LambdaChess

Software Requirements Specification

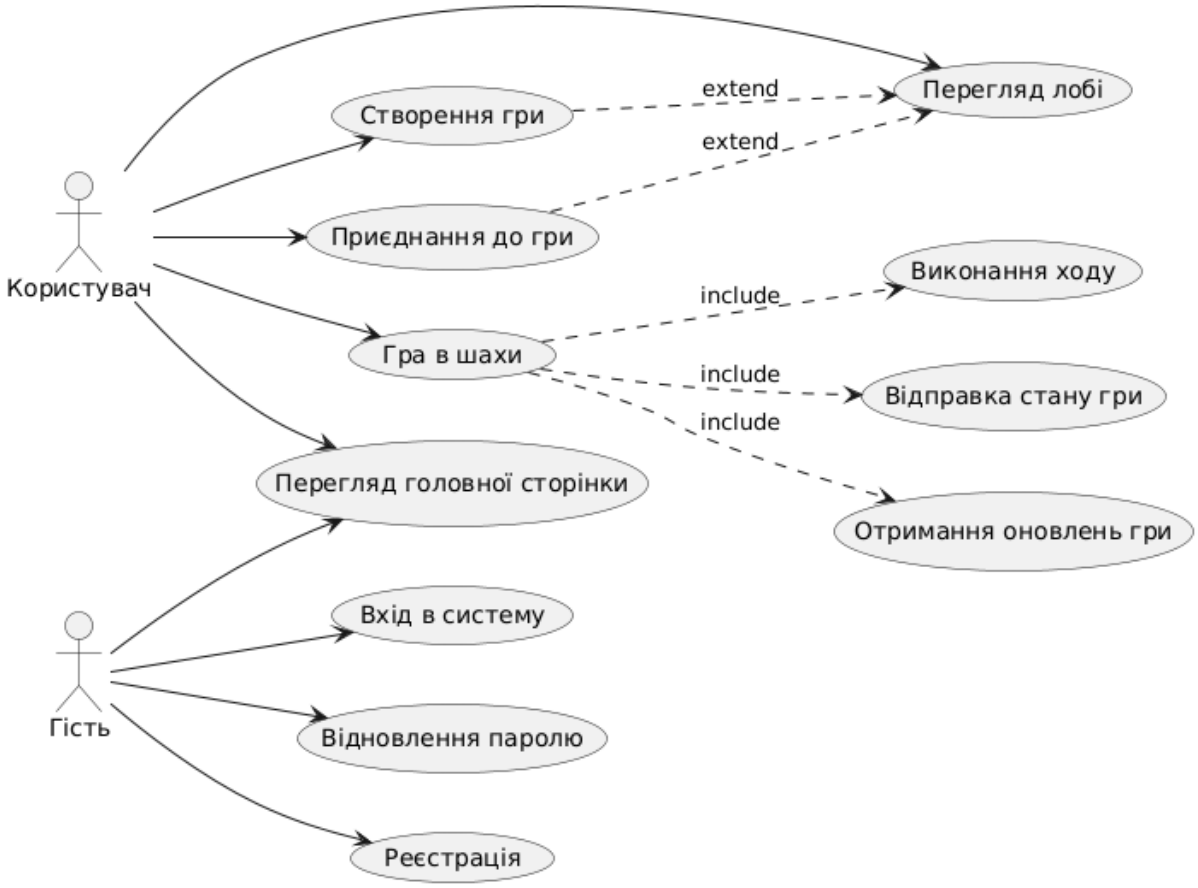
Опис системи

LambdaChess — це веб-застосунок для онлайн-гри в шахи, який дозволяє користувачам грати партії в реальному часі, переглядати історію ігор та керувати своїм профілем. Система забезпечує безпечну аутентифікацію, авторизацію на основі ролей та інтерактивний ігровий досвід через веб-інтерфейс.

Ролі в системі

- **Гість** — неавторизований користувач, який може переглядати головну сторінку, реєструватися та входити в систему
- **Користувач** — авторизований користувач, який може створювати та приєднуватися до шахових партій

Use Case Diagram



Use Case Таблиця

Актор	*Use Case*	*Опис*
Гість	Перегляд головної сторінки	Відкрити головну сторінку: Перейти на "/"
	Перегляд політики конфіденційності	Переглянути політику: Натиснути посилання "Privacy"
	Перегляд сторінки помилки	Відображення помилки: Система показує сторінку з RequestId при виникненні помилки
	Реєстрація	Створення облікового запису: Натиснути "Register" → Ввести Email, Password, ConfirmPassword → Натиснути "Зареєструватись"
	Вхід в систему	Авторизація: Натиснути "Login" → Ввести Email та Password → Натиснути "Увійти"
	Вихід з системи	Завершення сесії: Натиснути "Logout" в навігації
	Відновлення паролю	Відновити доступ: Натиснути "Forgot Password" → Ввести Email → Отримати посилання на Email → Ввести новий пароль
Користувач	Перегляд лобі	Переглянути доступні ігри: Натиснути "Game" → Відкриється сторінка з списком ігор де WhitePlayer або BlackPlayer = null
	Створення гри	Створити партію: На сторінці лобі натиснути "Create Game" → POST запит до /Game/Create → Система створює GameSession з поточним користувачем як WhitePlayer
	Приєднання до гри (через лобі)	Обрати гру: Натиснути на посилання гри в лобі → Перехід на /Game?gameId={id}
	Перегляд ігрової дошки	Відкрити гру: Перейти на /Game?gameId={id} → Система показує шахову дошку, інформацію про гравців, статус гри
	Приєднання до гри (SignalR)	Автоматичне приєднання: При відкритті ігрової сторінки виконується JoinGame(gameId) → Система додає користувача до SignalR групи
	Встановлення як чорного гравця	Зайняти місце: Якщо WhitePlayer зайнятий, а BlackPlayer вільний → Система встановлює BlackPlayerId = поточний користувач
	Встановлення як білого гравця	Зайняти місце: Якщо WhitePlayer вільний → Система встановлює WhitePlayerId = поточний користувач
	Виконання ходу	Зробити хід: Клікнути на фігуру → Клікнути на цільову клітинку → Chess.js валідує хід → Оновлюється PGN
	Відправка стану гри	Синхронізація: Після ходу викликається SendPGNGameState(gameId, newPGN) → Система перевіряє що новий PGN починається з поточного
	Отримання оновлень гри	Отримати зміни: Через SignalR отримати ReceivePGNGameState(gameState) → Оновити дошку та UI
	Сповіщення про приєднання	Інформування: При JoinGame система відправляє UserJoined(username) всім учасникам групи
	Обробка помилок гри	Отримати повідомлення: При помилках система відправляє Error(message) через SignalR конкретному користувачу

Перевірка повноти гри	Контроль доступу: Якщо WhitePlayer і BlackPlayer зайняті → Система відправляє помилку "Game session is full"
Запобігання самоприєднання	Валідація: Якщо користувач вже є WhitePlayer або BlackPlayer → Система не дублює приєднання
Валідація ігрової сесії	Перевірка існування: При будь-яких діях з грою система перевіряє чи існує GameSession з вказаним ID

Функціональні вимоги

1. Аутентифікація та авторизація

- Система повинна підтримувати реєстрацію нових користувачів через ASP.NET Core Identity
- Система повинна забезпечувати безпечну авторизацію з використанням cookies
- Система повинна підтримувати базову аутентифікацію без ролей (тільки авторизовані/неавторизовані користувачі)

2. Управління користувачами

- Користувачі повинні мати базовий профіль через Identity
- Система повинна зберігати базову інформацію користувачів

3. Ігрова функціональність

- Система повинна підтримувати створення нових шахових партій через GameController
- Користувачі повинні мати можливість приєднуватися до існуючих ігор через SignalR
- Система повинна забезпечувати гру в реальному часі через GameHub
- Система повинна зберігати стан гри в форматі PGN
- Система повинна відстежувати гравців (білі/чорні) в GameSession

4. Лобі та пошук ігор

- Система повинна відображати список доступних ігор через GameController
- Система повинна показувати ігри з незаповненими місцями (WhitePlayer або BlackPlayer = null)
- Система повинна використовувати Entity Framework для отримання ігор з Include для гравців

Нефункціональні вимоги

1. Продуктивність

- Час відгуку на ходи не повинен перевищувати 100мс
- Система повинна підтримувати до 1000 одночасних користувачів
- Сторінки повинні завантажуватися не більше ніж за 3 секунди

2. Безпека

- Всі паролі повинні бути захешовані
- Система повинна захищатися від XSS та CSRF атак
- Авторизація повинна використовувати безпечні cookies

3. Надійність

- Система повинна мати uptime не менше 99%
- Повинні бути резервні копії бази даних
- Система повинна коректно обробляти обрив з'єднання

4. Зручність використання

- Інтерфейс повинен бути адаптивним для мобільних пристроїв
- Система повинна підтримувати доступність для користувачів з обмеженими можливостями
- Час навчання роботи з системою не повинен перевищувати 15 хвилин

Технічні вимоги

Архітектура

- **Стиль архітектури**: Тришарова архітектура (3-tier/Layered Architecture)
- **Фреймворк**: ASP.NET Core 8.0 MVC
- **База даних**: SQLite
- **ORM**: Entity Framework Core
- **Реальний час**: SignalR
- **Аутентифікація**: ASP.NET Core Identity

Структура проекту (3 шари)

1. **Presentation Layer (Рівень представлення)**:
   - LambdaChess.Web.UI - MVC контролери, представлення, SignalR хаби
   - LambdaChess.Web.Controllers - додаткові контролери
2. **Business Logic Layer (Рівень бізнес-логіки)**:
   - LambdaChess.BLL.Services - сервіси та бізнес-логіка
3. **Data Access Layer (Рівень доступу до даних)**:
   - LambdaChess.DAL.Models - моделі даних
   - LambdaChess.DAL.Repositories.Abstractions - інтерфейси репозиторіїв
   - LambdaChess.DAL.Repositories.Implementations - реалізації репозиторіїв

Технологічний стек

- **Backend**: C# (.NET 8)
- **Frontend**: HTML5, CSS3, JavaScript, Bootstrap
- **База даних**: SQLite
- **Контейнеризація**: Docker
- **Шахова логіка**: Chess.js, Chessboard.js

Сценарії використання

Сценарій 1: Реєстрація нового користувача

1. Гість відкриває сторінку реєстрації
2. Вводить email, пароль та ім'я користувача
3. Система валідує дані та створює новий обліковий запис
4. Користувач отримує email для підтвердження
5. Після підтвердження користувач може авторизуватися

Сценарій 2: Створення та гра в шахи

1. Авторизований користувач переходить до лобі
2. Натискає кнопку "Створити гру"
3. Система створює нову ігрову сесію
4. Інший користувач приєднується до гри
5. Розпочинається шахова партія в реальному часі
6. Гравці по черзі роблять ходи
7. Партія завершується (мат, пат, нічия або здача)

Додаткова інформація

Azure: https://lambdachess-atdwgmh4hfbkauh8.germanywestcentral-01.azurewebsites.net/
GitHub: https://github.com/DDeviak/LambdaChess.git