SOFTWARE ENGINEERING

Project Report

Project by:

Root Structure

- node_modules/: Contains installed dependencies. Avoid modifying or including in version control.
- public/: Static assets like index.html, images, and icons. Do not include sensitive data here.
- src/: Core source code for the app.
    - assets/: Images, fonts, and other static files.
    - common/: Shared utilities or functions used across components.
    - components/: Modular UI components.
    - constants/: Centralized constants like API URLs and action types.
    - helpers/: Utility functions for reusable code.
    - locales/: Localization files for multi-language support.
    - pages/: Main app pages.
    - routes/: App routing configurations.
    - store/: Redux state management files.
    - auth/: Manages authentication logic.
    - Feature Modules: Organized folders for functionalities like calendar/, chat/, etc.
- **Key Files:**
    - App.js: Main app entry point for initialization and global settings.
    - config.js: API configurations; avoid exposing sensitive info here.
    - index.js: Bootstraps the app.
    - serviceWorker.js: Configures offline functionality.
    - setupTests.js: Prepares the app for testing.

Environment and Configuration

- Environment Variables (.env): Store sensitive keys (e.g., API keys, Firebase settings). Never commit .env files to version control.
- **Key Configuration Files:**
    - .gitignore: Exclude sensitive files from version control.
    - package.json: Manage dependencies and scripts. Regularly update to prevent vulnerabilities.
    - yarn.lock: Locks dependency versions for consistency.

Components Overview

1. Login Component:
    - Handles user login with form validation using Formik and Redux state for error handling.
    - Includes social login options.
2. Register Component:
    - Manages user registration with Formik and validation schemas.
    - Uses Redux for success/error messages.
3. Logout Component:

    - Logs out users by dispatching the logoutUser action and redirects to the login page.
4. UserProfile Component:
    - Allows users to view and edit their profiles.
    - Uses Formik for validation and updates Redux state.
5. ForgetPasswordPage Component:
    - Enables password reset via email submission.
    - Displays success/error messages from Redux.
6. LeadersBoard Component:
    - Displays a player table with options to add, edit, or delete entries.
    - Uses modals for form inputs and manages Redux actions for data handling.
7. eSportsHub Component:
    - Displays a paginated grid of games using Redux to fetch and store game data.
    - Includes loading spinners and pagination controls.

Steam Web API Documentation

The Steam Web API provides a set of endpoints to access data about Steam apps, users, in-game items, and more. Most API methods follow a common structure and require certain parameters to function correctly.

Common Elements

Required Parameters

- **key**: Your unique Steam Web API key. This is required for authorization and to ensure secure access. You can obtain a key here. Using an invalid or missing key will result in an HTTP 403 (forbidden) error.
- **format** (Optional): Specifies the output format for the API response. Options include:
    - **json** (default): Returns data in JSON format.
    - **xml**: Returns data in XML format.
    - **vdf**: Returns data in Valve Data Format.
- **language** (Optional): Specifies the ISO639-1 language code for the output language of all tokenized strings. If the requested language is unavailable, the API will default to English. If this parameter is omitted, tokenized strings will be returned as placeholders (e.g., "TF_Weapon_Jar").

Terminology

- **Tokenized String**: Refers to a placeholder string prefixed with # (e.g., #TF_Weapon_Jar). This string is replaced with the equivalent language-specific value from the game's Valve Data Format (VDF) file. For example, "TF_Weapon_Jar" represents "Jar Based Karate" in Team Fortress 2.

Game-Specific Interfaces

Certain games have specific API interfaces, identified by <ID>, which is the unique app ID for that game.

Generic Interfaces

- **IEconItems_<ID>**
    - **GetPlayerItems**: Lists items in a player's inventory.
    - **GetSchema**: Provides schema details for items in the game.
    - **GetSchemaURL**: Returns the URL for the item_game.txt file.
    - **GetStoreMetadata**: Retrieves metadata about the game's store.
    - **GetStoreStatus**: Returns the status of the game's store.

Dota 2 Interfaces

- **IDOTA2Match_<ID>**
    - **GetLeagueListing**: Provides information about DotaTV-supported leagues.
    - **GetLiveLeagueGames**: Lists live league matches with match details.
    - **GetMatchDetails**: Retrieves information about a particular match.
    - **GetMatchHistory**: Lists matches, with filters.
    - **GetTeamInfoByTeamID**: Returns details of teams in the game.
- **IEconDOTA2_<ID>**
    - **GetGameItems**: Retrieves Dota 2 in-game items.
    - **GetHeroes**: Lists heroes in Dota 2.
    - **GetRarities**: Lists item rarities.
    - **GetTournamentPrizePool**: Returns the prize pool for specific tournaments.

List of Game IDs

Some popular game IDs include:

- **240** - Counter-Strike: Source
- **440** - Team Fortress 2
- **570** - Dota 2

- **730** - Counter-Strike: Global Offensive
- **620** - Portal 2

For a full list, see the **GetAppList** method.

Backend

Documentation

Overview

The backend is a Node.js/Express server that acts as a bridge between the frontend and the Steam Web API. It handles user requests, manages authentication, processes data from Steam API, and communicates with the database to store and retrieve user-specific data.

Key Components

1. **Node.js**: The runtime environment for executing server-side JavaScript code.
2. **Express.js**: A web application framework for Node.js, used to create RESTful endpoints.
3. **Axios (or Fetch)**: Used to call the Steam Web API from the backend and manage data flow between the Steam API and the database.
4. **MongoDB (or SQL)**: Database for persisting user data, preferences, and app-specific information.
5. **Mongoose (or Sequelize)**: ORM for MongoDB/SQL, used to define schema and handle database operations in a structured way.
6. **Authentication**: Middleware using JWT (JSON Web Tokens) to secure access to sensitive routes.

API Endpoints

The backend provides a series of RESTful endpoints to handle requests from the frontend. Below are some key API routes and their expected functionality:

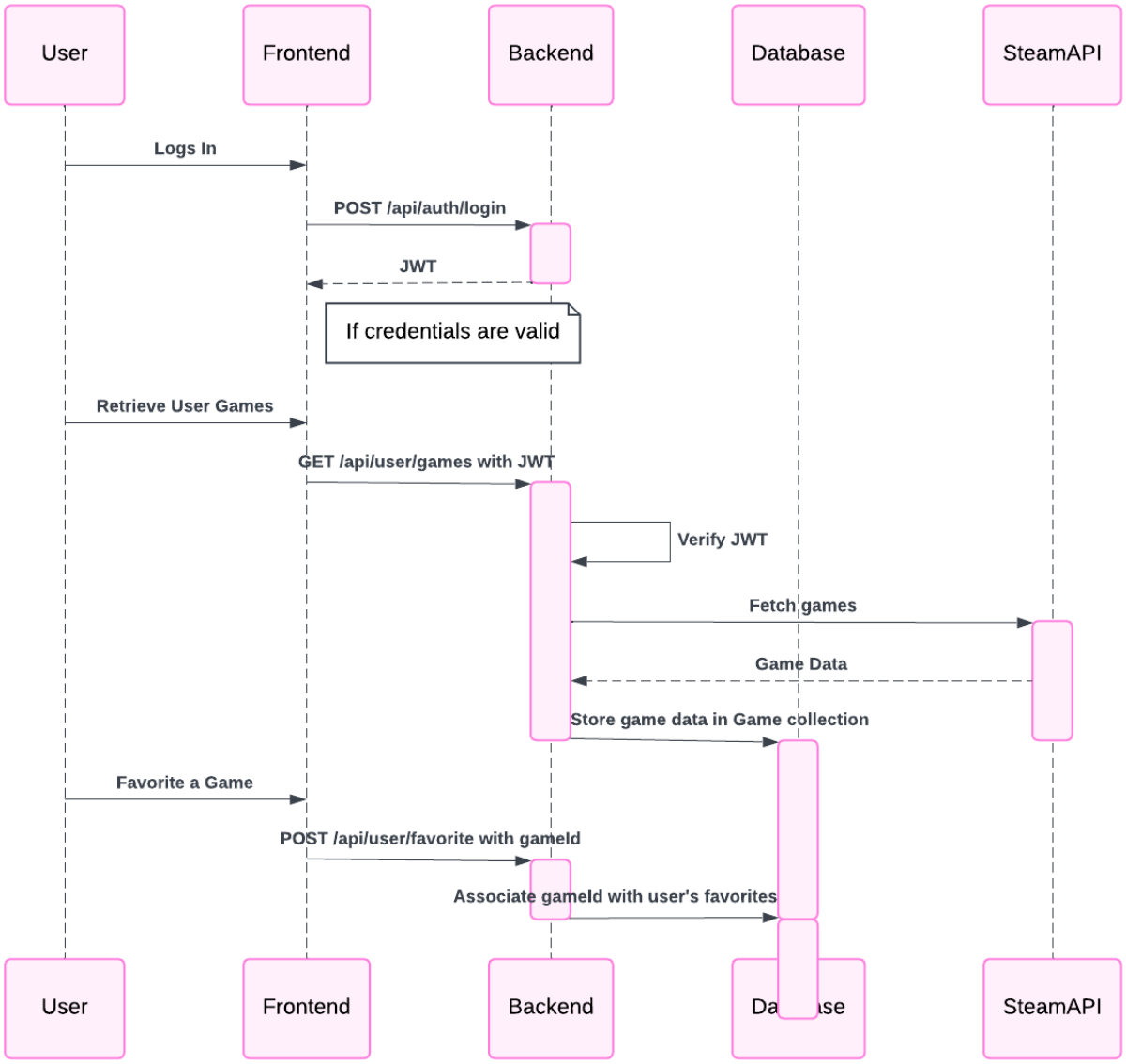
User Authentication

- **POST /api/auth/register**: Register a new user.
    - **Body**: { username, password, email }
    - **Description**: Validates and hashes the password before storing it in the database.
- **POST /api/auth/login**: Log in a user.
    - **Body**: { email, password }
    - **Description**: Verifies credentials, generates a JWT token, and returns it to the frontend.
- **POST /api/auth/logout**: Log out a user.
    - **Description**: Invalidates the token or removes it from the client.

User Profile & Game Data

- **GET /api/user/profile**: Retrieve user profile data.
    - **Headers**: { Authorization: Bearer <token> }

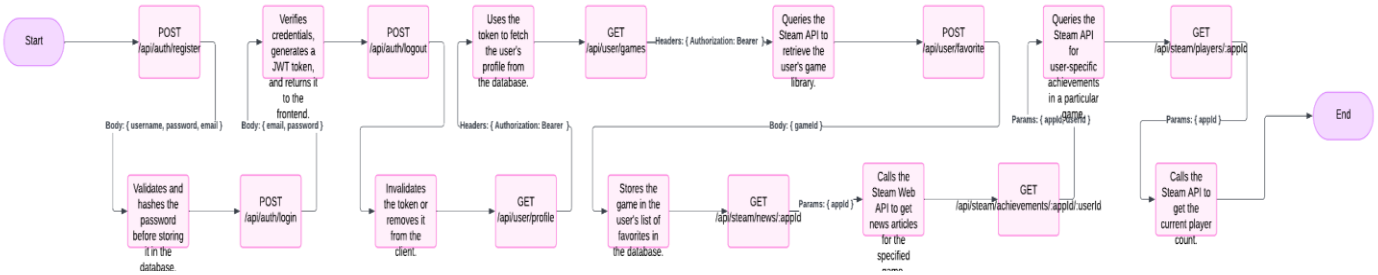
    - **Description**: Uses the token to fetch the user's profile from the database.
- **GET /api/user/games**: Get a list of games owned by the user.
    - **Headers**: { Authorization: Bearer <token> }
    - **Description**: Queries the Steam API to retrieve the user's game library.
- **POST /api/user/favorite**: Mark a game as a favorite.
    - **Body**: { gameId }
    - **Description**: Stores the game in the user's list of favorites in the database.



Steam Data & Stats

- **GET /api/steam/news/:appId**: Retrieve news for a specific game.
    - **Params**: { appId }
    - **Description**: Calls the Steam Web API to get news articles for the specified game.
- **GET /api/steam/achievements/:appId/:userId**: Retrieve user achievements for a game.

    - **Params**: { appId, userId }
    - **Description**: Queries the Steam API for user-specific achievements in a particular game.
- **GET /api/steam/players/:appId**: Retrieve the number of players currently playing a game.
    - **Params**: { appId }
    - **Description**: Calls the Steam API to get the current player count.



Middleware

- **Authentication Middleware**: Verifies JWT tokens to protect routes. Ensures only authenticated users can access sensitive data.
- **Error Handling Middleware**: Catches and manages errors, returning structured responses to the frontend to improve user experience.
- **Rate Limiting Middleware**: Limits the number of requests per user/IP to prevent abuse of Steam API requests.

Security Measures

- **JWT Authentication**: Secures endpoints by requiring valid JWT tokens for access.
- **Environment Variables**: API keys and database credentials are stored in a .env file and accessed via process.env to prevent exposure.
- **Data Validation**: Validates user input using libraries like Joi to prevent SQL injection and XSS attacks.
- **HTTPS**: Enforces HTTPS to secure data transmission.

Database Schema

Below is a proposed schema using MongoDB and Mongoose.

MongoDB (Mongoose) Schema

User Schema

```
const UserSchema = new mongoose.Schema({
  username: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  steamId: { type: String, required: true, unique: true },
  createdAt: { type: Date, default: Date.now },
  favorites: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Game' }]
});
```

Game Schema

```
const GameSchema = new mongoose.Schema({
  steamAppId: { type: Number, required: true, unique: true
  }, title: { type: String, required: true },
  genre: { type: String },
  developer: { type: String },
  releaseDate: { type: Date },
  description: { type: String },
  createdAt: { type: Date, default: Date.now },
});
```

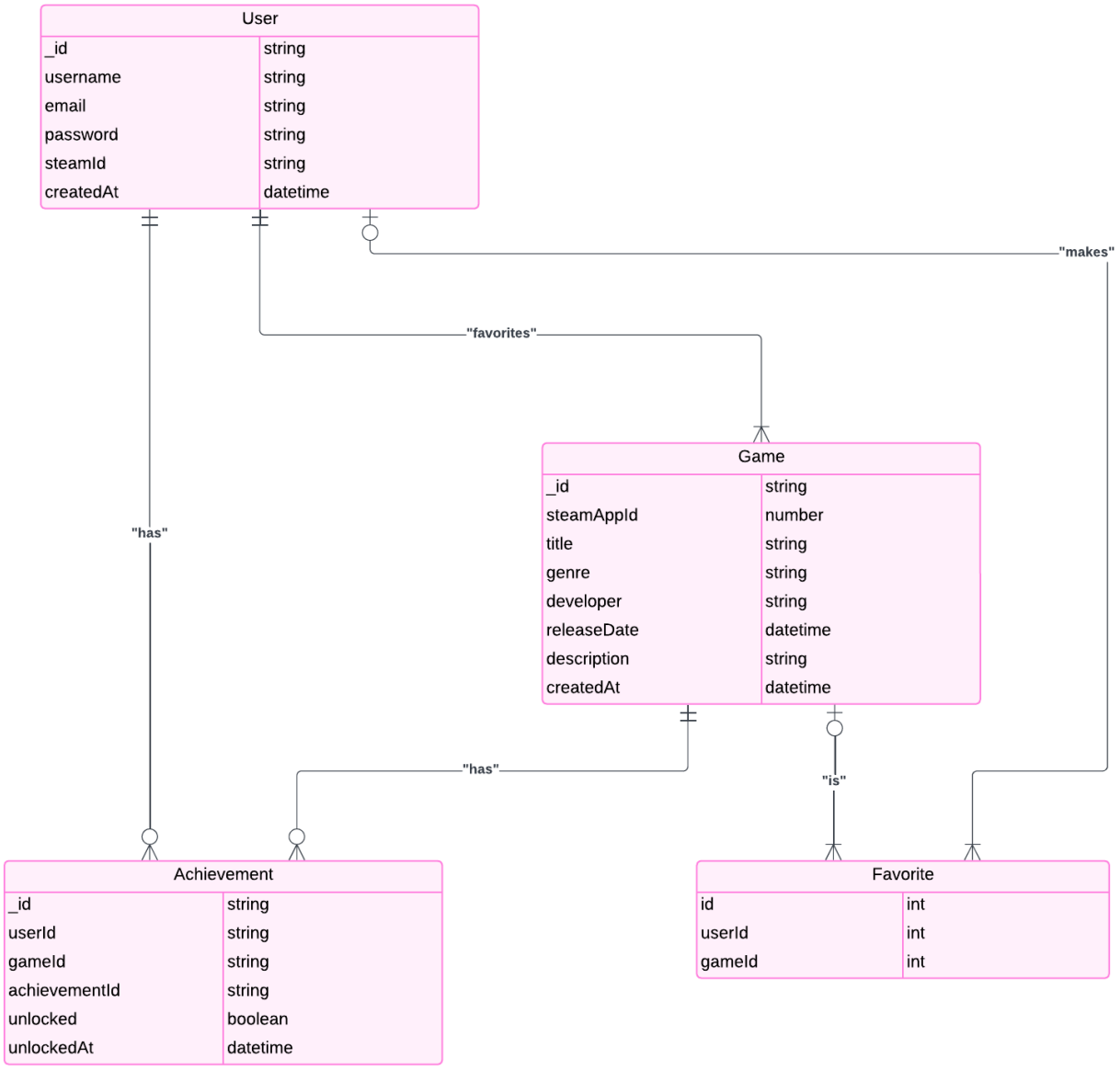
Favorite Schema (for SQL databases with relational tables)

If using a relational database, the favorites relationship can be established in a separate table:

```
CREATE TABLE Favorites (
  id INT PRIMARY KEY AUTO_INCREMENT,
  userId INT,
  gameId
  INT,
  FOREIGN KEY (userId) REFERENCES Users(id),
  FOREIGN KEY (gameId) REFERENCES Games(id)
);
```

Achievements Schema

```
const AchievementSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  gameId: { type: mongoose.Schema.Types.ObjectId, ref: 'Game' },
  achievementId: { type: String, required: true },
  unlocked: { type: Boolean, default: false },
  unlockedAt: { type: Date }
});
```



Data Flow

1. **User Logs In**: The frontend sends a login request to /api/auth/login. If credentials are valid, the backend generates a JWT and sends it to the frontend for secure, authenticated requests.

2. **Retrieve User Games**: The frontend calls /api/user/games with the JWT. The backend verifies the token, fetches games using the Steam API, and stores game data in the Game collection.
3. **Favorite a Game**: The user marks a game as a favorite, and the frontend sends a request to /api/user/favorite. The backend associates the gameId with the user's favorites field in the database.

Database Security

1. **Encryption**: Passwords are hashed (e.g., using bcrypt) before storing in the User schema.
2. **Access Control**: Role-based access can be implemented if there are admin and user roles.
3. **Validation**: Each schema field is validated with types and required checks to prevent injection and malformed data.
4. **Indexes**: Fields like email and steamAppId are indexed for faster querying and uniqueness, reducing the chance of duplicate data.

API Rate Limiting

To avoid abuse of Steam API requests, rate limiting middleware is applied to limit requests to key endpoints, especially those interacting with the Steam API. This protects both your server resources and prevents IP blocks from Steam's API.

Logging & Error Handling

1. **Logging**: Winston or Morgan can be used to log requests, responses, and errors, helping with debugging and tracking suspicious activities.
2. **Error Handling**: Centralized error handling middleware sends consistent error responses to the frontend, improving UX and debugging.