**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 document provides the testing requirements for the Next.js-based course management application, including authentication, tutorial management, course listing, payment flow, maintains tutorial on the cloud Cloudinary and more. It 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. 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 document outlines the testing strategies, tools, and test cases required to ensure the successful functionality and quality of the Next.js-based 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.
* **Integration Tests**: Test interactions between services (e.g., Clerk authentication, MongoDB).
* **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.

**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**

* **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**

* **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.
* The project is stable and ready for deployment.