**Software Requirements Document**

## **Product Perspective**

The MythBusters Project is a standalone web application designed to educate children aged 8-14 and adults about health misconceptions through interactive games and flashcards. It can be integrated into broader educational programs or school curricula to enhance health literacy, serving as a key component of a larger course project.

## **Product Functions**

The system will,

* Provide three distinct educational games (Hangman, Car Race, Balloon) with varying difficulty levels
* Offer interactive flashcards for myth debunking
* Manage user accounts to track progress and in-game currency
* Display leaderboards for competition
* Enable avatar and profile customization using earned currency.

## **User Characteristics**

The expected users and their skill levels described as below:

* Primary users are children aged 8-14 with basic computer literacy.
* Secondary users are adults with varying tech skills interested in health education.
* All users should navigate the web interface with minimal assistance.

## **Constraints**

The expected constraints in terms of platform, programming language and deadline listed in detail as below:

* **Platform:** Development is limited to a responsive web platform with no native mobile apps.
* **Programming Language:**
  + **Frontend**: TypeScript (React with Elastic UI)
  + **Backend:** Java (Spring Boot with Maven)
  + **Database:** PostgreSQL
* **Deadline:** The deadline is aligned with the project timeline, targeting completion by August 10, 2025.

## **System Features (Use Case Based)**

### **Use Case 1: Play Game**

* **Main Actor:** User
* **Goal:** Learn about health myths through interactive games and earn points.
* **Preconditions:**
  + The user is registered and logged in with an active account.
* **Main Flow:**

1. User logs in.
2. User selects a game (e.g., Hangman, Car Race, Balloon).
3. User chooses a difficulty level.
4. User answers questions related to health myths to earn points.
5. System saves the user's progress and updates points.

* **Postconditions:**
  + The user's game progress and points are saved and updated in the system.
* **Exceptions:**
  + **Network Error:** If the client fails to communicate with the server, the user is notified and progress may be stored locally or retried.
  + **API Failure:** If the backend cannot fetch questions or save scores, an error message is shown, and the user may retry.
  + **Unexpected Game Termination:** If the game crashes or the browser tab is closed unexpectedly, the system attempts to save partial progress.

**Use Case 2: View Leaderboard**

* **Main Actor:** User
* **Goal:** View competitive rankings among users.
* **Preconditions:**
  + The user is logged in.
* **Main Flow:**

1. User logs in.
2. User navigates to the leaderboard page.
3. System retrieves and displays the latest leaderboard rankings.
4. User returns to the main menu.

* **Postconditions:**
  + The user has viewed the latest available leaderboard data.
* **Exceptions:**
  + **No Data Available:** If there are no scores recorded yet, the system notifies the user that no leaderboard data is available.
  + **Server Error:** If the backend fails to retrieve leaderboard data, the user is shown an error message.
  + **Network Issue:** If there is a connection problem, the leaderboard cannot be displayed and a connection error message is shown.

**Use Case 3: Create Profile**

* **Main Actor:** User
* **Goal:** Register a new account by creating a user profile with name, profile photo, and optional details.
* **Preconditions:**
  + The user has selected "Sign Up" from the main dashboard (i.e., the user does not have an existing account).
* **Main Flow:**

1. User selects Sign Up from the main dashboard.
2. System displays the profile creation screen.
3. User enters required information (e.g., name) and optionally uploads a profile photo.
4. User may also fill in optional fields (e.g., age, gender, interests).
5. User confirms and submits the form.
6. System validates the data and creates the user's account and profile.
7. Upon success, the user is redirected to the main menu/dashboard with full access.

* **Postconditions:**
  + A new user profile and account are created and stored in the system.
  + The user is now considered "logged in" and has access to features like playing games, earning points, and viewing the leaderboard.
* **Exceptions:**
  + **Missing or Invalid Input:** If required fields are incomplete or contain invalid data, the system prompts the user to fix them.
  + **Photo Upload Error:** If the profile picture upload fails (e.g., wrong format/size), an error is shown and the user is asked to retry.
  + **Account Creation Failure:** If the system fails to save the new profile (e.g., server/database error), the user is notified and can retry.

### **Use Case 4: Buy Avatar**

* **Main Actor:** User
* **Goal:** Purchase a custom avatar using earned in-game currency.
* **Preconditions:**
  + The user is logged in.
  + The user has sufficient in-game currency to purchase an avatar.
* **Main Flow:**

1. User logs in.
2. User checks their available in-game currency.
3. User selects an avatar from the game configuration screen.
4. User confirms the purchase.
5. System deducts the currency and updates the user's profile with the new avatar.

* **Postconditions:** 
  + The user's profile reflects the new avatar, and the currency is updated.
* **Exceptions:**
  + **Insufficient Currency:** If the user does not have enough currency, the system notifies them to earn more before making a purchase.
  + **Transaction Error:** If the purchase fails due to a server or database issue, the user is notified and the transaction is canceled.
  + **Concurrency Conflict:** If the user's currency was modified during the purchase, the system revalidates and prompts the user to retry.

## **Non-Functional Requirements**

* **Usability**: Intuitive interface ensuring 80% of users find it easy and engaging (per UAT)
* **Performance**: Loads within 3 seconds with fluid interactions
* **Portability**: Compatible with modern web browsers
* **Reliability**: Stable with minimal downtime

## **External Interface Requirements**

The system interacts with the following external interfaces:

* **Web Browser (User Interface)**: The React frontend runs in the user's browser, sending HTTP requests and rendering the user interface.
* **Network Protocol (HTTP/HTTPS)**: All client-server communication occurs over secure HTTP/HTTPS connections.
* **Database System (PostgreSQL)**: A separate relational database system accessed by the backend for persistent storage.
* **Third-Party Libraries**: Includes Material UI (UI design), Jackson (JSON processing), SLF4J (logging), and Spring Security (authentication/authorization).

## **User Interfaces**

The MythBusters application follows a clean and intuitive flow starting from a Main Dashboard and guiding users through various core actions such as playing games, checking scores, and customizing their experience. Below are the main user interface screens:

**Main Dashboard**

* Acts as the central hub of the app.
* Key components:
  + Login / Sign Up area
  + Game selection panel (Hangman, Car Race, Balloon)
  + Leaderboard access
  + User profile view (avatar, score, logout)

### **Game Configuration Screen**

* Shown after a game is selected.
* Allows the user to:
  + Choose difficulty level (e.g., Easy / Medium / Hard)
  + Start the game

### **Game Screen (In-Game UI)**

* Displays the selected game (e.g., Hangman).
* Real-time interaction:
  + Health-myth-related questions
  + Game mechanics (guess letters, avoid obstacles, etc.)
  + Score and question progress (e.g., “Question 10/10”)

### **Result Screen (Game Over / Victory)**

* Shown after game completion or failure.  
  Displays:
  + Score earned
  + Total points
  + Option to replay or return to dashboard

### **Leaderboard Screen**

* Lists top users with:
  + Username
  + Scores
  + Ranks

## **Software Interfaces**

The MythBusters Project uses a client-server architecture with the following interfaces:

* **Frontend (React, TypeScript, Material UI)**: Communicates with the backend via RESTful APIs using Axios, manages UI with Material UI, and handles state with Redux or Context API.
* **Backend (Spring Boot, Java)**: Exposes REST APIs with Spring Web, interacts with PostgreSQL via Spring Data JPA and Hibernate, and secures endpoints with Spring Security.
* **Database (PostgreSQL)**: Connects to the backend via JDBC for storing user profiles, leaderboards, and avatars.
* **External Libraries**: Uses Jackson for JSON processing, SLF4J for logging, and Maven for dependency management.