# Mekelle University - YouTube

# MEKELLE UNIVERSITY

# ETHIOPIAN INSTITUTE OF TECHNOLOGY - MEKELLE

# SCHOOL OF COMPUTING

# DEPARTMENT OF SOFTWARE ENGINEERING

# Subject: SOFTWARE TESTING

Title: SOFTWARE REQUIREMENT SPECIFICATION

**Amare Birhane Assefa -----------------------------------------------------------ID No: URG/171995/12**

Submitted to: mesele

Submission day: 2/2/2025

# Software Requirements Specification (SRS)

Table of Contents

[Software Requirements Specification (SRS) 3](#_Toc187605191)

[1. Introduction 6](#_Toc187605192)

[1.1 Purpose and Scope 6](#_Toc187605193)

[1.2 Definitions, Acronyms, and References 6](#_Toc187605194)

[2. Overall Description 7](#_Toc187605195)

[2.1 Product Perspective 7](#_Toc187605196)

[2.2 Product Features 7](#_Toc187605197)

[2.3 User Characteristics 7](#_Toc187605198)

[2.4 Operating Environment 7](#_Toc187605199)

[2.5 Design and Implementation Constraints 7](#_Toc187605200)

[2.6 Assumptions and Dependencies 8](#_Toc187605201)

[3. Functional Requirements 8](#_Toc187605202)

[3.1 Authentication 8](#_Toc187605203)

[3.2 Profile Management 8](#_Toc187605204)

[3.3 Course Management 8](#_Toc187605205)

[3.4 Payments 8](#_Toc187605206)

[3.5 Logout 9](#_Toc187605207)

[4. Use Cases 9](#_Toc187605208)

[4.1 Use Case: User Registration 9](#_Toc187605209)

[4.2 Use Case: User Login 9](#_Toc187605210)

[4.3 Use Case: Update User Profile 10](#_Toc187605211)

[4.4 Use Case: Course Management (Admin) 11](#_Toc187605212)

[4.5 Use Case: Payment Processing 12](#_Toc187605213)

[5. Non-Functional Requirements 12](#_Toc187605214)

[5.1 Performance 12](#_Toc187605215)

[5.2 Scalability 12](#_Toc187605216)

[5.3 Security 13](#_Toc187605217)

[5.4 Usability 13](#_Toc187605218)

[5.5 Maintainability 13](#_Toc187605219)

[6. User Interface Requirements 13](#_Toc187605220)

[6.1. Dashboard 13](#_Toc187605221)

[6.2. Profile Update Form 13](#_Toc187605222)

[6.3. Course Management Page (for Admin) 13](#_Toc187605223)

[6.4. User Registration Page 14](#_Toc187605224)

[6.5. User Login Page 14](#_Toc187605225)

[6.6. Payment Page 14](#_Toc187605226)

[6.7. Logout Confirmation 14](#_Toc187605227)

[7. Interface Requirements 15](#_Toc187605228)

[7.1 User Interface 15](#_Toc187605229)

[7.2 API Endpoints 15](#_Toc187605230)

[7.3 Database Schema 15](#_Toc187605231)

[8. Appendices 15](#_Toc187605232)

[8.1 Future Enhancements 15](#_Toc187605233)

[8.2 References 16](#_Toc187605234)

# Software Requirements Specification (SRS)

# 1. Introduction

### 1.1 Purpose and Scope

**Purpose**

* Definition: The SRS aims to define the functional and non-functional requirements of a web-based user management system.
* Functionality: It highlights key functionalities such as user registration, login, profile updates, and secure logouts.

**Scope**

* Key Features:
  + Secure user authentication and authorization.
  + User registration with personal details and file uploads.
  + Secure login with email and password.
  + Fetching user profile information from a MongoDB database.
  + Updating personal information and passwords.
  + Course management, including adding and purchasing courses using Stripe integration.

**Analysis:**

* Strengths: The purpose and scope sections are clear and provide a solid foundation for the SRS. They effectively outline the system's objectives and functionalities.
* Improvements:
  + Include a brief description of the target user groups (e.g., students, administrators) to establish context.
  + Define the expected outcomes or benefits of the system to enhance understanding.

### 1.2 Definitions, Acronyms, and References

**Definitions**

* Key Terms:
  + JWT: JSON Web Token for authentication.
  + API: Application Programming Interface.
  + CRUD: Create, Read, Update, Delete operations.
  + MongoDB: NoSQL database for storing user data.

**References**

* Links to relevant documentation for technologies such as ReactJS, ExpressJS, and Axios.

**Analysis:**

* Strengths: This section is comprehensive and provides essential terminology that aids in understanding the document.
* Improvements:
  + Provide context on how these technologies interact within the system architecture.
  + Consider adding a glossary for less common terms.

# 2. Overall Description

### 2.1 Product Perspective

* The system operates as a web application that integrates a ReactJS frontend with a Node.js and Express backend, using MongoDB for data storage. JWT handles user authentication.

### 2.2 Product Features

1. User Registration: New users can register by providing name, email, password, and DOB.
2. User Login: Users log in using email and password, receiving a secure JWT.
3. Profile Management: Users can view and update their profiles.
4. Course Management: Admins can manage courses, while students can view and purchase them.

**Analysis:**

* Strengths: The product perspective clearly outlines the system's architecture and integration. The features are well-defined, providing a comprehensive view of the system's capabilities.
* Improvements:
  + Group related features for better organization (e.g., combine profile management features).
  + Include user scenarios or use cases to illustrate how users will interact with the system.

### 2.3 User Characteristics

* Admin Users: Can manage courses and view all user data.
* Students: Can register, log in, update their profiles, and purchase courses.

### 2.4 Operating Environment

* Frontend: ReactJS, running on modern web browsers (Chrome, Firefox, Safari).
* Backend: Node.js with ExpressJS.
* Database: MongoDB, running locally or on a cloud service like MongoDB Atlas.
* Authentication: JWT stored in local storage or cookies.

### 2.5 Design and Implementation Constraints

* JWT expiration will require users to reauthenticate periodically.
* Only secure HTTPS connections should be supported for production environments.
* Data validation must occur on both the frontend and backend to prevent malformed requests.

### 2.6 Assumptions and Dependencies

* Users must have valid credentials to access protected routes.
* MongoDB must be operational and connected for the application to function.
* Internet connectivity is required for both frontend and backend to communicate.

# 3. Functional Requirements

### 3.1 Authentication

* **Registration:**
  + Input: Name, email, password, DOB, and file upload.
  + Process: Validate and hash the password, store user data in the database.
  + Output: Confirmation message and token.
* **Login:**
  + Input: Email and password.
  + Process: Validate credentials, issue JWT.
  + Output: Login success or error message.

### 3.2 Profile Management

* **View Profile:**
  + Input: JWT in the authorization header.
  + Process: Fetch user data from the database.
  + Output: JSON with user details (password excluded).
* **Update Profile:**
  + Input: Updated user details and JWT.
  + Process: Validate JWT, update database record.
  + Output: Updated profile confirmation.

### 3.3 Course Management

* **Add Courses (Admin):**
  + Input: Title, description, price.
  + Output: Confirmation of course addition.
* **View Courses:**
  + Output: List of available courses.

### 3.4 Payments

* **Stripe Payment Integration:**
  + Input: Course ID.
  + Process: Create Stripe payment session.
  + Output: Redirect to payment page.

### 3.5 Logout

* Input: None.
* Process: Remove JWT from client storage (local storage or cookies).
* Output: Confirmation of logout and redirection to the login page.

# 4. Use Cases

### 4.1 Use Case: User Registration

**Use Case ID**: UC-01

**Actors**: User (a new user who wants to register for the system).

**Preconditions**:

* The user has access to the registration page of the application.
* The user has a valid email address and meets the registration requirements.

**Postconditions**:

* The user account is created in the database.
* The user receives a confirmation message.

**Main Flow**:

1. User navigates to the registration page.
2. User enters personal details: Name, Email, Password, Date of Birth, and optional file upload.
3. System validates the input:
   * Checks if the email format is valid.
   * Ensures the password meets strength requirements.
4. System hashes the password and stores the user data in the database.
5. System generates a confirmation message.
6. System sends a welcome email to the user (optional).
7. User is redirected to the login page or dashboard.

**Alternate Flow**:

* If input validation fails:
  + The system displays error messages for each invalid field, prompting the user to correct the information.

### 4.2 Use Case: User Login

**Use Case ID**: UC-02

**Actors**: User (a registered user attempting to log in).

**Preconditions**:

* The user has previously registered and has valid login credentials.

**Postconditions**:

* The user is authenticated and granted access to their profile.

**Main Flow**:

1. User navigates to the login page.
2. User enters their email and password.
3. System validates the credentials:
   * Checks if the email exists in the database.
   * Verifies the password against the stored hash.
4. System issues a JWT for session management.
5. User is redirected to the dashboard.
6. System displays a welcome message with user profile details.

**Alternate Flow**:

* If login fails:
  + The system displays an error message indicating invalid credentials.

### 4.3 Use Case: Update User Profile

**Use Case ID**: UC-03

**Actors**: User (an authenticated user who wants to update their profile).

**Preconditions**:

* The user is logged into the system.

**Postconditions**:

* The user's profile information is updated in the database.

**Main Flow**:

1. User navigates to the profile management page.
2. User views their current profile information.
3. User updates desired fields: Name, Email, Password, Date of Birth, and optional file upload.
4. System validates the input:
   * Ensures the email format is valid.
   * Checks password strength if updated.
5. System updates the user data in the database.
6. System displays a confirmation message of the successful update.

**Alternate Flow**:

* If input validation fails:
  + The system displays error messages for each invalid field, prompting the user to correct the information.

### 4.4 Use Case: Course Management (Admin)

**Use Case ID**: UC-04

**Actors**: Admin User (a user with administrative privileges).

**Preconditions**:

* The admin is logged into the system.

**Postconditions**:

* Courses are added, modified, or deleted in the database.

**Main Flow**:

1. Admin navigates to the course management page.
2. Admin selects an action:
   * Add a new course
   * Edit an existing course
   * Delete a course
3. For adding a course:
   * Admin enters details: Title, Description, Price.
   * System validates input.
   * System adds the course to the database.
   * System displays a confirmation message.
4. For editing a course:
   * Admin selects a course to edit.
   * Admin modifies course details.
   * System validates input.
   * System updates the course in the database.
   * System displays a confirmation message.
5. For deleting a course:
   * Admin selects a course to delete.
   * System asks for confirmation.
   * If confirmed, the system removes the course from the database.
   * System displays a confirmation message.

**Alternate Flow**:

* If input validation fails during adding or editing:
  + The system displays error messages for each invalid field, prompting the admin to correct the information.

### 4.5 Use Case: Payment Processing

**Use Case ID**: UC-05

**Actors**: User (a registered user purchasing a course).

**Preconditions**:

* The user is logged into the system and has selected a course to purchase.

**Postconditions**:

* The payment is processed, and the course is added to the user's account.

**Main Flow**:

1. User navigates to the course details page and selects "Purchase."
2. System redirects the user to the payment page (Stripe).
3. User enters payment details (credit card information).
4. System validates payment details.
5. System processes the payment through Stripe.
6. If payment is successful:
   * System updates the user's course enrollment in the database.
   * System displays a success message and confirmation of the purchase.
7. User is redirected to their course list or dashboard.

**Alternate Flow**:

* If payment fails:
  + The system displays an error message indicating the failure and prompts the user to retry.

# 5. Non-Functional Requirements

### 5.1 Performance

* Response time under 2 seconds for API requests.

### 5.2 Scalability

* The system should support up to 10,000 concurrent users without significant performance degradation.

### 5.3 Security

* Passwords must be hashed using bcrypt before storage.
* JWT must be signed using a secure secret key.
* HTTPS must be enforced in production environments.

### 5.4 Usability

* The system should have an intuitive UI with proper form validations and user-friendly error messages.

### 5.5 Maintainability

* Modular codebase for backend (controllers, routes).
* Consistent coding standards for frontend and backend.

# 6. User Interface Requirements

### 6.1. Dashboard

* **Purpose**: To provide users with an overview of their profile and quick access to various functionalities.
* **Elements**:
  + **Profile Summary**: Displays the user’s name, email, and date of birth.
  + **Navigation Menu**:
    - Link to Profile Management.
    - Link to Course Management (for admin users).
    - Link to Logout.
  + **Notifications Section**: Displays messages (e.g., account updates, course alerts).

### 6.2. Profile Update Form

* **Purpose**: To allow users to view and update their personal information.
* **Elements**:
  + **Pre-filled Input Fields**:
    - First Name
    - Last Name
    - Email
    - Date of Birth
    - Password (with an option to change)
    - File Upload (for profile picture)
  + **Validation Messages**: Real-time feedback for input validation (e.g., valid email format, password strength).
  + **Submit Button**: To save changes.
  + **Cancel Button**: To discard changes and return to the dashboard.

### 6.3. Course Management Page (for Admin)

* **Purpose**: To manage courses including adding, editing, or deleting courses.
* **Elements**:
  + **Course List**:
    - Table displaying all courses with columns for Title, Description, Price, and Actions (Edit/Delete).
  + **Add Course Button**: Opens a form to enter new course details.
  + **Edit Course Form**: Pre-filled with existing course details when editing.
  + **Delete Confirmation Dialog**: Prompt to confirm course deletion.

### 6.4. User Registration Page

* **Purpose**: To allow new users to create an account.
* **Elements**:
  + **Input Fields**:
    - Name
    - Email
    - Password
    - Date of Birth
    - File Upload (optional)
  + **Submit Button**: To register the user.
  + **Error Messages**: Display validation errors for each field if not filled correctly.

### 6.5. User Login Page

* **Purpose**: To authenticate users into the system.
* **Elements**:
  + **Input Fields**:
    - Email
    - Password
  + **Submit Button**: To log in the user.
  + **Error Messages**: Display invalid credentials error if login fails.
  + **Link to Registration Page**: For new users to create an account.

### 6.6. Payment Page

* **Purpose**: To process payments for course purchases.
* **Elements**:
  + **Course Details**: Displaying the selected course information (Title, Price).
  + **Payment Form**:
    - Input fields for credit card information (Card Number, Expiry Date, CVV).
    - Submit Button to process payment.
  + **Error Messages**: Display validation errors if payment details are incorrect.

### 6.7. Logout Confirmation

* **Purpose**: To confirm the user's action when logging out.
* **Elements**:
  + **Confirmation Message**: Asking if the user is sure they want to log out.
  + **Yes Button**: To confirm logout.
  + **No Button**: To cancel logout action and return to the dashboard.

# 7. Interface Requirements

### 7.1 User Interface

* Dashboard: Displays user profile details with options to update the profile or log out.
* Profile Update Form: Pre-filled form with current user details, allowing updates.

### 7.2 API Endpoints

* POST /api/auth/signup: Register a new user.
* POST /api/auth/login: Authenticate and issue JWT.
* GET /api/user/profile: Fetch user profile.
* PUT /api/user/update/:id: Update user profile.

### 7.3 Database Schema

* **User:**
* {
* "firstName": "String",
* "middleName": "String",
* "lastName": "String",
* "email": "String",
* "password": "String",
* "dob": "Date",
* "filePath": "String"
* }
* **Course:**
* {
* "title": "String",
* "description": "String",
* "price": "Number"
* }

# 8. Appendices

### 8.1 Future Enhancements

* Add roles for admin and regular users.
* Enable profile picture uploads and storage.
* Implement email verification during registration.
* Two-factor authentication.
* Admin dashboard for advanced analytics.
* Course categories and search filters.

### 8.2 References

* JWT Authentication
* Stripe API
* React Documentation
* Node.js Documentation
* ExpressJS Documentation
* MongoDB Documentation