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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка браузерної гри Who Am I з використанням технології Java Spring Boot

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

_____ Гриців О. М.
(підпис) (прізвище та ініціали)

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

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

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

Рецензент _____ Никитюк В. В.
(підпис) (прізвище та ініціали)

Тернопіль
2023

РЕФЕРАТ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2023, Сторінок 51, рисунків 17, презентація.

Тема: Розробка браузерної гри Who Am I з використанням технології Java Spring Boot.

У процесі розробки браузерної гри "Who Am I" з використанням технології Java Spring Boot, була проведена детальна робота над моделлю предметної області. Були розглянуті наступні основні аспекти гри:

1. Правила гри: Було розроблено та описано правила гри "Who Am I", які визначають механіку та цілі гри. Це включає процес вибору персонажів, проведення раундів, задавання запитань та вгадування персонажів.

2. Зв'язки між гравцями та системою: Було визначено, як гравці взаємодіють з грою та системою. Це включає взаємодію гравців під час запитань та відповідей, а також спілкування з системою для контролю гри та відстеження прогресу.

3. Механіка гри: Були розглянуті різні аспекти механіки гри, такі як раунди, лічильники часу та можливості отримання підказок. Ці елементи геймплею додають цікавість та виклик грі, стимулюючи гравців до стратегічного мислення та співпраці.

Результатом цієї роботи є веб-сайт, на якому гравці можуть насолоджуватись грою "Who Am I". Практичне значення цієї роботи полягає в тому, що вона надає можливість гравцям весело провести час, спілкуючись та взаємодіючи з іншими гравцями у цій захоплюючій грі.

ANNOTATION

Bachelor's qualification work. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, Specialty 121 "Software Engineering". TNTU, 2023, Pages 51, figures 17, presentation.

Topic: Development of the browser game "Who Am I" using Java Spring Boot technology.

In the process of developing the browser game "Who Am I" using Java Spring Boot technology, detailed work was carried out on the development of the domain model. The following key aspects of the game were considered:

1. Game rules: The rules of the "Who Am I" game were developed and described, defining the mechanics and objectives of the game. This includes the process of selecting characters, conducting rounds, asking questions, and guessing characters.

2. Player-system interactions: The interactions between players and the game system were defined. This includes the player's interaction during questions and answers, as well as communication with the system for game control and progress tracking.

3. Game mechanics: Various aspects of game mechanics were explored, such as rounds, timers, and the possibility of obtaining hints. These gameplay elements add interest and challenge to the game, encouraging players to think strategically and cooperate.

The outcome of this work is a website where players can enjoy the game "Who Am I". The practical significance of this work lies in providing players with an enjoyable experience, allowing them to have fun while interacting and communicating with other players in this captivating game.

ЗМІСТ

ВСТУП

Розробка комп'ютерних ігор є цікавою та захоплюючою галуззю в світі інформаційних технологій. У сучасному світі ігор, де мільйони гравців шукають нові способи розваги, настільні ігри здобувають особливу популярність. Вони створюють неперевершену атмосферу взаємодії та розваги, особливо в такий непростий час.

Цей проект присвячений розробці браузерної гри "Who Am I" у жанрі настільної гри, яка є особливо популярною в США. Гра "Who Am I" дозволяє гравцям насолодитися захоплюючою атмосферою гри в групі, де кожен має змогу відчути себе в ролі різних персонажів та вгадувати, хто вони є, шляхом задавання питань та розгадування загадок.

Використання технології Java Spring Boot у розробці цієї браузерної гри надає широкі можливості для створення потужної та масштабованої серверної частини, а також ефективної системи керування даними. Java Spring Boot є популярною технологією у сфері розробки веб-додатків, що дозволяє зручно реалізовувати та підтримувати гру.

Метою даного проекту є розробка високоякісної браузерної гри "Who Am I" з використанням технології Java Spring Boot, яка надасть користувачам незабутні враження та можливість насолодитися цією популярною настільною грою. У цьому документі будуть детально розглянуті етапи проєктування, розробки та тестування гри, а також описані основні компоненти та функціональні можливості, які будуть впроваджені.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

Компанія CGE Digital [1] (рис. 1.1) є відомим у світі розробником і видавцем настільних ігор. Вона має багатий досвід у цифровій адаптації настільних ігор і відома своїми інноваційними підходами до гейм дизайну. Компанія заснована командою талановитих розробників ігор, які мають страсть до створення унікальних і захоплюючих геймплейних досвідів.

Рис. 1.1 – Логотип конкурента CGE Digital

CGE Digital прагне поєднувати найкращі елементи настільних ігор з сучасними технологіями, створюючи веб-версії ігор, які можна грати в онлайн-середовищі. Компанія має великий асортимент успішних цифрових адаптацій настільних ігор, що охоплюють різноманітні жанри та тематики.

CGE Digital відзначається своїм професіоналізмом і фокусом на якості. Команда розробників володіє глибоким розумінням настільних ігор, їх механік і правил, та знає, як вдало перетворити їх у цифровий формат, зберігаючи весь шарм і веселощі, що властиві оригіналам.

Компанія активно співпрацює з авторами ігор, щоб забезпечити вірну та якісну цифрову інтерпретацію їх творінь. Вона також звертає особливу увагу на візуальне оформлення, звуковий супровід і користувацький досвід, щоб створити неперевершену гральну атмосферу.

CGE Digital прагне не тільки розробляти цифрові версії настільних ігор, але й

розширювати їх функціональні можливості, включаючи можливість грати з друзями по мережі, використовувати голосовий чат та взаємодію з глобальною спільнотою гравців. Однією із таких ігор є “Codenames Online”.

Codenames Online є однією з найуспішніших цифрових адаптацій настільної гри. Ця гра відтворює захоплюючий геймплей і неповторну атмосферу оригінальної настільної гри "Codenames", яка стала популярною серед шанувальників настільних ігор по всьому світу.

Codenames Online пропонує гравцям унікальний досвід, де вони можуть змагатися з друзями. Гра базується на простому, але цікавому принципі, де гравці поділені на дві команди і спробують відгадати кодові слова, що пов'язані з підказкою, наданою капітаном команди. Це вимагає стратегічного мислення, комунікації та розуміння партнерів, що робить гру захоплюючою та динамічною.

Codenames Online відрізняється від оригінальної настільної гри тим, що пропонує віддалену гру через Інтернет. Гравці можуть приєднатися до віртуального столу з будь-якого куточка світу, спілкуватися через відеозв'язок або чат і спільно розгадувати кодові слова. Додаткові функції, такі як таймери, підказки та статистика гри, роблять досвід гри ще більш захопливим та цікавим.

Codenames Online успішно поєднує класичні механіки настільної гри з передовими технологіями, що дозволяє гравцям насолоджуватися грою в будь-який час та в будь-якому місці. Ця цифрова адаптація забезпечує доступність гри для широкої аудиторії та підкреслює зусилля компанії CGE Digital у розвитку та просуванні настільних ігор у цифровому просторі.

1.2 Обґрунтування вибору напрямку розробки

Зростаючий попит на віртуальні форми розваг:

За останні роки спостерігається значний попит на віртуальні ігрові платформи, які надають можливість грати у настільні ігри в онлайн-режимі. Це свідчить про

зростаючу зацікавленість гравців у віртуальних формах розваг та їх бажання взаємодіяти з друзями незалежно від географічного розташування.

Потреба у глобальному спілкуванні:

У сучасному світі, коли люди перебувають на різних континентах, важливо мати засоби для віртуального спілкування та взаємодії з друзями. Віртуальні настільні ігри стають прекрасним способом об'єднати людей із різних куточків світу, дозволяючи їм насолоджуватися грою разом та спілкуватися незалежно від відстані.

Ринковий потенціал і конкурентні переваги:

Розробка браузерної гри Who Am I з використанням технології Java Spring Boot [2] відповідає потребам гравців, які мають бажання зіграти у популярну настільну гру з друзями в будь-якому місці світу. Враховуючи зростання популярності онлайн-ігор та високий ринковий потенціал настільних ігор у віртуальному середовищі, дана розробка має конкурентні переваги і може привернути увагу широкого кола гравців.

Враховуючи ці фактори, обрання напрямку розробки браузерної гри Who Am I з використанням технології Java Spring Boot є обґрунтованим та відповідає потребам сучасного гравцевого співтовариства.

2 ПРОЄКТУВАННЯ БРАУЗЕРНОЇ ГРИ

2.1 Розробка моделі предметної області

Використання візуалізації є важливим засобом забезпечення зрозуміння системи. Вона дозволяє перетворити абстрактні ідеї на щось конкретне, що можна "побачити". Моделювання дозволяє нам продемонструвати поведінку та бажану структуру програмного забезпечення. Коли ми визначаємо цілі програмної системи, ми отримуємо шаблон, який не тільки дозволяє розширити систему в майбутньому, але й мінімізує ризики, пов'язані з переробкою, частковими змінами або реалізацією.

Наведемо наступні правила за якими буде розроблено продукт і якими будуть користуватися користувачі під час гри [3]:

- Кожен гравець пропонує персонажа для інших гравців. Це може бути: людина, тварина, персонаж з фільмів чи серіалів і т.д.
- Персонажів між гравцями система розподіляє випадковим чином. Персонаж не може бути призначений людині яка його задала тільки у випадку якщо гравці задали кільки однакових персонажів.
- Коли всі персонажі були розподілені між гравцями гра починається автоматично.
- Гравці задають питання по черзі.
- Коли гравець задає питання всі інші відповідають.
- В гравців що відповідають на питання є три варіанти відповіді “Так”(YES), “Ні”(NO), “Не впевнений(а)”(NOT SURE).
- Якщо більшість відповідей позитивні то гравець продовжує задавати питання або може спробувати вгадати свого персонажа.
- Якщо більшість відповідей негативні то хід переходить до іншого граця.
- Коли гравці відповідають на запитання про відгадування персонажа в них є тільки два варіанти відповіті “Так”(YES) або “Ні”(NO).
- Персонаж вважається відгаданим якщо більшість відповідей позитивні.

- Позитивні відповіді це відповіді “Так”(YES) і “Не впевнений(а)”(NOT SURE).
- Гра продовжується до того часу поки є 2 і більше гравця.
- Гравець програє коли він залишається один в лобі.

2.2 Розробка бізнес моделі

Перед початком написання програмного коду важливо спроектувати бізнес модель, що дозволяє використовувати модель для візуалізації майбутньої програмної системи та всіх її компонентів та їх взаємодії. Аналіз вимог до програмного забезпечення також має велике значення, оскільки його результат можна представити у вигляді діаграми варіантів використання, використовуючи мову UML. Ця діаграма покаже взаємодію системи і кінцевого користувача. На ній можна побачити сутності та варіанти використання (use cases).

Було спроектовано наступну діаграму варіантів використання для гравця:

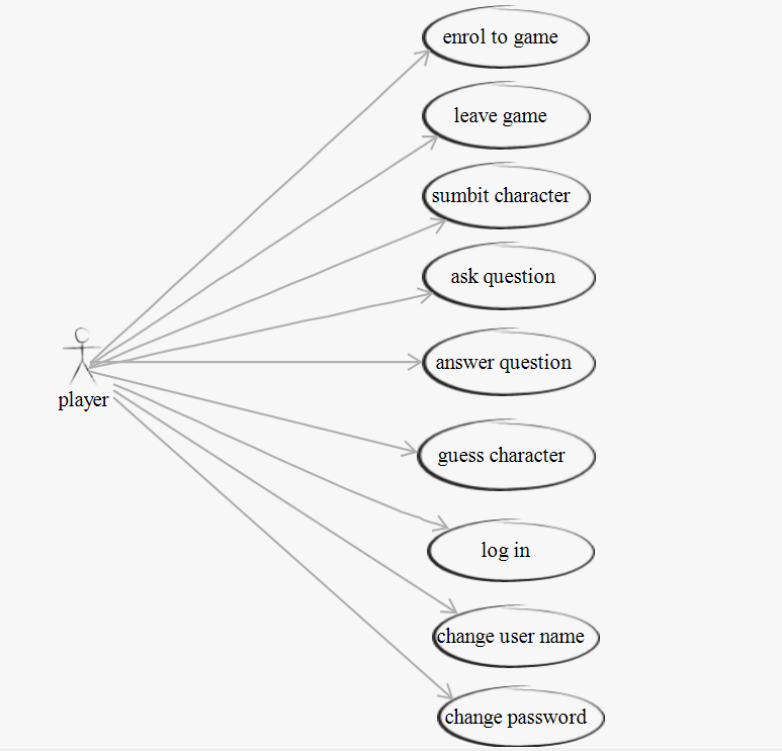

Рисунок 2.2.1 – Діаграма варіантів використання

На рисунку 2.2.1 бачимо що в програмній системі існуватиме тільки роль користувача. Йому доступні такі функції:

- Приєднатися до гри
- Покинути гру
- Ввести персонажа
- Задати запитання
- Відповісти на питання
- Вгадати персонажа
- Ввійти
- Змінити ім’я користувача
- Поміняти пароль

Ці функції, які доступні користувачеві в програмній системі, грають важливу роль у створенні зручного та функціонального інтерфейсу. Можливість приєднатися до гри дозволяє користувачеві залучитися до процесу і взаємодіяти з іншими гравцями. Покидання гри дає користувачу можливість зрушити зі своїм поточним станом, якщо потрібно. Введення персонажа, задання запитань, відповідь на питання та вгадування персонажа створюють групу функцій, які сприяють взаємодії користувача з грою і забезпечують можливість розважатися та грати. Ввійти, змінити ім'я користувача та поміняти пароль дозволяють користувачеві зручно управляти своїм обліковим записом і забезпечувати безпеку своїх особистих даних. В цілому, цей набір функцій впевнено виконує роль утримання користувача у програмній системі, забезпечуючи йому необхідний функціонал для комфортної та задоволеної взаємодії з програмою.

2.3 Проєктування архітектури

Після аналізування предметної області стало очевидним, які сутності будуть присутні в програмній системі та які атрибути цих сутностей.

На рисунку 2.3.1 представлено діаграму бази даних.

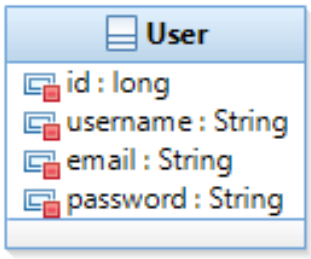

Рисунок 2.3.1 – Діаграма бази даних

З рисунку 2.3.1 модна побачити що в базі даних існує одна таблиця для збереження даних про користувача, її назва User. Як показано на діаграмі бази даних в користувача є чотири поля:

- Ідентифікатор користувача
- Ім’я користувача
- Електрона адреса
- Пароль

На рисунку 2.3.2 зображено діаграму основних класів що підтримують стабільну роботу гри.

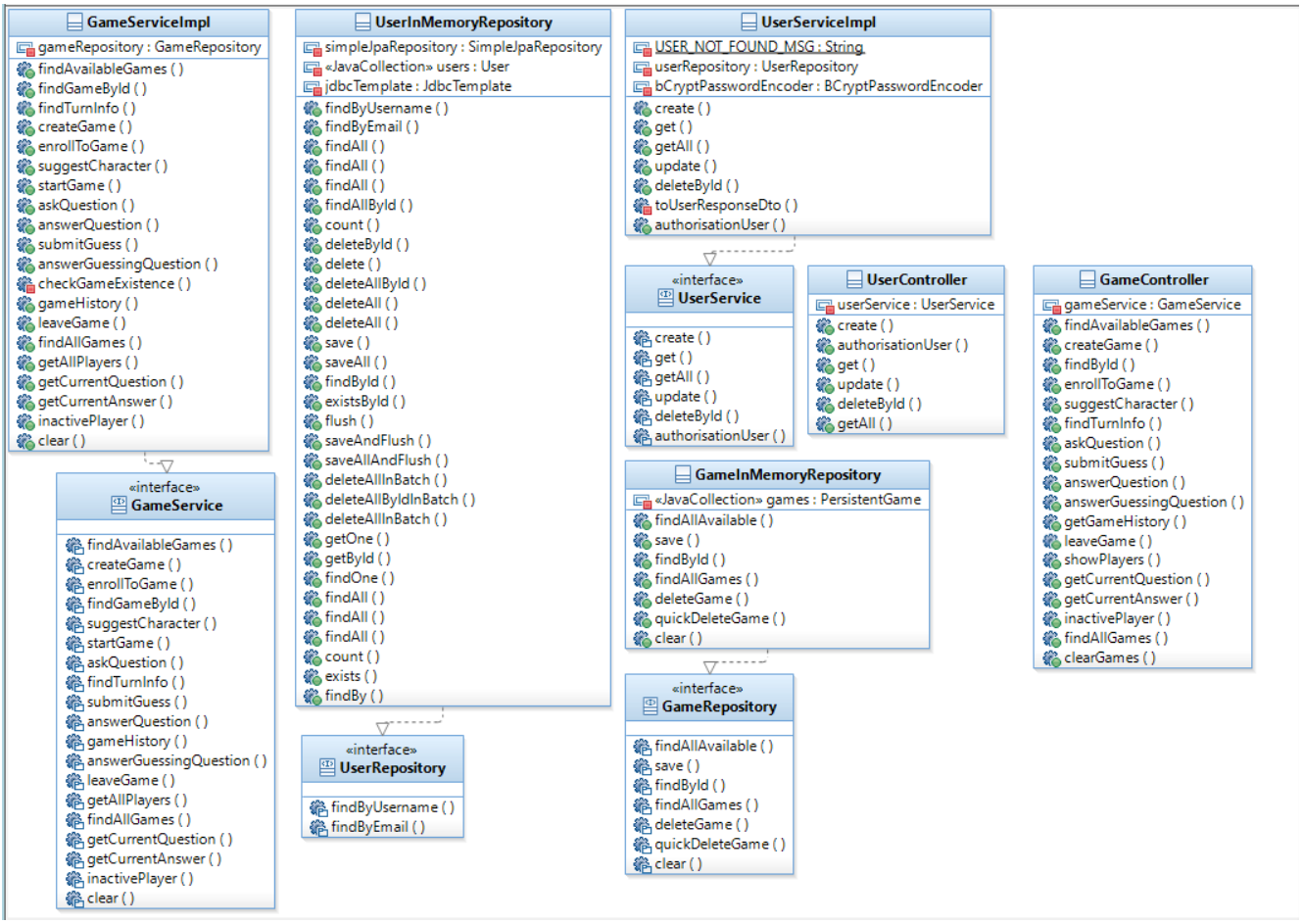

Рисунок 2.3.2 – Діаграма класів

На цій діаграмі зображено такі класи:

- GameController – Основна функція класу полягає в обробці запитів, пов'язаних з ігровим процесом. Він має ряд методів для взаємодії з грою, управління гравцями та отримання інформації про стан гри. Наприклад, метод findAvailableGames() дозволяє знайти доступні ігри, createGame() створює нову гру, а findById() повертає деталі гри за заданим ідентифікатором. Клас також має методи для приєднання до гри, надсилання пропозицій персонажів, задавання питань та відповідей, а також для отримання історії гри та керування неактивними гравцями. Контролер також надає методи для отримання загальної кількості гравців та очищення списку ігор. Всі ці методи використовуються для взаємодії з клієнтським додатком і забезпечують роботу ігрової системи.
- GameServiceImpl – реалізацією інтерфейсу GameService і містить логіку для керування грою. Він виконує різноманітні операції з ігровою логікою та

взаємодіє з GameRepository, щоб отримати доступ до даних ігор. Він надає методи для пошуку доступних ігор, створення нової гри, реєстрації гравців до гри, подання пропозицій про персонажа, запитань, відповідей, вгадування та інших операцій, пов'язаних зі станом гри.

- GameService – Інтерфейс описує контракт для сервісу гри "Who Am I?". Цей сервіс надає методи для взаємодії з грою, такі як створення гри, приєднання до гри, відправлення пропозиції щодо персонажу, початок гри, задання питань, отримання деталей про хід гри та інше.
- GameInMemoryRepository – Клас є реалізацією інтерфейсу GameRepository і використовується для зберігання і доступу до даних гри у пам'яті. Він містить список games, який зберігає об'єкти PersistentGame. Клас має методи для знаходження доступних ігор, зберігання гри, знаходження гри за ідентифікатором, видалення гри та очищення репозиторію. Крім того, клас має метод quickDeleteGame, який швидко видаляє гру без затримки. Клас позначений аннотацією @Repository для використання в контексті Spring.
- GameRepository – Інтерфейс визначає контракт для репозиторію гри. Він описує методи для отримання доступних ігор, збереження гри, отримання гри за її ідентифікатором, отримання списку всіх ігор, видалення гри та очищення репозиторію. Інтерфейс працює з об'єктами типу PersistentGame.
- UserController – Клас є контролером для управління користувачами в системі. Він виконує обробку HTTP-запитів, пов'язаних з операціями створення, отримання, оновлення та видалення користувачів.
- UserServiceImpl – Клас є реалізацією інтерфейсу UserService і забезпечує логіку для операцій з базою даних.
- UserService – Інтерфейс визначає контракт для сервісного рівня, що займається операціями з базою даних.
- UserInMemoryRepository – Клас є реалізацією інтерфейсу UserRepository і забезпечує зберігання та доступ до об'єктів типу User. Він зберігає користувачів у пам'яті та використовує JdbcTemplate для виконання SQL-запитів до бази даних.

- UserRepository – Цей клас представляє репозиторій для доступу до даних користувачів. Він розширює інтерфейс JpaRepository, що забезпечує стандартні методи для роботи з базою даних за допомогою JPA.

На рисунку 2.3.3 подано діаграму послідовності.

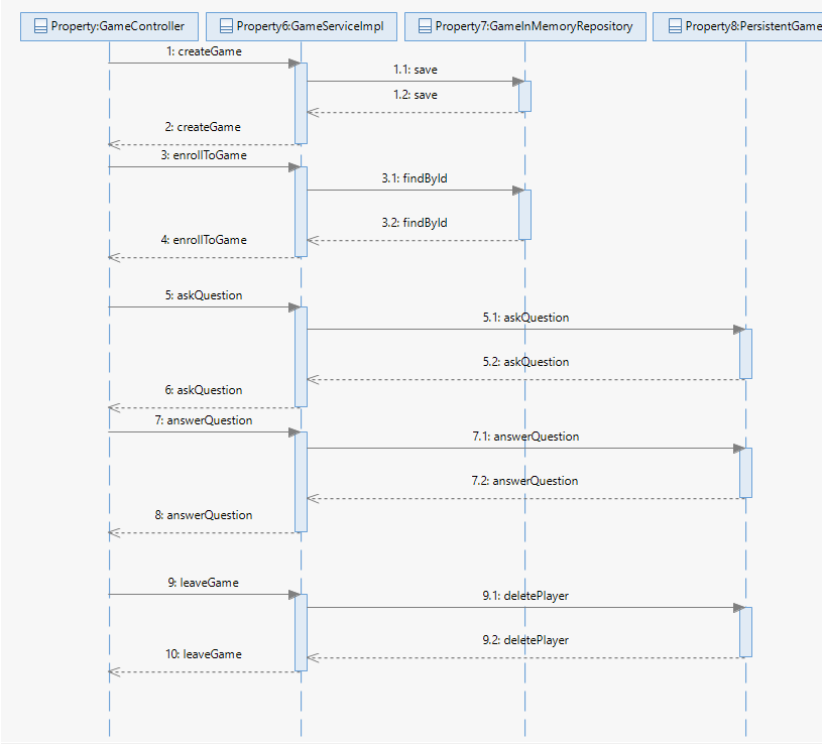

Рисунок 2.3.3 – Діаграма послідовності

Рисунок 2.3.3 ілюструє важливу взаємодію класів, яка становить основу для створення гри "Who Am I". Ці класи включають механізми під'єднання, постановки запитань, отримання відповідей на запитання та виходу з гри. Кожен з цих класів має свої власні відповідальності та функціональні можливості, але разом вони спільно працюють, щоб забезпечити повну функціональність системи "Who Am I".

Ці класи взаємодіють з клієнтським додатком, що дозволяє користувачам грати в гру, а також з базою даних, де зберігається інформація про гру та користувачів. Вони взаємодіють з базою даних для збереження та отримання різних даних, таких як стан гри, історія запитань та відповідей, а також профілі користувачів.

3 КОНСТРУЮВАННЯ БРАУЗЕРНОЇ ГРИ

3.1 Розробка ключових класів

Під час розробки програмної системи були створені ключові класи, що реалізують базову логіку системи, які відображені на діаграмі класів на рисунку 2.3.2. Розглянемо реалізацію цих класів:

Клас `GameController` має анотацію `@RestController`, що позначає його як контролер, який обробляє HTTP-запити. Анотація `@RequestMapping("/games")` вказує на те, що цей контролер відповідає на запити, що починаються з `/games`. Клас реалізовує наступні поля і методи:

- `private final GameService gameService`: поле, що представляє сервіс гри і ініціалізується через конструктор. Воно використовується для взаємодії з логікою гри.
- `@GetMapping`: метод `findAvailableGames` обробляє GET-запит на шляху `/games` і повертає список доступних ігор.
- `@PostMapping`: метод `createGame` обробляє POST-запит на шляху `/games` і створює нову гру з заданим гравцем та даними гри.
- `@GetMapping("/{id}")`: метод `findById` обробляє GET-запит на шляху `/games/{id}` і повертає деталі гри за її ідентифікатором.
- `@PostMapping("/{id}/players")`: метод `enrollToGame` обробляє POST-запит на шляху `/games/{id}/players` і приєднує гравця до гри за її ідентифікатором.
- `@PostMapping("/{id}/characters")`: метод `suggestCharacter` обробляє POST-запит на шляху `/games/{id}/characters` і приймає пропозицію персонажа від гравця для гри.
- `@GetMapping("/{id}/turn")`: метод `findTurnInfo` обробляє GET-запит на шляху `/games/{id}/turn` і повертає інформацію про поточний хід гри.
- `@PostMapping("/{id}/questions")`: метод `askQuestion` обробляє POST-запит на шляху `/games/{id}/questions` і задає запитання в грі.
- `@PostMapping("/{id}/guess")`: метод `submitGuess` обробляє POST-запит на

шляху `/games/{id}/guess` і надсилає вгадування гравця в гру.

- `@PostMapping("/{id}/answer")`: метод `answerQuestion` обробляє POST-запит на шляху `/games/{id}/answer` і надає відповідь на запитання в грі.
- `@PostMapping("/{id}/guess/answer")`: метод `answerGuessingQuestion` обробляє POST-запит на шляху `/games/{id}/guess/answer` і надає відповідь на запитання про вгадування персонажа в грі.
- `@GetMapping("/{id}/history")`: метод `getGameHistory` обробляє GET-запит на шляху `/games/{id}/history` і повертає історію чату гри.
- `@DeleteMapping("/{id}/leave")`: метод `leaveGame` обробляє DELETE-запит на шляху `/games/{id}/leave` і дозволяє гравцеві покинути гру.
- `@GetMapping("/all-players-count")`: метод `showPlayers` обробляє GET-запит на шляху `/games/all-players-count` і повертає загальну кількість гравців у всіх іграх.
- `@GetMapping("/{id}/questions/getQuestion")`: метод `getCurrentQuestion` обробляє GET-запит на шляху `/games/{id}/questions/getQuestion` і повертає поточне запитання в грі.
- `@GetMapping("/{id}/answer/getAnswer")`: метод `getCurrentAnswer` обробляє GET-запит на шляху `/games/{id}/answer/getAnswer` і повертає поточну відповідь в грі.
- `@PostMapping("/{id}/inactivePlayer")`: метод `inactivePlayer` обробляє POST-запит на шляху `/games/{id}/inactivePlayer` і помічає гравця як неактивного в грі.
- `@GetMapping("/all")`: метод `findAllGames` обробляє GET-запит на шляху `/games/all` і повертає список всіх ігор.
- `@PostMapping("/clear")`: метод `clearGames` обробляє POST-запит на шляху `/games/clear` і видаляє всі ігри зі списку.

Клас `GameServiceImpl` є реалізацією інтерфейсу `GameService` і має наступні поля та методи:

- `private final GameRepository gameRepository`: поле, що представляє репозиторій гри і ініціалізується через конструктор. Воно використовується

для доступу до даних гри.

- `@Override`: анотація, що позначає перевизначення методу з інтерфейсу `GameService`.
- `findAvailableGames()`: метод, який повертає список доступних ігор, використовуючи `gameRepository.findAllAvailable()`.
- `findGameById(String id)`: метод, який повертає деталі гри за її ідентифікатором, створюючи новий об'єкт `GameDetails` з результатом `checkGameExistence(id)`.
- `findTurnInfo(String id, String player)`: метод, який повертає інформацію про поточний хід гри за ідентифікатором гри та гравця, шляхом виклику `game.getTurnDetails()`.
- `createGame(String playerId, NewGameRequest gameRequest)`: метод, який створює нову гру з заданим гравцем та даними гри. Якщо немає доступних ігор, створюється нова гра і повертається об'єкт `GameDetails`. Якщо є доступні ігри, гравець додається до першої доступної гри або видається виняток `GameStateException`, якщо всі місця для гравців зайняті.
- `enrollToGame(String gameId, String playerId)`: метод, який приєднує гравця до гри за її ідентифікатором. Якщо гра знаходиться в стані очікування гравців, гравець додається до гри і повертається об'єкт `PlayerDetails`. Якщо гра не знаходиться в стані очікування гравців, видається виняток `GameStateException`.
- `suggestCharacter(String gameId, String player, CharacterSuggestion suggestion)`: метод, який приймає пропозицію персонажа від гравця для гри за її ідентифікатором. Якщо гра знаходиться в стані пропозиції персонажа, пропозиція додається до гри. Коли всі гравці надали пропозиції, гра починається.
- `startGame(String gameId)`: метод, який розпочинає гру за її ідентифікатором. Залежно від поточного стану гри, видаються винятки `GameStateException` або повертається об'єкт `GameDetails`.
- `askQuestion(String gameId, String player, String message)`: метод, який задає

питання в грі за її ідентифікатором. Якщо гра знаходиться в стані "GAME_IN_PROGRESS", питання додається до гри.

- `answerQuestion(String gameId, String player, QuestionAnswer answer)`: метод, який додає відповідь гравця на питання в грі за її ідентифікатором. Якщо гра знаходиться в стані "GAME_IN_PROGRESS", відповідь додається до гри.
- `submitGuess(String gameId, String player, Message guess)`: метод, який надсилає відгадку гравця в гру за її ідентифікатором. Якщо гра знаходиться в стані "GAME_IN_PROGRESS", відгадка додається до гри.
- `answerGuessingQuestion(String gameId, String playerId, QuestionAnswer answerGuess)`: метод, який додає відповідь гравця на запитання-відгадку в грі за її ідентифікатором. Якщо гра знаходиться в стані "GAME_IN_PROGRESS" і залишилося лише один гравець, гра видаляється.
- `checkGameExistence(String gameId)`: приватний метод, який перевіряє наявність гри за її ідентифікатором в репозиторії. Якщо гра існує, повертається об'єкт гри. В іншому випадку видається виняток `GameNotFoundException`.
- `gameHistory(String gameId)`: метод, який повертає історію чату гри за її ідентифікатором.
- `leaveGame(String gameId, String playerId)`: метод, який дозволяє гравцю покинути гру за її ідентифікатором. Якщо в грі залишається два гравці, один з них стає переможцем. Якщо в грі залишається менше трьох гравців і гра не знаходиться в стані "GAME_IN_PROGRESS" або "WAITING_FOR_PLAYERS", гра видаляється. Якщо в грі залишається лише один гравець гра повністю видаляється.
- `findAllGames()`: метод, який повертає список усіх існуючих ігор.
- `getAllPlayers()`: метод, який повертає загальну кількість гравців у всіх іграх.
- `getCurrentQuestion(String gameId, String playerId)`: метод, який повертає поточне питання для гравця у грі за її ідентифікатором.
- `getCurrentAnswer(String gameId, String playerId)`: метод, який повертає поточну відповідь для гравця у грі за її ідентифікатором.

- `inactivePlayer(String gameId, String playerId)`: метод, який перевіряє, чи є гравець неактивним у грі за її ідентифікатором. Якщо гравець неактивний, він покидає гру і повертається значення `true`. В іншому випадку повертається значення `false`.
- `clear()`: метод, який видаляє всі існуючі ігри з репозиторію.

Клас `GameInMemoryRepository` є реалізацією інтерфейсу `GameRepository` і забезпечує збереження та керування об'єктами `PersistentGame` в оперативній пам'яті. Основні методи цього класу включають:

- `findAllAvailable()`: повертає список усіх доступних ігор, які очікують гравців.
- `save(PersistentGame game)`: зберігає об'єкт `PersistentGame` і повертає його.
- `findById(String gameId)`: знаходить гру за її ідентифікатором і повертає її у вигляді `Optional<PersistentGame>`. Якщо гра не знайдена, видається виняток `GameNotFoundException`.
- `findAllGames()`: повертає список усіх існуючих ігор.
- `deleteGame(String gameId)`: видаляє гру за її ідентифікатором. Перед видаленням, тут є затримка в 7 секунд для симуляції обробки.
- `quickDeleteGame(String gameId)`: швидко видаляє гру за її ідентифікатором без затримки.
- `clear()`: очищає всі існуючі ігри з репозиторію.

Отож, вищенаведені класи це основні класи без яких гра не буде працювати, які використовуються для отримання і обробки даних про поточне коло гри. Тут не було наведено інтерфейсів тому, що в цьому немає ніякого смислу так як всі методи наведені в вищезгаданих класах.

3.2 Розробка GUI

Важливим аспектом Інтернет-бізнесу є розуміння значення дизайну інтерфейсу користувача (UI) та зручності використання веб-сайту. Добре структурований UI є важливим фактором успіху. Дизайн UI впливає на залучення користувачів та їхній досвід (UX). Комфорт і легкість навігації на веб-сайті визначають успіх бізнесу.

Дизайн UI охоплює візуальні аспекти веб-сайтів, такі як кольори, шрифти, зображення та анімація. Ці елементи створюють комфортне середовище для користувачів і полегшують переходи між сторінками. Вони також забезпечують зрозумілість вмісту, що підвищує конверсію та уникнення незручностей. Розробка UI спрямована на передбачення шляху користувача на веб-сайті та забезпечення доступності та послідовності. Інтерфейс користувача поєднує дизайн взаємодії, візуальний дизайн та архітектуру веб-сайту.

В моєму проекті присутні наступні сторінки: головна сторінка, сторінка очікування, сторінка для задання персонажа, сторінка запитання, сторінка відповіді на запитання, сторінка для спроби відгадати свого персонажа, сторінка відповіді на відгадування персонажа. Відобразимо ці сторінки на наступних рисунках.

На рисунку 3.2.1 показано сторінку головного меню.

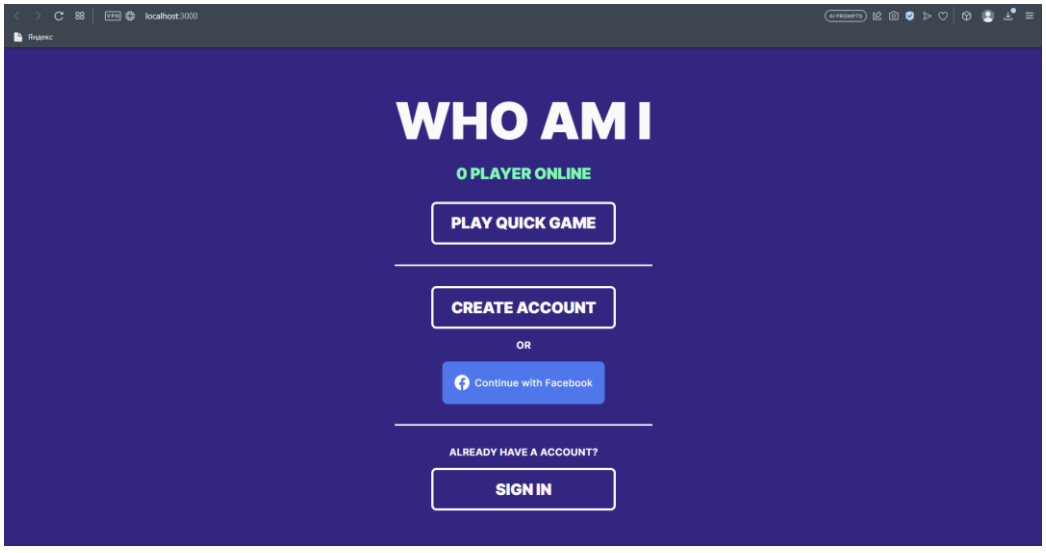

Рисунок 3.2.1 – Головна сторінка

На цій сторінці можна виконати вхід в систему або зіграти в швидку гру яка складається з чотирьох гравців.

Розглянемо наступну сторінку що зображена на рисунку 3.2.2 це сторінка очікування гравців.

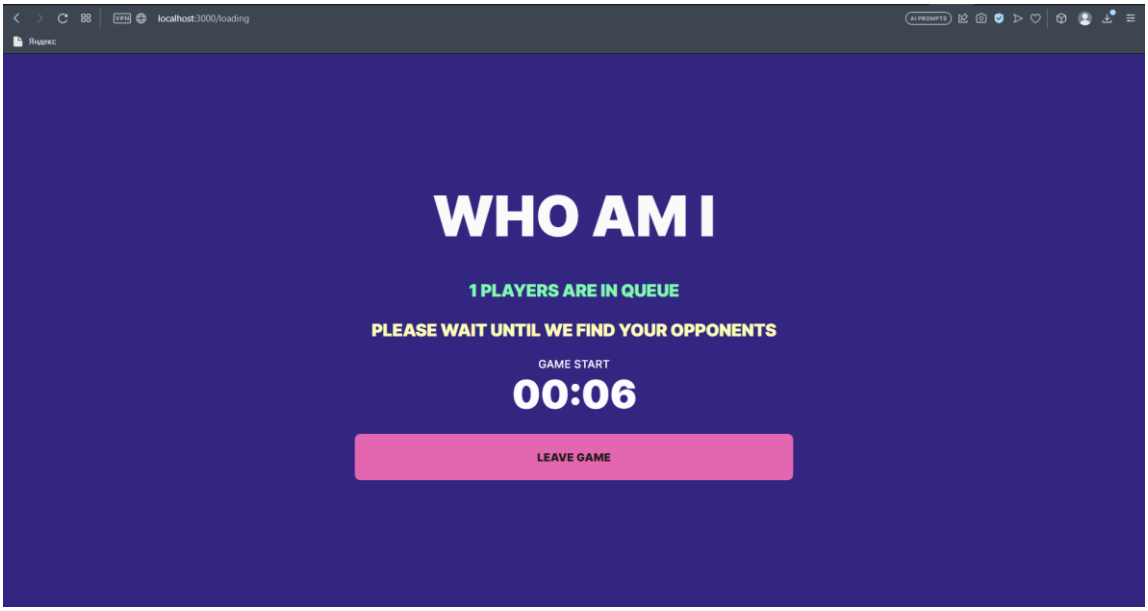

Рисунок 3.2.2 – Сторінка очікування гравців

На цій сторінці можна побачити скільки гравець чекає на інших гравців, а також якщо набридло чекати то можна скористатися кнопкою покинути гру.

Розглянемо наступну сторінку і це буде лобі що зображено на рисунку 3.2.3.

Рисунок 3.2.3 – Лобі

На цій сторінці зображено картки гравців, кнопку “ввести персонажа” і кнопку покинути гру. На рисунку 3.2.4 зображено модальне вікно для введення персонажа, а також зміна імені користувача що відображається. На рисунку 3.2.5 показано сторінку лобі після введення персонажа і зміною імені користувача.

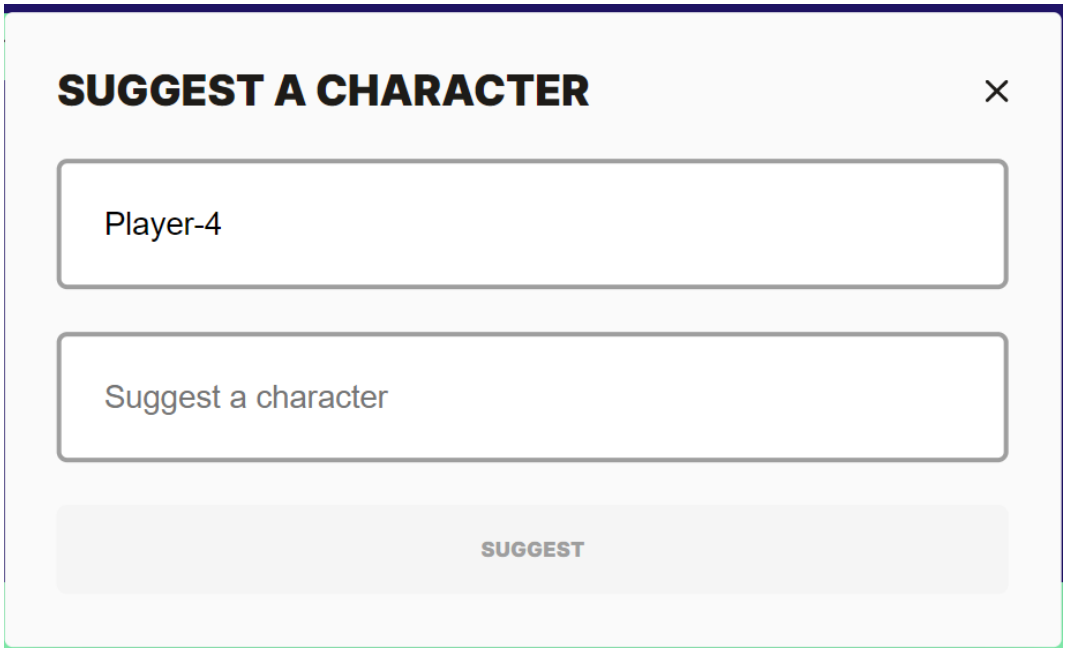

Рисунок 3.2.4 – Модальне вікно для запропонування персонажа

На цім рисунку можна побачити два поля для введення даних і неактивну кнопку “SUGGEST”. Ця кнопка робиться активною тільки в тім випадку коли всі поля заповнені.

Рисунок 3.2.5 – Лобі після запропонування персонажа гравцем

Розглянемо наступне вікно що зображено на рисунку 3.2.6.

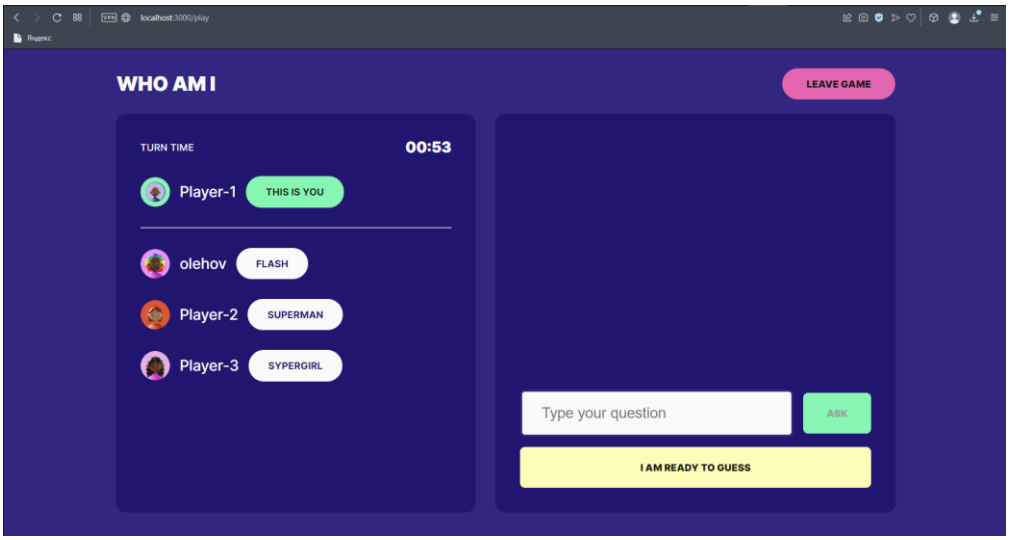

Рисунок 3.2.6 – Сторінка для запитання

На цій сторінці зображено поле для тексту і дві кнопки. Одна кнопка для відправлення запитання, а інша для спроби відгадати свого персонажа. Це модальне вікно зображене на рисунку 3.2.7.

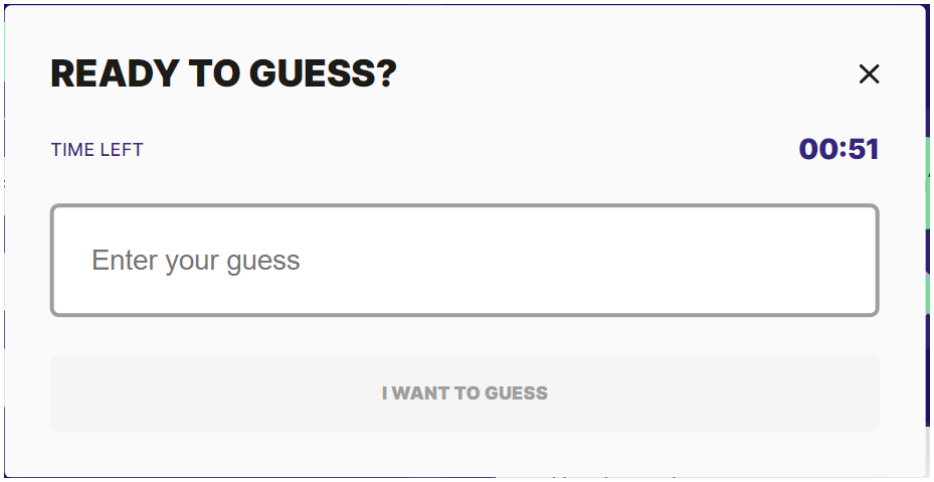

Рисунок 3.2.7 – Модальне вікно для спроби вгадати персонажа

На рисунку 3.2.8 зображено сторінку для відповіді на запитання.

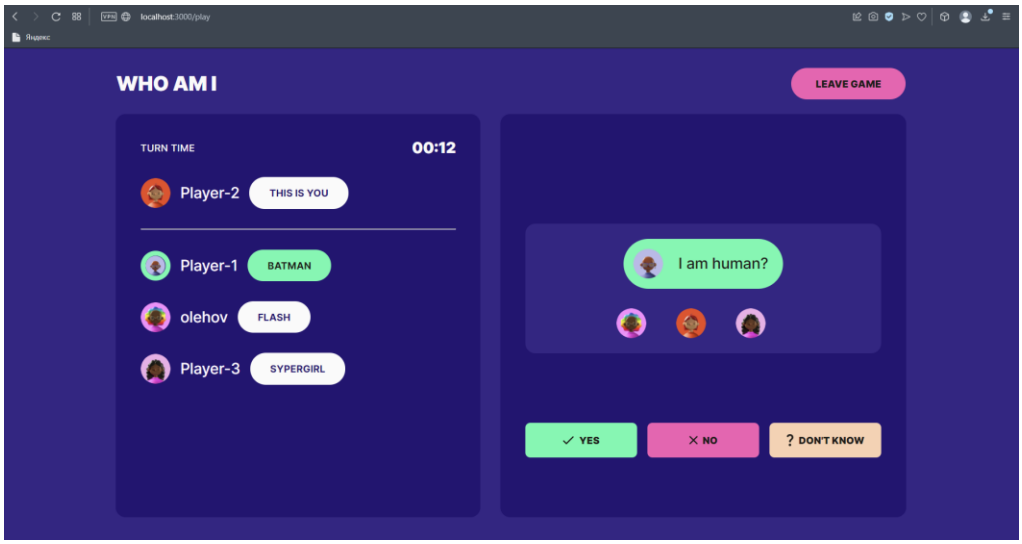

Рисунок 3.2.8 – Сторінка відповіді на запитання

На цій сторінці можна побачити три варіанти відповіді на запитання. Розглянемо сторінку на рисунку 3.2.9.

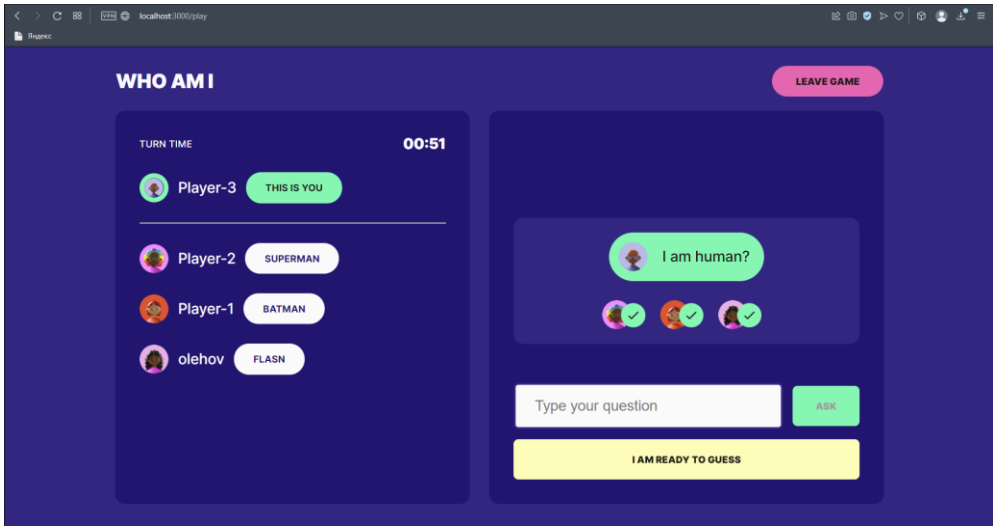

Рисунок 3.2.9 – Вигляд сторінки після відповіді на запитання

На цім рисунку можна побачити список із запитань і відповідей на запитання гравців. Судячи з рисунка 3.2.9 тут зображено що всі гравці дали позитивну відповідь на запитання, тому гравець продовжує задавати питання.

Розглянемо наступну сторінку 3.2.10.

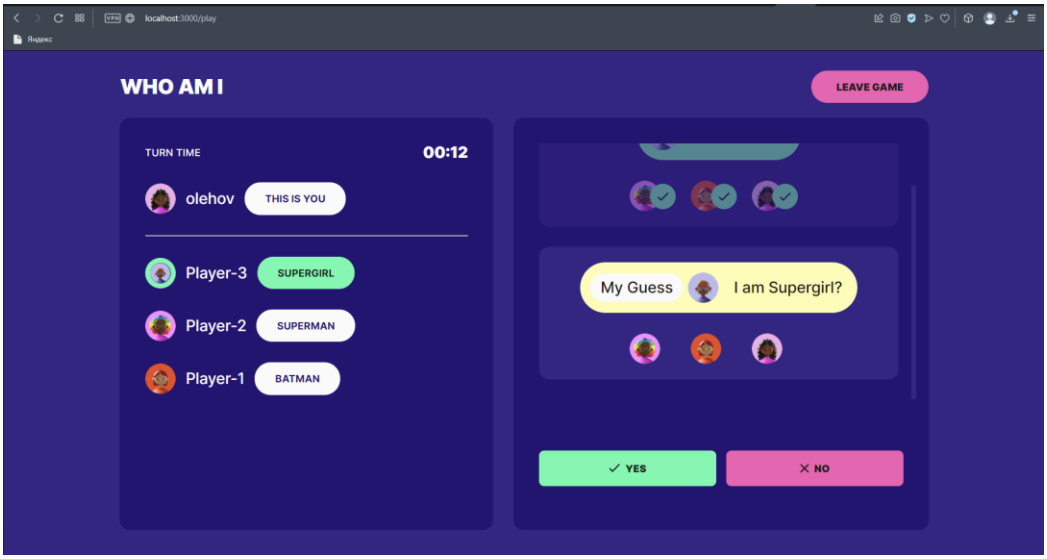

Рисунок 3.2.10 – Сторінка відповіді на спробу вгадати персонажа

Тут можна побачити тільки два варіанти відповіді на відміну від сторінки на рисунку 3.2.8. на цій сторінці гравець має дати конкретну відповідь “так” чи “ні”. Якщо більшість гравців відповідає “так”, то гравець бачить наступну сторінку що зображена на рисунку 3.2.11.

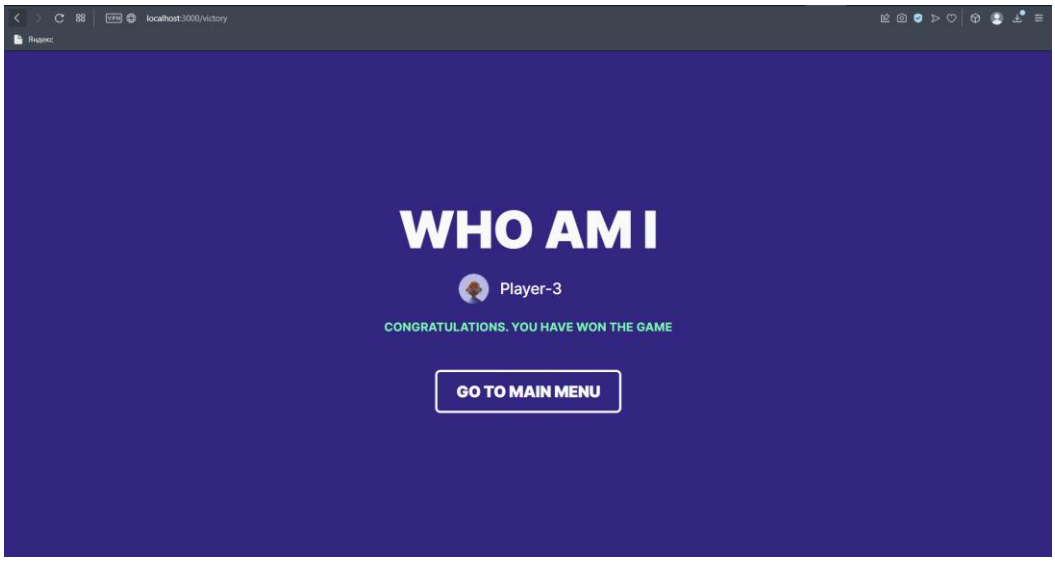

Рисунок 3.2.11 – Привітальна сторінка

На цій сторінці гравець бачить повідомлення про виграш в грі. Давайте розглянемо варіант що всі гравці завершили гру а залишився тільки один, то йому буде показано наступну сторінку, що зображена на рисунку 3.2.12.

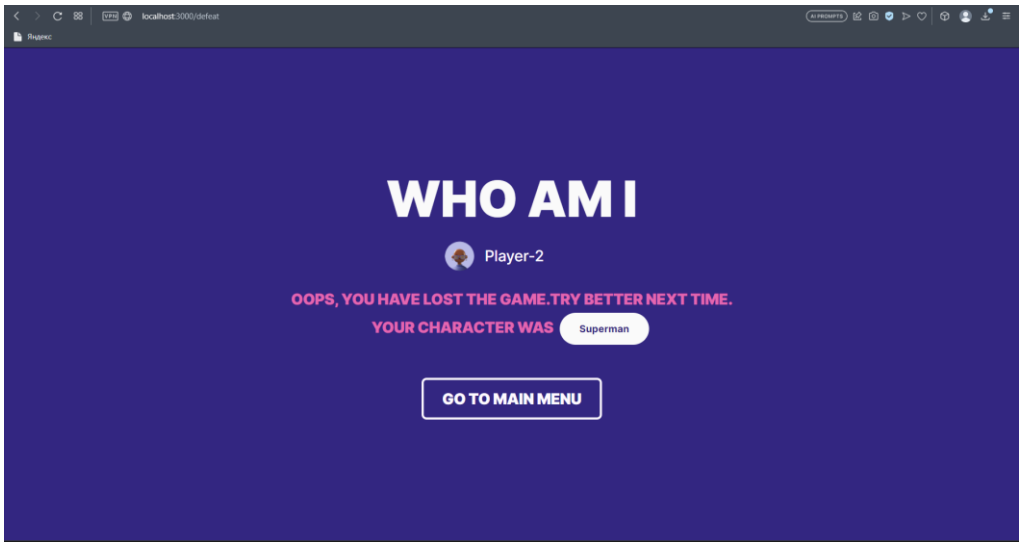

Рисунок 3.2.12 – Сторінка повідомлення про програш в грі

На цій сторінці показано повідомлення про програш в грі і персонаж який був в цього гравця. Також тут гра бажає успіху в наступних іграх.

3.3 Тестування програмного забезпечення та оцінка якості

Тестування програмного забезпечення та оцінка його якості відіграють важливу роль у розробці та експлуатації програмних продуктів. Цей процес включає перевірку функціональності, надійності, продуктивності та безпеки програмного забезпечення перед його випуском.

Основна мета тестування програмного забезпечення полягає в виявленні помилок, дефектів та несумісностей, які можуть виникнути під час його роботи. Це дозволяє розробникам виправити проблеми та забезпечити високу якість продукту перед випуском. Тестування також сприяє підтвердженню відповідності програмного забезпечення вимогам та специфікаціям, а також визначенню його поведінки в різних сценаріях використання.

Процес тестування може включати різні методи і підходи, включаючи модульне тестування, інтеграційне тестування, системне тестування та регресійне тестування.

Крім того, автоматизоване тестування стає все більш поширеним, оскільки воно дозволяє ефективно проводити тести та скорочує час, необхідний для перевірки програмного забезпечення.

Оцінка якості програмного забезпечення включає аналіз результатів тестування, вимірювання показників продуктивності, надійності та ефективності, а також оцінку відповідності функціональних вимог. Важливим аспектом оцінки якості є також зворотній зв'язок користувачів та їх задоволеність використанням програмного забезпечення.

Загальна мета тестування програмного забезпечення та оцінки його якості полягає в забезпеченні надійності, функціональності та задоволеності користувачів. Це допомагає покращити продукт, знизити ризики використання та забезпечити успішне функціонування програмного забезпечення в довгостроковій перспективі.

В розробці цього продукту я використовував Ручне тестування.

Ручне тестування [4] – це тип тестування програмного забезпечення, при якому тестовий кейс виконується тестувальником вручну без допомоги будь-яких автоматизованих інструментів.

Результати тестувань модна побачити на рисунках 3.2.1-3.2.12.

Отже, після успішного тестування гри і аналізу представлених результатів можна зробити висновок, що гра має добре реалізований ігровий інтерфейс, а також багато різноманітних сценаріїв для здійснення різних дій та повідомлення гравцям про стан гри. Крім того, функції гри є зручними та простими у використанні.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Характеристика життєдіяльності людини у системі “людина – машина – середовище існування”

Основним завданням цього розділу є детальний аналіз та характеристика життєдіяльності людини в контексті взаємозв'язку "людина - машина - середовище існування". Це включає вивчення впливу робочих умов, фізіологічних та психологічних особливостей людини, а також властивостей машин та оточуючого середовища на безпеку та здоров'я працівників. Дане завдання спрямоване на забезпечення безпеки праці, покращення умов праці та підвищення якості життя працівників [5].

Людина:

- Фізіологічні особливості організму людини: Досліджується вплив фізіологічних параметрів, таких як системи дихання, кровообігу, нервової системи, на пристосування людини до робочих умов та виявлення можливих фізіологічних обмежень.
- Психологічні аспекти: Розглядаються психологічні фактори, такі як стан емоційної стабільності, концентрація уваги, пам'ять, реакції на стрес, які можуть впливати на безпеку праці та продуктивність працівника.

Машина:

- Властивості та характеристики машин: Вивчається рівень автоматизації, надійності та безпеки машин, включаючи їхню конструкцію, ергономіку, системи управління та заходи безпеки, що забезпечують безпечну експлуатацію та взаємодію з працівником.
- Вплив машин на працівника: Аналізується вплив машин на фізичне та психологічне становище працівника, такий як фізичне навантаження, шум, вібрація, електромагнітні поля, освітлення тощо.
- Організація робочого місця: Розглядаються аспекти розташування обладнання, інструментів та устаткування на робочому місці з урахуванням

безпеки та ефективності праці. Аналізується раціональне використання простору, зручність доступу до необхідних елементів та заходи з покращення організації робочого простору.

Середовище існування:

- Вплив робочого середовища [6]: Вивчається вплив таких факторів, як температура, вологість, шкідливі речовини, пил, гази, радіація, на безпеку та здоров'я працівника, а також заходи з контролю та забезпечення безпеки.
- Ергономічні аспекти: Розглядаються аспекти організації робочого місця, розташування обладнання, меблів, інструментів, а також раціональне використання фізичних та психологічних можливостей працівника.

Детальний аналіз характеристики життєдіяльності людини у системі "людина – машина – середовище існування" дозволяє розуміти взаємозв'язок та взаємодію між цими елементами. Це сприяє розробленню та впровадженню ефективних заходів з охорони праці, покращенню умов праці та забезпеченню безпеки працівників. Знання про фізіологічні та психологічні особливості людини, характеристики машин та вплив робочого середовища сприяють зменшенню ризиків травматизму, захворювань та покращенню загального стану працівників, що впливає на їхню продуктивність та якість життя.

4.2 Вимоги безпеки до робочих місць користувачів ПК

Розробка програмного забезпечення - це складний процес, який вимагає не тільки професійних знань, але й забезпечення безпеки працівників. Метою цього завдання є визначення вимог безпеки до робочого місця розробника програмного забезпечення з метою запобігання можливим небезпекам та забезпечення комфорту та ефективності роботи.

Сучасні нормативні документи, що встановлюють вимоги безпеки праці під час використання працівниками персональних комп'ютерів, включають такі

офіційні акти:

- Відповідно до НПАОП 0.00-7.15-18 "Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями", розробникам програмного забезпечення рекомендується забезпечувати належні умови роботи з комп'ютером, включаючи правильне розташування робочого столу та стільця, належне освітлення та регулярні перерви для відпочинку очей [7].
- Відповідно до НПАОП 0.00-1.28-10 " Правила охорони праці під час експлуатації електронно-обчислювальних машин", розробникам програмного забезпечення слід дотримуватись правил безпеки для роботи з електроприладами, використовувати лише сертифіковане обладнання та забезпечувати його правильне розташування та заземлення [8].
- Згідно з ДСанПІН 3.3.2.007-98 "Вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин", важливо забезпечити правильну позицію тіла під час роботи з клавіатурою, мишею та іншими пристроями, а також встановити екран комп'ютера на відповідній висоті та під кутом, що не сприяє напруженню м'язів та виникненню захворювань опорно-рухового апарату [9].
- Згідно з ДСанПІН 3.3.2.007-98 "Вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин" необхідно забезпечити належне освітлення робочого місця та екрану комп'ютера, а також регулярно проводити перерви для відпочинку очей під час тривалої роботи перед екраном [9].
- Згідно з НПАОП 0.00-1.28-10 " Правила охорони праці під час експлуатації електронно-обчислювальних машин ", розробникам програмного забезпечення слід дотримуватись правил безпеки психологічного комфорту працівників, уникати стресових ситуацій та створювати позитивне робоче середовище [8].
- Захист інформації є важливим аспектом роботи розробника програмного забезпечення. Відповідно до закону України "Про захист

персональних даних", розробники мають забезпечувати захист конфіденційної та важливої інформації від несанкціонованого доступу та крадіжки.

Розглянемо наступні правила безпечної роботи на комп’ютері [10]:

- У приміщені щодня потрібно виконувати вологе прибирання.
- Приміщення потрібно провітрювати щогодини.
- Після кожної години роботи рекомендується робити перерву протяжністю в десять хвилин. Безперервна робота за комп’ютером для дорослої людини не повинна перевищувати двох годин. Під час перерви не варто сидіти в телефоні чи в якімось іншім електроннім пристрої.
- Постійно слідкуйте за станом монітора: він має бути чистим, без плям та пилу. Крім того, слідкуйте за чистотою своїх окулярів.
- Слідкуйте за поставою: ноги твердо стоять на підлозі чи на спеціальній підставці; стегна розташовані під прямим кутом до тулуба, а гомілки – під прямим кутом до стегон; сидіти потрібно прямо або злегка нахилившись вперед; пальці рук знаходяться на рівні зап’ястків або трохи нижче – у такому положенні вони найбільш рухливі; плечі мають бути розслаблені та вільно опущені, що сприяє розслабленню рук; відстань від очей до екрану монітора – не менше 55-60 см; центр екрану має знаходитися на рівні очей чи трохи нижче; рекомендується хоча б раз на день виконувати гімнастику для очей.
- Щоб попередити “синдром сухого ока”, моргайте кожні 3-5 секунд.
- Не використовуйте телевізор замість монітора, так як його випромінювання практично у сто разів перевищує випромінювання монітора.
- Обов’язково слідкуйте за диханням: воно має бути рівномірним, без затримок.
- Якщо шрифт занадто мілкий, то треба збільшити масштаб документу (наприклад до 150% або більше).
- Якщо з’явилося відчуття втоми, напруження, сонливість, тяжкості в очах, потрібно припинити роботу та хоча б трохи відпочити.

Відшкодування підприємствам, громадянам і державі збитків, завданих

порушенням вимог щодо охорони праці (ст. ЗО). Крім відшкодування шкоди працівникам (ст. 11) власник повністю відшкодовує збитки іншим підприємствам, громадянам і державі на загальних підставах у зв'язку з завданням шкоди при порушенні вимог щодо охорони праці [11]

Виконання вимог безпеки до робочих місць є необхідним та важливим аспектом забезпечення безпеки праці та збереження фізичного та психологічного здоров'я працівників. Це допомагає запобігти можливим травмам та захворюванням, які можуть виникнути в результаті некоректно організованого робочого місця. Дотримання вимог безпеки також сприяє підвищенню продуктивності та якості працівника.

Кожен робітник та організація, що забезпечує робоче місце, несуть відповідальність за дотримання цих вимог безпеки. Це включає в себе належне організування робочого простору, дотримання правил електробезпеки, забезпечення комфорту та безпеки під час взаємодії з комп'ютером, охорону зору, психологічну безпеку та захист інформації.

Розуміння та виконання цих вимог безпеки є важливою складовою успішного та відповідального професійного підходу до розробки програмного забезпечення. Це дозволяє створювати безпечне та продуктивне робоче середовище для працівника, сприяє його здоров'ю та добробуту, а також покращує якість розробки продукту і впливає на загальний успіх організації.

ВИСНОВКИ

Гра "Who Am I" є результатом моєї роботи як розробника і дизайнера, і я впевнений у її успішності та потенціалі. Результати тестування та аналіз представлених рисунків свідчать про високу якість реалізації користувацького інтерфейсу та відмінність у дизайні. Ця гра демонструє увесь мої творчий потенціал, знання процесу розробки та увагу до деталей.

Ігровий інтерфейс "Who Am I" пропонує комфортну та зручну навігацію, що дозволяє гравцям відчувати себе затягнутими у гру. Різноманітність передбачених сценаріїв для здійснення різних дій та повідомлення гравцям про стан гри забезпечує належну взаємодію між гравцями та грою. Дизайн гри також підкреслює її характер та внутрішню концепцію, створюючи неповторний візуальний досвід.

Гра "Who Am I" має потенціал привернути широке коло гравців і забезпечити їм задоволення від геймплею. Вона сполучає у собі розроблений інтерфейс, захоплюючий ігровий процес та естетичний дизайн, що створює цілісний та привабливий продукт. Результати тестування, представлені у вигляді рисунків, підтверджують високу якість гри та її вміння задовольнити гравців різного рівня інтересів та очікувань.

Гра "Who Am I" втілює моє бачення ідеальної гри, яка володіє якісним дизайном і захоплюючим ігровим досвідом. Я пишаюся своїм внеском як розробник і дизайнер у цей проект, і вірю, що гра зможе сподобатися багатьом гравцям та стати популярною. Завдяки вдалим тестуванням та детальному аналізу результатів, я маю впевненість у тому, що "Who Am I" є вдалим продуктом, який принесе задоволення і радість гравцям.

ПЕРЕЛІК ПОСИЛАНЬ

1. Про компанію CGE Digital. [Електронний ресурс] URL: https://czechgames.com/en/about-us/
2. Документація Spring Boot. [Електронний ресурс] URL: https://spring.io/projects/spring-boot
3. Основні правила гри. [електронний ресурс] URL: https://gameonfamily.com/who-am-i-game/
4. Про тестування. [Електронний ресурс] URL: https://www.zaptest.com/uk/ручне-тестування-що-це-таке-типи-проц
5. Закон України “Основи законодавства України про охорону здоров'я”. [Електронний ресурс] URL: https://zakon.rada.gov.ua/laws/show/2801-12#
6. Безпека життєдіяльності – Березюк О. В., Лемешев М. С, 90 с.
7. НПАОП 0.00-7.15-18 "Про затвердження вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями". [Електронний ресурс] URL: https://zakon.rada.gov.ua/go/z0508-18
8. НПАОП 0.00-1.28-10 "Правила охорони праці під час експлуатації електронно-обчислювальних машин". [Електронний ресурс] URL: http://online.budstandart.com/ua/catalog/doc-page?id_doc=27405
9. ДСанПІН 3.3.2.007-98 "Вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин". [Електронний ресурс] URL: https://zakon.rada.gov.ua/rada/show/v0007282-98
10. Правила безпечної роботи на комп'ютері [Електронний ресурс] URL: https://pedcollege.kiev.ua/index.php/77-robota-koledzhu/okhorona-pratsi/589-pravyla-bezpechnoi-roboty-na-kompiuteri.
11. В. Ц. Жидецький, В. С. Джигирей, О. В. Мельников. Основи охорони праці. — Вид. 2-е, стериотипне. — Львів: Афіша, 2000., 24 с.

ДОДАТОК А

Програмний код

Програмний код Класу GameController

```
@RestController
@RequestMapping("/games")
@RequiredArgsConstructor
public class GameController {

    private final GameService gameService;

    @GetMapping
    public List<PersistentGame> findAvailableGames(@RequestHeader(PLAYER) String
player) {
        return this.gameService.findAvailableGames();
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public GameDetails createGame(@RequestHeader(PLAYER) String player,
                                  @Valid @RequestBody NewGameRequest gameRequest)
{
        return this.gameService.createGame(player, gameRequest);
    }

    @GetMapping("/{id}")
    public ResponseEntity<GameDetails> findById(@PathVariable("id") String id) {
        return ResponseEntity.ok(this.gameService.findGameById(id));
    }

    @PostMapping("/{id}/players")
    public PlayerDetails enrollToGame(@PathVariable("id") String id,
                                      @RequestHeader(PLAYER) String player) {
        return this.gameService.enrollToGame(id, player);
    }

    @PostMapping("/{id}/characters")
    @ResponseStatus(HttpStatus.OK)
    public void suggestCharacter(@PathVariable("id") String id,
                                 @RequestHeader(PLAYER) String player,
                                 @Valid @RequestBody CharacterSuggestion
suggestion) {
        this.gameService.suggestCharacter(id, player, suggestion);
    }

    @GetMapping("/{id}/turn")
    public ResponseEntity<TurnDetails> findTurnInfo(@PathVariable("id") String id,
                                                    @RequestHeader(PLAYER) String
player) {
        return this.gameService.findTurnInfo(id, player)
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }

    @PostMapping("/{id}/questions")
    public void askQuestion(@PathVariable("id") String id,
                            @RequestHeader(PLAYER) String player, @Valid
```

```
@RequestBody Message message) {
        this.gameService.askQuestion(id, player, message.getMessage());
    }

    @PostMapping("/{id}/guess")
    public void submitGuess(@PathVariable("id") String id,
                            @RequestHeader(PLAYER) String player, @Valid
@RequestBody Message message) {
        this.gameService.submitGuess(id, player, message);
    }

    @PostMapping("/{id}/answer")
    public void answerQuestion(@PathVariable("id") String id,
                               @RequestHeader(PLAYER) String player, @RequestParam
QuestionAnswer answer) {
        this.gameService.answerQuestion(id, player, answer);

    }

    @PostMapping("/{id}/guess/answer")
    public void answerGuessingQuestion(@PathVariable("id") String id,
                                       @RequestHeader(PLAYER) String player,
@RequestParam QuestionAnswer answer) {
        this.gameService.answerGuessingQuestion(id, player, answer);
    }

    @GetMapping("/{id}/history")
    public ResponseEntity<HistoryChat> getGameHistory(@PathVariable("id") String
id) {
        return ResponseEntity.ok(this.gameService.gameHistory(id));
    }

    @DeleteMapping("/{id}/leave")
    public void leaveGame(@PathVariable("id") String id,
                          @RequestHeader(PLAYER) String player) {
        this.gameService.leaveGame(id, player);
    }

    @GetMapping("/all-players-count")
    public int showPlayers() {
        return this.gameService.getAllPlayers();
    }

    @GetMapping("/{id}/questions/getQuestion")
    public String getCurrentQuestion(@PathVariable("id") String id,
                                     @RequestHeader(PLAYER) String player) {
        return this.gameService.getCurrentQuestion(id, player);
    }

    @GetMapping("/{id}/answer/getAnswer")
    public String getCurrentAnswer(@PathVariable("id") String id,
                                   @RequestHeader(PLAYER) String player) {
        return this.gameService.getCurrentAnswer(id, player);
    }

    @PostMapping("/{id}/inactivePlayer")
    public boolean inactivePlayer(@PathVariable("id") String id,
                                  @RequestHeader(PLAYER) String player) {
        return this.gameService.inactivePlayer(id, player);
    }

    @GetMapping("/all")
    public List<PersistentGame> findAllGames(@RequestHeader(PLAYER) String
```

```
player){
        return this.gameService.findAllGames();
    }

    @PostMapping("/clear")
    public String clearGames(@RequestHeader(PLAYER) String player){
        this.gameService.clear();
        return "Games list cleared";
    }
}
```

Реалізація класу GameServiceImpl

```
@Service
@RequiredArgsConstructor
public class GameServiceImpl implements GameService {

    private final GameRepository gameRepository;

    @Override
    public List<PersistentGame> findAvailableGames() {
        return this.gameRepository.findAllAvailable();
    }

    @Override
    public GameDetails findGameById(String id) {
        return new GameDetails(checkGameExistence(id));
    }

    @Override
    public Optional<TurnDetails> findTurnInfo(String id, String player) {
        PersistentGame game = checkGameExistence(id);
        return Optional.of(game.getTurnDetails());
    }

    @Override
    public GameDetails createGame(String playerId, NewGameRequest gameRequest) {
        List<PersistentGame> availableGames = findAvailableGames();
        if (availableGames.isEmpty()) {
            PersistentGame game = new PersistentGame(playerId,
gameRequest.getMaxPlayers());
            return new GameDetails(gameRepository.save(game));
        } else {
            PersistentGame game =
checkGameExistence(availableGames.get(0).getGameId());
            if (game.getStatus().equals(GameStatus.WAITING_FOR_PLAYERS)) {
                game.enrollToGame(playerId);
            } else {
                throw new GameStateException("You cannot enroll to this game! All
player slots are taken");
            }
            return new GameDetails(game);
        }
    }

    @Override
    public PlayerDetails enrollToGame(String gameId, String playerId) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.getStatus().equals(GameStatus.WAITING_FOR_PLAYERS)) {
            return game.enrollToGame(playerId);
        }
        throw new GameStateException("You cannot enroll to this game! All player
```

```
slots are taken");
    }

    @Override
    public void suggestCharacter(String gameId, String player, CharacterSuggestion
suggestion) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.getStatus().equals(GameStatus.SUGGEST_CHARACTER)) {
            game.suggestCharacter(player, suggestion);
        } else {
            throw new GameStateException("You cannot suggest the character!
Current game state is: " + game.getStatus());
        }
        if (game.areAllPlayersSuggested()) {
            startGame(gameId);
        }
    }

    @Override
    public Optional<GameDetails> startGame(String gameId) {
        PersistentGame game = checkGameExistence(gameId);
        switch (game.getStatus()) {

            case GAME_IN_PROGRESS -> throw new GameStateException("Game already in
progress! Find another one to play!");

            case READY_TO_PLAY -> {
                game.startGame();
                return Optional.of(new GameDetails(game));
            }

            case SUGGEST_CHARACTER -> throw new GameStateException("Game can not
be started! Players suggesting characters! " +
                    "Waiting for other players to contribute their characters" +
                    "Players left: " +
                    game.getPLayers()
                            .stream()
                            .filter(randomPlayer ->
!randomPlayer.isSuggestStatus())
                            .map(PersistentPlayer::getNickname)
                            .collect(Collectors.toList()));

            case WAITING_FOR_PLAYERS -> throw new GameStateException("Game can not
be started!" +
                    " Waiting for additional players! " +
                    "Current players number: " + game.getPLayers().size() +
game.getMaxPlayers());
            default -> throw new GameStateException("Unrecognized state!");
        }
    }

    @Override
    public void askQuestion(String gameId, String player, String message) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.getStatus().equals(GameStatus.GAME_IN_PROGRESS)) {
            game.askQuestion(player, message);
        }
    }

    @Override
    public void answerQuestion(String gameId, String player, QuestionAnswer
answer) {
        PersistentGame game = checkGameExistence(gameId);
```

```
        if (game.getStatus().equals(GameStatus.GAME_IN_PROGRESS)) {
            game.answerQuestion(player, answer);
        }
    }

    @Override
    public void submitGuess(String gameId, String player, Message guess) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.getStatus().equals(GameStatus.GAME_IN_PROGRESS)) {
            game.askGuessingQuestion(player, guess);
        }
    }


    @Override
    public void answerGuessingQuestion(String gameId, String playerId,
QuestionAnswer answerGuess) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.getStatus().equals(GameStatus.GAME_IN_PROGRESS)) {
            game.answerGuessingQuestion(playerId, answerGuess);
        }
        if (game.getPLayers().size() == 1) {
            this.gameRepository.deleteGame(gameId);
        }
    }

    private PersistentGame checkGameExistence(String gameId) {
        if (gameRepository.findById(gameId).isPresent()) {
            return gameRepository.findById(gameId).get();
        }
        throw new GameNotFoundException(String.format(ROOM_NOT_FOUND_BY_ID,
gameId));
    }

    @Override
    public HistoryChat gameHistory(String gameId) {
        PersistentGame game = checkGameExistence(gameId);
        return game.getHistory();
    }

    @Override
    public void leaveGame(String gameId, String playerId) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.getPLayers().size() == 2) {
            game.makingWinner(playerId);
        }
        game.deletePlayer(playerId);
        if (game.getPLayers().size() <= 3 &&
!game.getStatus().equals(GameStatus.GAME_IN_PROGRESS) &&
!game.getStatus().equals(GameStatus.WAITING_FOR_PLAYERS))
            this.gameRepository.quickDeleteGame(gameId);
        if (game.getPLayers().size() == 1 &&
!game.getStatus().equals(GameStatus.WAITING_FOR_PLAYERS))
            this.gameRepository.deleteGame(gameId);
    }

    @Override
    public List<PersistentGame> findAllGames() {
        return this.gameRepository.findAllGames();
    }

    @Override
    public int getAllPlayers() {
```

```
        List<PersistentGame> allGames = findAllGames();
        int allPlayers = 0;
        if (!allGames.isEmpty()) {
            for (var game : allGames) {
                if (game.getPLayers() != null) {
                    allPlayers += game.getPLayers().size();
                }
            }
        }
        return allPlayers;
    }

    @Override
    public String getCurrentQuestion(String gameId, String playerId) {
        PersistentGame game = checkGameExistence(gameId);
        return game.getCurrentQuestion(playerId);
    }

    @Override
    public String getCurrentAnswer(String gameId, String playerId) {
        PersistentGame game = checkGameExistence(gameId);
        return game.getCurrentAnswer(playerId);
    }

    @Override
    public boolean inactivePlayer(String gameId, String playerId) {
        PersistentGame game = checkGameExistence(gameId);
        if (game.inactivePlayer(playerId)) {
            leaveGame(gameId, playerId);
            return true;
        }
        return false;
    }

    @Override
    public void clear() {
        this.gameRepository.clear();
    }
}
```

Реалізація інтерфейсу GameService

```
public interface GameService {

    List<PersistentGame> findAvailableGames();

    GameDetails createGame(String player, NewGameRequest gameRequest);

    PlayerDetails enrollToGame(String gameId, String playerId);

    GameDetails findGameById(String id);

    void suggestCharacter(String gameId, String player, CharacterSuggestion
suggestion);

    Optional<GameDetails> startGame(String gameId);

    void askQuestion(String gameId, String player, String message);

    Optional<TurnDetails> findTurnInfo(String id, String player);

    void submitGuess(String id, String player, Message guess);
```

```
    void answerQuestion(String id, String player, QuestionAnswer answer);

    HistoryChat gameHistory(String gameId);

    void answerGuessingQuestion(String id, String playerId, QuestionAnswer
answer);

    void leaveGame(String gameId, String playerId);

    int getAllPlayers();

    List<PersistentGame> findAllGames();

    String getCurrentQuestion(String gameId, String playerId);

    String getCurrentAnswer(String gameId, String playerId);

    boolean inactivePlayer(String gameId, String playerId);

    void clear();
}
```

Реалізація класу GameInMemoryRepository

```
@Repository
public class GameInMemoryRepository implements GameRepository {

    private final List<PersistentGame> games = new ArrayList<>();

    @Override
    public List<PersistentGame> findAllAvailable() {
        return this.games.stream()
                .filter(game ->
game.getStatus().equals(GameStatus.WAITING_FOR_PLAYERS))
                .collect(Collectors.toList());
    }

    @Override
    public PersistentGame save(PersistentGame game) {
        this.games.add(game);
        return game;
    }

    @Override
    public Optional<PersistentGame> findById(String gameId) {
        return Optional.ofNullable(this.games
                .stream()
                .filter(game -> game.getGameId().equals(gameId))
                .findFirst()
                .orElseThrow(() -> new GameNotFoundException("Game not found!")));
    }

    @Override
    public List<PersistentGame> findAllGames() {
        return this.games;
    }

    @Override
    public void deleteGame(String gameId) {
        try {
            Thread.sleep(7_000);
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        this.games.removeIf(game -> game.getGameId().equals(gameId));
    }

    public void quickDeleteGame (String gameId){
        this.games.removeIf(game -> game.getGameId().equals(gameId));
    }

    @Override
    public void clear() {
        this.games.clear();
    }

}
```

Реалізація інтерфейсу GameRepository

```
public interface GameRepository {

    List<PersistentGame> findAllAvailable();

    PersistentGame save(PersistentGame game);

    Optional<PersistentGame> findById(String id);

    List<PersistentGame> findAllGames();

    void deleteGame(String gameId);

    void quickDeleteGame(String gameId);

    void clear();
}
```

Реалізація класу PersistentGame

```
public class PersistentGame {

    private final String id;
    private final List<PersistentPlayer> players;
    private final int maxPlayers;
    private GameStatus gameStatus = GameStatus.WAITING_FOR_PLAYERS;
    private final List<PersistentPlayer> winners = new LinkedList<>();
    private Turn turn;
    private final HistoryChat history = new HistoryChat();
    private int playersLeft;
    private final List<QuestionAnswer> playersAnswers = new ArrayList<>();

    /**
     * Creates a new game (game room) and makes a first enrolment turn by a
current player
     * so that he won't have to enroll to the game he created
     *
     * @param hostPlayer player to initiate a new game
     */
    public PersistentGame(String hostPlayer, Integer maxPlayers) {
        this.id = String.format("%d-%d",
                Instant.now().toEpochMilli(),
                Double.valueOf(Math.random() * 999).intValue());
```

```
        this.players = new ArrayList<>(maxPlayers);
        this.maxPlayers = maxPlayers;
        this.players.add(new PersistentPlayer(hostPlayer, id, "Player-1"));
        this.turn = new TurnImpl(players);
        this.playersLeft = maxPlayers;
    }

    public String getGameId() {
        return this.id;
    }

    public int getMaxPlayers() {
        return maxPlayers;
    }

    public List<PersistentPlayer> getPLayers() {
        return players;
    }

    public GameStatus getStatus() {
        return gameStatus;
    }


    public List<PersistentPlayer> getWinnerList() {
        return winners;
    }

    /***
     *
     * @return data about current turn (current player, and other players)
     */
    public TurnDetails getTurnDetails() {
        return new TurnDetails(turn.getCurrentPlayer(), turn.getOtherPlayers());
    }

    /***
     *
     * @return PersistentPlayer whose turn is now
     */
    public PersistentPlayer getCurrentTurn() {
        return turn.getCurrentPlayer();
    }

    public List<PersistentPlayer> getOrderedPlayers() {
        return Optional.ofNullable(this.turn)
                .map(Turn::getAllPlayers)
                .orElse(this.players);
    }

    public PlayerDetails enrollToGame(String playerId) {
        PersistentPlayer player;
        if (players.stream().noneMatch((randomPlayer ->
randomPlayer.getId().equals(playerId)))) {
            player = new PersistentPlayer(playerId, this.id, "Player-" +
(players.size() + 1));
            players.add(player);
            this.turn = new TurnImpl(players);
            if (players.size() == maxPlayers) {
                gameStatus = GameStatus.SUGGEST_CHARACTER;
            }
            return PlayerDetails.of(player);
        } else {
```

```
            throw new GameStateException("Player already enrolled in this game!");
        }
    }

    public void suggestCharacter(String playerId, CharacterSuggestion suggestion)
{
        var player = players
                .stream()
                .filter(randomPlayer -> randomPlayer.getId().equals(playerId))
                .findFirst().orElseThrow(() -> new PlayerNotFoundException("Player
not found!"));
        if (!player.isSuggestStatus()) {
            player.setCharacter(suggestion.getCharacter());
            player.setNickname(suggestion.getNickname());
            player.setSuggestStatus(true);
        } else {
            throw new GameStateException("You already suggest the character");
        }
        if (players.stream().filter(PersistentPlayer::isSuggestStatus).count() ==
maxPlayers) {
            gameStatus = GameStatus.READY_TO_PLAY;
        }
    }

    public boolean areAllPlayersSuggested() {
        for (var player : this.players) {
            if (!player.isSuggestStatus()) {
                return false;
            }
        }
        return true;
    }

    public void startGame() {
        if (players.stream().filter(PersistentPlayer::isSuggestStatus).count() ==
maxPlayers) {
            assignCharacters();
            var currentPlayer = turn.getCurrentPlayer();
            currentPlayer.setPlayerState(PlayerState.ASK_QUESTION);
            players.stream()
                    .filter(randomPlayer ->
!randomPlayer.getId().equals(currentPlayer.getId()))
                    .forEach(randomPlayer ->
randomPlayer.setPlayerState(PlayerState.ANSWER_QUESTION));
            gameStatus = GameStatus.GAME_IN_PROGRESS;
        }
    }


    public void askQuestion(String playerId, String message) {
        // TODO: Show question
        var askingPlayer = players
                .stream()
                .filter(randomPlayer -> randomPlayer.getId().equals(playerId))
                .findFirst()
                .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

        cleanPlayersValues(players);

        if (askingPlayer.getPlayerState().equals(PlayerState.ASK_QUESTION)) {
            askingPlayer.setPlayerQuestion(message);
            askingPlayer.setEnteredQuestion(true);
```

```
            this.players.stream()
                    .filter(randomPlayer ->
!randomPlayer.getId().equals(askingPlayer.getId()))
                    .forEach(randomPlayer ->
randomPlayer.setPlayerState(PlayerState.ANSWER_QUESTION));

            this.history.addQuestion(message, playerId);

        } else {
            throw new TurnException("Not your turn! Current turn has player: " +
getCurrentTurn().getNickname());
        }
    }

    public void answerQuestion(String playerId, QuestionAnswer questionAnswer) {
        //TODO: show on screen questions and answers from history
        var askingPlayer = turn.getCurrentPlayer();
        var answeringPlayer = players
                .stream()
                .filter(randomPlayer -> randomPlayer.getId().equals(playerId))
                .findFirst()
                .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

        if (askingPlayer.equals(answeringPlayer)) {
            throw new TurnException("Not your turn for answering");
        }

        if (answeringPlayer.getPlayerState().equals(PlayerState.ANSWER_QUESTION))
{
            this.playersAnswers.add(questionAnswer);
            answeringPlayer.setEnteredAnswer(true);
            answeringPlayer.setPlayerAnswer(String.valueOf(questionAnswer));

            this.history.addAnswer(questionAnswer.toString(), playerId);
        }

        if (playersAnswers.size() == this.playersLeft - 1) {
            var positiveAnswers = playersAnswers
                    .stream()
                    .filter(answer -> answer.equals(QuestionAnswer.YES) ||
answer.equals(QuestionAnswer.DONT_KNOW))
                    .collect(Collectors.toList());

            var negativeAnswers = playersAnswers
                    .stream()
                    .filter(answer -> answer.equals(QuestionAnswer.NO))
                    .collect(Collectors.toList());

            if (positiveAnswers.size() < negativeAnswers.size()) {
                this.turn = this.turn.changeTurn();
            }
            this.playersAnswers.clear();
            this.playersLeft = this.players.size();
        }
    }

    public void askGuessingQuestion(String playerId, Message guess) {
        var askingPlayer = players
                .stream()
                .filter(randomPlayer -> randomPlayer.getId().equals(playerId))
                .findFirst()
```

```
                .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

        cleanPlayersValues(players);

        if (askingPlayer.getPlayerState().equals(PlayerState.ASK_QUESTION)) {
            askingPlayer.setPlayerState(PlayerState.GUESSING);
            askingPlayer.setPlayerQuestion(guess.getMessage());
            askingPlayer.setEnteredQuestion(true);
            askingPlayer.setGuessing(true);

            this.players.stream()
                    .filter(randomPlayer ->
!randomPlayer.getId().equals(askingPlayer.getId()))
                    .forEach(randomPlayer ->
randomPlayer.setPlayerState(PlayerState.ANSWER_GUESS));

            this.history.addQuestion("Guess: " + guess.getMessage(), playerId);

        } else {
            throw new TurnException("Not your turn! Current turn has player: " +
getCurrentTurn().getNickname());
        }
    }

    public void answerGuessingQuestion(String playerId, QuestionAnswer
askQuestion) {
        var askingPlayer = turn.getCurrentPlayer();
        var answeringPlayer = players
                .stream()
                .filter(randomPlayer -> randomPlayer.getId().equals(playerId))
                .findFirst()
                .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

        if (askingPlayer.equals(answeringPlayer)) {
            throw new TurnException("Not your turn for answering");
        }

        if (answeringPlayer.getPlayerState().equals(PlayerState.ANSWER_GUESS)) {
            this.playersAnswers.add(askQuestion);
            answeringPlayer.setEnteredAnswer(true);
            answeringPlayer.setPlayerAnswer(String.valueOf(askQuestion));

            this.history.addAnswer(askQuestion.toString(), playerId);
        }

        if (playersAnswers.size() == this.playersLeft - 1) {
            var afkAnswers = playersAnswers
                    .stream()
                    .filter(answer -> answer.equals(QuestionAnswer.DONT_KNOW))
                    .collect(Collectors.toList());

            if (afkAnswers.size() != playersAnswers.size()) {
                var positiveAnswers = playersAnswers
                        .stream()
                        .filter(answer -> answer.equals(QuestionAnswer.YES))
                        .collect(Collectors.toList());

                var negativeAnswers = playersAnswers
                        .stream()
                        .filter(answer -> answer.equals(QuestionAnswer.NO))
                        .collect(Collectors.toList());
```

```
                if (positiveAnswers.size() >= negativeAnswers.size()) {
                    askingPlayer.setPlayerState(PlayerState.GAME_WINNER);
                    this.winners.add(askingPlayer);
                    deletePlayer(askingPlayer.getId());
                    if (this.players.size() == 1) {
                        players.get(0).setPlayerState(PlayerState.GAME_LOOSER);
                    }
                } else {
                    this.turn = this.turn.changeTurn();
                }
            } else {
                askingPlayer.setPlayerState(PlayerState.ASK_QUESTION);
            }
            this.playersAnswers.clear();
            this.playersLeft = this.players.size();
        }
    }

    public void deletePlayer(String playerId) {
        if (gameStatus.equals(GameStatus.GAME_IN_PROGRESS)) {
            var leavingPlayer = this.players
                    .stream()
                    .filter(player -> player.getId().equals(playerId))
                    .findFirst()
                    .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

            if (!leavingPlayer.isEnteredAnswer() &&
!leavingPlayer.isEnteredQuestion() &&
this.turn.getCurrentPlayer().isEnteredQuestion()) {
                if
(leavingPlayer.getPlayerState().equals(PlayerState.ANSWER_QUESTION)) {
                    answerQuestion(playerId, QuestionAnswer.DONT_KNOW);
                }
                if
(leavingPlayer.getPlayerState().equals(PlayerState.ANSWER_GUESS)) {
                    answerGuessingQuestion(playerId, QuestionAnswer.DONT_KNOW);
                }
            }
        }
        if(!this.turn.getCurrentPlayer().isEnteredQuestion()){
            this.playersLeft = this.players.size() - 1;
        }

        this.players.removeIf(player -> player.getId().equals(playerId));
        this.turn.removePLayer(playerId);
    }


    private void assignCharacters() {
        var availableCharacters = players.stream()
                .map(PersistentPlayer::getCharacter)
                .collect(Collectors.toList());

        final var random = new Random();

        for (int i = availableCharacters.size() - 1; i >= 1; i--) {
            Collections.swap(availableCharacters, i, random.nextInt(i + 1));
        }

        for (int i = 0; i <= players.size() - 1; i++) {
            int j = 0;
```

```
            var player = players.get(i);
            if (availableCharacters.size() == 1) {
                if (player.getCharacter().equals(availableCharacters.get(j))) {
                    int randomNum = (int) (Math.random() * players.size() - 1);
                    player.setCharacter(players.get(randomNum).getCharacter());

players.get(randomNum).setCharacter(availableCharacters.get(j));
                } else {
                    player.setCharacter(availableCharacters.get(j));
                }
            } else {
                while (player.getCharacter().equals(availableCharacters.get(j)) &&
j < availableCharacters.size() - 1) {
                    j++;
                }
                player.setCharacter(availableCharacters.get(j));
            }
            availableCharacters.remove(j);
        }

    }

    private void cleanPlayersValues(List<PersistentPlayer> players) {
        players.forEach(randomPlayer -> {
            randomPlayer.setEnteredAnswer(false);
            randomPlayer.setEnteredQuestion(false);
            randomPlayer.setPlayerQuestion(null);
            randomPlayer.setPlayerAnswer(null);
        });
    }

    public HistoryChat getHistory() {
        return history;
    }

    public String getCurrentQuestion(String playerId) {
        for (var player : this.players) {
            if (player != null && player.getId().equals(playerId)) {
                if (player.getPlayerQuestion() == null) {
                    return "Player " + player.getNickname() + " didn't ask";
                }
                return player.getNickname() + " asked: " +
player.getPlayerQuestion();
            }
        }
        throw new PlayerNotFoundException("Player is not found");
    }

    public String getCurrentAnswer(String playerId) {
        for (var player : this.players) {
            if (player != null && player.getId().equals(playerId)) {
                if (player.getPlayerAnswer() == null) {
                    return "Player " + player.getNickname() + " didn't answer";
                }
                return player.getNickname() + " answered: " +
player.getPlayerAnswer();
            }
        }
        throw new PlayerNotFoundException("Player is not found");
    }

    public boolean inactivePlayer(String playerId) {
        var inactivePlayer = this.players
```

```
                .stream()
                .filter(player -> player.getId().equals(playerId))
                .findFirst()
                .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

        if (inactivePlayer.getPlayerState().equals(PlayerState.ANSWER_QUESTION) ||
inactivePlayer.getPlayerState().equals(PlayerState.ANSWER_GUESS)) {
            if (this.turn.getCurrentPlayer().isEnteredQuestion()) {
                inactivePlayer.incrementInactiveCounter();

                if
(this.turn.getCurrentPlayer().getPlayerState().equals(PlayerState.ASK_QUESTION)) {
                    answerQuestion(playerId, QuestionAnswer.DONT_KNOW);
                } else if
(this.turn.getCurrentPlayer().getPlayerState().equals(PlayerState.GUESSING)) {
                    answerGuessingQuestion(playerId, QuestionAnswer.DONT_KNOW);
                }
            }
            return inactivePlayer.getInactiveCounter() == 3;
        }
        if(inactivePlayer.getPlayerState().equals(PlayerState.ASK_QUESTION) ||
inactivePlayer.getPlayerState().equals(PlayerState.GUESSING)){
            this.playersLeft = this.players.size() - 1;
        }
        return true;
    }


    public void makingWinner(String playerId) {
        var leavingPlayer = this.players
                .stream()
                .filter(player -> player.getId().equals(playerId))
                .findFirst()
                .orElseThrow(() -> new
PlayerNotFoundException(String.format(PLAYER_NOT_FOUND, playerId)));

        if (!(leavingPlayer.getPlayerState().equals(PlayerState.GAME_WINNER))) {
            for (var player : this.players) {
                if (!player.getId().equals(playerId)) {
                    player.setPlayerState(PlayerState.GAME_WINNER);
                    this.winners.add(player);
                }
            }
        }
    }

}
```

ДОДАТОК Б

Диск з розробленою програмою