Software Requirements Document (SRD) for Bible Trivia Web App

1. Introduction

1.1 Project Overview

The Bible Trivia Web App is an interactive, web-based game designed to engage users with Bible-related questions across different difficulty levels and categories. The game can be played in single-player or multiplayer mode, and includes an admin panel for managing questions, categories, and game settings. GPT 4o mini integration allows the admin to generate trivia questions dynamically.

1.2 Objectives

• Provide an engaging platform for Bible enthusiasts to test and improve their knowledge.
• Support multiple game modes (single-player and multiplayer).
• Implement an intuitive UI for seamless gameplay.
• Enable AI-generated and manually uploaded trivia questions.
There should be an option to upload mp4 video files for avatar based question master
• Provide an admin panel for content management.

1.3 Scope

• **Game modes:** Single-player and multiplayer (up to 3 players).
• **Game levels:** Beginner, Intermediate, Advanced.
• **Question types:** Multiple-choice.
• **Admin functionalities:** AI-generated questions, manual upload via spreadsheet, category
management.
• **Leaderboard & scoring system.**
• **Responsive UI for web and mobile.**

2. **Functional Requirements**

2.1 Game Setup

• Players can choose from different categories:
- Theme-Based Bible Trivia
- Old Testament

- New Testament
- Bible Stories
- Famous People in the Bible

• Players start with 0 points.
• The game duration is either time-based (10-15 minutes) or question-based (e.g., 10 questions).

2.2 Gameplay Mechanics

• Each question has a 20-second timer.
• Players answer questions via multiple-choice buttons or text input.
• A correct answer earns 1 point; a Bonus Round question earns 2 points.
• The final score is displayed at the end of the game.
• In case of a tie, a tie-breaker question is presented.

2.3 Multiplayer Mode

• Supports up to 3 players.
• Real-time game synchronization to display questions and responses across all players.
• Leaderboard updates dynamically.

2.4 Admin Panel

• AI-Generated Questions: Admin can specify the number of questions, category, and
difficulty level to generate trivia questions via ChatGPT 4.o mini.
• Manual Upload: Admin can upload questions in bulk using a structured spreadsheet.
• Edit/Delete Questions: Manage and update the question database.
• Category Management: Add, edit, or delete trivia categories.

2.5 Audio/Visual Feedback

• Correct answers trigger a “Correct!” sound and a green checkmark.
• Incorrect answers trigger a “Wrong!” sound and a red X.
• Winning players receive celebratory sound and animations.

3. Technical Specifications
3.1 Tech Stack

• **Frontend:** React, TypeScript, Tailwind CSS
• **Backend:** Node.js, Express.js
• **Database:** MySQL/MongoDB/PostgreSQL
• **Real-time Features:** WebSockets for multiplayer mode
• **AI Integration:** OpenAI ChatGPT 4.o mini API
• **Authentication:** Firebase Auth or JWT
• **Storage:** S3 or Firebase for question bank

3.2 Software Development Standards

• Code Quality: Follows Clean Architecture principles.
• Security: JWT authentication, input validation, and SQL injection prevention.
• Version Control: GitHub (feature branching strategy).
• CI/CD: Automated deployment via GitHub Actions.

4.1 Option 1: Multi-Player Version

Features:
• Multi -player mode only.
• AI-generated and manually uploaded questions.
• Admin panel for question management.
• Responsive web UI.

AI Quiz Master (Master Eli)
Real-world rewards (books, caps, T-shirts)
Digital certificate for full completion

Welcome screen > Meet Ai avatar Master Eli> Choose category>Avatar asks questions.

If Correct:
Avatar says (voice or text): "Splendid! As Noah built the Ark with faith, you build your knowledge!"
(Animation: Big happy smile, hand raise
Gestures the answer is wrong)
If Wrong:
Avatar says: "A brave attempt, but the builder of the Ark was Noah! Fear not, for wisdom grows with each question.
"
(Animation: Gentle nod, comforting expression)

Check for bugs or errors in both frontend and backend.
Make sure all API routes in the backend are working and reachable from the frontend.
Ensure the frontend components properly fetch and render data.
Validate that all forms, buttons, timers, and interactions work.

Ensure there are no React state issues or missing dependency arrays in `useEffect`.

Make sure the backend has propThe game er error handling and doesn’t crash on bad input.

Return a cleaned, test-ready version of the full code and tell me how to test it locally.”