**Software Requirements Specification (SRS)**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to outline the functional and non-functional requirements for The Movie Library API. This API is movies datebase and will allow users to manage movies and actors, including features such as creating, reading, updating, deleting, searching, and sorting.

**1.2 Scope**

This document covers the requirements for the Movie and Actor API, including all endpoints and their expected behaviors. It does not cover performance, security, or scalability requirements.

**1.3 Definitions, Acronyms, and Abbreviations**

* **API:** Application Programming Interface
* **CRUD:** Create, Read, Update, Delete
* **JSON:** JavaScript Object Notation
* **HTTP:** Hypertext Transfer Protocol

**1.4 References**

* [Sinatra Documentation](http://sinatrarb.com/documentation.html)
* [RSpec Documentation](https://rspec.info/documentation/)

**2. Overall Description**

**2.1 Product Perspective**

The Movie Library API is a standalone application that provides endpoints for managing movies and actors. It is built using Sinatra and follows RESTful principles.

**2.2 Product Functions**

* Manage movies (create, read, update, delete)
* Manage actors (create, read, update, delete)
* Search for movies and actors
* Sort movies and actors

**2.3 User Classes and Characteristics**

* **Admin:** Can perform all CRUD operations on movies and actors.
* **User:** Can search and view movies and actors.

**2.4 Operating Environment**

* The API will run on a server with Ruby and Sinatra installed.
* The database used is assumed to be SQLite for development and PostgreSQL for production.

**2.5 Design and Implementation Constraints**

* The API must use JSON for data interchange.
* Authentication is required for all endpoints except the welcome message.

**2.6 Assumptions and Dependencies**

* Users have basic knowledge of HTTP and RESTful APIs.
* The database schema is predefined and includes tables for movies and actors.

**3. Functional Requirements**

**3.1 Movie Management**

**3.1.1 List Movies**

* **Requirement ID:** R001
* **Description:** The system shall list all movies.
* **Endpoint:** GET /movies
* **Response:** JSON array of movie objects
* **Preconditions:** Movies exist in the database.

**3.1.2 Add Movie**

* **Requirement ID:** R002
* **Description:** The system shall add a new movie.
* **Endpoint:** POST /movies
* **Request Body:** JSON object with movie details
* **Response:** JSON object of the created movie
* **Preconditions:** None.

**3.1.3 Edit Movie**

* **Requirement ID:** R003
* **Description:** The system shall return movie data for editing.
* **Endpoint:** GET /movies/:id/edit
* **Response:** JSON object of the movie
* **Preconditions:** A movie with the specified ID exists in the database.

**3.1.4 Update Movie**

* **Requirement ID:** R004
* **Description:** The system shall update an existing movie.
* **Endpoint:** PATCH /movies/:id
* **Request Body:** JSON object with updated movie details
* **Response:** JSON object of the updated movie
* **Preconditions:** A movie with the specified ID exists in the database.

**3.1.5 Delete Movie**

* **Requirement ID:** R005
* **Description:** The system shall delete an existing movie.
* **Endpoint:** DELETE /movies/:id
* **Response:** JSON object with a success message
* **Preconditions:** A movie with the specified ID exists in the database.

**3.1.6 Search Movies**

* **Requirement ID:** R006
* **Description:** The system shall search for movies based on name, director, main actor, or year.
* **Endpoint:** GET /movies/search
* **Query Parameters:** search (string)
* **Response:** JSON array of matching movie objects
* **Preconditions:** None.

**3.1.7 Sort Movies**

* **Requirement ID:** R007
* **Description:** The system shall sort movies based on specified criteria.
* **Endpoint:** GET /movies/sort
* **Query Parameters:** sort (string)
* **Response:** JSON array of sorted movie objects
* **Preconditions:** None.

**3.2 Actor Management**

**3.2.1 List Actors**

* **Requirement ID:** R008
* **Description:** The system shall list all actors.
* **Endpoint:** GET /actors
* **Response:** JSON array of actor objects
* **Preconditions:** Actors exist in the database.

**3.2.2 Add Actor**

* **Requirement ID:** R009
* **Description:** The system shall add a new actor.
* **Endpoint:** POST /actors
* **Request Body:** JSON object with actor details
* **Response:** JSON object of the created actor
* **Preconditions:** None.

**3.2.3 Edit Actor**

* **Requirement ID:** R010
* **Description:** The system shall return actor data for editing.
* **Endpoint:** GET /actors/:id/edit
* **Response:** JSON object of the actor
* **Preconditions:** An actor with the specified ID exists in the database.

**3.2.4 Update Actor**

* **Requirement ID:** R011
* **Description:** The system shall update an existing actor.
* **Endpoint:** PATCH /actors/:id
* **Request Body:** JSON object with updated actor details
* **Response:** JSON object of the updated actor
* **Preconditions:** An actor with the specified ID exists in the database.

**3.2.5 Delete Actor**

* **Requirement ID:** R012
* **Description:** The system shall delete an existing actor.
* **Endpoint:** DELETE /actors/:id
* **Response:** JSON object with a success message
* **Preconditions:** An actor with the specified ID exists in the database.

**3.2.6 Search Actors**

* **Requirement ID:** R013
* **Description:** The system shall search for actors based on name, description, birthdate, or oscar status.
* **Endpoint:** GET /actors/search
* **Query Parameters:** search (string)
* **Response:** JSON array of matching actor objects
* **Preconditions:** None.

**3.2.7 Sort Actors**

* **Requirement ID:** R014
* **Description:** The system shall sort actors based on specified criteria.
* **Endpoint:** GET /actors/sort
* **Query Parameters:** sort (string)
* **Response:** JSON array of sorted actor objects
* **Preconditions:** None.

**4. Non-Functional Requirements**

**4.1 Performance**

* The API should respond to requests within 200ms under normal load.

**4.2 Usability**

* The API should be easy to use and follow RESTful principles.

**4.3 Reliability**

* The API should have an uptime of 99.9%.

**4.4 Security**

* All endpoints should be protected with basic authentication, except the welcome message.

**4.5 Maintainability**

* The code should be well-documented and follow best practices for readability and maintainability.

**5. Appendices**

**5.1 Glossary**

* **CRUD:** Create, Read, Update, Delete
* **JSON:** JavaScript Object Notation
* **HTTP:** Hypertext Transfer Protocol