Audio Editor Web Application

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for the Audio Editor Web Application. This application allows users to upload, edit, and process audio files with various features such as merging, splitting, volume control, and applying audio effects. The application will be developed using **ReactJS** for the frontend, **Java Spring Boot** for the backend, and **Firebase** for storage.

1.2 Scope

This web application provides an interactive and user-friendly interface for audio editing. The system allows users to:

- Upload and manage audio files
- Play, stop, and control playback
- Merge and split audio files
- Adjust volume levels
- Apply effects like fade in, fade out
- Export edited audio files
- Use microphone recording

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, testers, project managers, and stakeholders involved in the development of the Audio Editor Web App. Readers should refer to:

- **Section 2** for system overview
- **Section 3** for detailed functional and non-functional requirements
- **Section 4** for system architecture and design

1.4 Definitions and Acronyms

- **ReactJS** – a JavaScript library for building user interfaces
- **Spring Boot** – a Java framework for backend development
- **Firebase** – a cloud-based storage and database service
- **API** – Application Programming Interface
- **UI** – User Interface

2. Overall Description

2.1 Product Perspective

The Audio Editor Web App is a standalone application that integrates with Firebase for storage and leverages RESTful APIs for backend processing. It provides real-time audio manipulation capabilities through a web interface.

2.2 Product Features

- **Audio Upload & Storage**: Users can upload audio files and store them in Firebase.
- **Playback Controls**: Play, pause, and stop functionalities.
- **Audio Editing**: Merge, split, adjust volume, and apply effects.
- **Recording Feature**: Capture audio via microphone.
- **Export & Download**: Users can export and download the edited audio file.

2.3 User Characteristics

- **General Users**: Users with basic audio editing needs.
- **Advanced Users**: Users requiring more refined editing capabilities.

3. Specific Requirements

3.1 Functional Requirements

3.1.1 User Interaction

- The system shall allow users to upload audio files.
- The system shall provide controls for playing, stopping, and pausing audio.
- The system shall support merging two or more audio files.
- The system shall support splitting audio at specific timestamps.
- The system shall allow volume adjustments.
- The system shall allow applying fade in and fade out effects.
- The system shall allow users to export the edited file.

3.1.2 Storage & Processing

- The system shall store uploaded files in Firebase.
- The system shall process audio files via Spring Boot APIs.
- The system shall allow real-time editing without requiring file downloads.

3.2 Non-Functional Requirements

- **Performance**: The system should process audio files within 3 seconds for basic operations.
- **Usability**: The UI should be intuitive and responsive.
- **Scalability**: The system should handle concurrent users and process multiple files simultaneously.
- **Compatibility**: The application should be accessible via modern browsers (Chrome, Firefox, Edge, Safari).

4. System Architecture

4.1 Architecture Overview

The system follows a client-server architecture:

- **Frontend**: ReactJS for UI components.
- **Backend**: Spring Boot RESTful API for processing requests.
- **Storage**: Firebase for audio file storage.
- **Audio Processing**: Implemented via Java Spring Boot services.