndex].outfits[0].sex[selectedGender ? "male" : "female"].head[headIndex],
    body: classes[selectedClassIndex].outfits[0].sex[selectedGender ? "male" : "female"].body[bodyIndex],
    legs: classes[selectedClassIndex].outfits[0].sex[selectedGender ? "male" : "female"].legs[legsIndex],
  };

  const stringifiedData = JSON.stringify(configData);

  const encoder = new TextEncoder();
  const dataBuffer = encoder.encode(stringifiedData);

  try {
    const hashBuffer = await crypto.subtle.digest("SHA-256", dataBuffer);

    const hashArray = Array.from(new Uint8Array(hashBuffer));
    const hashHex = hashArray.map(byte => byte.toString(16).padStart(2, "0")).join("");

    setHash(hashHex);
  } catch (error) {
    console.error("Error generating hash:", error);
  }
};
```

Рисунок 3.38 – Компонент CharacterCreator

```
return (
  <div
    style={{
      display: "flex",
      justifyContent: "center",
      alignItems: "center",
      height: "100vh",
      textAlign: "center",
      padding: "20px",
      position: "relative",
    }}
  >
    <LanguageSwitch language={language} setLanguage={setLanguage} />

    {step === 1 ? (
      <div>
        <h2>{t.chooseRace}</h2>
        <RaceSelection
          races={races}
          selectedRaceIndex={selectedRaceIndex}
          handlePrevRace={handlePrevRace}
          handleNextRace={handleNextRace}
          handleNext={handleNext}
          t={t}
        />
      </div>
    ) : step === 2 ? (
      <div>
        <h2>{t.chooseGender}</h2>
        <GenderSelection
          races={races}
          selectedRaceIndex={selectedRaceIndex}
          handleGenderSelect={handleGenderSelect}
          t={t}
        />
```

Рисунок 3.39 – Компонент CharacterCreator

```
        </div>
      ) : step === 3 ? (
        <div>
          <h2>{t.chooseClass}</h2>
          <ClassSelection
            classes={classes}
            selectedClassIndex={selectedClassIndex}
            handlePrevClass={handlePrevClass}
            handleNextClass={handleNextClass}
            setStep={setStep}
            t={t}
          />
        </div>
      ) : (
        <div>
          <h2>{t.customizeOutfit}</h2>
          <div>
            <CharacterCustomization
              races={races}
              selectedRaceIndex={selectedRaceIndex}
              selectedGender={selectedGender !== null ? selectedGender : true}
              classes={classes}
              selectedClassIndex={selectedClassIndex}
              headIndex={headIndex}
              bodyIndex={bodyIndex}
              legsIndex={legsIndex}
              handlePrevItem={handlePrevItem}
              handleNextItem={handleNextItem}
              setHeadIndex={setHeadIndex}
              setBodyIndex={setBodyIndex}
              setLegsIndex={setLegsIndex}
            />
          </div>
```

Рисунок 3.40 – Компонент CharacterCreator

```
            <CharacterNameInput
              characterName={characterName}
              setCharacterName={setCharacterName}
              t={t}
            />
            <ImageGenerator
              characterName={characterName}
              t={t}
              generateHash={generateHash}
            />
            {hash && (
              <div className="hash-display">
                <h4>Generated Hash: {hash}</h4>
              </div>
            )}
          </div>
        )}
      </div>
    );
};

export default CharacterCreator;
```

Рисунок 3.41 – Компонент CharacterCreator

В залежності від кроку, користувач бачить відповідний інтерфейс, який був прописаний у відповідних компонентах. В сукупності він об’єднує всі компоненти та керує логікою всього процесу створення зовнішності ігрового персонажа.

Компонент CharacterCreator підключено у файлі main.tsx, який рендерить весь проєкт (рис. 3.42)

src > main.tsx

```tsx
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './CharacterCreator.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)

```

Рисунок 3.42 – Файл main.tsx

3.4 Тестування генератора зовнішності ігрових персонажів

Тестування є важливим етапом розробки будь-якого проєкту, так як воно гарантує виконання функціональних вимог, перевіряє стабільність застосунку, та покращує його надійність.

Для тестування такого роду веб-застосунків, як генератор зовнішності ігрових персонажів було обрано автоматизоване тестування. Автоматизоване тестування, на відміну від ручного тестування, спрощує процес виявлення помилок за допомогою спеціальних програм, що знижує витрати та час на цикл тестування [15].

Можна виділити кілька основних переваг автоматизованого тестування:

1. Швидкість. Автоматизоване тестування відбувається набагато швидше аніж вручну. Особливо це враховується при великих проєктах чи часто повторюваних тестах.
2. Повторюваність. Тести можна запускати безліч разів без переписування вручну
3. Відсутність людського фактору. Тести повністю автоматизовані, тому на етапі коли вони працюють, людський фактор не відіграє жодної ролі.
4. Ранні виявлення багів. Автоматизоване тестування дуже корисне на ранніх стадіях коду та допомагає виявити помилки.
5. Тести складних сценаріїв. Автоматизоване тестування чудово підійде для складних розгалужених проєктів де присутні десятки варіантів взаємодії та функціоналу.

Для проведення автоматизованого тестування, було обрано обрано застосунок SeleniumIDE.

Інтегроване середовище розробки Selenium (SeleniumIDE) — це просте у використанні розширення для браузера, яке записує дії користувача в браузері за допомогою існуючих команд Selenium із параметрами, визначеними контекстом кожного елемента [16]. Це чудовий спосіб вивчити синтаксис Selenium. Він доступний для Google Chrome, Mozilla Firefox і Microsoft Edge.

На момент розробки генератора ігрових персонажів для гри D&D, застосунок SeleniumIDE було видалено з магазину розширень Google Chrome, тому я використовуватиму його у браузері Mozilla Firefox.

Для початку треба додати SeleniumIDE до розширень нашого браузера, для цього переходимо на офіційний сайт та обираємо опцію «Firefor download» (рис. 3.43). Після цього встановлюємо SeleniumIDE в магазині розширень браузера (рис. 3.44).

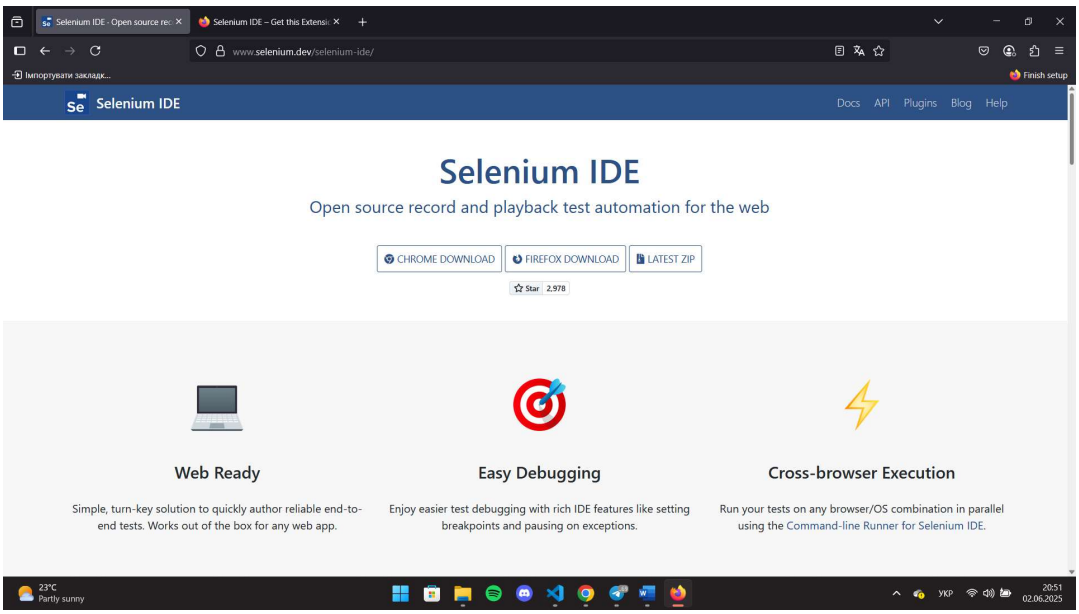

Рисунок 3.43 – Сайт SeleniumIDE

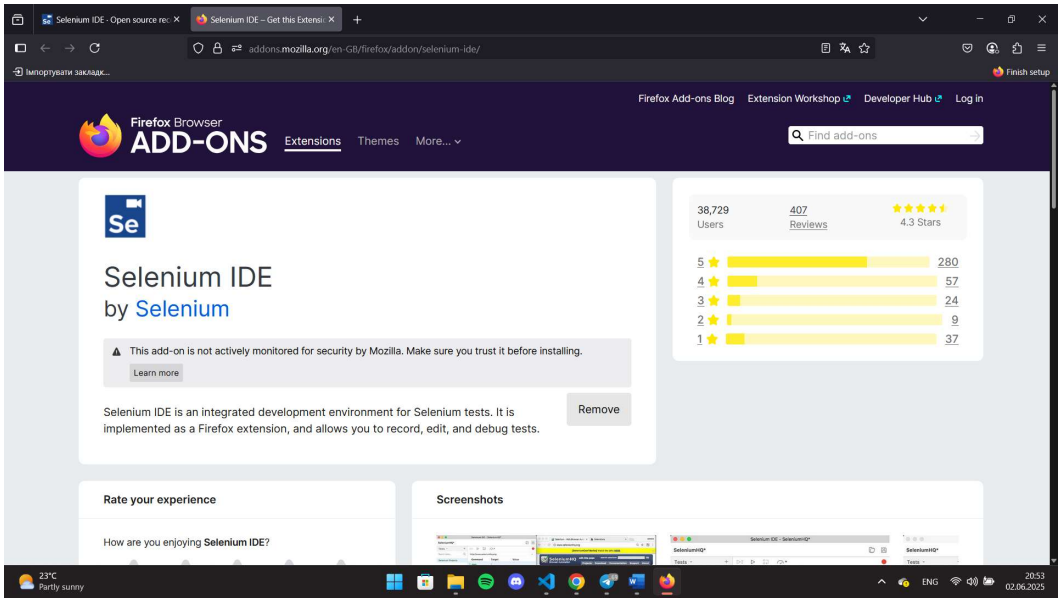

Рисунок 3.44 – SeleniumIDE у магазині розширень Firefox

Після встановлення застосунок SeleniumIDE можна запустити прямо з браузера (рис. 3.45)

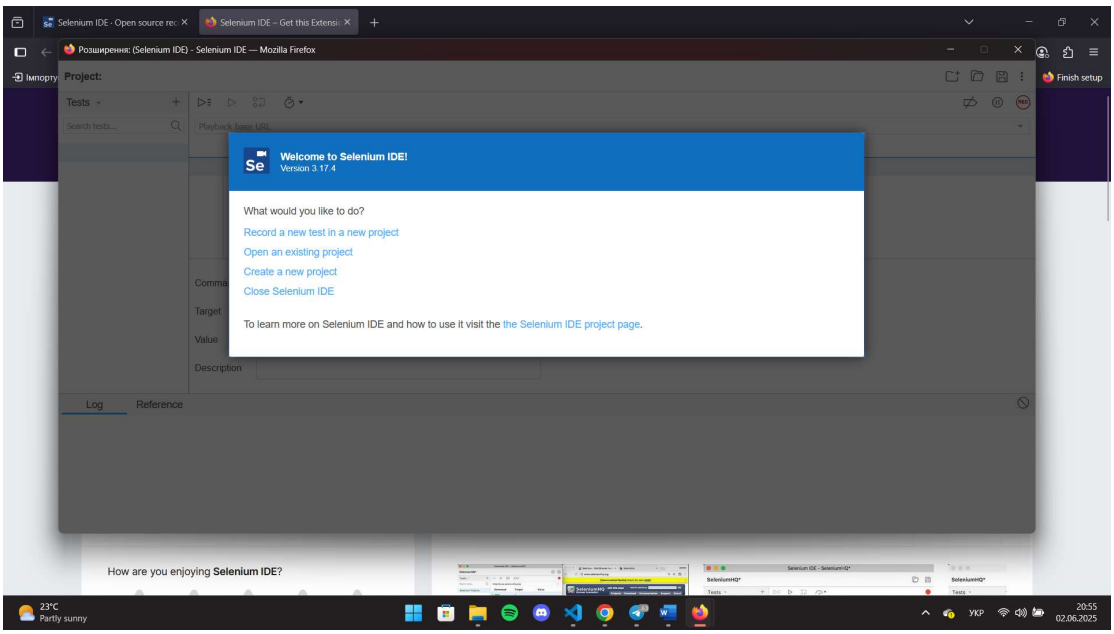

Рисунок 3.45 – Стартовий екран SeleniumIDE

Першим кроком йде створення проекту, де ми будемо записувати та зберігати усі наші тести. Для цього обираємо «Create new project» та вводимо назву для нашого проекту (рис. 3.46).

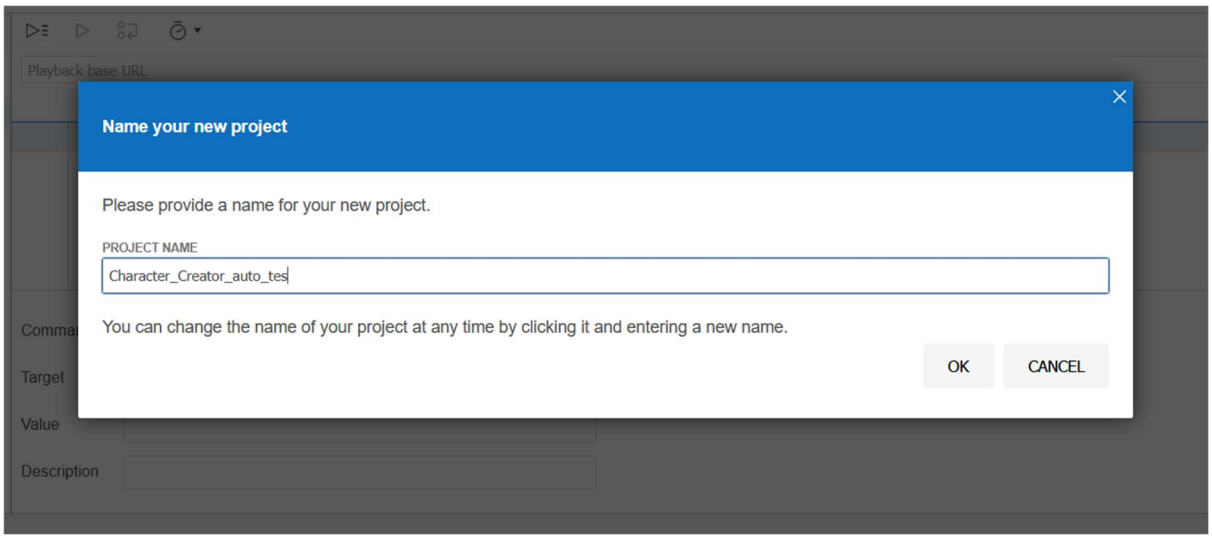

Рисунок 3.46 – Створення проєкту у SeleniumIDE

Після створення перед нами відкривається робоча площа програми (рис. 3.47). Зліва у нас знаходиться стовпець для наших створених тестів. По центру знаходиться основна зона роботи. В ній будуть розміщуватись кроки наших тестів, які будуть там записуватись та відтворюватись. Вгорі є рядок для введення URL-адреси. Внизу розташовані логи, для відслідковування виконання кроків поточного тесту.

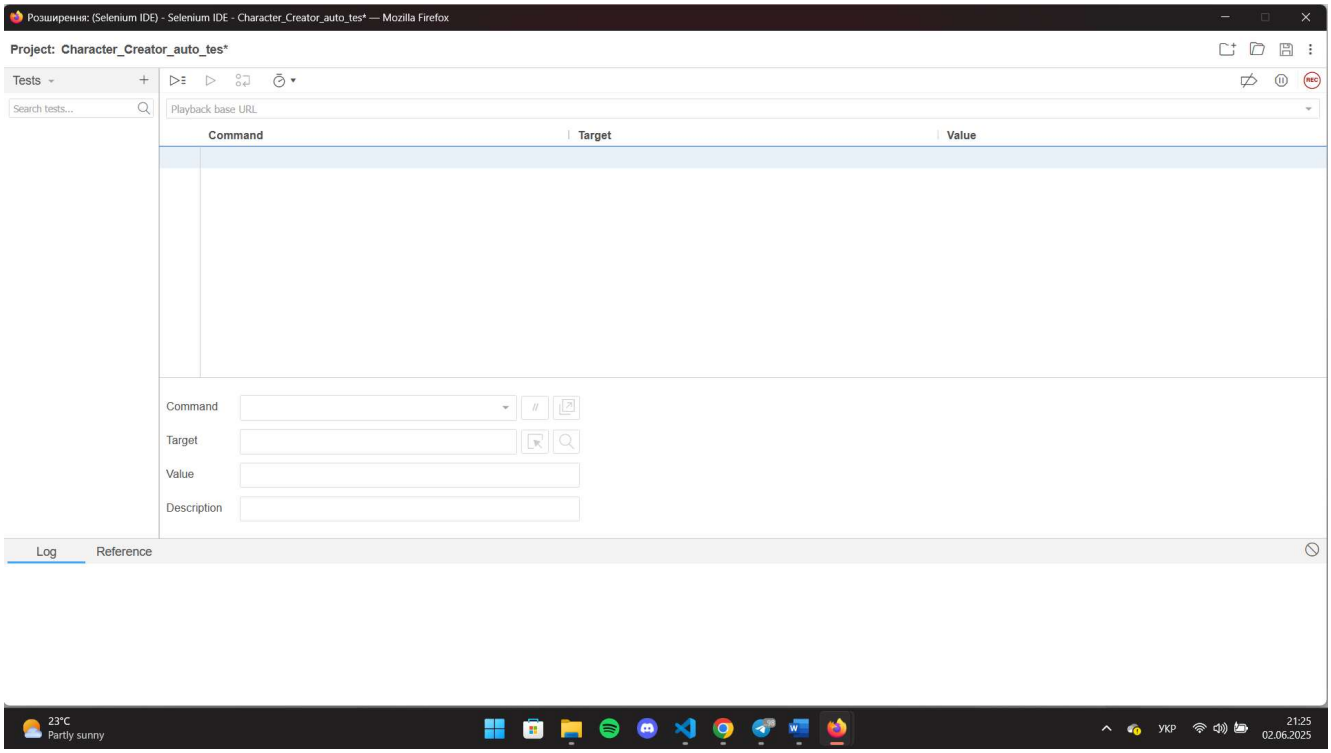

Рисунок 3.47 – Робоча площа SeleniumIDE

Принцип роботи SeleniumIDE полягає в тому, що ми після введення адреси, розпочинаємо запис. У нас автоматично відкривається вікно браузера з цією введеною адресою і всі наші подальші дії, як от кліки на елементи у графічному інтерфейсі, введення символів на клавіатурі чи змінення масштабу вікна, усі ці дії записуються. Після проведення дій, які підходять під суть нашого тесту, ми зупиняємо запис. Після цього, ми можемо відтворювати цей запис безліч разів, щоб перевірити чи бот зможе автоматично пройти по тим самим крокам, які ми записали

раніше. Важливо, проводити запис акуратно та з великою точністю, щоб не виникало випадкових кліків чи покривань курсором. А також, щоб кроки нашого тесту, не містили в собі елементи які змінюються (наприклад змінні рекламні блоки), чи такі елементи як реєстрація, а також reCAPTCHA, тому що жоден бот не може пройти цю перевірку.

Так як генератор зовнішності ігрових персонажів «Character Creator» в основному лінійний застосунок, то всі тести будуть подібними між собою, і відрізнятимуться лише набором комбінацій. Було обрано три комбінації зовнішностей персонажів, для проведення автоматизованого тестування:

1. Ельфійка магічного класу
2. Дворфійка бойового класу
3. Людина жінка спритного класу

Також тести включатимуть у себе введення кастомного імені персонажа, завантаження готового зображення та відкриття цього зображення через спливаюче вікно завантажень у браузері

Створимо усі тести у лівій панелі SeleniumIDE та назвемо їх відповідними комбінаціями зовнішності: «elf-female-mage», «dwarf-female-warrior» та «human-female-agile» (рис. 3.48)

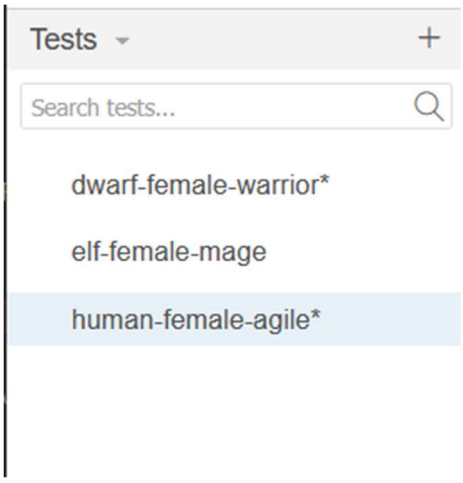

Рисунок 3.48 – Створенні тести для різних комбінацій зовнішностей

Для проведення першого тесту, вносимо адресу генератора зовнішності ігрових персонажів у рядок для адреси, та натискаємо кнопку запису праворуч вгорі.

Проходимо кроки для створення ельфійки магічного класу, обираємо рандомні елементи одягу, вводимо кастомне ім’я, натискаємо кнопку готово та відкриваємо зображення через спливаюче вікно. Після всіх дій, зупиняємо запис.

Натискаємо кнопку для запуску тексту, і автоматично з’являється вікно де програються щойно зроблені нами дії. Тестувальник обирає расу ельфа, жіночу стать, магічну групу класів та перегортає декілька варіантів одягу. Вводить ім’я Erika, натискає кнопку готово та відкриває зображення у спливаючому вікні.

У робочій площі SeleniumIDE, тест тепер відображається зеленим і всі його кроки також відображені зеленим кольором, що свідчить про те, що автоматизований тест пройшов успішно (рис. 3.49). Також у логах можна переглянути як проходив кожен крок тесту

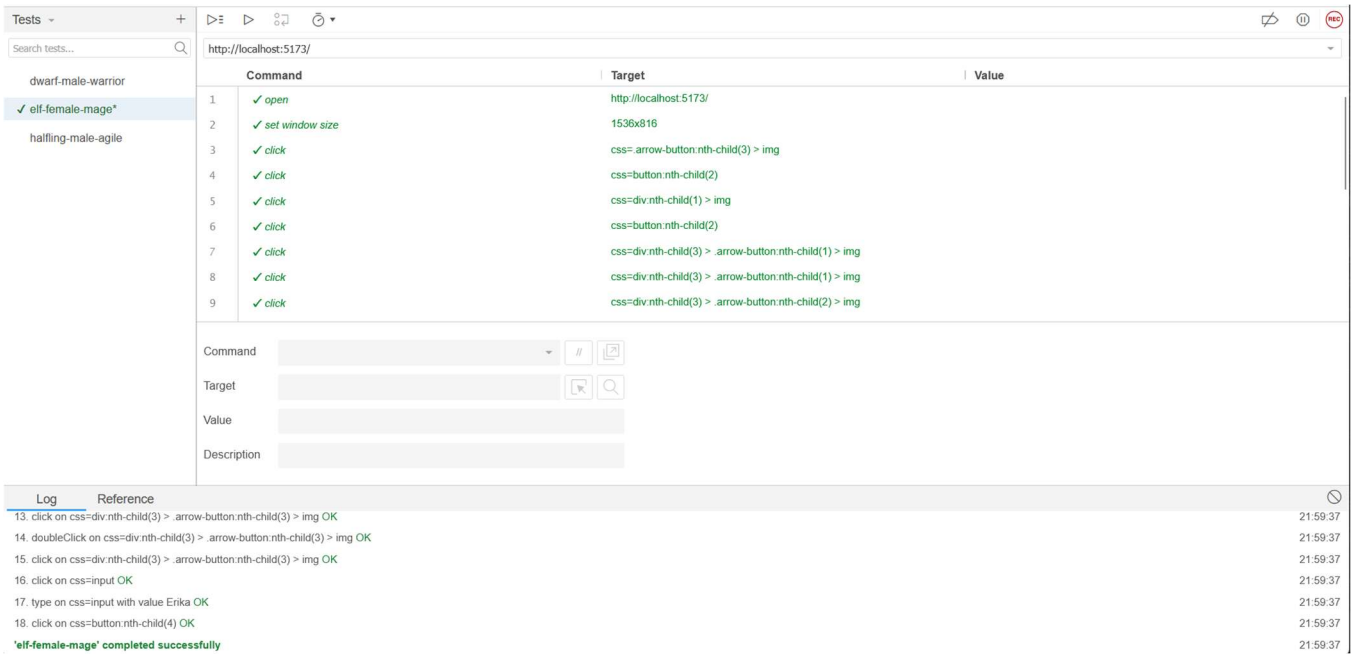

Рисунок 3.49 – Успішний тест, зі створенням ельфійки магічного класу

Створимо за таким самим принципом комбінацію дворфійку бойового класу з іменем Brannet. Починаємо запис, обираємо расу дворф, стать жінка, категорію

бойових класів, обираємо варіанти одягу, вписуємо ім’я та завантажуємо зображення. Зупиняємо запис.

Запустивши тест, бачимо що всі кроки виконались успішно та тест відмічений як вдалий (рис. 3.50).

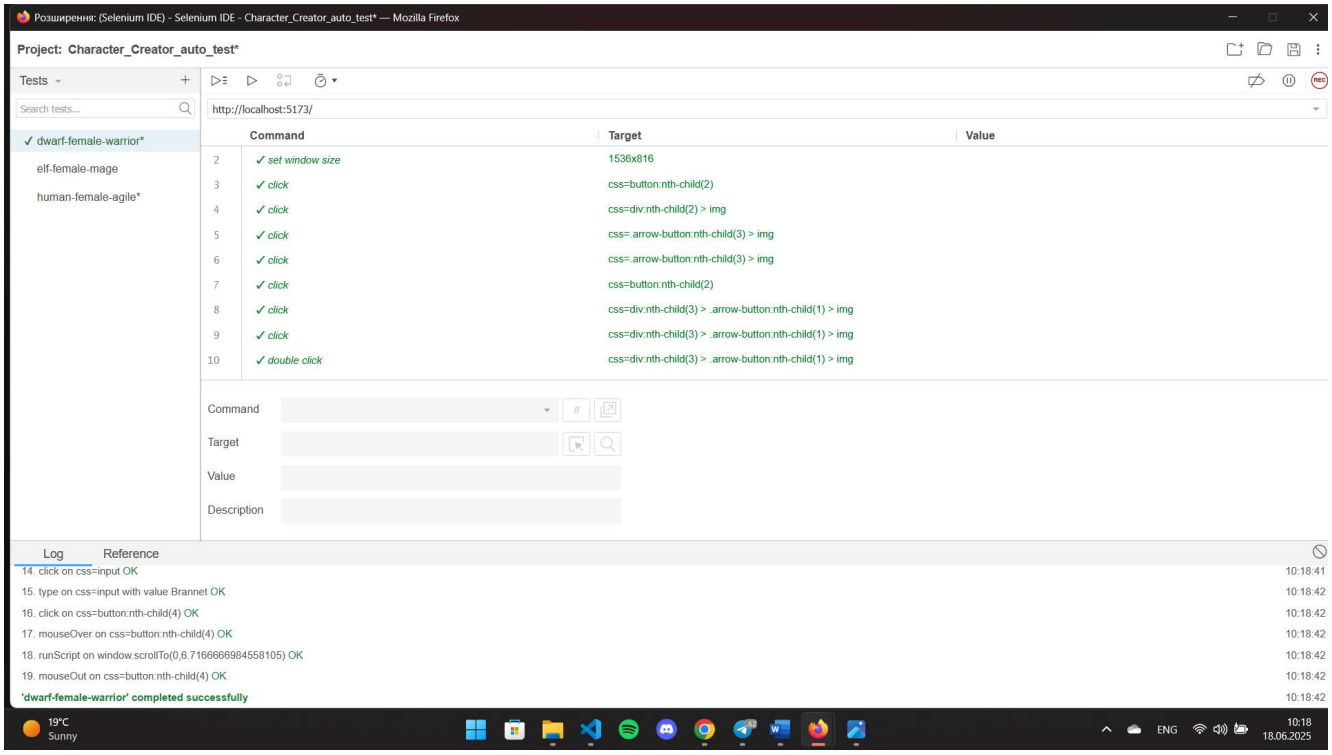

Рисунок 3.50 – Успішний тест, зі створенням дворфійки бойового класу

Останнім тестом, стане створення людини жінки спритного класу на ім’я Mary. Проводимо всі ті самі маніпуляції лише обравши расу людини та спритну групу класів.

Після запуску, цей тест так само показав себе успішно і усі його кроки виявились валідними (рис. 3.51).

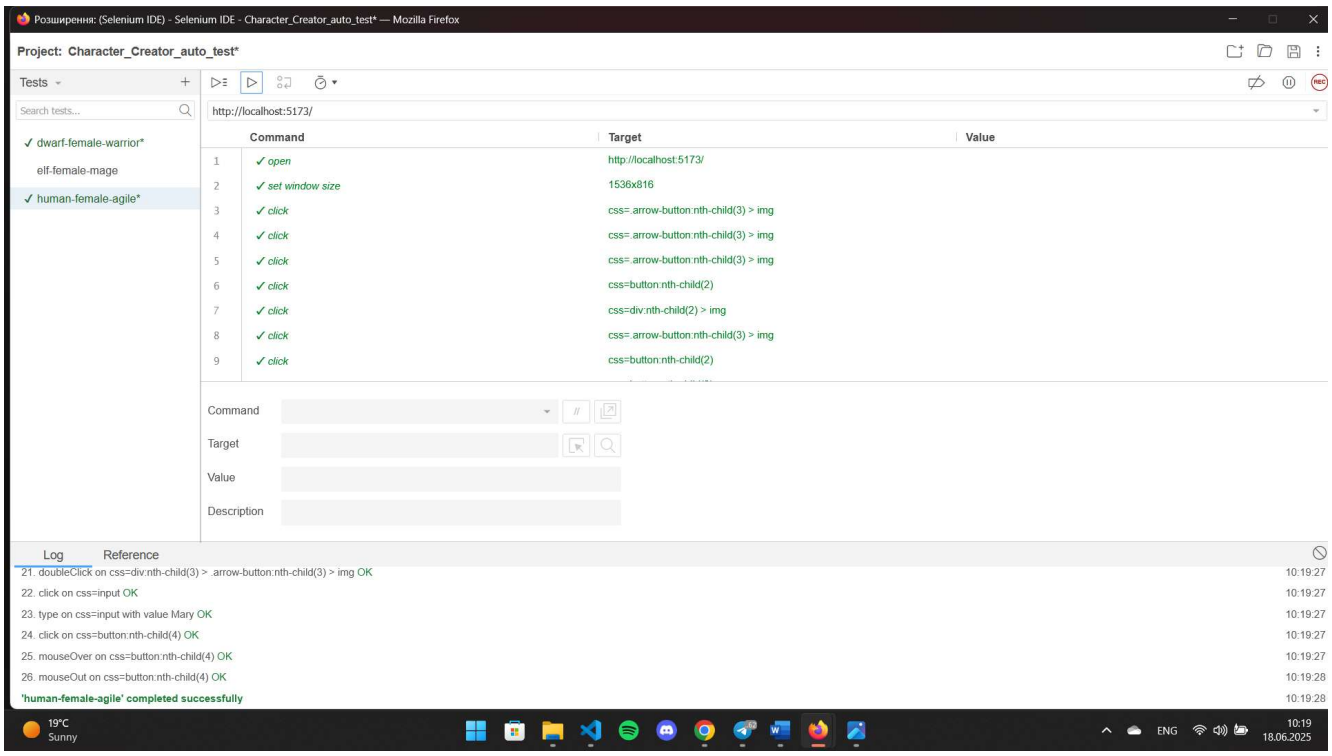

Рисунок 3.51 – Успішний тест, зі створенням людини спритного класу

Провівши автоматизоване тестування за допомогою застосунку SeleniumIDE, було складено три успішні тести зі створення різних комбінацій зовнішностей для ігрових персонажів. Готові тести були відпрацьовані швидко та без затримок, що показало чудову працездатність генератора зовнішності ігрових персонажів для гри D&D.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при обмороженні.

Під час службових відряджень, особливо у віддалені або екстремальні місцевості, існує значний ризик для працівника обмороження або інших небезпек. Недотримання правил безпеки та недостатня підготовка можуть призвести до серйозних наслідків, включаючи адміністративні заходи, які можуть застосувати компанія або організація.

Обмороження — це пошкодження тканин внаслідок тривалої дії низької температури. Це серйозне травмування, яке може спричинити руйнування клітин та органів, особливо у разі недостатньої швидкості надання допомоги. Ризик обмороження підвищується за певних умов, таких як сильний вітер, вологість, низька температура та носіння вологого або мокрого одягу. Важливо розуміти, що обмороження може статися навіть при температурі 3-5°C, якщо поєднуються ці фактори. Найбільш вразливими частинами тіла є кінцівки (руки, ноги), ніс і вуха, адже ці частини швидше за все піддаються охолодженню [16].

Існує 4 ступеня обмороження:

- I ступінь — ураження шкіри, що проявляється в порушенні кровообігу, шкіра блідне, знижена чутливість. Після розігрівання з'являється біль, набряк і свербіж, шкіра облущується, і потерпілий зазвичай одужує без серйозних наслідків.
- II ступінь — супроводжується некрозом шкіри. Під час відновлення кровообігу шкіра набуває червоно-синього кольору, з'являються пухирці, що містять прозору рідину, болі та лихоманка.
- III ступінь — серйозні пошкодження, які призводять до тромбозу судин, некрозу шкіри та м’яких тканин. Пухирці стають темно-бурими, а пацієнт відчуває сильний біль і загальну слабкість.
- IV ступінь — змертвіння всіх шарів тканин, включаючи кістки [16].

Першочергові заходи при наданні домедичної допомоги за обмороженням, згідно з наказом Міністерства охорони здоров'я України (Наказ № 441 від 09.03.2022), включають: [17]:

1) Переконатися у відсутності небезпеки для себе та інших перед наданням допомоги.
2) Припинити вплив низьких температур.
3) Заспокоїти постраждалого та пояснити подальші дії.
4) Викликати екстрену медичну допомогу.
5) У разі загального переохолодження — усунути несприятливі зовнішні фактори, перемістити постраждалого у тепле приміщення, зняти мокрий одяг та забезпечити постраждалому теплі напої.
6) Якщо є ознаки відмороження, потрібно обережно зняти мокрий одяг і взуття, накласти стерильні марлеві пов'язки на уражені ділянки без тиску, і знерухомити кінцівки.
7) Важливо уникати масажу, розтирання або використання джерел тепла на уражених ділянках, а також не пошкоджувати пухирці [17].

Ці правила допоможуть запобігти погіршенню стану постраждалого і значно підвищують шанси на відновлення здоров'я, якщо надано правильну та своєчасну допомогу.

4.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора.

На перший погляд, питання естетичного та ергономічного оформлення робочого місця може здатися незначущим, однак насправді це має прямий вплив як на здоров'я і самопочуття працівників, так і на ефективність виконаної роботи.

Особливо це важливо для працівників, чия діяльність пов'язана з комп'ютерними технологіями, оскільки тривала робота за комп'ютером у некомфортних умовах може призвести до фізичних і психологічних проблем. Оператор комп’ютера, який не почувається комфортно, не може виконувати завдання з належною якістю, що негативно позначається на результатах роботи.

Робоче місце — це зона, де працівник проводить значну частину свого робочого часу. Раціональна організація робочих місць допомагає зменшити фізичний дискомфорт, знизити рівень втоми та підвищити продуктивність праці. За результатами досліджень, правильно організоване робоче місце може підвищити продуктивність на 15-25% [18].

Правильна організація робочого місця сприяє зменшенню втоми, підвищенню працездатності та зниженню загального дискомфорту. Дослідження показують, що при ергономічно організованому робочому місці продуктивність праці може збільшитися на 15-25% [19]. Тому правильне оформлення робочого місця є важливим аспектом у забезпеченні здоров’я працівників і підвищенні їхньої ефективності.

Організація робочого місця включає кілька ключових аспектів:

1. Правильне розміщення робочого місця у виробничому приміщенні з урахуванням природного освітлення. Робочі місця повинні бути орієнтовані таким чином, щоб світло з вікон падало збоку, переважно зліва, щоб уникнути відблисків на екрані і не спричиняти дискомфорту.
2. Вибір ергономічного робочого положення та меблів, що відповідають антропометричним характеристикам людини. Врахування таких факторів допомагає створити комфортні умови для роботи, зменшуючи навантаження на спину, шию та очі.
3. Розташування обладнання на робочих місцях повинно бути раціональним, щоб працівник мав легкий доступ до всіх необхідних інструментів, не перевантажуючи простір.

4. Урахування особливостей трудової діяльності. Для працівників, які працюють з відеотерміналами, вимоги до організації робочого місця строго регламентуються стандартами, такими як ДНАОП 0.00-1.31-99 [19], що визначають оптимальні відстані між екранами та меблями.

Згідно з вимогами, площа для одного робочого місця з персональним комп’ютером повинна становити не менше 6 м², а об'єм приміщення — не менше 20 м³. Важливо дотримуватись таких норм:

- Відстань від робочого місця до стін зі світловими прорізами повинна бути не менше 1 метра.
- Між бічними поверхнями відеотерміналів має бути відстань не менше 1,2 метра.
- Між тильною поверхнею одного відеотермінала і екраном іншого повинна бути відстань не менше 2,5 метра.
- Ширина проходів між рядами робочих місць повинна бути не менше 1 метра [19, 20].

Ці вимоги є частиною стандартів, які регулюють правильне компонування робочих місць для забезпечення ефективності праці, зниження рівня втоми та покращення здоров'я працівників.

ВИСНОВКИ

В ході виконання цієї дипломної роботи мною було розроблено генератор зовнішності ігрових персонажів для гри D&D.

В результаті, вдалось створити генератор зі своїм унікальним оформленням, яке чудово підходить до атмосфери гри Dungeons&Dragons, та в якому поєднуються зручні функції для генерування власних цікавих стилів для персонажів.

Вибір технології ReactJS чудово сказався на кінцевому продукті. Серверний рендеринг React дозволяє швидко підвантажувати усі необхідні елементи, що було доведено під час автоматизованого тестування, де всі кроки тестів відбувались з мінімальною затримкою. Компонентна складова React, дала змогу грамотно розділити усі елементи проєкту на складові, та лише допоможе розвивати генератор зовнішності ігрових персонажів у майбутньому.

Було розроблено генератор зовнішності ігрових персонажів, з лінійною системою вибору. Генератор включає в себе послідовні вибори раси, статі, категорії класів та елементів одягу. Вибір елементів одягу, базується на всіх попередніх етапах, та підтягує потрібний одяг. Також реалізовано функції завантаження зображення під введеним іменем та генерування хеш-коду. Було розроблено дві версії мови для веб-застосунку: українська та англійська. Усі малюнки також були виконані у стилі, який підходить до гри D&D.

Моя бакалаврська робота, враховує всі етапи розробки програмного забезпечення: аналіз вимог, архітектурне проєктування, конструювання, розробка та тестування. Створений проєкт також відображає передовий підхід до розробки веб-застосунків, тенденція яких це оптимізація та велике різноманіття у виборі контенту. Саме це і демонструє генератор зовнішності ігрових персонажів «Character Creator».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Підземелля та Дракони: історія створення та що за гра [Електронний ресурс] – Режим доступ до ресурсу: https://books.google.com.ua/books/about/Dungeons_Dragons_The_Making_of_Original.html?id=SOWm0AEACAAJ&source=kp_book_description&redir_esc=y

2. Книга гравця [Електронний ресурс] – Режим доступ до ресурсу: https://www.scribd.com/document/709709597/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0-%D0%93%D1%80%D0%B0%D0%B2%D1%86%D1%8F

3. TaleSpire у Steam [Електронний ресурс] – Режим доступ до ресурсу: https://store.steampowered.com/app/720620/TaleSpire/

4. Owlbear Rodeo [Електронний ресурс] – Режим доступ до ресурсу: https://www.owlbear.rodeo/

5. D&D Wiki [Електронний ресурс] – Режим доступ до ресурсу: https://www.dandwiki.com/wiki/Main_Page

6. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, Я.І. Кінах, Г.Б. Цуприк - Тернопіль: ТНТУ, 2017-38 с.

7. React - Опис UI [Електронний ресурс] – Режим доступ до ресурсу: https://react.dev/learn/describing-the-ui

8. TypeScript for JavaScript Programmers [Електронний ресурс] – Режим доступ до ресурсу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

9. Про середовище програмування C# [Електронний ресурс] – Режим доступ до ресурсу: https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp/

10. Що таке PostgreSQL? [Електронний ресурс] – Режим доступ до ресурсу: https://www.postgresql.org/about/

11. Lemon Frog on Behance [Електронний ресурс] – Режим доступ до ресурсу: https://www.behance.net/lemonfrog

12. CyrillicGoth – Колекція українських шрифтів [Електронний ресурс] – Режим доступ до ресурсу: https://www.ukrfonts.com/info/index.php?v=19&id=1050

13. Все про професію UI/UX дизайнера [Електронний ресурс] – Режим доступ до ресурсу: https://dan-it.com.ua/uk/blog/vse-pro-profesiyu-ui-ux-dyzajnera/

14. Що таке CSS? [Електронний ресурс] – Режим доступ до ресурсу: https://w3schoolsua.github.io/css/index.html#gsc.tab=0

15. Автоматизоване тестування [Електронний ресурс] – Режим доступ до ресурсу: https://newline.tech/test-automation-when-why-and-who-needs-it_uk/

16. SeleniumIDE [Електронний ресурс] – Режим доступ до ресурсу: https://www.selenium.dev/selenium-ide/

17. ТЕМА 9. Медицина катастроф. Долікарська допомога при обмороженні [Електронний ресурс] – Режим доступ до ресурсу: https://dl.tntu.edu.ua/content.php?cid=299869

18. НАКАЗ Про затвердження порядків надання домедичної допомоги особам при невідкладних станах 28 березня 2022 р. за № 356/37692 [Електронний ресурс] – Режим доступ до ресурсу: https://zakon.rada.gov.ua/laws/show/z0356-22#n669

19. Конспект лекцій з курсу: Охорона праці в праці [Електронний ресурс] – Режим доступ до ресурсу: https://elartu.tntu.edu.ua/bitstream/123456789/17891/1/KonspektOPG.pdf

20. Сьомка С. В. Ергономіка та ергодизайн: підручник / С. В. Сьомка; за ред. М-ва культури України. – Київ: Ліра-К, 2022. – 618 с. – ISBN 978-617-7507-19-1.

21. ДСТУ 7951:2015. Дизайн і ергономіка. Крісло оператора. Загальні ергономічні вимоги: нац. стандарт України / Укр. н.-д. ін-т дизайну та ергономіки. – Київ: УкрНДНЦ, 2016. – 10 с. – [Електронний ресурс]. – Назва з екрана. – Режим доступу: https://dnaop.com/html/61769/doc-ДСТУ_7951_2015, вільний. – Дата звернення: 08.06.2025.

ДОДАТКИ

Додаток А – Тези конференцій

VIII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.4
Фаль О. –ст. гр. СП-43
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ТА ДИЗАЙН ІГРОВОГО ДОДАТКУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ REACT JS

Науковий керівник: к.т.н., доц. каф. ПІ Стоянов Ю.М.

Fal O.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT AND DESIGN OF A GAME APPLICATION USING REACT JS TECHNOLOGY

Supervisor: PhD Yurii Stoianov

Ключові слова: інженерія програмного забезпечення, React, JavaScript, ігровий додаток
Key words: software engineering, React, JavaScript, game application

Моя мета полягає в створенні ігрового додатку, який не лише забезпечить комфорт і задоволення користувачів, але й революціонізує галузь за допомогою передових технологій React JS [1]. Мова йде про створення інтерактивного ігрового середовища, де гравці зможуть насолоджуватися широким вибором функціональних можливостей, від інтелектуальних викликів до створення власних дизайнів для ігрових персонажів з використанням новітніх технологій.

React JS надає широкий набір інструментів для розробки сучасних інтерфейсів користувача. Компонентна архітектура дозволяє ефективно повторно використовувати код, а віртуальний DOM забезпечує високу продуктивність додатку навіть при значних обсягах динамічних змін. Завдяки гнучкості React та його екосистемі, ми можемо легко впроваджувати інтерактивні елементи, керування станом (через хуки або сторонні бібліотеки), а також інтеграцію з іншими технологіями.

Мій підхід заснований на синергетичному поєднанні інновацій та відповідального підходу до розробки, що гарантує постійне поліпшення продукту. Відкритість до новаторства і співпраці з експертами галузі для забезпечення найвищої якості і задоволення потреб наших користувачів.

За допомогою React JS та глибокого розуміння JavaScript [2] можна досягти забезпечення найсучасніших рішень для ігрового додатку, що дозволить завоювати прихильність гравців по всьому світу та подарувати їм незабутній досвід у сфері ігрових можливостей.

Література:

1. React JS, основні поняття. URL: https://uk.legacy.reactjs.org/docs/hello-world.html.
2. Вступ до JavaScript. URL: https://uk.javascript.info/intro.

Додаток Б – Лістинг коду основних компонентів генератора зовнішності ігрових персонажів

Лістинг 1 – код RaceSelection

```
import React from "react";
import arrowRight from "./assets/arrow-right.png";
import arrowLeft from "./assets/arrow-left.png";
import { Race } from "./types/types";
import {  Translation } from "./types/types";

interface RaceSelectionProps {
  races: Race[];
  selectedRaceIndex: number;
  handlePrevRace: () => void;
  handleNextRace: () => void;
  handleNext: () => void;
  t: Translation;
}

const RaceSelection: React.FC<RaceSelectionProps> = ({
  races,
  selectedRaceIndex,
  handlePrevRace,
  handleNextRace,
  handleNext,
  t,
}) => {
  return (
    <div>
      <div
        style={{
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
        <button onClick={handlePrevRace} className="arrow-button">
          <img src={arrowLeft} alt="Next" />
        </button>
        <div>
          <img
            src={races[selectedRaceIndex].maleImage}
            alt={t.nameRace[selectedRaceIndex]}
            style={{ width: "200px" }}
          />
          <p>{t.nameRace[selectedRaceIndex]}</p>
        </div>
        <button onClick={handleNextRace} className="arrow-button">
          <img src={arrowRight} alt="Next" />
        </button>
```

```
      </div>
      <button onClick={handleNext} style={{ marginTop: "10px" }}>
        {t.next}
      </button>
    </div>
  );
};

export default RaceSelection;
```

Лістинг 2 – код GenderSelection

```
import React from "react";
import { Race } from "./types/types";
import { Translation } from "./types/types";
interface GenderSelectionProps {
  races: Race[];
  selectedRaceIndex: number;
  handleGenderSelect: (gender: true | false) => void;
  t: Translation;
}

const GenderSelection: React.FC<GenderSelectionProps> = ({
  races,
  selectedRaceIndex,
  handleGenderSelect,
  t,
}) => {
  return (
    <div>
      <div
        style={{
          display: "flex",
          justifyContent: "center",
          gap: "20px",
          marginTop: "20px",
        }}
      >
        <div
          onClick={() => handleGenderSelect(true)}
          style={{ cursor: "pointer" }}
        >
          <img
            src={races[selectedRaceIndex].maleImage}
            alt={t.male}
            style={{ width: "150px" }}
          />
          <p>{t.male}</p>
        </div>
        <div
          onClick={() => handleGenderSelect(false)}
          style={{ cursor: "pointer" }}
```

```
        >
          <img
            src={races[selectedRaceIndex].femaleImage}
            alt={t.female}
            style={{ width: "150px" }}
          />
          <p>{t.female}</p>
        </div>
      </div>
    </div>
  );
};

export default GenderSelection;
```

Лістинг 3 – код ClassSelection

```
import React from "react";
import arrowRight from "./assets/arrow-right.png";
import arrowLeft from "./assets/arrow-left.png";
import { CharacterClass, Translation } from "./types/types";

interface ClassSelectionProps {
  classes: CharacterClass[];
  selectedClassIndex: number;
  handlePrevClass: () => void;
  handleNextClass: () => void;
  setStep: React.Dispatch<React.SetStateAction<number>>;
  t: Translation;
}

const ClassSelection: React.FC<ClassSelectionProps> = ({
  classes,
  selectedClassIndex,
  handlePrevClass,
  handleNextClass,
  setStep,
  t,
}) => {
  return (
    <div>
      <div
        style={{
          display: "flex",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
        <button onClick={handlePrevClass} className="arrow-button">
          <img src={arrowLeft} alt="Prev" />
```

```
        </button>
        <div>
          <img
            src={classes[selectedClassIndex].icon}
            alt="Class Icon"
            className="icon-img"
          />
          <p>{t.nameClass[selectedClassIndex]}</p>
        </div>
        <button onClick={handleNextClass} className="arrow-button">
          <img src={arrowRight} alt="Next" />
        </button>
      </div>
      <button onClick={() => setStep(4)} style={{ marginTop: "10px"
}}>
        {t.next}
      </button>
    </div>
  );
};

export default ClassSelection;
```

Лістинг 4 – код CharacterCustomization

```
import React from "react";
import arrowRight from "./assets/arrow-right.png";
import arrowLeft from "./assets/arrow-left.png";
import { CharacterClass, Race } from "./types/types";

interface CharacterCustomizationProps {
  races: Race[];
  selectedRaceIndex: number;
  selectedGender: boolean;
  classes: CharacterClass[];
  selectedClassIndex: number;
  headIndex: number;
  bodyIndex: number;
  legsIndex: number;
  handlePrevItem: (
    setIndex: React.Dispatch<React.SetStateAction<number>>,
    itemArray: string[]
  ) => void;
  handleNextItem: (
    setIndex: React.Dispatch<React.SetStateAction<number>>,
    itemArray: string[]
  ) => void;
  setHeadIndex: React.Dispatch<React.SetStateAction<number>>;
  setBodyIndex: React.Dispatch<React.SetStateAction<number>>;
  setLegsIndex: React.Dispatch<React.SetStateAction<number>>;
}
```

```
const CharacterCustomization: React.FC<CharacterCustomizationProps>
= ({
  races,
  selectedRaceIndex,
  selectedGender,
  classes,
  selectedClassIndex,
  headIndex,
  bodyIndex,
  legsIndex,
  handlePrevItem,
  handleNextItem,
  setHeadIndex,
  setBodyIndex,
  setLegsIndex,
}) => {
  const raceName = races[selectedRaceIndex].race;
  const genderKey = selectedGender ? "male" : "female";
  const selectedOutfit = classes[selectedClassIndex].outfits.find(

    (outfit) => outfit.race === raceName

  );


  if (!selectedOutfit) {
    return (
      <div style={{ color: "red" }}>
        ❌ Outfit not found for race <strong>{raceName}</strong>
      </div>
    );
  }

  const sexOutfit = selectedOutfit.sex[genderKey];

  return (
    <div
      style={{
        display: "flex",
        justifyContent: "center",
        alignItems: "center",
      }}
    >
      <div
        style={{
          display: "flex",
          flexDirection: "column",
          justifyContent: "space-between",
          height: "400px",
        }}
      >
        <button onClick={() => handlePrevItem(setHeadIndex,
sexOutfit.head)} className="arrow-button">
```

```
          <img src={arrowLeft} alt="Prev Head" />
        </button>
        <button onClick={() => handlePrevItem(setBodyIndex,
sexOutfit.body)} className="arrow-button">
          <img src={arrowLeft} alt="Prev Body" />
        </button>
        <button onClick={() => handlePrevItem(setLegsIndex,
sexOutfit.legs)} className="arrow-button">
          <img src={arrowLeft} alt="Prev Legs" />
        </button>
      </div>

      <div
        id="character-container"
        style={{
          position: "relative",
          width: "375px",
          height: "500px",
          overflow: "hidden",
        }}
      >
        <img
          src={sexOutfit.classGroup}
          alt="Class Group"
          className="character-image"
          style={{
            width: "100%",
            height: "100%",
            objectFit: "cover",
            position: "absolute",
            top: "0",
            left: "0",
          }}
        />
        <img
          src={sexOutfit.head[headIndex]}
          alt="Head"
          className="character-image"
          style={{
            position: "absolute",
            top: "0",
            left: "0",
            width: "100%",
            height: "100%",
            objectFit: "cover",
          }}
        />
        <img
          src={sexOutfit.body[bodyIndex]}
          alt="Body"
          className="character-image"
          style={{
            position: "absolute",
```

```
            top: "0",
            left: "0",
            width: "100%",
            height: "100%",
            objectFit: "cover",
          }}
        />
        <img
          src={sexOutfit.legs[legsIndex]}
          alt="Legs"
          className="character-image"
          style={{
            position: "absolute",
            top: "0",
            left: "0",
            width: "100%",
            height: "100%",
            objectFit: "cover",
          }}
        />
      </div>

      <div
        style={{
          display: "flex",
          flexDirection: "column",
          justifyContent: "space-between",
          height: "400px",
        }}
      >
        <button onClick={() => handleNextItem(setHeadIndex,
sexOutfit.head)} className="arrow-button">
          <img src={arrowRight} alt="Next Head" />
        </button>
        <button onClick={() => handleNextItem(setBodyIndex,
sexOutfit.body)} className="arrow-button">
          <img src={arrowRight} alt="Next Body" />
        </button>
        <button onClick={() => handleNextItem(setLegsIndex,
sexOutfit.legs)} className="arrow-button">
          <img src={arrowRight} alt="Next Legs" />
        </button>
      </div>
    </div>
  );
};

export default CharacterCustomization;
```

Лістинг 5 – код CharacterCreator

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

```
import LanguageSwitch from "./AdditionalFunctions/LanguageSwitch";
import translations, { Language } from
"./AdditionalFunctions/translations";
import RaceSelection from "./RaceSelection";
import GenderSelection from "./GenderSelection";
import ClassSelection from "./ClassSelection";
import CharacterCustomization from "./CharacterCustomization";
import CharacterNameInput from
"./AdditionalFunctions/CharacterNameInput";
import ImageGenerator from "./AdditionalFunctions/ImageGenerator";
import classesJson from "./helpers/jsons/classes.json";
import raceJson from "./helpers/jsons/race.json";
import { Race, CharacterClass } from "./types/types";

const races: Race[] = raceJson;
const classes: CharacterClass[] = classesJson;

const CharacterCreator: React.FC = () => {
  const [language, setLanguage] = useState<Language>("ua");
  const t = translations[language];

  const [step, setStep] = useState<number>(1);
  const [selectedRaceIndex, setSelectedRaceIndex] =
useState<number>(0);
  const [selectedGender, setSelectedGender] = useState<boolean |
null>(null);
  const [selectedClassIndex, setSelectedClassIndex] =
useState<number>(0);
  const [headIndex, setHeadIndex] = useState<number>(0);
  const [bodyIndex, setBodyIndex] = useState<number>(0);
  const [legsIndex, setLegsIndex] = useState<number>(0);
  const [characterName, setCharacterName] = useState<string>("");
  const [hash, setHash] = useState<string>("");

  const handlePrevRace = () => {
    setSelectedRaceIndex((prev) => (prev === 0 ? races.length - 1 :
prev - 1));
  };

  const handleNextRace = () => {
    setSelectedRaceIndex((prev) => (prev === races.length - 1 ? 0 :
prev + 1));
  };

  const handleNextClass = () =>
    setSelectedClassIndex((prev) => (prev + 1) % classes.length);

  const handlePrevClass = () =>
    setSelectedClassIndex((prev) => (prev === 0 ? classes.length - 1
: prev - 1));

  const handleGenderSelect = (gender: boolean) => {
    setSelectedGender(gender);
```

```
    setStep(3);
  };

  const handleNext = () => setStep(2);

  const handlePrevItem = (
    setState: React.Dispatch<React.SetStateAction<number>>,
    items: string[]
  ) => {
    setState((prev) => (prev === 0 ? items.length - 1 : prev - 1));
  };

  const handleNextItem = (
    setState: React.Dispatch<React.SetStateAction<number>>,
    items: string[]
  ) => {
    setState((prev) => (prev + 1) % items.length);
  };

  const generateHash = async () => {
  const configData = {
    race: races[selectedRaceIndex].maleImage,
    gender: selectedGender,
    class: classes[selectedClassIndex].nameClass,
    head: classes[selectedClassIndex].outfits[0].sex[selectedGender
? "male" : "female"].head[headIndex],
    body: classes[selectedClassIndex].outfits[0].sex[selectedGender
? "male" : "female"].body[bodyIndex],
    legs: classes[selectedClassIndex].outfits[0].sex[selectedGender
? "male" : "female"].legs[legsIndex],
  };

  const stringifiedData = JSON.stringify(configData);

  const encoder = new TextEncoder();
  const dataBuffer = encoder.encode(stringifiedData);

  try {
    const hashBuffer = await crypto.subtle.digest("SHA-256",
dataBuffer);

    const hashArray = Array.from(new Uint8Array(hashBuffer));
    const hashHex = hashArray.map(byte =>
byte.toString(16).padStart(2, "0")).join("");

    setHash(hashHex);
  } catch (error) {
    console.error("Error generating hash:", error);
  }
};

  return (
    <div
```

```
      style={{
        display: "flex",
        justifyContent: "center",
        alignItems: "center",
        height: "100vh",
        textAlign: "center",
        padding: "20px",
        position: "relative",
      }}
    >
      <LanguageSwitch language={language} setLanguage={setLanguage}
/>

      {step === 1 ? (
        <div>
          <h2>{t.chooseRace}</h2>
          <RaceSelection
            races={races}
            selectedRaceIndex={selectedRaceIndex}
            handlePrevRace={handlePrevRace}
            handleNextRace={handleNextRace}
            handleNext={handleNext}
            t={t}
          />
        </div>
      ) : step === 2 ? (
        <div>
          <h2>{t.chooseGender}</h2>
          <GenderSelection
            races={races}
            selectedRaceIndex={selectedRaceIndex}
            handleGenderSelect={handleGenderSelect}
            t={t}
          />
        </div>
      ) : step === 3 ? (
        <div>
          <h2>{t.chooseClass}</h2>
          <ClassSelection
            classes={classes}
            selectedClassIndex={selectedClassIndex}
            handlePrevClass={handlePrevClass}
            handleNextClass={handleNextClass}
            setStep={setStep}
            t={t}
          />
        </div>
      ) : (
        <div>
          <h2>{t.customizeOutfit}</h2>
          <div>
            <CharacterCustomization
              races={races}
```

```
              selectedRaceIndex={selectedRaceIndex}
              selectedGender={selectedGender !== null ?
selectedGender : true}
              classes={classes}
              selectedClassIndex={selectedClassIndex}
              headIndex={headIndex}
              bodyIndex={bodyIndex}
              legsIndex={legsIndex}
              handlePrevItem={handlePrevItem}
              handleNextItem={handleNextItem}
              setHeadIndex={setHeadIndex}
              setBodyIndex={setBodyIndex}
              setLegsIndex={setLegsIndex}
            />
          </div>
          <CharacterNameInput
            characterName={characterName}
            setCharacterName={setCharacterName}
            t={t}
          />
          <ImageGenerator
            characterName={characterName}
            t={t}
            generateHash={generateHash}
          />
          {hash && (
            <div className="hash-display">
              <h4>Generated Hash: {hash}</h4>
            </div>
          )}
        </div>
      )}
    </div>
  );
};

export default CharacterCreator;
```

Додаток В – Диск з роботою