

MEKELLE UNIVERSITY

SCHOOL OF COMPUTING EIT-M

DEPARTMENT OF SOFTWARE ENGINEERING

SOFTWARE TESTING AND QUALITY ASURSNCE INDIVIDUAL ASSINGMENT

TILAHUN GOITOM ……….EITM/UR170315/12

NOTE ONLY FOR THE PAYMENT FLOW IS PROVIEDED WITH DITAILED TEST CASES AND GOES ALL THE WAY THROUGH THE STEPS FOR NOW

**Software Testing Requirements Specification (SRS) for PeerCourse-Next.js Project**

**Table of Contents**

1. **Introduction**

1.1 Purpose of Testing  
1.2 Scope of the Project  
1.3 Definitions, Acronyms, and Abbreviations  
1.4 References  
1.5 Overview

1. **Test Strategy**

2.1 Types of Testing  
2.2 Test Levels  
2.3 Test Design  
2.4 Test Tools and Environment

1. **Test Requirements**

3.1 Functional Testing  
3.2 Non-Functional Testing  
3.3 Integration Testing  
3.4 User Acceptance Testing (UAT)

1. **Test Cases**

4.1 Authentication Testing  
4.2 Course Management Testing  
4.3 Tutorial Management Testing  
4.4 Stripe Payment Flow Testing  
4.5 Error Handling Testing  
4.6 UI/UX Testing

1. **Test Execution**

5.1 Test Execution Strategy  
5.2 Test Environment Setup  
5.3 Test Execution Plan

1. **Defect Management**

6.1 Defect Life Cycle  
6.2 Defect Reporting  
6.3 Defect Severity and Priority

1. **Test Completion Criteria**

7.1 Completion Criteria  
7.2 Exit Criteria

1. **Appendices**

8.1 Glossary  
8.2 Test Data  
8.3 Test Traceability Matrix

**1. Introduction**

**1.1 Purpose of Testing**

This is supposed to provides the testing requirements for the Next.js-based course management application\_(PEER-COURSES), including authentication, tutorial management, course listing, payment flow, maintains tutorial files on the cloud Cloudinary and more so that all possible defect could be hunted based on these specifications. It also outlines the strategy, test cases, environment, and defect management approach to ensure that the application functions as intended and meets user needs.

**1.2 Scope of the Project**

The project allows users to browse, purchase, and view tutorials in various courses and generate thier own to the course they payed for or say joined to . Users can unlock course content through Stripe payments, and course-related materials are managed and displayed on the front end. The authentication is managed by Clerk, and the backend uses Mongoose and MongoDB Atlas for data storage.

**1.3 Definitions, Acronyms, and Abbreviations**

* **UAT**: User Acceptance Testing
* **Stripe**: Payment gateway used for processing course payments.
* **Clerk**: Authentication service for user management.
* **Mongoose**: A Node.js library for MongoDB object modeling.

**1.4 References**

* [Stripe Documentation](https://stripe.com/docs)
* [Clerk Authentication](https://clerk.dev/docs)
* [Next.js Documentation](https://nextjs.org/docs)

**1.5 Overview**

This resource bases and outlines the testing strategies, tools, and test cases required to ensure the successful functionality and quality of the Next.js-based, PEER-COURSE project.

**2. Test Strategy**

**2.1 Types of Testing**

* **Unit Testing**: Testing individual components, such as API endpoints, payment integration, and database models.
* **Integration Testing**: Ensuring that all components work together correctly, like Clerk authentication, Stripe payments, and MongoDB integration.
* **Functional Testing**: Validating that each feature (e.g., user login, tutorial display) works as expected.
* **UI Testing**: Ensuring that the front-end is user-friendly, responsive, and displays the right content.
* **Security Testing**: Ensuring that authentication (Clerk) and payment (Stripe) are secure and that no sensitive data is exposed.
* **Performance Testing**: Checking for latency, load handling, and response times.

**2.2 Test Levels**

* **Unit Tests**: Test individual functions/components of the application (e,g login

. 2 signup 3 payment 4 course fecher 5 tutorial adder 6 listing tutorial 7 unlocking course

* **Integration Tests**: Test interactions between services (e.g., Clerk authentication, MongoDB stripe for payment ).
* **End-to-End Tests**: Test complete user journeys like course purchase or tutorial access.
* **Regression Tests**: Ensure that new updates don't break existing functionality.

**2.3 Test Design**

Tests will be designed around the user stories and features of the application. Each test will include:

* **Test Name**
* **Test Description**
* **Pre-conditions**
* **Test Steps**
* **Expected Results**

**2.4 Test Tools and Environment**

* **Testing Framework**: Jest (Unit, Integration, and E2E tests)
* **UI Testing**: Cypress
* **Mocking**: Mock Service Worker (MSW) for mocking API responses
* **Environment**: Test environment will replicate production with a dedicated database.

**3. Test Requirements**

**3.1 Functional Testing**

* **Login**: Ensure users can log in through Clerk using Google or custom login.
* **Course Listing**: Verify courses are displayed, with appropriate data such as course title, image, and payment amount.
* **Tutorial Listing**: Ensure tutorials related to a course are displayed correctly, with links to content URLs.
* **Unlock Course**: Verify that users can unlock a course after a successful Stripe payment.
* **Stripe Payment Flow**: Test the entire payment process, including success and failure paths
* Authenticating with clerk which will handle sign-up, session .

**3.2 Non-Functional Testing**

* **Security**: Test that sensitive user data, such as payment information, is properly encrypted and not exposed.
* **Performance**: Ensure pages load quickly, and the system can handle a large number of concurrent users.

**3.3 Integration Testing**

* **Clerk Authentication**: Ensure Clerk integrates correctly with the app and provides user data when needed.
* **MongoDB and Mongoose**: Test that data is saved, retrieved, and updated correctly in MongoDB.
* **Stripe**: Ensure the payment process works with Stripe, including refund functionality.

**3.4 User Acceptance Testing (UAT)**

* **Usability**: Ensure the system is intuitive for users to sign in, browse courses, and unlock content.
* **Functionality**: Ensure that all features, like viewing tutorials and purchasing courses, work as expected.

**4. Test Cases**

**4.1 Authentication Testing**

* **Test Case 1**: Verify that users can sign up with Google.
* **Test Case 2**: Verify that users can log in with custom credentials.
* **Test Case 3**: Verify that only authenticated users can access courses.

**4.2 Course Management Testing**

* **Test Case 4**: Verify that courses are listed correctly.
* **Test Case 5**: Verify that a user can view a specific course page.

**4.3 Tutorial Management Testing priority-high**

* **Test Case 6**: Verify that tutorials for a course are displayed.
* **Test Case 7**: Verify that tutorial links open the correct content URL.

**4.4 Stripe Payment Flow Testing -priority-high**

* **Test Case 8**: Verify that users can unlock a course after a successful payment.
* **Test Case 9**: Verify that failed payments are handled correctly (e.g., showing an error message).

**4.5 Error Handling Testing**

* **Test Case 10**: Verify that API errors (e.g., failed fetch) are handled gracefully.
* **Test Case 11**: Verify that missing content or courses displays appropriate error messages.

**4.6 UI/UX Testing**

* **Test Case 12**: Verify that the homepage loads correctly and displays courses.
* **Test Case 13**: Verify that course images and content display correctly.

**5. Test Execution**

**5.1 Test Execution Strategy**

Tests will be executed using Jest for unit and integration tests and Cypress for E2E testing. Each test will be part of a continuous integration (CI) pipeline to ensure that every push to the codebase is validated.

**5.2 Test Environment Setup**

Tests will be executed in a dedicated environment using:

* Mocked API responses (MSW)
* Dedicated test database (MongoDB Atlas)
* Test Stripe keys for payments

**5.3 Test Execution Plan**

Test cases will be executed in the following order:

1. Unit tests for API and components
2. Integration tests for Clerk, MongoDB, and Stripe
3. UI/UX testing in the staging environment
4. Final end-to-end testing in production-like environments

**6. Defect Management**

**6.1 Defect Life Cycle**

Each defect will go through the following life cycle:

* New
* Assigned
* In Progress
* Fixed
* Closed

**6.2 Defect Reporting**

Defects will be reported via the project’s issue tracker (e.g., GitHub Issues, Jira) with the following details:

* Defect ID
* Description
* Severity
* Priority
* Steps to reproduce

**6.3 Defect Severity and Priority**

Defects will be categorized into:

* **Critical**: Stops the application from functioning.
* **Major**: Major functionality is broken, but the system can continue to run.
* **Minor**: Small issues that don’t significantly affect the user experience.

**7. Test Completion Criteria**

**7.1 Completion Criteria**

Testing will be considered complete when:

* All critical and major defects are resolved.
* All high-priority test

cases have passed.

* User Acceptance Testing has been signed off.

**7.2 Exit Criteria**

The project will exit the testing phase when:

* Regression testing confirms that no new bugs have been introduced.
* When 70%of the test cases are in please and successful
* The project is stable and ready for deployment.

TEST CASES SAMPLE THE FIRST AND THE SECONF PRIORITAZED THE PAYMENT AND THE AUTHENTICATION

* **Test Case 8**: Verify that users can unlock a course after a successful payment.

Assumptions 1Test case 1 Test case 2, test case 3 test case 4 and all the necessary test cases that depends on

**Step 2 Initiate the Payment Process:**

* Click the "Unlock" or "Buy" button on the locked course.
* Verify that the payment form/modal redirects to Stripe's checkout page or displays a payment gateway interface.

**Step 4 a- Enter Valid Payment Details(valide test data)**

Use Stripe’s provided **test credit card number** (Visa: 4242 4242 4242 4242

MasterCard: 5555 5555 5555 4444

American Express: 3782 8224 6310 005

Discover: 6011 1111 1111 1117

JCB: 3530 1113 3330 0000).

* Fill in valid test details for expiration date, CVV, and ZIP code \_any valid future date, 3 decimal digits and 5 digits decimal.
* Submit the payment.
* Then finally log the output ( e.g successfully paid ,)

Note the above inputs fields are tests one input field at a time

**Step 5 Verify Payment Success:**

* Ensure the app redirects to a confirmation or success page after payment.
* Confirm that the payment status is updated in the database (e.g., course status changed to “unlocked” and Stripe transaction ID saved).
* Check for a success toast or confirmation message (e.g., "Course unlocked successfully!").

**Steps 6 Access the Unlocked Course:**

* Go back to the course list or course details page.
* Ensure the course is no longer marked as "locked" and is accessible.
* Verify all course content (e.g., tutorials, resources) is viewable.

**Steps 7 Validate with Stripe Dashboard:**

* Log into the Stripe dashboard (test mode).
* Confirm the payment entry appears in the dashboard with the correct amount and description.

**Step 8 Expected Result:**

* The user is redirected to a confirmation page, sees a success message, and the course is marked as unlocked and accessible.

**Step4 b verfiy it return for**

* An error message is displayed to the user.
* The course remains locked and inaccessible.
* No successful payment entries should appear in the Stripe dashboard.
* **Test Case 9**: Verify that failed payments are handled correctly (e.g., showing an error message).

Assumption 1

**Step 1 Initiate the Payment Process:**

* Click the "Unlock" or "Buy" button on the locked course.
* Verify that the payment form/modal redirects to Stripe's checkout page or displays a payment gateway interface.

**Step 2 Enter Invalid Payment Details:**

* Use Stripe’s provided **test credit card number for declined payments** (e.g., 4000 0000 0000 9995 and any other numbers other than the test numbers).
* Fill in valid test details for expiration date, CVV, and ZIP code.
* Submit the payment. GOTO STEP 3,4
* Use Stripe’s provided **test credit card number for payments** (e.g., 424242442424242).
* Fill in valid test details for expiration date, CVV, and ZIP code.

1 not future date GOTO STEP 3 ,4

2 test for less than 3 digit( 21) ,more then 3 digit(83739), and non digit (hey) for CVV

GOTO STEP 3,4

3 Test for less than 5 digit(4783),more than 5 digit (589393) and non digi (zip123)t for

ZIP GOTO STEP 3,4

* Submit the payment.
* Note every alerts and logs

**Step 3 Confirm the Course Status:**

* Go back to the course list or course details page.
* Ensure the course is still marked as "locked" and remains inaccessible.

**Step 4 Validate with Stripe Dashboard:**

* Log into the Stripe dashboard (test mode).
* Confirm the failed payment entry appears in the dashboard with the correct amount and reason for failure.

### ****Test Case 1: Verify that users can sign up with Google****

#### Assumptions:

* Google sign-up is integrated using Clerk.
* The system handles both successful and failed sign-up attempts.

#### Test Steps:

**1.Navigate to the Sign-Up Page**:

* + Open the application and click "Sign Up."

**2.Select Google Sign-Up**:

* + Click the "Continue with Google" button.

**3a.Provide Valid Google Credentials**:

* + Use a valid test Google account (e.g., testuser@gmail.com) to sign up.
  + Authenticate successfully and allow permissions.

**Expected Outcome** (Valid Input):

* + The user is redirected to the course/home page.
  + A new record for the user is created in the database, containing:
    - Name
    - Email
    - Unique Google ID
  + A success toast is displayed (e.g., "Sign-up successful!").

**Invalid Inputs (Google Sign-Up Fails)**:

* + Try signing up with a disabled Google account.
    - Expected: An error message is displayed (e.g., "Google account not accessible"), and no user is created.
  + Try canceling the Google authentication process midway.
    - Expected: The process stops, and the user is returned to the sign-up page.
  + Use an incorrect email/password combination.
    - Expected: Google prompts for re-entry of credentials without proceeding.

### ****Test Case 2: Verify that users can log in with custom credentials****

#### Assumptions:

* The system supports email/password-based authentication via Clerk.

#### Test Steps:

##### ****Valid Inputs (Login Success)****:

1. **Navigate to the Login Page**:
   * Open the application and click "Log In."
2. **Enter Valid Credentials**:
   * Email: validuser@test.com
   * Password: ValidPass123!
3. **Submit**:
   * Click the "Log In" button.
4. **Expected Outcome**:
   * The user is redirected to the course list/home page.
   * A success toast appears (e.g., "Login successful!").
   * Session details (e.g., tokens or cookies) are stored securely.

##### ****Invalid Inputs (Login Fails)****:

1. **Incorrect Email Format**:
   * Email: invalidemail.com (missing @).
   * Expected: Error displayed (e.g., "Invalid email format.").
2. **Unregistered Email**:
   * Email: unknownuser@test.com.
   * Password: AnyPassword123!.
   * Expected: Error displayed (e.g., "Account not found.").
3. **Incorrect Password**:
   * Email: validuser@test.com.
   * Password: WrongPass456!.
   * Expected: Error displayed (e.g., "Invalid credentials.").
4. **Short Password**:
   * Email: validuser@test.com.
   * Password: 123 (less than 8 characters).
   * Expected: Error displayed (e.g., "Password must be at least 8 characters long.").
5. **Empty Fields**:
   * Email: (empty).
   * Password: (empty).
   * Expected: Error displayed (e.g., "Email and password are required.").

### ****Test Case 3: Verify that only authenticated users can access courses****

#### Assumptions:

* The application uses session-based or token-based authentication to restrict access to courses.

#### Test Steps:

##### ****Access Without Logging In****:

1. **Navigate to Course List or Details Page**:
   * Open the application and try to view the course list or a specific course while logged out.
   * Expected:
     + User is redirected to the login/sign-up page.
     + A message is displayed (e.g., "Please log in to access courses.").

##### ****Access After Logging In****:

1. **Log In with Valid Credentials**:
   * Use Google or custom login credentials to log in successfully.
2. **Access Course List**:
   * Navigate to the course list.
   * Expected:
     + Paid/unlocked courses are accessible.
     + Locked courses display an "Unlock" button.
3. **Log Out and Re-Access**:
   * Log out and try to access the course list again.
   * Expected:
     + User is redirected to the login page.
     + Locked courses remain inaccessible.

##### ****Test with Tampered Sessions****:

1. **Manually Modify the Auth Token**:
   * Tamper with the stored authentication token or session cookie.
   * Expected:
     + The system invalidates the token and redirects the user to the login page.
     + A message is displayed (e.g., "Session expired. Please log in again.").

### ****Test Case 4: Validate Database Integrity for Authentication****

#### Assumptions:

* All user data (e.g., email, password hash) is stored securely.

#### Test Steps:

**1.Verify Password Storage**:

* + Check the database after signing up or logging in.
  + Expected:
    - Passwords are stored as secure hashes (e.g., bcrypt).
    - No plaintext passwords should be visible.

**2.Test Duplicate Email Prevention**:

* + Try signing up with an existing email (e.g., validuser@test.com).
  + Expected:
    - Error displayed (e.g., "Email already in use.").

**Validate Email Uniqueness**:

* + Attempt to register multiple accounts using the same email address but different cases (e.g., ValidUser@test.com and validuser@test.com).
  + Expected:
    - System treats emails as case-insensitive.
    - Error displayed for duplicate email registration.

### ****Traceability Matrix****

| **Requirement** | **Test Case ID** | **Test Steps** | **Valid Test Data** | **Invalid Test Data** | **Expected Behavior** |
| --- | --- | --- | --- | --- | --- |
| Users can sign up with Google | TC-1 | Navigate to the sign-up page, select "Continue with Google," and authenticate successfully. | Test Google account: testuser@gmail.com | Disabled account, canceled sign-up midway. | User redirected to home, record created in DB, success toast displayed; invalid sign-up shows error messages. |
| Users can log in with custom credentials | TC-2 | Navigate to the login page, enter valid email/password, and submit. | Email: validuser@test.com, Password: ValidPass123! | Invalid email format, unregistered email, wrong password, short password, or empty fields. | Successful login redirects to home with session saved; invalid attempts display appropriate error messages. |
| Only authenticated users can access courses | TC-3 | Attempt to access courses without logging in, log in, and retry access. | Valid login credentials (via Google or custom email/password). | Attempt access while logged out or with tampered session token. | Redirect to login for unauthorized access; authenticated users see paid/unlocked courses and locked courses with "Unlock" buttons. |
| Payments unlock courses | TC-8 | Initiate payment process, enter valid payment details, and confirm the success of unlocking a course. | Stripe test cards (Visa: 4242 4242 4242 4242, etc.), valid expiration, CVV, ZIP. | Declined cards (e.g., 4000 0000 0000 9995), invalid expiration, CVV, ZIP, or other invalid details. | Successful payment unlocks the course, changes status in the DB, and displays a confirmation; invalid payments show errors. |
| Failed payments handled correctly | TC-9 | Attempt payment with invalid details and confirm the system response. | N/A | Invalid Stripe test cards, invalid CVV (e.g., 21, hey), invalid ZIP (e.g., 1234, zip123), invalid expiration date. | Course remains locked, error message shown to the user, and failed payment entry logged in the Stripe dashboard. |
| User data is stored securely and validated | TC-4 | Verify password storage in DB, test duplicate email prevention, and case-insensitive email handling. | Valid user data (validuser@test.com, password hashed with bcrypt). | Attempt duplicate email registration, same email with different cases (e.g., ValidUser@test.com). | Passwords stored securely (hashes), duplicate email registration is prevented, and email validation is case-insensitive. |

### ****Test Data****

| **Input Field** | **Valid Test Data** | **Invalid Test Data** | **Expected Behavior** |
| --- | --- | --- | --- |
| **Email** | user@test.com | invalidemail.com, unknown@.com, (empty field) | Error for invalid format, empty email, or unregistered email. |
| **Password** | ValidPass123! | 123, password, Pass!, (empty field) | Error for short (<8 characters), weak, or empty passwords. |
| **Google Account** | Test Google account: testuser@gmail.com | Disabled Google account, canceled authentication, or incorrect credentials. | Sign-up fails and shows an error for inaccessible accounts, canceled authentication, or invalid credentials. |
| **Auth Token** | Valid session token | Tampered session token or expired token. | System invalidates the token, redirects to login, and displays an appropriate error message. |
| **Card Number** | Stripe test cards: 4242 4242 4242 4242 | Declined cards: 4000 0000 0000 9995, invalid numbers (e.g., 4242 4242, testcard) | Error displayed and course remains locked. |
| **Expiration Date** | Any valid future date | Past dates or invalid format (e.g., 13/25, test). | Error displayed and course remains locked. |
| **CVV** | 3-digit number (e.g., 123) | Invalid CVV: less than 3 digits (e.g., 12), more than 3 digits (e.g., 1234), non-digits (e.g., cvv123, abc). | Error displayed and course remains locked. |
| **ZIP Code** | 5-digit number (e.g., 12345) | Invalid ZIP: less than 5 digits (e.g., 1234), more than 5 digits (e.g., 123456), non-digits (e.g., zip123). | Error displayed and course remains locked. |

This traceability matrix ensures that all test cases are traceable to their respective requirements and tested with both valid and invalid input data.