# **Design Document - StarShip Game Project**

## **Theme**

This project is a React-based game application that includes a Phaser game, Redux state management, and a game-over modal. The application is designed to provide an engaging user experience with a simple and intuitive interface.

## **Requirements**

### **Functional Requirements**

1. **User Authentication**:
   * Users must log in or register to access the game.
   * Authentication is implemented using JWT (JSON Web Tokens).
   * Passwords are securely hashed using bcrypt.
2. **Game Mechanics**:
   * Users control a starship to destroy meteors.
   * Fire from the starship triggers sound effects and removes meteors upon collision.
   * Tracks and displays the player’s score and remaining lives.
3. **Game Over Modal**:
   * Displays a message when the game ends.
   * Provides options to replay or navigate to the home page.
4. **Interactive Alerts**:
   * SweetAlert integration for intuitive error handling and user notifications.
5. **Database Integration**:
   * User data (login credentials) and scores are stored in a MongoDB database.
   * Supports retrieval of the last saved score for returning players.
6. **Error Handling**:
   * Detects and displays errors related to game logic or authentication failures.

7. **Redux State Management:**

* + Uses Redux to manage global state, including user authentication, scores, and game persistence.

### **Non-Functional Requirements**

* Responsive user interface.
* Scalable database integration with MongoDB.
* Secure session management using industry-standard practices.
* Smooth and engaging gameplay with real-time Phaser rendering.

## **Workflow**

1. **User Registration/Login**:
   * New users register with an email, username, and password.
   * Returning users log in with their credentials.
   * JWT is generated upon successful login or registration.
2. **Game Access**:
   * Authenticated users navigate to the game page.
   * Unauthorized access is redirected to the login page with a SweetAlert prompt.
3. **Gameplay**:
   * Users play the game, controlling a starship to destroy meteors.
   * Collision between starship fire and meteors triggers a sound effect and removes the meteor.
   * The game tracks scores and remaining lives.
4. **Game Over**:
   * When the game ends, a modal displays the final score and options to replay or return to the home page.
5. **Score Management**:
   * Player scores are stored in the MongoDB database.
   * The last saved score is retrieved and displayed upon login.

## **Components**

### **Entities**

#### **User Entity**

The User entity represents the registered players of the game and includes the following attributes:

* **Username**: The unique identifier for the user within the game.
* **Email**: The user's email address, which must be unique.
* **Password**: The hashed password for secure authentication.
* **DateTime**: The date and time the user was registered.

Schema:

* username: String (required)
* email: String (required, unique)
* password: String (required)
* dateTime: Date (default: Date.now)

#### **Score Entity**

The Score entity tracks the scores achieved by users during gameplay and includes the following attributes:

* **Email**: The email of the user associated with the score.
* **Username**: The username of the user associated with the score.
* **Score**: The numeric score achieved in a game session.
* **Date**: The date and time the score was recorded.

Schema:

* email: String (required)
* username: String (required)
* score: Number (required)
* date: Date (default: Date.now)

## **Main Components**

1. **Game.tsx**
   * Handles game initialization and user authentication checks.
   * Displays Phaser-based gameplay.
   * Displays the "Game Over" modal with replay/home options.
2. **StarshipGame.tsx**
   * Renders the Phaser game.
   * Manages game logic, including meteor collision detection and sound effects.
3. **Modal.tsx**
   * Displays the "Game Over" modal with final scores and navigation options.
4. **Login.tsx & Register.tsx**
   * Login: Validates user credentials and generates a JWT.
   * Register: Validates and hashes user data before storing in MongoDB.
5. **authService.ts**
   * Manages authentication logic, including JWT validation and storage.
6. **MongoDB Integration**
   * Handles storing and retrieving user data and scores

**7. Redux State Management**

* + **store.ts**: Configures the Redux store with reducers and middleware.
  + **slices/userSlice.ts**: Manages user authentication state.
  + **slices/gameSlice.ts**: Manages game-related state, such as scores and lives..

## **Technologies Used**

* **Frontend**: React, Redux, TypeScript, Phaser, SweetAlert.
* **Backend**: Node.js, Express.js.
* **Database**: MongoDB.
* **Authentication**: JWT, bcrypt.

## **Development Plan**

### **Milestones**

1. **Authentication Module**:
   * Implement login and registration pages.
   * Secure passwords with bcrypt and authenticate with JWT.
2. **Game Mechanics**:
   * Develop the Phaser game with starship and meteors.
   * Add collision detection, sound effects, and scoring logic.
3. **Game Over Modal**:
   * Design and integrate the modal with replay/home options.
4. **Database Integration**:
   * Set up MongoDB for user data and score storage.
   * Implement data retrieval for returning players.
5. **Error Handling and Alerts**:
   * Add SweetAlert prompts for errors and notifications.

## **Contact**

For inquiries or contributions, please contact yosishor28@gmail.com.