Solution Architecture Document

1. Project Description

Purpose: The main purpose of this project is to help people quickly choose which film to watch by either picking a random one or searching for a specific title. Additionally, authenticated users can add their favorite films to the system and share them with other users.

Scope: The system's primary functions are:

- to fetch and display film details at random or via title search;
- to allow registered users to add and share their favorite films with the community.

Users: The target audience includes all individuals, regardless of age or gender, who enjoy watching, discovering, and sharing films.

Key Features: The system includes several features:

- display random information about a film;
- search for a film by title;
- user registration and login;
- add personal (user-submitted) films;
- update and delete the user's own personal account information (e.g., email, username).

2. Project Architecture

Architecture Style: The system is built using a monolithic architecture, meaning all components are part of a single, unified application.

It also follows a client-server model, where the frontend (client) communicates with the backend (server) via API requests.

Components:

- backend:
    - *programming language*: C#;
    - *platform*: .Net;
    - *framework*: ASP.Net Core;
- database: SQLite;

- frontend:
    - *programming language*: JavaScript;
    - *library*: React.
- documentation tools: Swagger, Docfx, Storybook.

Interaction Between Components: Data is transferred between the client and server via a REST API.

Diagrams:

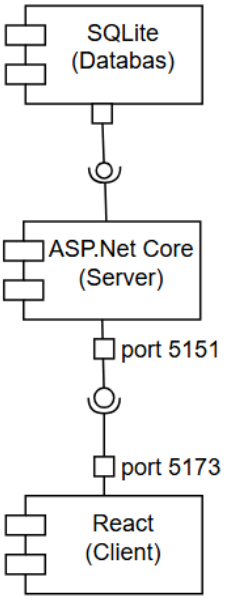

Figure 2.1 – Component Diagram

Component Diagram (fig. 2.1) includes:

- React (Client): Requires an interface from the server and communicates with it via HTTP. It runs on a development port 5173.
- ASP.NET Core (Server): Provides an interface (API) for the client and requires a connection to the database. It runs on a backend port 5151.
- SQLite (Database): Provides data access for the server through a local database connection.

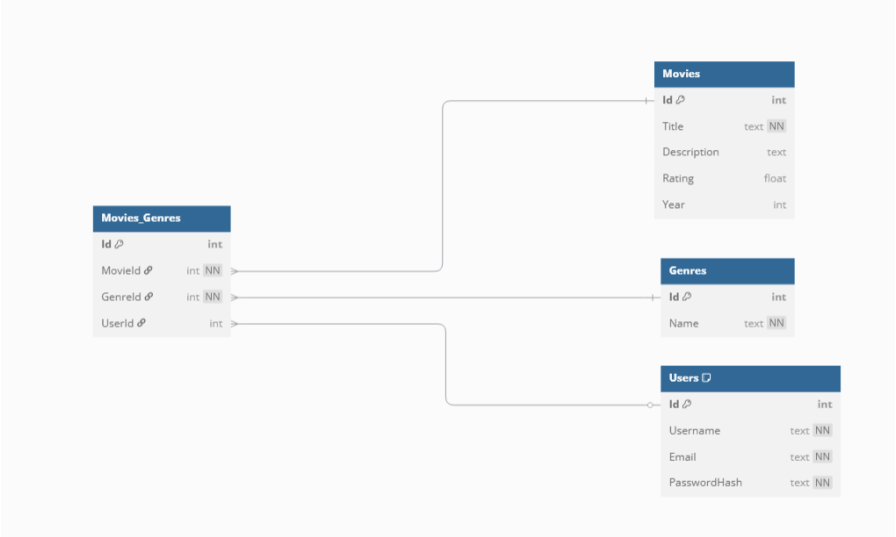

Figure 2.2 – Database Diagram

Database Diagram (fig. 2.2) contains four tables:

- *Genres*: Stores available movie genres (e.g., Action, Comedy, Drama).
- *Movies*: Stores detailed information about each movie, such as title, description, release year, and the user who added it.
- *Users*: Contains information about users registered in the system, including username, email, and password hash.
- *Movies_Genres*: A join table that implements a many-to-many relationship between movies, genres and users. Each record links one movie to one genre, assigned by one user.

3. Infrastructure Required for This Architecture

Hosting: The project is currently deployed on a local host environment.

Environment Setup:

To run and develop the system, the following environment setup is required:

Frontend:

- React.js (v19)
- Node.js (v20.x)
- Package Manager: npm
- Browser: Google Chrome or any modern browser

Backend:

- ASP.NET Core (v8.0)
- .NET SDK (v8.0 or higher)

- SQLite;

General Tools:

- Git (version control);
- Visual Studio Code or Visual Studio (IDE);

Environment Variables:

- frontend: API keys or connection strings stored in a .env file.
- backend: CORS settings and database path configured in appsettings.Development.json

CI/CD: Currently not implemented.

Monitoring & Logging:

Frontend: No monitoring or logging system currently in place.

Backend: Uses the default ASP.NET Core logging system to track warnings, errors, and runtime events.

Security: The project implements the following security measures:

- CORS policy configuration to control cross-origin requests and ensure that only trusted origins can access the backend;
- User authentication using cookie-based authentication for session management;
- Password hashing using PBKDF2 with HMAC-SHA512 to securely store user credentials.