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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра інженерії програмного забезпечення
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка інтерактивної системи менеджменту персонажів рольової гри з використанням Django REST фреймворку

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Гук В. А. (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 Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Кафедра інженерії програмного забезпечення
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

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

студенту Гук Віталій Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтерактивної системи менеджменту персонажів рольової гри з використанням Django Rest фреймворку.

Керівник роботи к.т.н., ст. викл. Стоянов Юрій Миколайович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*18*» *квітня* 2025 року № *4/7-333*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Система з інтерактивними змінними листами персонажів та їх параметрами для проведення настільної гри дракони та підземелля.

4. Зміст роботи (перелік питань, які потрібно розробити) аналіз предметної області, аналіз конкурентів, визначення вимог, визначення інструментів, методологій та технологій, моделювання системи та створення діаграм, розробка веб-застосунку, тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів) діаграми варіантів використання для всіх акторів, діаграми класів, діаграма послідовності, дизайн сайту, програмний код, слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	к.т.н., доцент кафедри МТ Лазарюк Валерій Володимирович.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	20.04.2025	
2	Аналіз конкурентів	23.04.2025	
3	Основна частина	05.06.2025	
4	Безпека життєдіяльності, основи охорони праці	06.06.2025	
5	Попередній захист	09.06.2025	
6	Підготовка пояснювальної записки	15.06.2025	
7	Підготовка презентації та доповіді		
8	Нормоконтроль, рецензування		
9	Антиплагіат		

Студент _____ Гук В. А.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Кваліфікаційна робота бакалавра виконана Гуком Віталієм Андрійовичем. студентом групи СП-41 Тернопільського національного технічного університету імені Івана Пулюя, присвячена розробці інтерактивної системи менеджменту персонажів рольової гри Дракони та Підземелля. Робота обсягом 76 сторінок, яка включає 24 рисунків, 1 таблицю, 6 додатків та бібліографію з 24 джерел.

Ключові слова: Dungeons and Dragons, Django, Веб-застосунок, REST, Українська локалізація.

Мета дипломної роботи полягає в аналізі предметної області та розробки українськогомовного веб-застосунку на основі поєднання моделей настільної гри Dungeons and Dragons для створення та менеджменту листа персонажа з використанням сучасних веб-технологій. На разі не існує іншого українськомовного програмного забезпечення, яке б достатньо обхоплювало гнучкість аспектів гри Дракони та Підземелля, яка набуває все більшої популярності в українському просторі.

При розробці програмного забезпечення було виділено багато уваги створенню зручного та інтуїтивного користувацького інтерфейсу і надання якомога широкого доступу до параметрів та навичок персонажа. Було використано методи об’єктно-орієнтованого проектування та моделювання предметно області фреймворку Django.

Застосунок рекомендується до використання в ігрових спільнотах, клубах та звичайний гравці настільної гри. Проєкт не є комерційній та створений для покращення якості проведення гри

Ця дипломна робота вносить великий вклад в розвиток українськомовної ігрової спільноти, внісши веб-застосунок, який поєднує в собі функціональність листа персонажа з гнучкістю ігрових механік гри Dungeons and Dragons. Проєкт задовольняє потребу в описаному вище локалізованому інструменті гравців, який з кожним днем стає все більше і більше.

ABSTRACT

The bachelor's qualification thesis, completed by Vitalii Andriiovych Huk, a student of group SP-41 of the Ivan Puluj National Technical University of Ternopil, is dedicated to the development of an interactive character management system for the tabletop role-playing game Dungeons & Dragons. The work consists of 76 pages, including 24 figures, 1 tables, 6 appendices, and a bibliography of 24 sources.

Keywords: Dungeons and Dragons, Django, Web Application, REST, Ukrainian localization.

The aim of this thesis is to analyze the subject domain and to develop a Ukrainian-language web application based on the models of the Dungeons & Dragons tabletop game for creating and managing character sheets using modern web technologies. Currently, there is no other Ukrainian-language software that comprehensively supports the flexibility of the Dungeons & Dragons system, which is gaining increasing popularity in the Ukrainian-speaking community.

During the development process, significant attention was given to creating a user-friendly and intuitive interface, as well as providing comprehensive access to character parameters and abilities. The development applied object-oriented design methods and subject area modeling using the Django framework.

The application is recommended for use by gaming communities, hobby clubs, and individual tabletop players. The project is non-commercial and was created to improve the quality of gameplay for such a complex system as Dungeons & Dragons, specifically for the Ukrainian-speaking community.

This thesis makes a significant contribution to the development of the Ukrainian gaming community by introducing a web application that combines the functionality of a character sheet with the flexibility of Dungeons & Dragons game mechanics. The project meets the growing demand for a localized digital tool that continues to gain popularity among players every day.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

D&D – Dungeons and Dragons

ПЗ – Програмне забезпечення

ВЗ – Веб-застосунок

БД – База даних

UI – User Interface

UX – User Experience

JS – JavaScript

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

JSON – JavaScript Object Notation

URL – Uniform Resource Locator

API – Application Programming Interface

REST – Representational State Transfer

HTML – HyperText Markup Language

SCSS – Sassy Cascading Style Sheets

SQL – Structured Query Language

ORM – Object-Relational Mapping

ВСТУП

В сьогоднішніх реаліях інформаційної ери, ефективність та зручність доступу до зберігання, перегляду та редагування даних грає велику роль в будь якій сфері життя, включаючи й розваги. Якщо дані подані в зручному для користування вигляді, збільшується якість роботи з ними і відповідно зменшує кількість часу, стресу і сил для виконання цілі. Крім того важливо, що такі інструменти були доступні для всіх у спеціалізованому та локалізованому форматі.

Одна з таких сфер діяльності це ігри та розваги. Сьогодні на просторах української ігрової спільноти дуже стрімко популяризується настільна гра Dungeons and Dragons, яка декілька років тому в нас була популярна лише в дуже вузьких кругах. Сьогодні ця тенденція стрімко змінюється, проте на фоні цього немає жодного локалізованого під український ринок, який б достатньо реалізував всі можливості листа персонажа, який грає основну роль в проведені розваги, яка основується на взаємодії героїв гравців з світом та між собою.

Для реалізації такого інструменту необхідно використовувати потужні методи об’єктно орієнтованого проектування та моделювання через комплексний зв’язок моделей та їх даних, з яких складається лист персонажа. Також для реалізації такого проєкту необхідні високий рівень знання відповідних технологій для створення та маніпуляцій таких об’єктів.

Мета даної роботи створити такий інструмент, який задовільнить потребу в зберіганні та редагування аспектів персонажа у зручному для пересічного користувача. Також важливою частиною проєкту полягає в надані як і вже готових об’єктів з яких складається персонаж, так і створених самим користувачем для власного користування. Через це важливо реалізувати систему так, щоб цей процес був максимально легким в використанні, так як під час гри в D&D події відбуваються доволі швидко в відносно великій групі людей(в середньому від 3 до 6 гравців). Непотрібні затримки в часі можуть вплинути на якість проведення гри

та зменшити і так обмежений час її проведення, бо процес її проведення займає багато часу.

Дослідження цієї проблеми проводилось як і на особистому досвіді участі в грі, так і на відгуках та побажаннях інших гравців з різних країн світу, де ця гра популярна вже досить багато часу. Було зібрано багато проблем з оригінальним паперовим та новішим електронним варіантом листа персонажа, з якими стикаються звичайні гравці.

На основі цього дослідження було розроблено підхід до інтерактивного листа персонажа, адаптований до потреб україномовної спільноти. Запропоновано рішення, яке враховувало змінні вимоги користувачів та пропонує інтерфейс на українській мові, автоматизацію введення зв’язаних даних, динамічна зміна аспектів персонажа та зручну структуру подання його інформації. Удосконалено спосіб побудови листа персонажа через врахування мовної специфіки та реальних сценаріїв використання існуючих аналогів.

В результаті проведеного дослідження були реалізовані у вигляді веб-застосунку для створення, перегляду та редагування листів персонажів гравців рольової гри D&D. Застосунок орієнтований на україномовних користувачів та враховує потреби і новачків, і досвідчених в цій сфері ігрових ровах. Для цього застосунок надає прості та ефективні рішення, які усувають часті проблеми в предметній області, характерні для паперових або іншомовних аналогів. Вибір технологій був проведений на основі зважения наявних факторів масштабованості, структурованості даних та зручності реалізації логіки взаємопов’язаних об’єктів. Для зберігання даних вибрано реляційну БД MySQL, що забезпечує ефективність при використані з великою кількістю зв’язних моделей та об’єктів. Щодо розробки, то було обрано фреймворк Django через зручність та ефективність створення зв’язаних моделей та побудови орієнтованих REST сервісів. Ці обрані інструменти для розробки веб-застосунків забезпечили гнучку та надійну архітектуру застосунку. Напрацювання цього дослідження були подані для публікації у вигляді тези в конференції, саму тезу можна переглянути в ДОДАТОК А.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНТЕРАКТИВНОЇ СИСТЕМИ ДЛЯ МЕНЕДЖМЕНТУ ПЕРСОНАЖІВ D&D

1.1 Аналіз існуючих конкурентів

Попри популярність настільної гри D&D в світі та зростаючу з кожним днем кількість гравців в українській спільноті збільшується, доступні ПЗ для створення, редагування та перегляду листів персонажів часто мають суттєві обмеження щодо української локалізації, а точніше її відсутність, і гнучкості та зручності використання. Це створює чи малу проблему в наявності такої платформи та створює потребу та попит на розробку такого застосунку. З цією метою було проаналізовано низку наявних аналогів.

1.1.1 D&D Beyond

D&D Beyond – це обмежено безкоштовна платформа для менеджменту персонажа від авторів настільної гри D&D, через що їхні сервіси високої якості. Сам веб-застосунок має потужний і широкий функціонал для зручного управління своїми персонажами. На рисунку 1.1 зображено зовнішній вигляд головної сторінки листа персонажа.

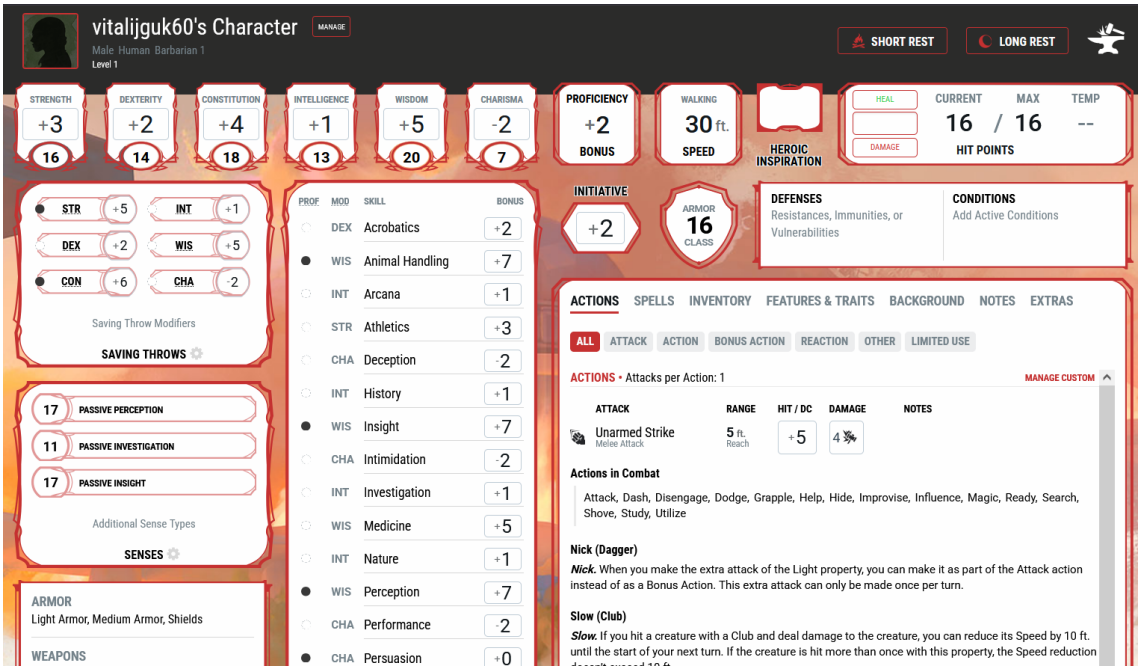


Рисунок 1.1 – Зовнішній вигляд листа персонажа в D&D beyond.

Також є адаптований мобільний додаток з тією ж назвою. Після створення свого цифрового листа персонажа можна:

– Повний доступ та інтеграція з офіційними джерелами та книгами правил;

– Керувати параметрами персонажа та спорядженням;

– Можливість кидати куби та ділитись ними з своїми товаришами по грі;

– Швидко та легко підвищувати рівень персонажа не пропускаючи ніяких кроків;

– Можливість персоналізувати зовнішній вигляд деяких елементів;

– Мати доступ до своїх персонажів будь де в мобільному застосунку D&D Beyond [1].

В поєднанні ці фактори створюють потужний інструмент для поставленої цілі, проте є суттєві мінуси:

– Багато обмежень функціоналу() для безкоштовної версії;

– Дуже велику кількість контенту з інших книг правил треба докупляти за велику ціну, щоб використовувати;

– Немає можливості додавання власного контенту;

– Громісткий та заплутаний UI;

– Не оптимізований сайт, який повільно працює.

– Відсутність української локалізації та метричної системи числення

1.1.2 Roll20

Roll20 – це частково безкоштовна незалежна від офіційного автора настільної гри платформа для проведення онлайн ігор D&D. В її функціональні можливості входять створення ігрової мапи, комунікація між гравцями онлайн, створення персонажів магазин асетів для проведення гри [2]. Функціонал доволі обмежений та основна увага акцентована на проведені гри в режимі онлайн. Щодо можливості створення персонажа, то застосунок задовольняє лише дуже базові потреби гравців. На рисунку 1.2 зображено зовнішній вигляд листа персонажа в Roll20.

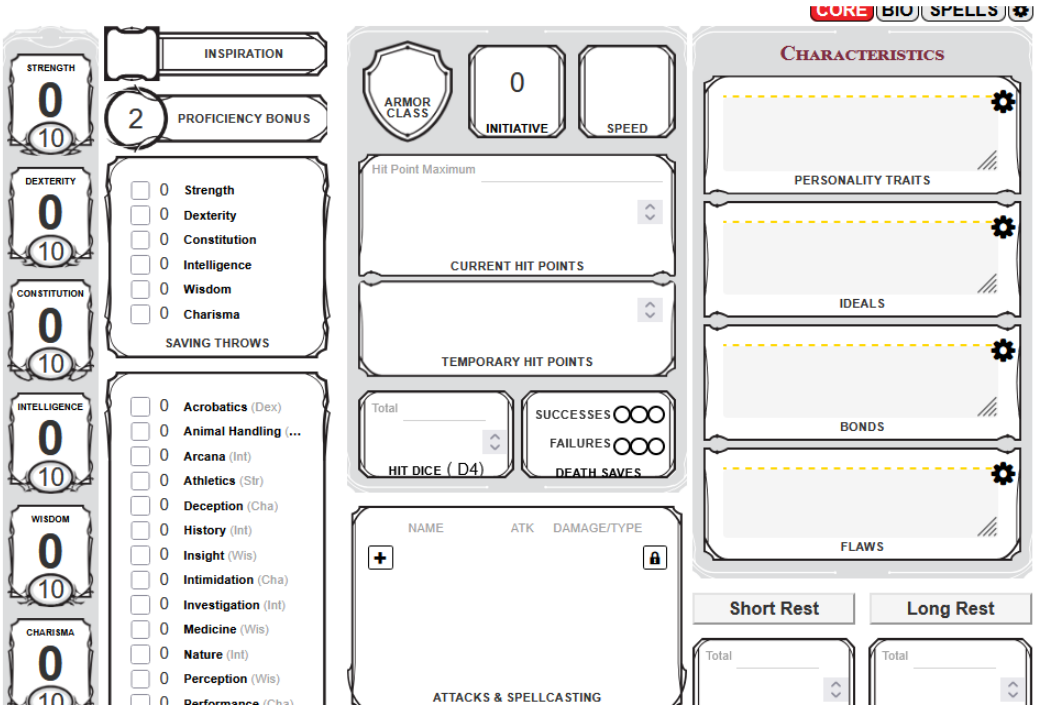


Рисунок 1.2 – Зовнішній вигляд листа персонажа в Roll20

Сам лист персонажа(див. рис. 1.2) не зручний у використані, заплутаний та виступає як копія паперового листа персонажа. Все заповнюється в ручну, ніяких

зв’язків та моделей. Крім того є багато обмежень в безплатній версії аккаунта, які доступні лише в вищих двох рівнях підписки. Також немає українсько локалізації, яка заважає якісному користуванню застосунком.

1.1.3 Інші конкуренти

Існує достатньо інтерпретацій листів персонажа у цифровому вигляді, такі як FightClub, OrcPub2, Dungeon Master’s Vault. Всі вони реалізовують різні ступені автоматизації заповнення, можливості додавання власних об’єктів та інтегрування з ігровими кампаніями. Між ними всіма найпоширеніші є такі проблеми:

– Складність в освоєні;

– Недостатня свобода заповнення листа персонажа;

– Відсутність всіх необхідних полів в листі персонажа;

– Низька швидкодія;

– Відсутність локалізації;

– Відсутність оновлень застосунку.

Також варто згадати, що є декілька веб-застосунків, які надають достатньо функціоналу для взаємодії з листом персонажа, роте розташовані на російських доменах. Через відсутність такого локалізованого під український ринок веб-застосунку, українська спільнота змушена вибирати або сайт з російським доменом, або платну платформу без української мови.

1.2 Визначення вимог та специфікації ПЗ

Обов’язковим та надзвичайно важливим кроком в розробці ПЗ є визначення його вимог, на основі яких буде розроблений застосунок. Добре пропрацьовані вимоги дозволяють:

– виробити загальне розуміння між Замовником і Розробником;

– визначити рамки проекту;

– точніше визначити фінансові і часові характеристики проекту;

– убезпечити Замовника від ризиків отримати продукт, в якому він не зможе працювати

– убезпечити Розробника від ризиків потрапити в ситуацію "неконтрольованого розмиття меж", яке може призвести до непередбачених витрат ресурсів понад початкові очікування [3].

Під час аналізу ПЗ описані властивості, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться у розробці. Вимоги визначаються в процесі аналізу вимог та фіксуються в специфікації вимог, діаграмах прецедентів та інших артефактах процесу аналізу та розробки вимог [4].

Також варто пам’ятати, що ПЗ розробляється для користувача, тому дуже важливо звертати на це увагу приймаючи рішення про розробку застосунку. У випадку цього проекту цільовою аудиторією(користувачами) є гравці в настільну гру D&D, які шукають українськомовну платформу для зберігання, створення, редагування та перегляду особистих листів персонажів з можливість створення особистих об’єктів.

Функціональні вимоги регламентують функціонування або поведінка системи (behavioral requirements). Функціональні вимоги відповідають на питання "що повинна робити система" в тих або інших ситуаціях. Функціональні вимоги визначають основний "фронт робіт" Розробника, і встановлюють цілі, завдання і сервіси, що надаються системою замовникові [5]. В розроблюваному в межах цієї роботи веб-застосунок вимагає особливо обережно описані функціональні вимоги,

так як платформа складається з багатьох зв’язків між різними її елементами на одній сторінці.

Проаналізувавши вимоги до ПЗ було визначено такі функціональні вимоги:

– Користувач повинен мати можливість створити обліковий запис та увійти в вже існуючий, щоб получити доступ до системи;

– Можливість створити персонажа зі всіма обов’язковими параметрами та внести всі другорядні параметри опісля за потреби;

– Система повинна визначати деякі параметри персонажа автоматично;

– Персонажа можна видаляти;

– Користувач повинен мати можливість створювати декількох персонажів та переглядати список всіх персонажів, які належать йому;

– Усі параметри персонажа, які не обраховуються автоматично можуть бути змінені користувачем;

– Користувач повинен мати доступ до перегляду всіх параметрів свого персонажа;

– Можливість завантажувати в систему та з неї персонажів в форматі json.

Перечислені функціональні вимоги повністю покривають потреби користувачів з цільової аудиторії.

Нефункціональні вимоги, відповідно, регламентують внутрішні і зовнішні умови або атрибути функціонування системи. К. Вігерс виділяє наступні основні групи нефункціональних вимог:

– Зовнішні інтерфейси (External Interfaces)

– Атрибути якості (Quality Attributes)

– Обмеження (Constraints) [5].

В розроблюваному в межах цієї роботи веб-застосунок вимагає детального планування характеристик вимог, які не пов’язані з конкретними функціями, а є загальними для всього проекту. Через велику кількість моделей, даних, зв’язків та інших елементів випливає багато можливих проблем з системою: недосвідчений користувач може бути розгублений в інтерфейсі застосунку, перенавантаження, негнучкість, безпека.

Проаналізувавши ці потенційні проблеми визначено низку нефункціональних вимог, яких потребує система:

– UI повинен бути зручний, інтуїтивний та не перенавантажений;

– Застосунок повинен працювати стабільно, без затримок;

– Система повинна бути стійка до вводу непередбачуваних даних;

– ПЗ має бути сумісне з популярними сучасними браузерами;

– Об’єкти системи з яких складається персонаж повинні бути розширювальні;

– Дані користувача повинні зберігатись захищено(авторизація та обмеження доступу).

1.3 Визначення архітектури, методології та технологій розробки

1.3.1 Архітектура та методологія

Дуже важливою частиною життєвого циклу ПЗ ефективно підібрана архітектура та степінь її опису. Архітектура ПЗ – це подання всього проекту з виділенням важливих характеристик і затушовування деталей, включає в себе найважливіші статичні і динамічні аспекти системи [6]. Важливість та вага вибору архітектури базується на тому, що це вибір структури та “логістики” проєкту. Як і в інших структурах підприємств та проєктів, ці два фактори впливають на всі його аспект безпосередньо так як ті, хто реалізовують застосунок з нею бачить повну карту розробки всього ПЗ, що дозволяє швидко, ефективно та якісно виконувати поставлене завдання.

Проаналізувавши вимоги та параметри розроблюваної системи було обрано трирівневу архітектуру. Її суть полягає в поділ системи на три логічні рівні:

– Рівень представлення –рівень, через який користувача взаємодіє з системою та відправляє запити;

– Логічний рівень – рівень, в якому реалізовується логіка роботи застосунка, така як обробка запитів, робота з моделями, реалізація прикладної логіки;

– Рівень даних – рівень, в якому відбувається взаємодія з БД.

Щодо обраної методології, проаналізувавши природу застосунку у вигляді комплексної комбінації різних моделей та рішень, з яких почастинно складається основний функціонал застосунку, було обрано Agile. Це гнучкий підхід, згідно з ним, працівники мають право на помилки, допущені при плануванні й оцінюванні проєкту. Адже все передбачити неможливо. Метою методології agile є мінімізація прорахунків і деталей, залишених поза увагою. Тут реально передбачити потенційні зміни чи доповнення, безболісно вбудувавши їх у робочий процес [7]. Хоч і Agile більше підходить до роботи в команді, велика кількість різних моделей та комплексний зв’язок між ними ідеально підходять під цю методологію.

1.3.2 Компоненти та вибір технологій

В результаті проведеного аналізу також були обрані компоненти та відповідні до них технології для реалізації ПЗ. Звісно, вибір архітектури, як складання мапи роботи є дуже важливою частиною життєвого циклу, проте без правильно підібраних інструментів його реалізація буде неможливою. Крім того варто зважати, що в кожної технології є свої сильні та слабкі сторони, зважаючи на які і потрібно обирати ці інструменти.

Для реалізації цієї системи було обрано такі компоненти:

– Фронтенд: HTML, SCSS, JavaScript;

– Бекенд: Django, JavaScript;

– База даних: MySQL.

З цього переліку можна побачити і обрану трирівневу архітектуру. Рівень представлення – компонент фронтенд, рівень логіки – компонент бекенд, рівень

даних – компонент база даних. Технології для реалізації компонентів були обрані в результаті аналізу сучасних технології.

SCSS або Sass – це мова стилів, яка компілюється у CSS. Вона дозволяє використовувати змінні, вкладені правила, міксини, функції та багато іншого, зберігаючи повну сумісність із синтаксисом CSS. Sass допомагає підтримувати великі таблиці стилів у впорядкованому стані та спрощує повторне використання дизайну як у межах одного проєкту, так і між різними проєктами [8]. Використання цієї мови стилів суттєво полегшує написання стилів, надає їм гнучкість, а також робить стилі об'єктно орієнтованими.

Рисунок 1.3 – Логотип Sass [8]

Django – потужний високорівневий веб-фреймворк для мови програмування Python. Він дозволяє швидко, легко та ефективно розробляти веб-застосунки. Сам фреймворк має чудову репутацію на ринку та серед розробників. В нього багато потужних аспектів, які забезпечують все, що може бути необхідно для створення веб-застосунків.

Рисунок 1.4 – Логотип Django [9]

Сильних сторін фреймворку Django чимало, до них відносяться:

– ORM(Object-Relational Mapper) система, яка дозволяє виконувати операції з БД кодом python;

– Власна, повна та безпечна аутентифікація та вбудована робота з аккаунтами, групами та дозволами;

– Можливість створення темплейтів, на основі яких можна будувати HTML сторінки;

– Власні потужна бібліотека з формами для рендеренгу та валідації;

– Зручний та прости URL роутинг;

– Наявність вбудованої адмін панелі;

– Захист від SQL ін’єкцій, крос-сайт маніпуляцій, віддаленого виконання коду та “Вкрадення кліків” [9].

Також Django має потужну архітектурну модель MVT(Model-View-Template), який в рази покращує структуру коду, його чистоту та робить розширення і модифікацію вже існуючого коду набагато легшими процесами. Її суть полягає в тому, щоб забезпечити чітке розділення логіки, даних, та відображення. Для кожного є свій окремий файл для їхньої реалізації, які зручно імпортувати та використовувати. Цими файлами є: Model, View та Template.

Model відповідає за описання об’єктів, які відображають структуру даних в БД. Також об’єкт model має в собі широко описаний функціонал описаний в собі для створення класу: різні види полів класу, заготовані класи всередині моделі для уточнення детальніше аспектів для взаємодією зі всією системою. З цих моделей створюється міграція, в результаті якої створюється вже готова БД.

Views відповідає за рендер сторінок. Саме в них дані з БД підготовляються до передачі в URL та в подальшому в HTML. Також views відповідають за обробку HTTP-запитів і відповідно за надсилання відповіді на них користувачу. Якщо узагальнити, то це є сервісний шар, в якому і виконується основна логіка проєкту.

Template відповідає за оформлення користувацького інтерфейсу, а саме HTML сторінок. Фреймворк має вбудований функціонал для створення шаблонів для сторінок. Він дозволяє описати каркас основної сторінки з можливістю

створення різних секцій в них, які можуть бути доповненні при описані нової сторінки на основі даного шаблону. Побачивши цей функціонал можна зазначити, що це реалізація одного з принципів ООП – поліморфізм. Можливість створення шаблонів в рази полегшує розширення та доповнення сторінок та забезпечує читабельний структурований код, розбитий на блоки.

JavaScript – це скриптова або програмна мова, яка дозволяє реалізувати складні функції на веб-сторінках. Вона також є третім шаром у стандартних веб-технології, два з яких – HTML та CSS [10]. В цьому проєкті JavaScript використано для створення слухачів подій та прив’язування до них скриптів для створення, видалення, редагування та зберігання об’єктів при взаємодії з HTML сторінкою.

В результаті виконання цього розділу було здійснено аналіз предметної області, вивчено існуючі аналоги веб-застосунків для менеджменту персонажів Dungeons & Dragons, а також визначено їхні переваги та недоліки. Проведено аналіз потреб цільової аудиторії та сформульовано функціональні й нефункціональні вимоги до майбутньої системи. Визначено архітектурний підхід, методологію розробки та обрано відповідні технології. Це заклало теоретичну та практичну основу для ефективного проектування і реалізації веб-застосунку, орієнтованого на україномовну ігрову спільноту.

РОЗДІЛ 2 МОДЕЛЮВАННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ ДЛЯ МЕНЕДЖМЕНТУ ПЕРСОНАЖІВ D&D

Проєктування програмного забезпечення, зокрема веб-застосунків, нерозривно пов’язане з побудовою формалізованих моделей, які описують як логіку взаємодії користувачів із системою, так і структуру її внутрішніх компонентів. У контексті розробки інтерактивного веб-застосунку для менеджменту персонажів рольової гри Dungeons & Dragons особливу увагу слід приділити побудові моделей, які забезпечують правильну взаємодію численних пов’язаних сутностей, що формують лист персонажа. Моделювання дозволяє не лише візуалізувати структуру майбутньої системи, а й своєчасно виявити її слабкі сторони до етапу реалізації.

У цьому розділі детально розглянуто побудову діаграм варіантів використання, класів та послідовностей, що відображають логіку взаємодії користувача з системою та забезпечують повну структуризацію її компонентів. Такий підхід дає змогу досягти високого рівня узгодженості між функціональними вимогами та архітектурною реалізацією системи, що, в свою чергу, є основою для стабільної, масштабованої та зручної у використанні платформи.

2.1 Моделювання діаграм варіантів використання

Моделюючи ПЗ обов’язковим та дуже важливим кроком є визначення всіх можливостей взаємодії користувача з системою. У випадку веб-застосунку зазвичай є як мінімум два види акторів, хоча зазвичай більше, так як в різних середовищах є різні ролі, які виконують люди. Для зображення таких сценаріїв існує діаграма варіантів використання.

Модель варіантів використання – це повна специфікація всіх можливих шляхів використання системи (ВВ). Ця специфікація може ввійти в укладений із замовником договір. Модель ВВ допомагає визначити границі системи, описуючи все, що вона повинна робити для користувачів [11].

Модель використання складається з різних структур, які репрезентують різні частинки пов’язаних з функціональними вимогами від лиця користувача. В діаграмі використання може є чотири типи об’єктів:

– Актор

– Система

– Зв’язок

– Коментар

Актор – це об’єкт, який репрезентує людину, яка користуватиметься функціоналом системи. Акторів може бути декілька в одному середовищі, які хоч і взаємодіють з системою, проте не є її частиною. Стандартом для веб-застосунків є наявність як мінімум двох акторів – адмін та користувач. Загально прийняте позначення для цього об’єкта – чоловічок.

Система – це об’єкт, який репрезентує модельовану систему. У випадку цієї роботи – веб-застосунок. Сама система містить різні прецеденти з якими взаємодіють актори. Зображується цей об’єкт прямокутником з назвою системи зверху.

Зв’язок – це об’єкт, який репрезентує взаємодію між прецедентом та актором. Важливо пам’ятати, що актор завжди має бути пов’язаний з прецедентом, або це означатиме, що актор не користується системою, проте прецедент не обов’язково на пряму пов’язаний з актором та натомість може мати зв’язок з іншим прецедентом. Існує чотири види зв’язків: асоціація, розширення, виключення та узагальнення. Кожен з них відповідає за природу взаємодії між прецедентами та акторами. Сам об’єкт зображений лінією з кінцем, який варіюється між різними видами зв’язків.

Коментар не є обов’язковим, але поліпшує читабельність та легкість розуміння діаграми. Цей об’єкт містить в собі розширення функціональності та

уточнення у вигляді заміток та дописів. При моделювання складною та комплексної системи їх варто добавляти, щоб запобігти затримкам при подальшій розробці.

При проектуванні було визначено два актори: Адміністратор та Користувач. Для кожного окремо було створено відповідну діаграму варіантів використання, які описують прецеденти та зв’язки з ними.

Користувач – це актор, який виконує роль звичайного пересічного користувача. Він має доступ до всього функціоналу для створення, видалення та редагування листів персонажів. До прецедента редагування листів персонажа входить також й управління користувацьким контентом. Користувацьким контентом є об’єкти листа персонажа, які користувач може створити власноруч та якими може користуватись лише їх автор. Саму діаграму варіантів використання зображено на рисунку 2.1.

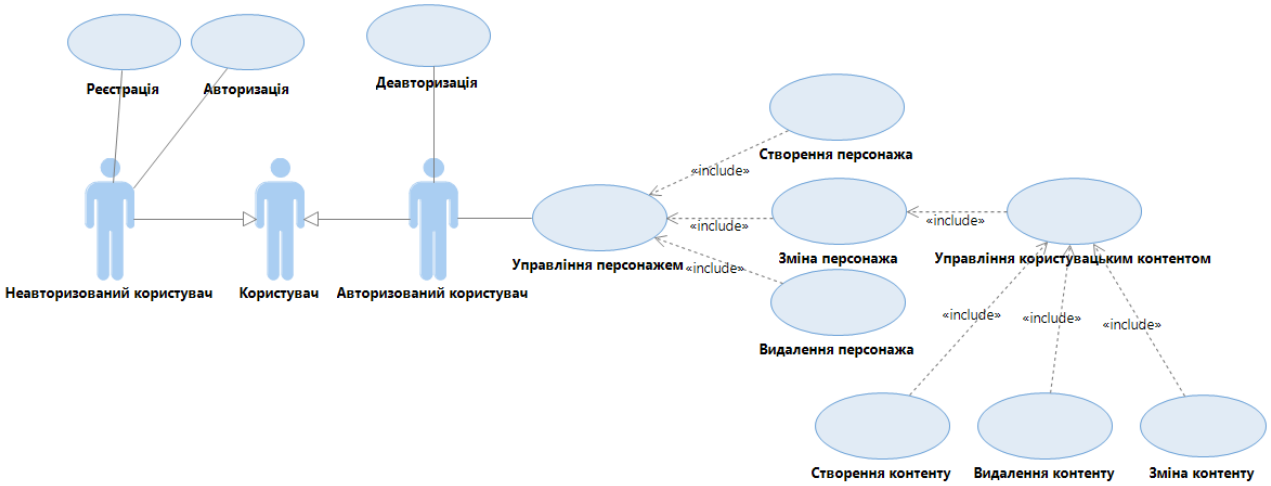


Рисунок 2.1 – Діаграма варіантів використання для користувача

Адміністратор – це актор, який модерує контент наявний в системі. Крім того, що він успадковує функціонал користувача, він може створювати контент, який буде доступний всім користувачам системи. Також він може створювати статті, які будуть показані на головній сторінці. Весь функціонал, доступний з адмін панелі. Саму діаграму варіантів використання зображено на рисунку 2.2.

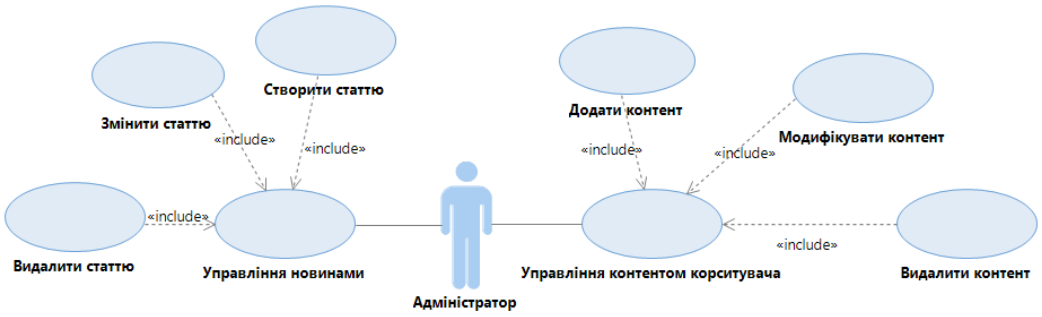


Рисунок 2.2 – Діаграма варіантів використання для адміністратора

Визначивши акторів, прецеденти та зв’язки між ними в результаті отримав діаграму варіантів використання, яка описує зрозуміло для любої людини взаємодію наявних акторів. На основі такої діаграми легко можна побудувати інші діаграми продовживши моделювання системи. Крім того ці діаграми поможуть розробникам краще розуміти роботу веб-застосунку, що в рази полегшить процес інтеграції в кодовому варіанті та зменшить необхідний час для реалізації програмної частини коду.

2.2 Моделювання діаграми класів

В такій комплексній системі як інтерактивний лист персонажа D&D більша частина роботи відбуваються на основі даних. Щоб змоделювати ці данні потрібно описати моделі. Самі по собі це класи з відповідними полями та методами. Саме тому всі класи в цій системі це моделі, які є майбутніми одиниця об’єктів застосунку. У цьому розділі розглядається побудова діаграми класів, яка відображає основні структури(класи) предметної області та зв’язки між ними. Діаграма класів дозволяє структурувати систему на логічному рівні та визначити ключові об'єкти, їх атрибути й взаємодію.

Class diagram (діаграма класів) – основна діаграма для створення коду програми. За допомогою діаграми класів створюється внутрішня структура системи, описується спадкування і взаємне положення класів один щодо одного.

Логічні структури, такі як класи – це лише заготовки, на основі яких потім будуть визначені фізичні об’єкти [12].

Сама діаграма класів складається з пакунків, класів та зв’язків між ними. Ці класи мають назву та описані атрибути та методи. Самі об’єкти мають вигляд прямокутників, розділених на три секції згадані вище. Також ці класи можуть бути абстрактними, які на пряму не використовуються і служать каркасом для побудови похідних класів.

Атрибути – це властивості та характеристики об’єкта, які описують його параметри. У контексті програмування – це змінні, які є частиною класу та зберігають в собі данні про нього. Крім того класи мають різні рівні доступу до методів та полів: private, public і protected, які позначають -, + та # відповідно. На основі них визначається інкапсуляція частини класу, тобто якого роду доступ мають різні аспекти системи.

Public – це режим доступу, який відкриває атрибут для всього середовища. Використати таке поле зі створеного відповідного об’єкту можна з будь якої частини системи.

Private – це режим доступу, який закриває атрибут від всього середовища. Таке поле можна використовувати тільки всередині його класу, а поза його межами він буде недосяжним.

Protected – це режим доступу, який відкриває атрибут тільки до похідних класів. Таке поле можна використати поза межами класу тільки з успадкованих класів.

Методи – це функціонал класу. Вони описують логіку, що може робити клас та як він це робитиме. Також методи часто повертають якісь дані та мають параметри, які передаються туди. Також методи можуть мати різні рівні інкапсуляції перекислених вище.

Хоч в нас і описані всі ці класи, проте самі по собі вони мало що можуть зробити. Система складається з багатьох класів, які зв’язані між собою та взаємодіють один з одним. Є чотири види цих відносин:

– Асоціація;

– Агрегація;

– Композиція;

– Успадкування.

Асоціація – це зв’язок між класами, який показує, що один об’єкт використовує інший та має до нього доступ, але не містить його. Цей зв’язок рахується звичайним та позначає прості відносини між структурами. Також цей зв’язок може мати назву ролі та мультиплікацію. Позначається лінією зі стрілкою на кінці який вказує пов’язаний на клас.

Агрегація – це зв’язок між класами, який показує, що один об’єкт є частиною іншого, але незалежний від нього, так як може існувати окремо від нього. Позначається лінією з пустим ромбом на кінці який вказує пов’язаний на клас.

Композиція – це зв’язок між класами, який показує, що один об’єкт є частиною іншого, залежний від нього та не може існувати без нього. Позначається лінією з заповненим ромбом на кінці який вказує пов’язаний на клас.

Успадкування – це зв’язок між класами, який показує, що один об’єкт успадковується від іншого(батьківського). При цьому похідний клас успадковує поля та методи головного класу. Позначається лінією з трикутником на кінці який вказує на батьківський клас.

При моделюванні та проектуванні системи було сформовано діаграми класів. Вони покривають всі моделі застосунку та формують зв’язки між ними. Переглянути діаграму можна на рисунку 2.3.

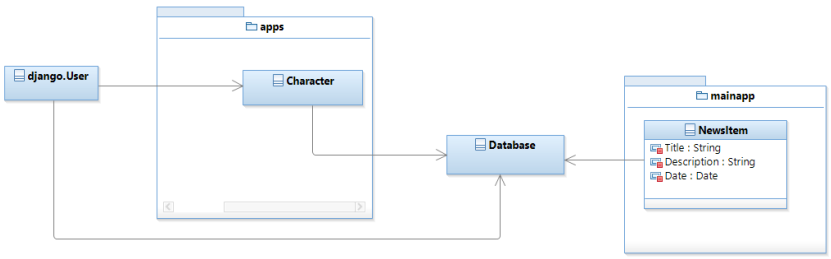


Рисунок 2.3 – Діаграма класів системи

Ця діаграма містить низку класів, з яких складається веб-застосунок. Варто зазначити, що об’єкт character дуже об’ємний, через що його вміст скрито для

кращого перегляду діаграми. Нижче буде наведено розписану діаграму класів для пакунка character. В системі є два робочих пакунка apps та mainapp.

Apps містить пакунки з реалізацією відділу системи. У випадку розроблюваного застосунку він містить apps.user та apps.character. Перший з перечисленого містить в собі реалізацію системи, яка пов’язана з користувачем. Було створено власного користувача, який успадковує функціонал абстрактного класу AbstractUser з Django. В успадкованому класі додано поля photo, date, ban та методи switchBan, loadPhoto.

Основні атрибути і методи для класу user:

– photo: фотографія користувача;

– date: дата створення аккаунта користувачем;

– ban: прапорець чи заблокований користувач в системі;

– switchBan(): зміна статусу доступу користувача на протилежний;

– loadPhoto(): завантаження фотографіх користувача.

Цей набір атрибутів та методів забезпечує гнучкість реалізації та повноту можливостей аккаунта користувача при програмуванні, регуляції та використанні системи.

Щодо пакету mainapp, то він містить лише одну модель – newsItem. Вона відповідає за зберігання об’єкту для новин. Самі новини будуть додаватись адміном. В цій моделі є лише атрибути. Їх є три: Title, Description, Date.

Атрибути для класу NewsItems:

Title: Заголовок статті;

Description: Опис статті, який містить її текст;

Date: Дата публікації статті.

Варто зазначити, що найбільше функціоналу описано на діаграмі класів для character. Цей пакет містить майже всі можливості веб-застосунку та має найбільшу кількість комплексних зв’язків. На рисунку 2.4 зображено цю діаграму класів, яка є розширенням діаграми зображено на рисунку 2.5, а саме класу character.

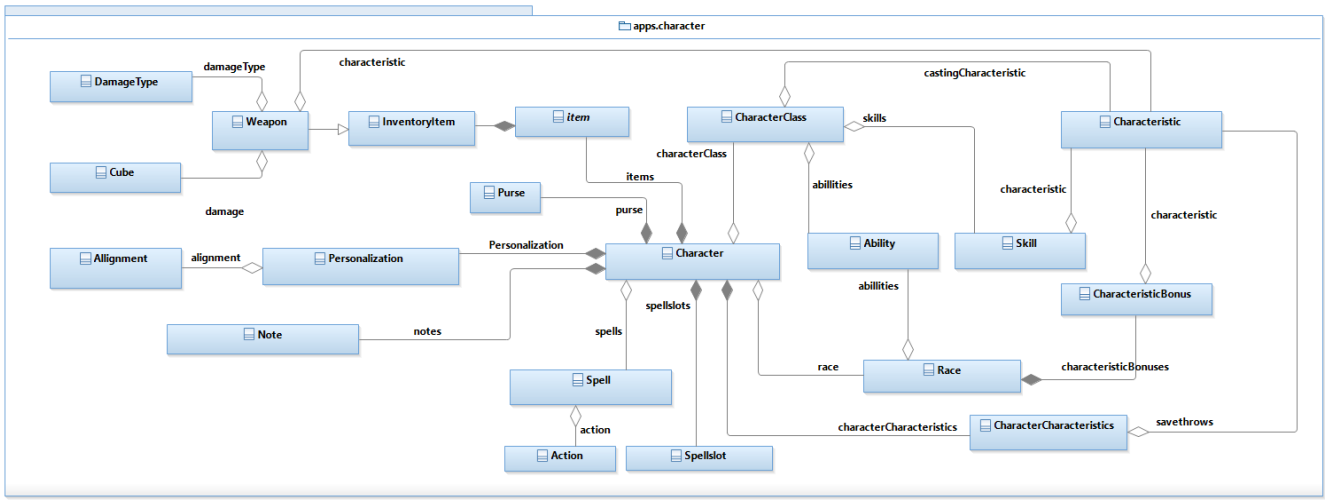


Рисунок 2.4 – Діаграма класів для пакету apps.character

Ця діаграма містить 19 класів, які покривають весь функціонал необхідний для реалізації системи для менеджменту листів персонажа D&D. За для зручності задля зручності читання атрибути та методи було приховано. Повну діаграму зображено в додатку А. Основним класом цієї діаграми є клас character – клас який репрезентує вже складений лист персонажа. Він має пов’язані структури, наприклад клас, раса, характеристики, заклинання та інші. Для кращого розуміння класів сформовано таблицю 2.1 з коротким описом кожного класу.

Таблиця 2.1 – короткий опис класів в пакунку character

№	Назва класу	Опис класу
1	Character	Це основний клас, який представляє ігрового персонажа та містить всі його параметри, які складаються з різних аспектів в єдиний кульмінаційний клас.
2	CharacterClass	Визначає клас персонажа (наприклад, воїн, маг). Він визначає, в чому умілий персонаж, які його здібності та яка його роль в команді.

Продовження таблиці 2.1

3	CharacterRace	Визначає расу персонажа (наприклад, орк, ельф, людина). Він з якого етносу походить ігровий персонаж, які його сильні та слабкі сторони його походження та які вродженні здібності він має.
4	Skill	Визначає навичку персонажа (наприклад, переконливість, скритність, атлетизм). За правилами гри в персонажа є 18 основних навичок які забезпечують наявність відповідної навички на всі можливі ситуації.
5	Ability	Визначає здібність персонажа (наприклад можливість бачити в темноті). Їх можна отримати від обраної раси, класу або добавити власноруч.
6	Characteristic	Один з найважливіших класів в системі. Визначає характеристики персонажа (наприклад, сила, гнучкість, інтелект). Ці характеристики визначають фізіологічні параметри персонажа. На основі них базується наскільки вправно персонаж володіє навичками та вправно виконує дії.
7	CharacteristicBonus	Цей клас використовується лише з Race класом та без нього існувати не може. Він визначає прибавку до характеристик раси.
8	CharacterCharacteristic	Цей клас специфікує характеристики під персонажа. Не плутати з класом characteristic, який відображає шість характеристик з правил гри. Цей клас натомість прив’язаний до персонажа та визначає значення його характеристик.

Продовження таблиці 2.1

9	Spellslot	Визначає рівні заклинань персонажа. Існує десять таких рівнів (заклини та дев’ять рівнів заклинань) і кожне заклинання має відповідний рівень. Цей параметр визначає, скільки разів персонаж може використати заклинань відповідного рівня між відпочинком.
10	Spell	Визначає заклинання. Вони можуть бути заздалегідь створені в системі та можуть бути створені користувачем, який буде бачити тільки автор.
11	Action	Визначає дію персонажа. В оригінальних правилах гри цих дій є декілька для різних видів дії. Основними є основна дія, додаткова та вільна.
12	Personalization	Визначає переконання, погляди та зовнішність персонажа (ідеали, страхи, колір очей, колір шкіри).
13	Alignment	Визначає відношення персонажа до світу та основу його рішень. Є різні комбінації поведінки між видом характеру (добрий, нейтральний, злий) та його стилем (законопослушний, нейтральний, хаотичний). Загалом цим комбінації є дев’ять.
14	Note	Визначає нотатки персонажа. Так як настільна гра має багато аспектів та рольового відіграшу, наявність нотатків обов’язкова, щоб слідкувати за подіями.
15	Purse	Визначає гаманець персонажа. В грі є валюта з різними видами монет. В основних правилах на яких базується система їх є п’ять видів: мідні, срібні, золоті, елктрумні та платинові копійки, які перечислені в порядку зростання цінності. Є відповідний курс переводу однієї монети в інший.

Продовження таблиці 2.1

16	InventoryItem	Визначає предмет інвертаря персонажа.
17	Weapon	Визначає зброю персонажа як зброю і як предмет. Сам клас успадковує InventoryItem та додає параметри, як характерні зброї, такі як шкода, вид шкоди та інші.
18	DamageType	Визначає вид шкоди (електрична, рубана, колюча). Їх є велика кількість, проте вони сталі.
19	Cube	Визначає доступні куби, які може використовувати гравець. Їх загалом є сім, а саме: к4, к6, к8, к10, к12, к20 та к100. Кожен з них репрезентує кість з відповідною кількістю граней.

Хоч і коротко опису достатньо для більшості класів, проте деякі потребують ширшого опису. Причиною цьому є комплексність деяких класів та х зв’язків в системі. Для кращого розуміння зацікавленими в розробці особами розроблюваного застосунку необхідна детально описати такі класи. Після проведення аналізу існуючих класів, детально було описано такі клас Character.

У моделі Character представлено повну структуру персонажа. Вона охоплює не лише базові характеристики персонажа, як-от ім’я, клас чи расу, а й додаткові – включаючи інвентар, особисті риси, заклинання та слоти для їх використання. Модель активно взаємодіє з іншими таблицями бази даних через зовнішні ключі та зв’язки типу many-to-many, що дозволяє гнучко зберігати складну інформацію про ігрову сутність.

Атрибути:

– user: посилання на користувача систем з якого було створено персонажа;

– name: ім’я персонажа з обмеженням в довжині;

– lvl: рівень персонажа, значення за замовчуванням – 1 та область значень для змінної від 1 до 20 включно;

– experience: очки досвіду персонажа, значення за замовчуванням – 1. На основі цього значення піднімається рівень;

– race: посилання на об’єкт, який зазначає расову приналежність персонажа;

– characterClass: посилання на об’єкт, який зазначає клас персонажа;

– armorClass: рівень броні персонажа, на основі якого визначається чи атака попала по ньому та чи пробила броню;

– addictionalSpeed: додаткова швидкість персонажа, яка є додатком до швидкості, зазначено в расі персонажа, значення за замовчуванням 0 та може мати значення як і мінусові, так і додатні;

– maxHitPoints: максимальна кількість очків здоров’я персонажа, не можу бути від’ємним і значення за замовчуванням 0;

– currentHitPoints: поточні очки здоров’я персонажа, значення за замовчуванням рівне максимальній кількості очок здоров’я персонажа;

– spells: список посилань на заклинання, які знає персонаж.

Також варто зазначити, що система має багато геттерів для отримання доступу до змінних. Крім того в багатьох класах зустрічаються методи save та __str__. Ці функції належать до класу, від якого походять всі моделі в Django Django.Models.model. __str__ повертає зазначене значення при отримані доступу до об’єкта через ORM систему. Save в свою чергу відповідає за кастомне збереження об’єктів в системі.

Також при проектуванні розроблюваного ПЗ було використано патерни для кращого результату. Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми [13].

Саме для цього застосунку було використано патерн “Декоратор”. Декоратор – це структурний патерн проектування, який надає змогу динамічно додавати об’єктам новий функціонал, загортаючи їх у “оболонки”. Таке пояснення обумовлюється тим, що цільовий об’єкт розміщується у іншому об’єкті-обгортці,

який надає базу поведінку об’єктові, а потім додає до результату щось нове. Обидва об’єкти мають загальний інтерфейс, тому для користувача немає різниці з яким працювати, з чистим чи загорнутим об’єктом [14].

При проєктуванні систему патерн Декоратор застосований для реалізації видів предметів. InventoryItem виступає, як базовий компонент патерну “Декоратор”. Цей клас описує загальні характеристики будь-якого предмета інвертаря. На разі є тільки одне покриття для цього класу – Weapon. В майбутньому можна буде добавити інші види предметів, не порушуючи існуючу структуру, в чому і є ціль цього структурного патерну. Клас Weapon виконує роль декоратора, який додає йому нові атрибути, не змінюючи базовий клас безпосередньо. Він зберігає посилання на об’єкт InventoryItem. Сам спроектований патерн зображений на рисунку 2.5.

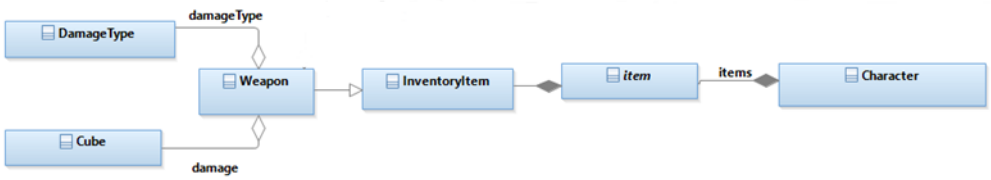


Рисунок 2.5 – Використання патерну “Декоратор”

Застування цього патерну забезпечує гнучкість структури та відкритість до розширення, оскільки можна створювати нові типи предметів, реалізовані як окремі декоратори.

2.3 Моделювання діаграми послідовності

У процесі проектування ПЗ важливо не лише визначити її структуру, а й описати взаємодію між об’єктами. Для цього використовуються діаграма послідовності, які відображають порядок виклику методів та передачі повідомлень у межах певного сценарію.

Діаграма послідовності використовується в основному для того, щоб показати взаємодію між об'єктами в послідовному порядку, в якому вони відбуваються. Вони є корисними для комунікації того, як взаємодіють різні бізнес-об'єкти. Окрім того діаграма послідовності на бізнес-рівні може бути використана як документ з вимогами для передачі вимог до майбутньої реалізації системи [16].

При моделюванні розроблюваної системи було створено діаграму послідовності для створення персонажа. Вона показує покроковий сценарій створення нового листа персонажа з демонстрацією взаємодії між різними частинами застосунка. Саму діаграму зображено на рисунку 2.6.

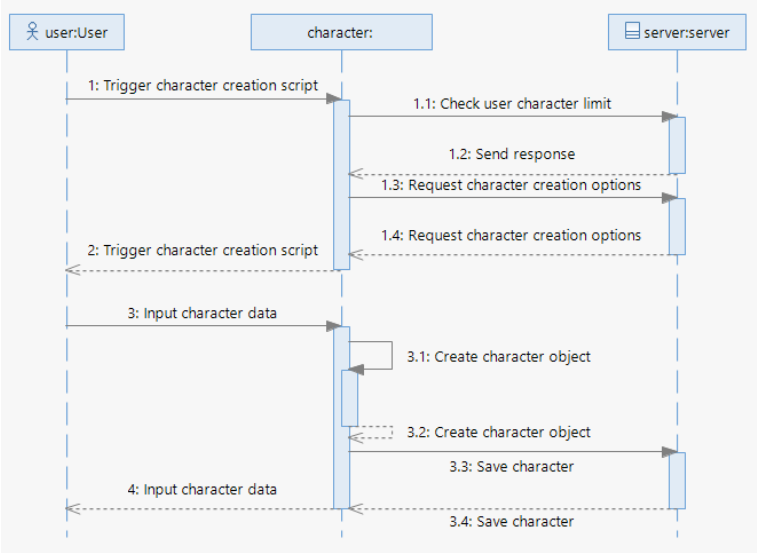


Рисунок 2.6 –Діаграма послідовності створення листа персонажа

На діаграмі послідовності зображено процес створення листа персонажа у веб-застосунку. Взаємодія відбувається між трьома основними об’єктами: User, Character і Server.

Послідовність дій зображена на діаграмі:

– Користувач надсилає запит за створення персонажа в модуль Character;

– Модуль Character перевіряє чи не перевищено ліміт кількості листів персонажа користувачем надсилаючи відповідний запит в модуль Server;

– Модуль Character відправляє відповідь, якщо вона позитивна, то відправляє ще один запит в Server на отримання об’єктів з БД;

– Модуль Server надсилає необхідні данні;

– Користувач заповнює поля для створення базового листа персонажа;

– Модуль Character створює відповідний об’єкт та зберігає його в БД.

В результаті цією послідовності створюється новий лист персонажа, який прив’язаний до відповідного користувача.

В результаті проведеного моделювання було створено комплексне уявлення про структуру система. Діаграми ВВ дозволили визначити акторів і їх взаємодію, діаграми класів – структуру і зв’язки між об’єктами, а діаграми послідовності – логіку виконання основних сценаріїв. Також було побудовано діаграму бази даних, яка слугує основою для ефективного зберігання інформації про персонажів.

РОЗДІЛ 3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ МЕНЕДЖМЕНТУ ПЕРСОНАЖІВ D&D

3.1 Ініціалізація проєкту та його налаштування

Щоб почати розробляти веб-застосунок потрібно спершу його ініціалізувати. На роль IDE було обрано Visual Studio Code з плагінами для роботи з мовою програмування Python. Перед початком розробки також потрібно створити віртуальне середовище python та встановити Django бібліотеку. Так як реалізація спроєктованого проєкту відбувається на Django, цей процес швидкий і простий. Щоб ініціалізувати проєкт в фреймворку можна використати команду startproject. В результаті виконання її створюється початковий проєкт, який можна доповнювати. Сама команда має обов'язкові параметри: назва проєкту та назва головного додатку

В проєкті Django можна створити довільну кількість додатків, який є свого роду пакетом з функціоналом. Фреймворк має відповідну команду, щоб додати структурований додаток – startapp. Сама команда також має параметр – назва додатку.

У випадку розробки ПЗ для цієї кваліфікаційної роботи система складається з трьох додатків. Ці додатки це: apps, mainapp та DnDProject. На рисунку 3.1 зображено сформовану структуру проєкту.

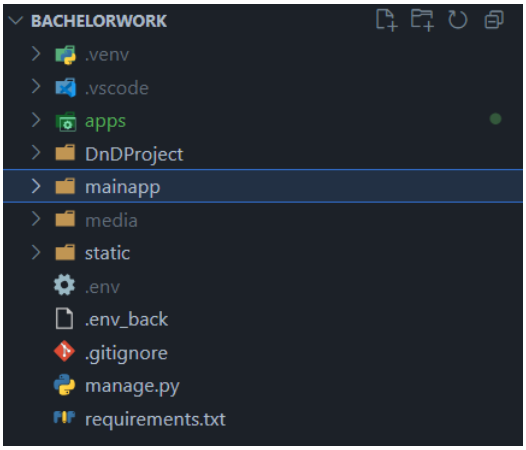


Рисунок 3.1 – Структура проєкту

Пояснення директорій та файлів в проєкту:

– .venv – Віртуальне середовище python;

– .vscode – Унікальна директорія для Visual Studio Code в якій зберігаються налаштування проєкту;

– apps – Додаток, в якому реалізовуються додатки для різних частин проєкту (у випадку розроблюваного проєкту – character), які формують основну логіку роботи веб-застосунку;

– DnDProject – основна директорія проєкту Django, в якому зберігаються налаштування та пов’язуються всі додатки;

– Mainapp – додаток, в якому реалізовані другорядні файли та функціонал;

– media – Папка з медіа файлами;

– .env – дані необхідні для підключення БД;

– manage.py – файл, з якого запускається застосунок;

– requirements.txt – Список бібліотек, необхідних для реалізації веб-застосунку.

Після формування всіх аспектів структури проєкту необхідно налаштувати розроблюваний застосунок в файлі DnDProject/settings.py. Для налаштування потрібно ввести всі додатки в масив INSTALLED_APPS та зазначити шлях до створених static та media директорій. Також було додано компресори для JS та SCSS.

Django Compressor – це потужний додаток Django, який об'єднує і стискає файли CSS і JavaScript. Він допомагає оптимізувати веб-продуктивність, зменшуючи кількість HTTP-запитів і мінімізуючи розмір файлів. Django Compressor легко інтегрується зі статичною системою обробки файлів Django і підтримує різні алгоритми стиснення та препроцесори [16].

3.2 Реалізація структурних моделей

Одним із ключових етапів розробки веб-застосунку є проєктування структури даних, що реалізується за допомогою моделей. У фреймворку Django моделі є об'єктно-реляційним відображенням (ORM) сутностей предметної області, які зберігаються в базі даних. Саме через моделі визначається логічна структура даних, їхні властивості та взаємозв’язки між ними. Створення моделей дозволяє розробнику зосередитись на логіці застосунку, не переймаючись деталями SQL-запитів, оскільки Django автоматично генерує відповідний код для роботи з базою даних. У цьому підрозділі розглянуто процес побудови моделей, які відображають основні об’єкти системи, їхні атрибути та зв’язки відповідно до вимог предметної області.

Моделі в проєкті побудованому з фреймворком Django описуються в окремому файлі models.py. Кожен додаток має такий файл. Самі ж моделі, створені розробником, успадковують структуру з вбудованого в Django класу Model. Для створення атрибутів моделей існує набір вбудованих в фреймворк типів даних, яких більш ніж достатньо для реалізації любої моделі. Крім того також є набір класів в класі Model, який успадковують створенні моделі. Ці класи можна перевизначити за допомогою поліморфізму. Найчастіше це Meta, який використовується для відображення моделі в адмін панелі.

Варто розглянути основну модель системи – character. Вона описує ігрового персонажа користувача, реалізована як клас Django-моделі та містить ключові поля, що охоплюють загальні параметри персонажа, бойові характеристики, класову та расову приналежність, а також пов’язані об’єкти, необхідні для повноцінного функціонування у грі. Модель має зовнішній ключ до моделі користувача (User), що дозволяє зберігати зв’язок між гравцем і його персонажами. Основні ігрові параметри включають ім’я персонажа, рівень, досвід, броню, швидкість та кількість очок здоров’я. Для визначення расових і класових особливостей використовуються зв’язки з моделями Race та CharacterClass, які у

свою чергу містять інформацію про здібності, бонуси та навички. Одним із атрибутів є поле spells, реалізоване як зв'язок «багато до багатьох» із моделлю Spell, що дозволяє зберігати перелік відомих персонажу заклинань. Метод getSpeed() обчислює фактичну швидкість пересування персонажа на основі расових даних і додаткових модифікаторів. На рисунку 3.2 зображено цієї реалізацію моделі.

```python
# Character
class Character(models.Model):
    user = models.ForeignKey('auth.User', null=True, on_delete=models.CASCADE, related_name='characters', verbose_name='User')
    name = models.CharField(max_length=100, default="new character", verbose_name='Character Name')
    lvl = models.IntegerField(default=1, verbose_name='Character Level')
    experience = models.IntegerField(default=0, verbose_name='Experience Points')
    race = models.ForeignKey(Race, null=True, on_delete=models.SET_NULL, related_name='characters', verbose_name='Race')
    characterClass = models.ForeignKey(CharacterClass, null=True, on_delete=models.SET_NULL, related_name='characters', verbose_name='Class')
    armorClass = models.IntegerField(default=10, verbose_name='Armor Class')
    additionalSpeed = models.IntegerField(default=0, verbose_name='Additional Speed')
    maxHitPoints = models.IntegerField(default=0, verbose_name='Maximum Hit Points')
    currentHitPoints = models.IntegerField(default=0, verbose_name='Current Hit Points')
    temporaryHitPoints = models.IntegerField(default=0, verbose_name='Temporary Hit Points')
    spells = models.ManyToManyField(Spell, blank=True, related_name='characters', verbose_name='Known Spells')

    def getSpeed(self):
        return self.race.speed+self.additionalSpeed


    class Meta:
        verbose_name = "Character"
        verbose_name_plural = "Characters"
```

Рисунок 3.2 – Програмна реалізація моделі Character

До моделі також прив'язані допоміжні сутності, що створюються автоматично при ініціалізації об'єкта. Завдяки механізму сигналів Django (post_save), після створення персонажа формуються пов'язані об'єкти: гаманець (Purse), блок персоналізації (Personalisation) та слоти для заклять (SpellSlot) для кожного рівня заклять від 0 до 9. Така автоматизація сприяє зручному розгортанню ігрової логіки без необхідності ручного створення цих об'єктів. На рисунку 3.3 зображено один з таких класів.

```python
class Purse(models.Model):
    character = models.OneToOneField(Character, on_delete=models.CASCADE, verbose_name="Related character", related_name="purse")
    copper = models.IntegerField(default=0, verbose_name='Copper Coins')
    silver = models.IntegerField(default=0, verbose_name='Silver Coins')
    gold = models.IntegerField(default=0, verbose_name='Gold Coins')
    electrum = models.IntegerField(default=0, verbose_name='Electrum Coins')
    platinum = models.IntegerField(default=0, verbose_name='Platinum Coins')

    class Meta:
        verbose_name = "Purse"
        verbose_name_plural = "Purses"
```

Рисунок 3.3 – Реалізація моделі Purse

Також варто зазначити, що в системі використовується вбудований в Django модель користувача. Її функціоналу достатньо для потреб застосунку та не має необхідності створювати власну модель користувача.

3.3 Створення дизайну та його реалізація

Візуальна складова веб-застосунку відіграє важливу роль у забезпеченні зручності взаємодії користувача з системою. На цьому етапі розробки основна увага була зосереджена на побудові інтерфейсу, який би поєднував функціональність із доступністю, а також відповідав загальній стилістиці рольової гри.

Процес створення дизайну охоплював розробку макетів основних сторінок, підбір кольорової палітри та шрифтів, а також компонування вмісту у логічні блоки. Основною метою було досягнення інтуїтивного розміщення елементів керування, забезпечення легкого доступу до основної інформації про персонажа та інтерактивних можливостей редагування даних. Дизайн було розроблено з використанням платформи figma. Також варто зазначити, що дизайн англійською мовою, бо figma конфліктує з використанням української розкладки з цим шрифтом, при реалізації веб-застосунку інтерфейс на українській мові.

Почнемо з головної сторінки, яка є вхідною точкою в систему. У верхній частині розташовано навігаційну панель, яка містить кнопки переходу до основних розділів — «Main» та «Characters». Справа відображається нікнейм користувача, кнопка виходу із системи, а також іконка профілю. Така структура дозволяє зручно перемикатися між основними функціями застосунку без перевантаження інтерфейсу.

Центральну частину сторінки поділено на два блоки: ліворуч розміщено секцію "News", де представлені новини або оновлення проєкту у вигляді окремих карток, а праворуч — блок "Characters", що містить список створених персонажів

користувача. Кожен блок візуально виділений темним фоном та світлим текстом, що сприяє кращому сприйняттю вмісту. На рисунку 3.4 зображено дизайн головних сегментів головної сторінки. Переглянути повну версію дизайну цієї сторінки можна в ДОДАТОК Г.

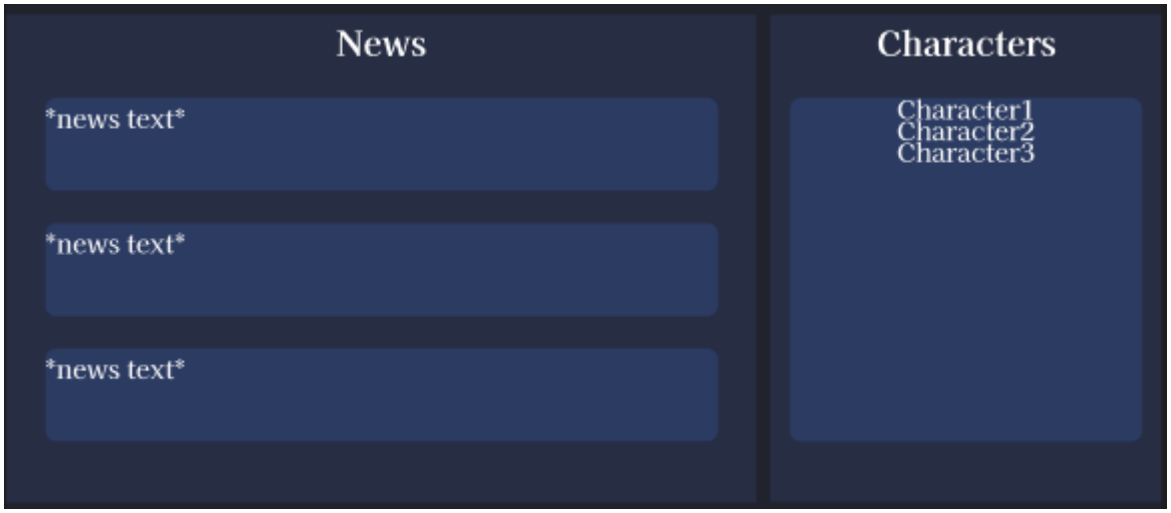


Рисунок 3.4 – дизайн головної сторінки

Також є сторінка зі списком створених листів авторизованим користувачем. Сама сторінка містить елемент з списком попередньо створених персонажів. Який можна прокручувати, якщо він займає більше, ніж виділено на то місця. Крім того кожен елемент, який репрезентує персонажа має кнопку видалення персонажа та є загальна кнопка на його створення. На рисунку 3.5 зображено цей список. Переглянути повну версію дизайну цієї сторінки можна в ДОДАТОК Г.

Рисунок 3.5 – Список створених персонажів

Ну і звісно головна сторінка листа персонажа, яка є найкоплекснішою. Вона містить в собі детальну інформацію про створеного персонажа. Він витриманий у темно-синій гамі, що забезпечує зосередженість користувача на контенті та відповідає загальному фентезійному стилю застосунку.

У верхній частині знаходиться інформаційний блок із базовими даними про персонажа: ім’я, раса, клас (разом із підкласом), а також рівень. Поруч виводяться основні бойові характеристики — броня, швидкість, бонус до майстерності (proficiency), а також кількість життів (HP). Такий блок забезпечує швидкий доступ до ключових параметрів персонажа під час гри. На рисунку 3.6 зображено цей інформаційний блок.

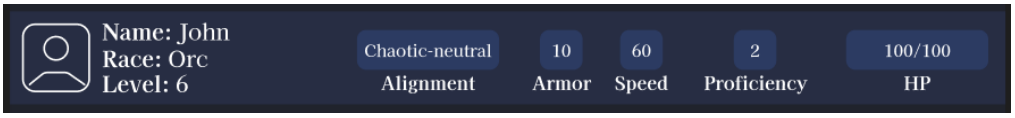


Рисунок 3.6 – Дизайн блоку з базовими даними про персонажа

Зліва розміщено панель характеристик (Strength, Dexterity, Intelligence, Wisdom, Charisma) з відповідними числовими значеннями та бонусами. Під кожною характеристикою наведено пов’язаний перелік навичок із модифікаторами. Це дозволяє гравцю легко орієнтуватися у сильних і слабких сторонах персонажа. На рисунку 3.7 зображено блок з характеристиками і навичками персонажа.

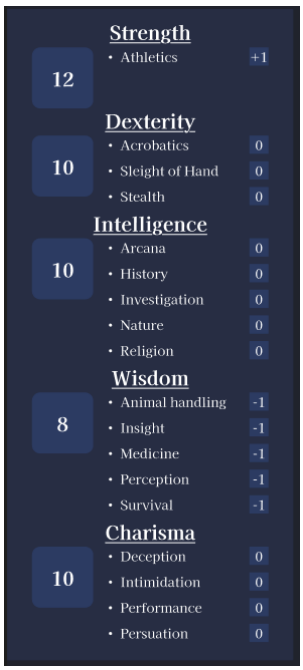


Рисунок 3.7 – Дизайн блоку з характеристиками і навичками персонажа.

Центральну частину займає набір вкладок, які перемикають між різними аспектами персонажа: Abilities, Inventory, Personality, Notes та Spells. На зображенні активною є вкладка Abilities, що містить дві основні секції — список атак і перелік здібностей. В залежності від обраної вкладки вміст під ними буде мінятись на відповідний. На рисунку 3.8 зображено одну з цих вкладок. Всі варіації цих вкладок можна переглянути в додатку Д.

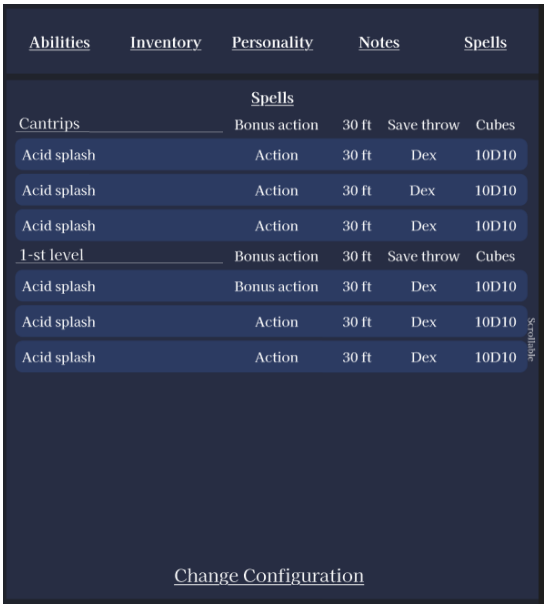


Рисунок 3.8 – Дизайн блоку з змінними вкладками

Загалом інтерфейс поєднує інформативність, логічну структуру та тематичне оформлення, сприяючи комфортному користуванню під час підготовки до гри або безпосередньо в її процесі. Повний дизайн цієї сторінки можна переглянути в ДОДАТОК Г.

Для реалізації графічного інтерфейсу було використано технології HTML, CSS (із застосуванням SCSS для структурованої стилізації) та шаблони Django. Перш за все було сформовано шаблон сторінки wrapper, який є прототипом для всіх сторінок в веб-застосунку. Він складається з різних блоків, які можна змінювати при успадкуванні, як при поліморфізмі. На рисунку 3.9 зображено один з таких блоків.

```
{% compress css %}
    {# Блок призначений для внутрішніх scss стилів, які потрібно вставити в head #}
    <link rel="stylesheet" href="{% static 'mainapp/scss/style.scss' %}" type="text/x-scss">
    {% block head_custom_styles %}

    {% endblock %}
{% endcompress %}
```

Рисунок 3.9 – Блок для підключення стилів сторінки

Структура head_custom_styles знаходиться в компресорі, який в свою чергу знаходиться в шапці сторінки. Через це вміст цього блоку буде знаходитись в шапці, наче їх прописали там. Все що треба зробити, щоб сторінка успадковувала цей шаблон необхідно на початку сторінки вказати, що вона розширює шаблонну сторінку, в нашому випадку wrapper.html. Також для зручності можна завантажити змінну з налаштувань, яка вказує на відносний шлях до статичної папки, в якій згідно правилам побудови проєкту зберігаються файли css та js. Щоб заповнити необіхдний блок потрібно повторити структуру, як описаний відповідний блок. Переглянути повні лістинги коду згаданих сторінок (шаблонної та головної) можна в ДОДАТОК Б. На рисунку 3.10 зображено приклад використання блоку head_custom_styles на головній сторінці.

```
{% extends 'wrapper.html' %}
{% load static %}
{% block head_custom_styles %}
    <link rel="stylesheet" href="{% static 'mainapp/scss/homepage.scss' %}" type="text/x-scss">
{% endblock %}
```

Рисунок 3.10 – Приклад заповнення блоку head_custom_styles

З використанням такого підходу процес створення сторінок сильно полегшується. Крім того за рахунок використання такого підходи сторінку та її параметри можна гнучко змінювати.

Також для створення сторінок було використано інструменти, які надає SCSS. Як було згадано раніше, він надає можливість успадковувати, включати, переписувати стилі, що є дуже корисним при розробці веб-застосунку. Щоб не змінювати численні стилі сторінок при необхідності введення змін до вигляду системи та не повторювати одні і ті самі стилі було створено файл з батьківськими стилями, які використовуються часто на сторінках. Стилі, які можуть бути успадковані та розширені мають перед назвою допис @mixin. Щоб успадкувати ж такий стиль, потрібно імпортувати цей файл в місце, де потрібно його використати та в класі, який перейме його опис має містити команду @include з назвою батьківського класу. Крім того SCSS має можливість оголошення змінних, в які можна записати необхідні значення та використовувати їх. Ці можливості в рази полегшують та покращують змінність стилів сторінок веб-застосунку при внесенні змін. Вміст файлу з батьківськими стилями та глобальними змінними надано в ДОДАТОК Б.

3.4 Реалізація бізнес логіки системи

У цьому підрозділі розглядається процес впровадження основних бізнес-правил та логіки, що лежать в основі функціонування системи. Реалізація бізнес-

логіки є ключовим етапом розробки, оскільки вона забезпечує відповідність роботи системи вимогам бізнес-процесів та сприяє досягненню поставлених цілей.

Спершу було сформовано низку url посилань в файлі urls.py. В ньому знаходиться список посилань веб-застосунку та відповідні views функції, які виконуються, якщо користувач переходить по посиланню. Цей список передається в файл urls.py основного пакету системи.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('mainapp.urls')),
    path('', include('apps.character.urls')),
    re_path(r'^media/(?P<path>.*)$', serve, {'document_root': settings.MEDIA_ROOT}),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Рисунок 3.11 – Вміст файлу urls.py для додатку character

Після цього створено views для пов’язування БД та веб-сторінки. В них за допомогою ORM від Django система отримує дані з БД у вигляді об’єктів. За потреби також з цих об’єктів формують комбінації різних об’єктів для зручності використання. Після цього сторінка рендериться функцією render, в яку передаються три параметри: request, url посилання на очікувану сторінку та контекст, який містить всі змінні, які використовуватимуться на сторінці. Після цього сторінка завантажується та користувач попадає на неї.

Також система потребувала низку скриптів для створення, редагування та видалення параметрів персонажа з використанням користувацького графічного інтерфейсу. Для цього було використано мову програмування JS. Проте перед тим як їх писати важливо, щоб елементи сторінки, які будуть містити якийсь функціонал та використовувати який з описаних скриптів повинні мати вказане ID. Також це ID має бути унікальне для кожного елемента, якщо він є частиною списку таких самих елементів. При розробці ПЗ для таких елементів вказано ID параметр в розмітці сторінки в форматі *назва елементу*–*id об’єкта в БД*. Якщо ж потрібна більш специфічне визначення, то друга частина замінювалось на необхідне слово.

Більшість скриптів була побудована з використанням слухача івентів прив’язаного до окремого елемента. Для цього спочатку потрібно знайти

відповідний елемент за ID аргументом з використанням методу getElementById або за назвою класу з використанням методу querySelector. Після цього до знайденого об’єкта елемента додавався слухач подій з вказаним типом події. У випадку розробки цієї системи використовувався тільки input через специфіку використання скриптів в веб-застосунку.

Після додання слухача подій найчастіше витягувались необідні дані з найденого елементу, такі як його назва, id, значення чи інші. З цих даних формувався HTTP запит в якому зазначались такі параметри:

– URL посилання, яке виконається в системі;
– Тип методу (POST, DELETE, UPDATE);
– Геттери;
– Тіло, яке містить змінні, які передаються в систему;

Також якщо запит має відповідь, то обробляться опісля. На рисунку 3.12 зображено один з таких скриптів.

```
let timer;
const character = document.getElementById("page-info");

// Event listener for characteristic fields
document.querySelectorAll(".characteristic-value").forEach(characteristic => {
    characteristic.addEventListener('input', function (){
        // Clears any outstanding timer
    clearTimeout(timer);
    // Sets new timer that may or may not get cleared
    timer = setTimeout(() => {
        const name = this.id;
        const value = this.value;
        fetch("save/characteristic/",{
            method: "POST",
            headers: {
                'Content-Type': 'application/json',
                'X-CSRFToken': getCookie('csrftoken'),
            },
            body: JSON.stringify({
                name: name,
                value: value,
            })
        }).then(response => response.json());
    }, 2000);
    });
});
```

Рисунок 3.12 – Скрипт для зміни характеристик персонажа

Щодо заповнення даних, то для цього процесу було використано два способи. Перший – це використання форм для заповнення та їх подання кнопкою “submit”. Такий спосіб використовувався при створенні нових об’єктів зі сторони

користувача. У таких випадках відкриється спливаюче вікно в якому знаходяться обов’язкові поля для заповнення. Після введення даних користувач підтверджує форму, яка вже попадає в систему. Крім того варто зауважити, що було використано не звичайні форми, а ті, що надає Django. Його перевага в тому, що він створений для взаємодії з Django, з його функціями, класами та об’єктами. На рисунку 3.15 зображено одну з таких форм.

```
class CharacterCreationForm(forms.Form):
    name = forms.CharField(label="Character name", widget=forms.TextInput(attrs={'class': 'text-input'}), max_length=15)
    race = forms.ModelChoiceField(label="Race", queryset=Race.objects.all(), widget=forms.Select(attrs={'class': 'select-input'}))
    character_class = forms.ModelChoiceField(label="Class", queryset=CharacterClass.objects.all(), widget=forms.Select(attrs={'class': 'select-input
    strengthValue = forms.IntegerField(label="Strengh", widget=forms.NumberInput(attrs={'class': 'number-input'}))
    dexterityValue = forms.IntegerField(label="Dexterity", widget=forms.NumberInput(attrs={'class': 'number-input'}))
    constitutionValue = forms.IntegerField(label="Constitution", widget=forms.NumberInput(attrs={'class': 'number-input'}))
    intelligenceValue = forms.IntegerField(label="Intelligence", widget=forms.NumberInput(attrs={'class': 'number-input'}))
    wisdomValue = forms.IntegerField(label="Wisdom", widget=forms.NumberInput(attrs={'class': 'number-input'}))
    charismaValue = forms.IntegerField(label="Charisma", widget=forms.NumberInput(attrs={'class': 'number-input'}))
```

Рисунок 3.13 – Форма в форматі Django для створення нового листа персонажа

В результаті використання вище згаданих технік, метедологій та інструментів, розробка відбувалась у швидкому та легкому темпі. Крім того він виглядає організовано, структуровано та зрозуміло.

3.5 Тестування розробленої системи

Тестування програмного забезпечення є невід'ємною частиною життєвого циклу розробки програмних систем. Воно спрямоване на виявлення помилок, оцінку якості продукту та забезпечення його відповідності функціональним і нефункціональним вимогам. Ефективне тестування дозволяє знизити ризики, пов'язані з експлуатацією ПЗ, підвищити надійність і задоволення користувача, а також скоротити витрати на виправлення дефектів на пізніх етапах розробки. Під час виконання кваліфікаційної роботи було проведено низку тестувань, а саме ручне та автоматичне тестування.

Ручне тестування проводилось для тестування процесу реєстрації та авторизації в системі, так як це важлива частина любого веб-застосунку. Також при ручному тестуванню перевірялась роботу полів, які користувач може змінювати без відправляння форми. Такі операції відбуваються при зміні вже існуючих об’єктів з використанням користувацького графічного інтерфейсу. Такий вибір обґрунтований тим, що цей функціонал не має високої варіативності та не часто змінний.

Також було створено низку юніт-тестів, які перевіряють валідацію даних в системі та роботу деяких функцій. Таке тестування забезпечить коректну роботу таких аспектів коду та дозволить при бекенд розробці перевіряти чи всі необхідні функції та перевірки працюють правильно та чи не виникає ніяких помилок при змінні програмної частини системи.

Щодо автоматичного тестування, то його було проведено на функціоналі створення, видалення та редагування об’єктів. Вибір обґрунтований тим, що система має дуже багато простору для розширення. Через це доцільно буде мати автоматичні тести, які можна буде запустити при кожній змінні та перевірити чи зміни не перешкоджають роботі функціоналу веб-застосунку. Таким чином буде забезпечено захист від випадків, коли через зміну однієї частини системи випливає помилка в іншій частині не поміченою.

На автоматичне тестування було виділено найбільше уваги через те, що вони доступні в наступних ітераціях. Самі класи тестів в selenium мають два типи: LiveServerTestCase та StaticLiveServerTestCase. З них двох при проведені тестування використовувався StaticLiveServerTestCase. Причина в тому, що LiveServerTestCase не завантажує статичні файли, наприклад JS скрипти, через що велика частина функціоналу не працює.

Сама ж структура тестових класів складається з:

– Метод setup: Метод, який налаштовує середовище перед виконанням кожного тесту;

– Метод teardown: Метод, який визначає поведінку середовища після виконання тесту;

– Тести, описані тестувальником.

При формуванні тестів було створено setUp метод, який надає всі необхідні аспекти для проходження тестів: створення сталих даних в БД з фікстур (світогляд, характеристики, куби та навички), створення об’єкта користувача від лиця якого будуть виконуватись тести та створення об’єктів, яких при звичайні роботі системи створює адміністратор (клас та раса персонажа). Також дуже важливо визначити браузерний драйвер, на якому працюватиме середовище. В його ролі було обрано ChromeDriver. Цей вибір обґрунтований тим, що найширше використовуваний браузер – Google Chrome, так як при обмежених ресурсах для тестування ПЗ необхідно забезпечити тестування для середовища, яке найширше використовується користувачами. На рисунку 3.14 зображено setUp для всіх описаних тестів.

```python
class TestCharacterList(StaticLiveServerTestCase):
    def setUp(self):
        # створення тестового користувача
        self.user = User.objects.create_user(username='testuser', password='password123')
        self.user.is_active = True
        self.user.save()

        self.user = User.objects.all()

        call_command('loaddata', 'load_alignment')
        call_command('loaddata', 'load_characteristics')
        call_command('loaddata', 'load_cubes')
        call_command('loaddata', 'load_skill')

        self.charClass = CharacterClass.objects.create(name="Воїн")
        self.race = Race.objects.create(name="Людина")

        self.charClass.save()
        self.race.save()

        self.charClass = CharacterClass.objects.create(name="Чаклун")
        self.race = Race.objects.create(name="Орк")

        self.charClass.save()
        self.race.save()

        self.browser = webdriver.Chrome()
```

Рисунок 3.14 – Налаштування середовища тестування setUp

Щодо написання самих тестів, то було протестовано весь основний функціонал, який доступний звичайному користувачу. Якщо узагальнити, то ці тести перевіряють сценарії створення, редагування та видалення об’єктів. Сам процес складається з пошуку об’єкта в розмітці сторінки та взаємодія з ним (натиснути, обрати, написати). Після цього результати пройденого сценарію

порівнюються з очікуваним результатом, виділяючи частинки сторінки, які вказують на те, що тест пройшов успішно, наприклад по назві сторінки. Крім того дуже важливо, щоб користувач перед виконанням цих тестів був авторизований, так як більшість функціоналу не доступна неавторизованим користувачам. Через це на початку кожного сценарію виконується авторизація тестового користувача в системі.

Під час тестування було перевірено ключовий функціонал системи, включаючи процеси реєстрації, авторизації, створення та редагування даних. Застосовано як ручні, так і автоматизовані методи тестування, що дозволило оцінити стабільність і надійність роботи інтерфейсу. Проведені перевірки підтвердили відповідність поведінки системи очікуваним результатам, а також допомогли виявити і виправити окремі недоліки. Загалом тестування підтвердило готовність системи до подальшого використання.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

У сучасних умовах розвитку технологій, а разом з тим і зростання технологічної складності процесів та інтенсифікації праці особливо актуальності набирають питання забезпечення безпеки життєдіяльності та охорони праці. Звісно ця сфера важлива для любого роду праці, включно з розробкою ПЗ. Забезпечення гідного рівня захисту життя і здоров’я людини в процесі її професійної діяльності є одним з ключових чинників сталого розвитку суспільства та ефективної роботи підприємства.

Безпека життєдіяльності — одна із наймолодших наук, яка спрямована на вирішення основної проблеми сучасного суспільства України — збереження здоров'я населення. За змістом дисципліна «Безпека життєдіяльності» формує світогляд майбутнього фахівця, який у своїй повсякденній праці повинен будувати передумови запобігання нещасним випадкам, захворюванням та усувати негативний вплив шкідливостей на здоров'я людини в умовах виконання виробничих завдань, її існування в життєвому середовищі та в надзвичайних ситуаціях [20].

4.1 Шляхи покращення життєдіяльності людини.

Покращення життєдіяльності людини є важливою складовою формування безпечного, здорового та продуктивного суспільства. У контексті сучасних викликів – зростання техногенного навантаження, погіршення екологічного стану довкілля, підвищення рівня стресу та зниження фізичної активності –необхідність розробки і впровадження ефективних шляхів забезпечення гармонійних умов існування людини набуває особливої актуальності.

Життєдіяльність - діяльність особи, що передбачає її повсякденне функціонування у спосіб та в обсязі, звичайних для неї, зокрема шляхом самообслуговування, орієнтації, пересування, спілкування, контролю за своєю поведінкою, навчання, трудової діяльності [19].

Початок нового століття надзвичайно загострив проблему забезпечення оптимальної життєдіяльності кожної людини та суспільства в цілому. Техногенні, природні, медико-біологічні, соціальні, екологічні, інформаційні, військові надзвичайні ситуації, тероризм - формують той ступінь критичного ризику, що визначає середовище перебування людини як потенційно небезпечне [21].

Щоб запобігти цьому важливо покращувати якість життєдіяльності людини. Є багато способів досягнення цього, які охоплюють всі аспекти життя людей.

Шляхи покращення життєдіяльності людини:

1. Оптимізація середовища існування людини:
   - встановлення регіональної безпеки, яка вміщує функцію раннього попередження негативних тенденцій та передбачає гарантії їх мінімізації;
   - розміщення і розвиток матеріального виробництва на певній території повинні здійснюватися відповідно до її еколого економічної збалансованості;
   - забезпечення науковими й управлінськими кадрами за визначеним рівнем професіоналізму та компетенції;
   - формування довгострокової єдиної державної політики у сфері забезпечення безпеки, освіти та ін. [21].

2. Забезпечення комфортного майбутнього та збільшення впевненості в ньому формуванням стабільної економіки, цілісності державних кордонів та дотримання суспільних цінностей [21];

3. Формування особистої безпеки кожної людини;

4. Оптимізація способу життя на основі суб’єктивних чинників та оптимізація складових “формули” здоров’я:

- Режим праці та відпочинку;
- Харчування;
- Рухова активність;
- Емоційно-психічний стан;
- Загартування;
- Доброзичливість, милосердя, гумор [22];

5. Забезпечення гідних умов виробництва та побуту [22];

6. Розвиток первинної медичної допомоги і профілактики

Покращення життєдіяльності людини передбачає створення умов, які сприяють фізичному, психічному та соціальному добробуту. Це досягається через гармонізацію взаємодії людини з природним, виробничим і побутовим середовищем, мінімізацію дії шкідливих і небезпечних чинників, розвиток інфраструктури, орієнтованої на безпеку та комфорт, а також формування здорового способу життя. Важливою умовою є також просвітницька діяльність і системне навчання з питань безпеки, що сприяє усвідомленому ставленню до власного здоров’я, праці й довкілля. Таким чином, життєдіяльність покращується не лише шляхом технічних і організаційних змін, а й через формування культури безпеки та особистої відповідальності кожного за своє життя і здоров’я.

4.2 Вимоги до профілактичних медичних оглядів для працівників ПК

Метою проведення обов'язкових профілактичних медичних оглядів є запобігання розповсюдженню інфекційних та небезпечних захворювань, динамічне спостереження за станом здоров'я працюючого населення [23].

Роботодавець зобов'язаний за свої кошти забезпечити фінансування та організувати проведення попереднього (під час прийняття на роботу) і періодичних (протягом трудової діяльності) медичних оглядів працівників, зайнятих на важких роботах, роботах із шкідливими чи небезпечними умовами праці або таких, де є

потреба у професійному доборі, щорічного обов'язкового медичного огляду осіб віком до 21 року. За результатами періодичних медичних оглядів у разі потреби роботодавець повинен забезпечити проведення відповідних оздоровчих заходів. Медичні огляди проводяться відповідними закладами охорони здоров'я, працівники яких несуть відповідальність згідно із законодавством за відповідність медичного висновку фактичному стану здоров'я працівника. Порядок проведення медичних оглядів визначається центральним органом виконавчої влади, що забезпечує формування державної політики у сфері охорони здоров'я [24].

При проведені медогляду обов’язково повинен брати участь терапевт. Періодичність проведення медичного огляду базується на шкідливих та небезпечних факторах, з якими зустрічається працівник. Працівники ПК підлягають під фактор “Зорово-напружені роботи, що пов’язані з безперервним стеженням за екраном відеотерміналів (дисплеїв)” під номером 6.2.3. Для нього є дві характеристики: менше 4 годин (за 8-годинну зміну) та більше 4 годин (за 8-годинну зміну) [24].

Зорово-напружені роботи, що пов’язані з безперервним стеженням за екраном відеотерміналів (дисплеїв) менше 4 годин (за 8-годинну зміну) мають такі вимоги [24]:

– Періодичність оглядів у закладі охорони здоров’я: 1 раз на рік;

– Фах лікарів, що беруть участь у медичних оглядах: офтальмолог, невропатолог;

– Медичні протипоказання (на доповнення до загальних медичних протипоказань):

1. Гострота зору не менше 0,5 Д на одному оці та 0,2 на другому при попередньому профогляді; не менше 0,4 на одному оці та 0,2 на другому оці при повторних періодичних медоглядах;
2. Аномалії рефракції: при попередньому медогляді - міопія не більше 8,0 Д, гіперметропія не більше 8,0 Д, астигматизм не більше 3,0 Д; при попередньому медогляді: міопія не більше 8,0

Д, астигматизм не більше 4,0 Д при повторному періодичному медогляді;

3. Зниження акомодації нижче вікових норм;
4. Порушення кольоропочуття, якщо колір несе інформаційне навантаження;
5. Лагофтальм;
6. Хронічні запальні або алергічні захворювання захисного апарата та оболонок очного яблука;
7. Захворювання зорового нерва, сітківки;
8. Наростаючий офтальмотонус;
9. Глаукома;
10. Епілепсія та синкопальні стани.

Зорово-напружені роботи, що пов’язані з безперервним стеженням за екраном відеотерміналів (дисплеїв) більше 4 годин (за 8-годинну зміну) мають такі вимоги [24]:

– Періодичність оглядів у закладі охорони здоров’я: 1 раз на рік;

– Фах лікарів, що беруть участь у медичних оглядах: офтальмолог, невропатолог;

– Медичні протипоказання (на доповнення до загальних медичних протипоказань):

1. Гострота зору не менше 0,5 Д на одному оці та 0,2 на другому при попередньому профогляді; не менше 0,4 на одному оці та 0,2 на другому оці при повторних періодичних медоглядах;
2. Аномалії рефракції: при попередньому медогляді - міопія не більше 8,0 Д, гіперметропія не більше 8,0 Д, астигматизм не більше 3,0 Д; при попередньому медогляді: міопія не більше 8,0 Д, астигматизм не більше 4,0 Д при повторному періодичному медогляді;
3. Зниження акомодації нижче вікових норм;

4. Порушення кольоропочуття, якщо колір несе інформаційне навантаження;
5. Лагофтальм;
6. Хронічні запальні або алергічні захворювання захисного апарата та оболонок очного яблука;
7. Захворювання зорового нерва, сітківки;
8. Наростаючий офтальмотонус;
9. Глаукома;
10. Епілепсія та синкопальні стани.

Отже, профілактичні медичні огляди для працівників, які працюють з персональними комп'ютерами, є обов'язковими та регламентуються чинним законодавством з метою раннього виявлення професійно зумовлених захворювань, оцінки працездатності та запобігання їхньому прогресуванню. Працівники ПК підпадають під категорію «зорово-напружених робіт» та повинні проходити періодичні медичні огляди щороку з обов'язковою участю офтальмолога і невропатолога. Залежно від тривалості роботи за екраном (менше або більше 4 годин за зміну), визначено конкретні медичні протипоказання, серед яких порушення гостроти зору, аномалії рефракції, зниження акомодації, патології очей і нервової системи. Своєчасне проходження таких оглядів сприяє збереженню здоров'я працівників та підвищенню безпеки й ефективності праці.

ВИСНОВКИ

В результаті виконання цього розділу було здійснено аналіз предметної області, вивчено існуючі аналоги веб-застосунків для менеджменту персонажів Dungeons & Dragons, а також визначено їхні переваги та недоліки. Проведено аналіз потреб цільової аудиторії та сформульовано функціональні й нефункціональні вимоги до майбутньої системи. Визначено архітектурний підхід, методологію розробки та обрано відповідні технології. Це заклало теоретичну та практичну основу для ефективного проектування і реалізації веб-застосунку, орієнтованого на україномовну ігрову спільноту.

Після проведеного моделювання було створено комплексне уявлення про структуру система. Діаграми ВВ дозволили визначити акторів і їх взаємодію, діаграми класів – структуру і зв’язки між об’єктами, а діаграми послідовності – логіку виконання основних сценаріїв. Також було побудовано діаграму бази даних, яка слугує основою для ефективного зберігання інформації про персонажів.

Під час тестування було перевірено ключовий функціонал системи, включаючи процеси реєстрації, авторизації, створення та редагування даних. Застосовано як ручні, так і автоматизовані методи тестування, що дозволило оцінити стабільність і надійність роботи інтерфейсу. Проведені перевірки підтвердили відповідність поведінки системи очікуваним результатам, а також допомогли виявити і виправити окремі недоліки. Загалом тестування підтвердило готовність системи до подальшого використання.

У результаті роботи було створено веб-застосунок для управління персонажами рольової гри Dungeons & Dragons. Система дозволяє користувачам створювати, переглядати та редагувати персонажів із збереженням ключових характеристик, рас, класів та інших параметрів. Інтерфейс адаптований для зручної взаємодії, а зберігання даних організоване через базу даних із чіткою структурою. Застосунок орієнтований на україномовних користувачів та може використовуватись як персональний інструмент для гравців і майстрів гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) What is D&D Beyond [Електронний ресурс] – Режим доступу до ресурсу: https://www.dndbeyond.com/resources/1793-what-is-d-d-beyond;

2) Roll20 Crash Course [Електронний ресурс] – Режим доступу до ресурсу: https://help.roll20.net/hc/en-us/articles/360039223834-Roll20-Crash-Course;

3) Процес аналізу вимог [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=98816;

4) Вимоги до програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/5130988/page:2/;

5) Поняття вимоги. Класифікація вимог [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=98814;

6) Архітектурно-центрований процес розробки ПЗ [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=213462;

7) Що таке Agile [Електронний ресурс] – Режим доступу до ресурсу: https://brainrain.com.ua/uk/chto-takoe-agile-ua/;

8) Sass: documentation [Електронний ресурс] – Режим доступу до ресурсу: https://sass-lang.com/documentation/;

9) Learn Django [Електронний ресурс] – Режим доступу до ресурсу: https://learndjango.com/courses/django-for-beginners/introduction/;

10) What is JavaScript? [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/docs/Learn_web_development/Core/What_is_JavaScript/;

11) Процес ППЗ, керований варіантами використання [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=213464;

12) М.Р. Петрик, Ф.Я. Мудрик Архітектура програмного забезпечення (на базі використання CASE-засобів IBM(sad)) навчальний посібник, Тернопіль: ТНТУ імені Івана Пулюя, 2017. 100с;

13) Що таке патерн [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/what-is-pattern;

14) Декоратор [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/decorator;

15) The sequence diagram [Електронний ресурс] – Режим доступу до ресурсу: https://developer.ibm.com/articles/the-sequence-diagram/;

16) Django-compressor [Електронний ресурс] – Режим доступу до ресурсу: https://best-of-web.builder.io/library/django-compressor/django-compressor;

17) Методичні вказівки до виконання дипломної роботи освітнього рівня – бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

18) Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329;

19) Про охорону здоров’я: Закон України від 19 листопада 1992 р. №2801-XII [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2801-12 – Дата звернення: 05.06.2025;

20) Потенційна безпека життєдіяльності людини. Основні поняття. [Електронний ресурс] – Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=299141;

21) Скобло Ю. С, Соколовська Т. Б., Мазоренко Д. І., Б 40 Тіщенко Л. М., Троянов М. М. Безпека життєдіяльності: Навчальний посібник для вищих навчальних закладів III— IV рівнів акредитації. — Київ: Кондор, 2003. — 424 с

22) Безпека життєдіяльності: навчальний посібник / О. В. Березюк, М. С. Лемешев. – Вінниця: ВНТУ, 2011. – 204 с;

23) Про затвердження Порядку проведення медичних оглядів працівників певних категорій: Наказ Міністерства охорони здоров’я України від 21 травня 2007 р. № 246, [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0639-02 – Дата звернення: 05.06.2025.

24) Про охорону праці: Закон України від 14 жовтня 1992 р. № 2694-XII [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12 – Дата звернення: 06.06.2025;

ДОДАТКИ

ДОДАТОК А. Тези конференції

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

УДК 004.4
Гук В.–ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

СТВОРЕННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ КЕРУВАННЯ ПЕРСОНАЖАМИ РОЛЬОВОЇ ГРИ ЗА ДОПОМОГОЮ DJANGO REST FRAMEWORK

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

Huk V.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF AN INTERACTIVE ROLE-PLAYING GAME CHARACTER MANAGEMENT SYSTEM USING DJANGO REST FRAMEWORK

Supervisor: PhD Yurii Stoianov

Ключові слова: інженерія програмного забезпечення, Django, Дракони та підземелля, REST
Key words: software engineering, Djanho, Dungeons and dragons, REST

Dungeons & Dragons — це настільна рольова гра, що стала класикою жанру та основою для багатьох сучасних RPG-систем. У цій грі кожен гравець створює унікального персонажа, а ведучий (Майстер) керує сюжетом та світом гри [1]. В Україні ця форма дозвілля набуває все більшої популярності, однак через її відносну новизну на ринку відчувається дефіцит локалізованих і зручних цифрових інструментів для гри.

Особливо важливою частиною є лист персонажа, де зберігається вся інформація про героя: його особистість, здібності, навички, володіння зброєю, спорядження тощо [2]. Ведення такого листа на папері може бути незручним, ненадійним і обмеженим. До того ж, у гравців часто є кілька персонажів, що значно ускладнює їх менеджмент. Щоб вирішити цю проблему, я розробив веб-додаток, який спрощує створення, редагування та збереження персонажів у цифровому форматі. Цей проєкт — повноцінна система для керування персонажами рольової гри, яка охоплює всі ключові аспекти: характеристики, навички, магію, спорядження та інше.

У реалізації я використав Django Rest Framework — сучасний інструмент для створення API. Його гнучкість, масштабованість та активна екосистема дозволяють швидко створювати надійні бекенд-рішення, які легко інтегруються з будь-яким фронтендом [3]. Завдяки цьому система залишається зрозумілою, розширювальною та готовою до використання як у веб, так і в мобільних додатках.

Література:

1. Підземелля і дракони (dungeons and dragons). URL: https://yak.koshachek.com/articles/pidzemellja-i-drakoni-dungeons-dragons.html
2. Створення персонажа для Dungeons and Dragons. URL: https://dnd.in.ua/stvorennya-personazha-dlya-dungeons-and-dragons/
3. Django REST framework. Офіційна документація. URL: https://www.django-rest-framework.org.

ДОДАТОК Б. Лістинги кодів розмітки та глобальні стилі

Лістинг коду шаблону розмітки сторінок wrapper.html:

```
{% load static %}
{% load compress %}
{% spaceless %}
   <!DOCTYPE html>
   <html lang="uk">
   <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>{% spaceless %}{% block title %}{% endblock %}{% endspaceless %}</title>
      <meta name="description" content="{% spaceless %}{% block description %}{% endblock %}{%
endspaceless %}">

      {% compress css %}
         {# Блок призначений для внутрішніх scss стилів, які потрібно вставити в head #}
         <link rel="stylesheet" href="{% static 'mainapp/scss/style.scss' %}" type="text/x-scss">
         {% block head_custom_styles %}

         {% endblock %}
      {% endcompress %}

      {# Блок призначений для зовнішніх (cdn) css стилів, які потрібно вставити в head #}
      {% block head_external_styles %}
      {% endblock %}

      {% compress js %}
         {# Блок призначений для внутрішніх js скриптів, які потрібно вставити в head #}
         {% block head_custom_scripts %}
         {% endblock %}
      {% endcompress %}
      {# Блок призначений для зовнішніх (cdn) js скриптів, які потрібно вставити в head #}
      {% block head_external_scripts %}
      {% endblock %}
   </head>
   <body>
   <input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf_token }}">
   <header>
      {% include 'mainapp/header.html' %}
   </header>
   <main>
      {% block body %}
      {% endblock %}
   </main>
   <footer>
      {% include 'mainapp/footer.html' %}
   </footer>
   {% compress js %}
      <!-- <script src="{% static 'mainapp/js/script.js' %}"></script> -->
      {# Блок призначений для внутрішніх js скриптів, які потрібно вставити в body #}
      {% block extra_js %}
      {% endblock %}
   {% endcompress %}
```

```
    {# Блок призначений для зовнішніх (cdn) js скриптів, які потрібно вставити в body #}
    {% block external_extra_js %}
    {% endblock %}

    </body>
    </html>
{% endspaceless %}
```

Лістинг коду розмітки головної сторінки index.html:

```
{% extends 'wrapper.html' %}
{% load static %}
{% block head_custom_styles %}
    <link rel="stylesheet" href="{% static 'mainapp/scss/homepage.scss' %}" type="text/x-scss">
{% endblock %}
{% block body %}
    <img class="big-picture"src="media/images/landscape.jpg" alt="landscape" class="big-picture">
    <div class="main-content">
        <div class="section-news">
            <p class="section-title">News</p>
            <div class="scrollable-items">
                {% for item in recentNews %}
                    <div class="item">
                        <p>{{item.Title}}</p>
                        <p>{{item.Description}}</p>
                    </div>
                {% endfor %}
            </div>
        </div>
        <div class="section-characters">
            <p class="section-title">Characters</p>
            <div class="scrollable-items">
                {% for item in characters %}
                    <a class="item" href="character/{{item.id}}">
                        <p> {{item.name}} {{item.lvl}} level {{item.characterClass}}</p>
                    </a>
                {% endfor %}
            </div>
        </div>
    </div>
    <div class="section-about">
        <h1 class="section-title">About Project</h1>
        <p> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam non urna nec nisi facilisis tincidunt.</p>
    </div>
{% endblock %}
```

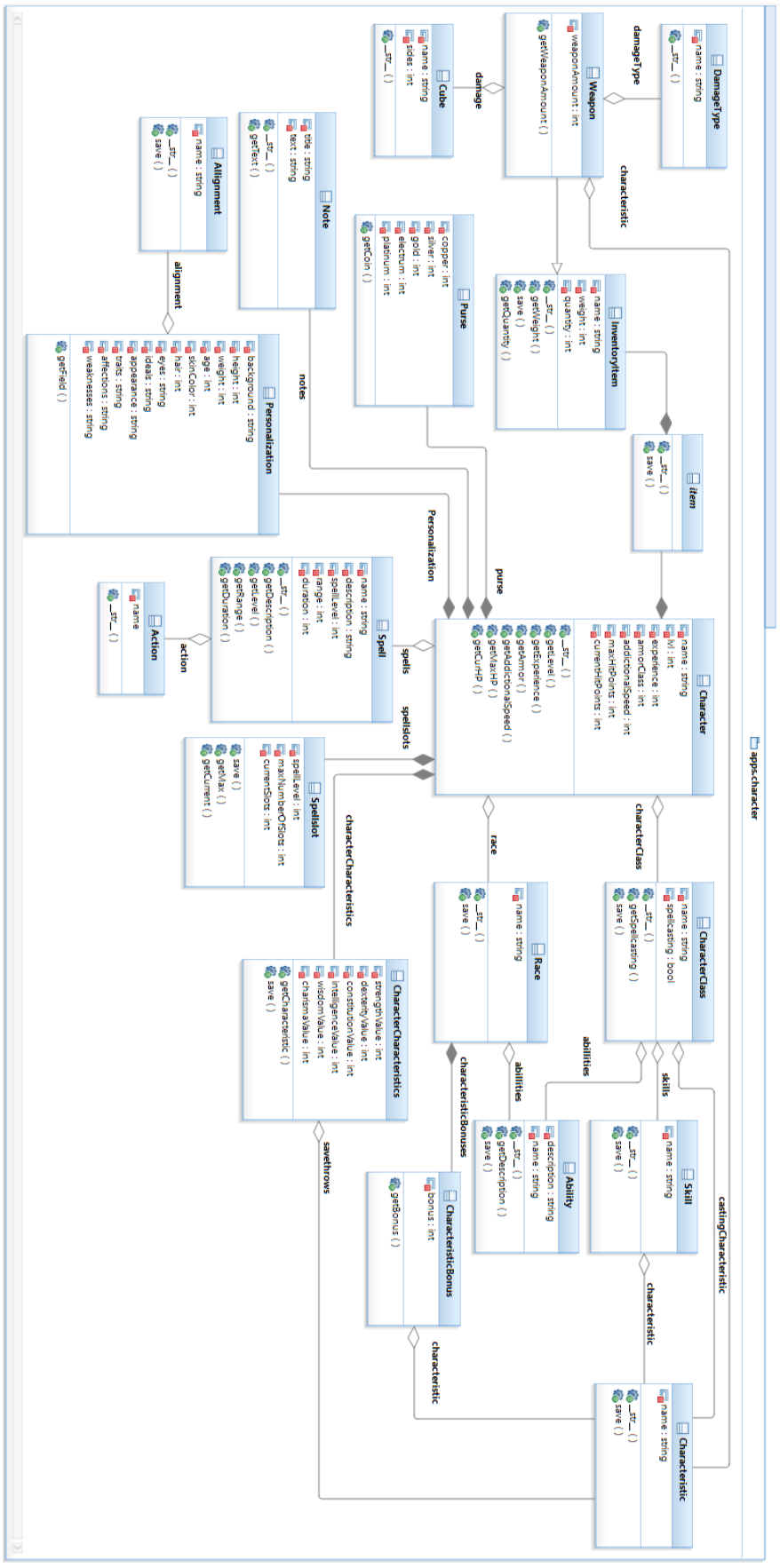
Лістинг коду глобальних стилів:

```
// Global functions
@font-face {
    font-family: KaiseiHarunoUmi;
    src: url(https://fonts.google.com/share?selection.family=Kaisei+HarunoUmi);
}

@mixin round-field{
    background-color: $field-color;
    font-size: $regular-text-size;
    background-color: $field-color;
    color: $font-color;
    border-color: $field-color;
    border-radius: 10px;
    padding: 10px 20px;
    text-align: center;
    height: 40px;
    font-weight: 300;
    box-sizing: content-box;
    font-family: $font;
    border-style: solid;
    border-width: 0;
}

@mixin section{
    background-color:$section-color;
    height: calc(100% - 40px);
    width: calc(100% - 40px);
    border-radius: 5px;
    border-collapse: separate;
    padding: 20px
}

@mixin section-title{
    text-align: center;
    text-decoration-line: underline;
    text-decoration-color: $font-color;
    text-decoration-thickness: 1px;
    font-size: 30px;
    font-weight: bold;
    top: 0;
    background-color: $section-color;
    margin: 0;
    padding: 0;
}

@mixin scrollable-items{
    display: flex;
    flex-direction: column;
    gap: 7px;
    overflow-y: scroll;
    text-justify: justify;
    text-align: justify;
    padding: 10px;
    border-radius: 10px;
}
```

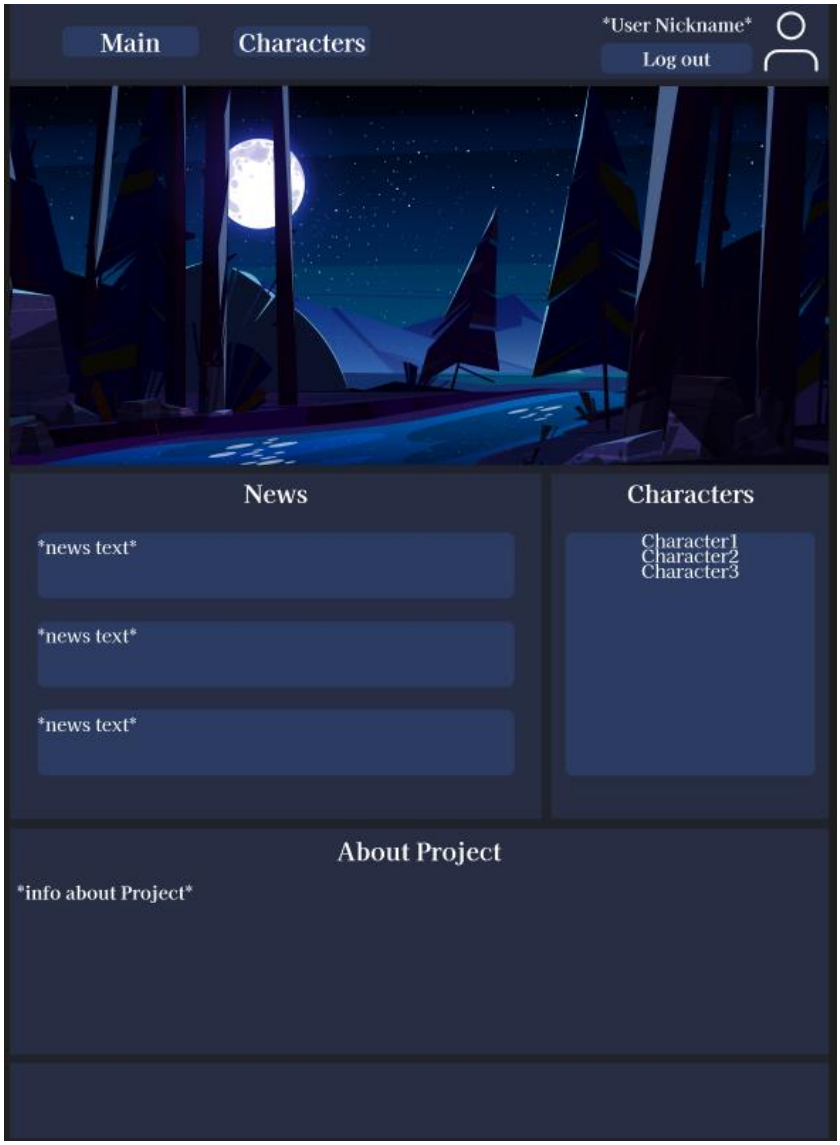
```
@mixin popup{
    display: none;
    width: 100vw;
    height: 100vh;
    top: 0;
    left: 0;
    position: fixed;
    z-index: -1;
    background-color: rgba(32, 35, 44, 0.8);
    align-items: center;
    justify-content: center;

    .popup-content{
        height: 60%;
        width: 40%;
        border-radius: 20px;
        background-color: $section-color;
        opacity: 1;
        padding: 40px;

        form{
            display: flex;
            flex-direction: column;
            align-items: center;
            overflow-y: auto;
            height: 100%;
            width: 100%;
            gap: 20px;
        }

        div{
            display: flex;
            flex-direction: column;
            width: 80%;
            height: 100px;

            input{
                font-size: $regular-text-size;
            }

            select{
                font-size: $regular-text-size;
            }
        }

        .submit-button{
            @include round-field;
            border-style: solid;
            font-family: $font;
        }
    }
}

@mixin button{
    @include round-field;
    display: flex;
```

```
    justify-content: center;  /* горизонтальне центрування */
    align-items: center;      /* вертикальне центрування */
    height: 40px;
    width: 10%;
    margin: 10px 20px;
    align-self: center;
    background-color: $field-color;
    border-style: solid;
    border-color: $field-color;
    color: $font-color;
    font-family: $font;
    font-size: 24px;
    cursor: pointer;
}


$background-color: #20232C;
$font-color: #FFFFFF;
$section-color: #272D43;
$field-color: #2C3B62;
$regular-text-size: 30px;
$font: "KaiseniHarunoUmi"
```

ДОДАТОК В. Діаграма класів для пакунка apps.character

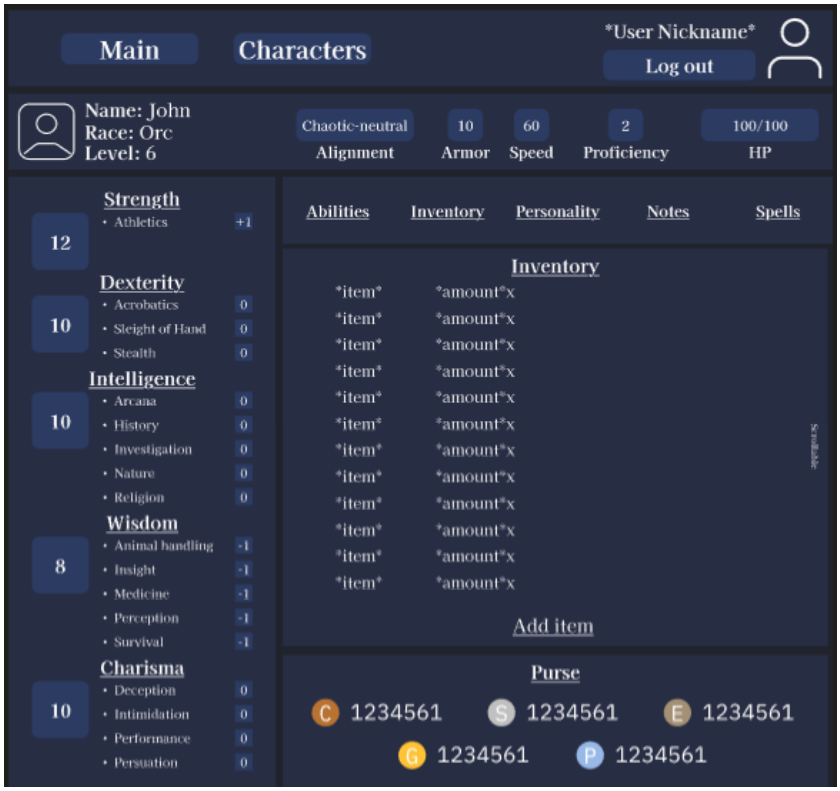


ДОДАТОК Г. Дизайн веб-застосунку



Дизайн головної сторінки

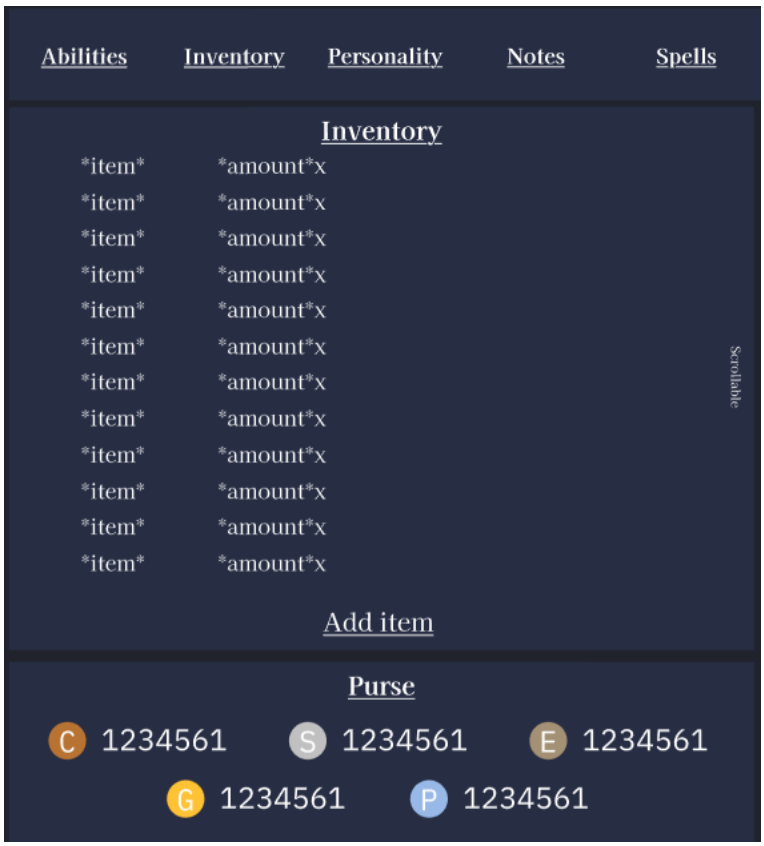
Дизайн сторінки зі списком персонажів



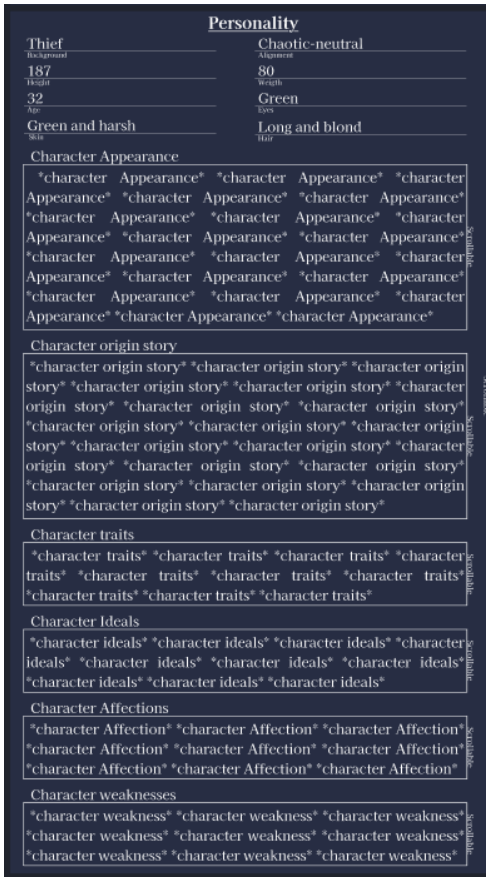
Дизайн сторінки листа персонажа

ДОДАТОК Д. Варіації вкладок в листі персонажа

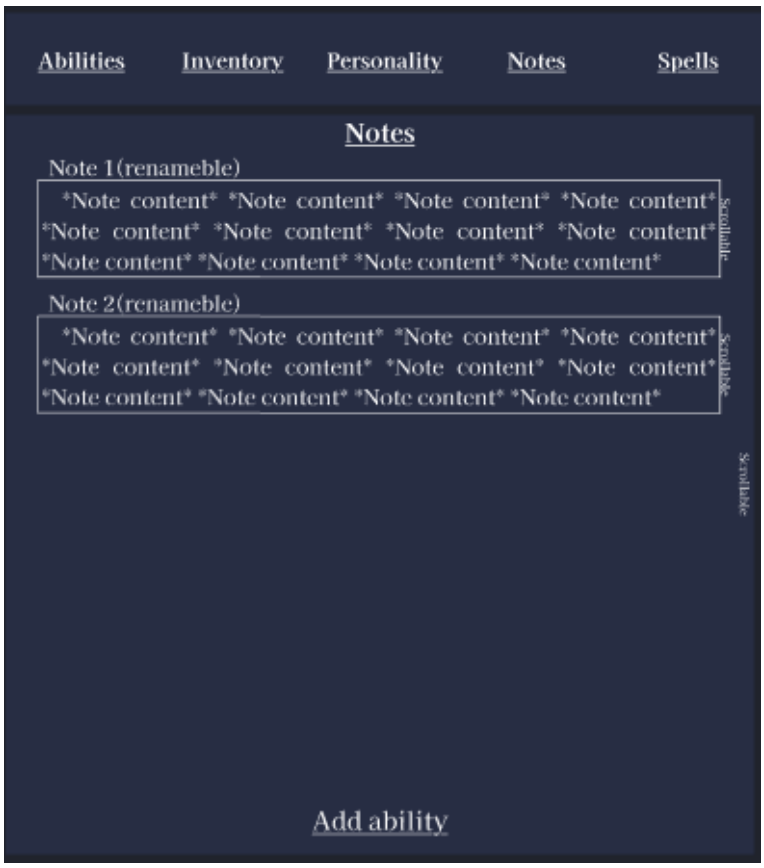
Вкладка Здібності



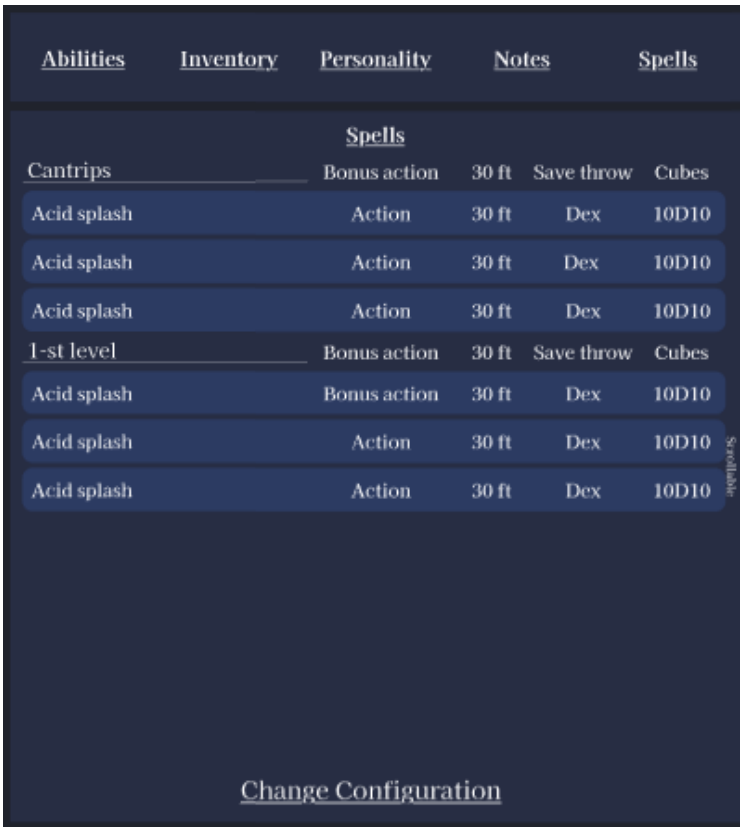
Вкладка Інвертар



Вкладка Персоналізації



Вкладка нотатків



Вкладка Заклинань

ДОДАТОК Е. Диск з роботою