**Intro App Dev React App Test Plan**

**1) TEST PLAN IDENTIFIER**

Document Version:

1. (Initial release)

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 01/11/2023 | 1.0 | Initial release | Crystal Sawers |

**2) REFERENCES**

* README.md for React app repo: Explains how to set up and run the frontend environment, the components used, and how it interacts with the API.
* README.md for Node.js API: Provides details on the backend setup/environment, endpoints, and data models.

**3 INTRODUCTION**

**Purpose:**

This Test Plan outlines the quality assurance and testing strategy for the Gym Workout Database React App. The document serves as a Master Test Plan, covering various aspects of the application, from data management to user interface validation.

**Scope:**

The plan complements the overarching Software Project Plan by detailing the testing methodologies for CRUD operations on users, workouts, exercises, and equipment, as well as data validation and navigation functionalities. It does not encompass performance and security testing. Resource and budget constraints have been considered, aligning the test plan with the main project objectives.

**4) TEST ITEMS (FUNCTIONS)**

* Creating a new user
* Updating an existing user
* Deleting an existing user
* Creating a new workout
* Updating an existing workout
* Deleting an existing workout
* Creating a new exercise
* Updating an existing exercise
* Deleting an existing exercise
* Creating new equipment
* Updating existing equipment
* Deleting existing equipment
* Displaying tables of data: Users, Workouts, Exercises, Equipment
* Input and Data Validation when creating a user, workout
* Navigation bar which navigates to data for users

**5) SOFTWARE RISK ISSUES**

**A. Third-Party Libraries and Packages**

The application uses several third-party libraries for both the Node API and React frontend. There's a risk that these packages may update and cause compatibility issues.

**B. Interfacing Software**

The React app interfaces with a Node API. Changes or updates in the API might break existing functionality in the frontend.

**C. Tooling and Packages**

There may be a learning curve associated with understanding specific libraries or packages that the application relies on, causing a delay in testing efforts.

**D. Complex Functions**

The CRUD operations and data validation functions can be complex and have multiple edge cases that need thorough testing.

**E. Historical Issues**

If any components of the app have had issues in the past, such as bugs found during unit testing, they should be flagged for more rigorous testing.

**F. Documentation**

While READMEs are available for both the API and React app, some modules or features might not be as well-documented, increasing the risk of misunderstanding during testing.

**Inherent Software Risks:**

**A. Safety**

Data integrity and validation are crucial as the application deals with user-generated workout and exercise data.

**B. Multiple Interfaces**

The application involves interaction between the frontend and backend, each having its components and libraries, increasing the complexity of the test scope.

**C. Impacts on Client**

Malfunctions can lead to data loss or inaccuracies, affecting the user's experience and trust in the application.

**D. Regulatory Compliance**

While not directly applicable for a gym workout database, it's essential to be aware of any data storage or privacy regulations that might come into play, such as GDPR if your app has a global user base.

**Misunderstanding of Original Requirements:**

Vague or unclear requirements can lead to incorrect implementations and should be flagged early on.

**Historical Defects:**

Pay attention to areas where defects have clustered in the past, as these are often indicators of future issues.

**Brainstorming Sessions:**

It's advisable to hold brainstorming sessions to collectively identify what could go wrong. Typical starting points for these discussions could be questions like, "What worries me about this project/application?"

**6) FEATURES TO BE TESTED**

**1. User Management (Risk: High)**

Creating a new user

Updating an existing user

Deleting an existing user

**Why High Risk:** User management is central to the application. Mistakes here can lead to data loss or unauthorized data access.

**2. Workout Management (Risk: Medium)**

Creating a new workout

Updating an existing workout

Deleting an existing workout

**Why Medium Risk:** While important, workout data is less critical than user data. However, errors can still significantly impact the user experience.

**3. Exercise Management (Risk: Medium)**

Creating a new exercise

Updating an existing exercise

Deleting an existing exercise

**Why Medium Risk:** Similar to workouts, exercise data is important for the user but less critical than user information.

**4. Equipment Management (Risk: Low)**

Creating new equipment

Updating existing equipment

Deleting existing equipment

**Why Low Risk:** While useful, equipment data is the least critical and its absence or errors are less likely to impair basic app functions.

**5. Data Display (Risk: Medium)**

Displaying tables of data: Users, Workouts, Exercises, Equipment

**Why Medium Risk:** Accurate data display is important for user trust and effective use of the application, but it's not as critical as user or workout data.

**6. Input and Data Validation (Risk: High)**

Input validation when creating a user or a workout

**Why High Risk:** Incorrect validation can lead to invalid data, affecting both user management and workout tracking.

**7. Navigation (Risk: Low)**

Navigation bar which navigates to data for users

**Why Low Risk:** Navigation is essential for a good user experience but does not pose a risk to data integrity or security.

**7) FEATURES NOT TO BE TESTED**

To be decided, test out what’s not working in react app first

**8) APPROACH (STRATEGY)**

**Overall Strategy:**

The testing approach will be phased, starting with Unit Testing to validate individual pieces of code, followed by Component Testing with Cypress for user interface elements, Integration Testing to ensure that the individual components work well together, and finally System Testing to confirm the overall functionality.

**Tools:**

* Cypress for Component Testing
* Jest/React Testing Library for Unit Testing
* Postman for API/Integration/System Testing

**Special Training:**

Jest and Testing Library will require some learning for configuring the testing environment and writing simple unit tests. Online documentation and tutorials will be used for training.

**Metrics Collected:**

Number of test cases passed/failed

Coverage percentage

Time taken for test execution

**Configuration Management:**

Git will be used for source code version control.

Test cases and results will be documented.

**Configurations Tested:**

Local development environment

Staging environment

**Hardware and Software:**

Testing will be carried out on multiple browsers (Chrome, Firefox, Safari).

Backend testing will include database and server validations.

**Regression Testing:**

Regression testing will be done at each test level.

Tests will be rerun after each significant change to ensure existing functionalities are not broken.

**Un-testable Elements:**

Features or requirements that are found to be untestable will be documented and communicated to the development and product management teams for clarification.

**Special Requirements:**

Due to inter-dependencies, user management tests should be run before workout and exercise management tests.

**Meetings:**

Weekly meetings will be held to discuss test progress, challenges, and next steps.

**9) ITEM PASS/FAIL CRITERIA**

**Unit Testing:**

* All test cases must be completed.
* Code coverage should be at least 85%.
* No critical defects should be open; minor defects should be less than 5.
* Integration Testing (Including API Testing with Postman):
* All API endpoints should return the expected status codes and outputs.
* Less than 2% failed tests allowed for non-critical functionalities.
* No failures allowed for critical functionalities like user authentication and data integrity.

**System Testing:**

* All lower-level plans should be completed.
* No critical bugs should be open.
* System should be able to handle specified load.

**Component Testing with Cypress:**

* All test cases must be executed.
* Visual checks should align with expected UI/UX designs.

**Exploratory Testing:**

No showstopper or critical defects should be found.

**General:**

* Overall defect rate should be less than 5% of total test cases.
* Severity 1 defects (showstoppers, critical defects) should be 0.

**Metrics for Defects:**

Defect: A defect is a deviation from the expected result which may or may not result in a failure.

Failure: A failure is the manifestation of a defect as observed by the end-user, causing incorrect operation, system crashes, or data corruption.

**10) SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS**

**Suspension Criteria:**

Testing will be suspended under the following conditions:

* Critical Defects: If more than 3 critical defects are found that compromise the core functionality of the application, testing will be halted for immediate fixing and review.
* System Downtime: If the test environment becomes unstable or unavailable, testing will be paused until it is restored.
* Dependency Failure: If an external system or service that the application is dependent upon fails or becomes unavailable.
* Data Corruption: If a defect leads to the corruption of the test data, thereby invalidating the test results.
* Resource Unavailability: Lack of essential testing resources such as hardware, software, or personnel.

**Resumption Requirements:**

Testing activities will resume when:

* Critical Defects: All identified critical defects have been fixed and verified.
* System Stability: The test environment is stable and operational.
* Dependencies: All external systems or services are confirmed operational.
* Data Integrity: Test data has been restored or replaced, and its integrity confirmed.
* Resource Availability: All essential resources are available for testing.

**Ghost Errors:**

Care will be taken to avoid ghost errors, which can occur because of defects identified but not immediately fixed. These can falsely appear as new defects but are actually the result of previous, unaddressed issues.

**11) TEST DELIVERABLES**

* **Test Plan Document:** This current document, outlining the scope, approach, resources, and schedule of the testing activities.
* **Test Execution Document:** This document details the steps, environment, and outcomes for each test case executed, serving as a comprehensive log of the testing phase.
* **Test Summary Report:** A high-level overview that summarizes the testing activities, results, and key metrics, intended for stakeholders to quickly assess the state of the application.

**12) REMAINING TEST TASKS**

This test plan focuses on the core functionalities of the Gym Workout Database React App such as user, workout, exercise, and equipment management. Several features and components are out of scope for this iteration and will be considered in future test plans.

These include:

* **User Authentication:** This version of the test plan does not include testing for user authentication features such as sign-in, sign-up, OAuth, two-factor authentication, or password recovery. These features are planned to be tested in subsequent cycles.
* **Performance Testing:** While the current test plan does involve component and integration testing, performance testing like load and stress testing are not included and will be part of future test plans.
* **Accessibility Testing:** Accessibility features and compliance with standards like WCAG are not covered in this test plan but are slated for future testing cycles.
* **Multi-platform Testing:** This iteration only tests the application on a predetermined set of platforms and browsers. Extensive multi-platform and multi-browser testing will be conducted in the future.
* **Localization and Internationalization:** These aspects are currently out of scope and will be considered for future testing.
* **Deployment and URL Accessibility:** The React app will eventually be deployed to a specific URL for user access. However, testing related to deployment and URL accessibility is not covered in this current test plan.
* **Missing API Endpoints/Features:** The following API endpoints are currently not included in this test plan due to them not yet being implemented. They will be considered for future testing.
* /api/v1/workoutlogs
* /api/v1/musclegroups
* /api/v1/auth/register
* /api/v1/auth/login
* /api/v1/auth/logout

By identifying these areas, we aim to prevent any resource wastage on functionalities that are not within the scope of this test plan and set clear expectations for stakeholders regarding the test coverage.

**13) ENVIRONMENTAL NEEDS**

* **Special Hardware:** No special hardware required, but tests should be performed on different devices in the web interface as it’s a web application like smartphones, tablets, and desktops to ensure responsiveness.
* **Test Data:** Pre-filled database, which has been recently updated using Render and PostgreSQL, with sample workout and user information for testing CRUD operations. No special collection requirements but should cover edge cases.
* **Component Testing:** Each major component (User Profiles, Workouts, Exercises, Equipment) should be subjected to individual tests along with integrated tests.
* **Power Requirements:** Standard power needs, as testing will be conducted on regular computing systems.
* **Software Versions:** Clearly specify the versions of React, Node.js, PostgreSQL database, and any other major libraries you're using.
* **System Access:** Testing should be done in a separate test environment to avoid disrupting the main application. Database rollbacks or backups are to be performed to reset the state after testing.

**14) STAFFING AND TRAINING NEEDS**

As it’s a solo testing project, I'll handle all the training on tools like Cypress and Postman using online resources. I am responsible for all aspects of testing and training.

**15) RESPONSIBILITIES**

I oversee the entire testing process, identify and set risk levels for each feature, and decide which features will be tested. I'll set the testing strategy and manage my time effectively for both development and testing.

**16) SCHEDULE**

Given that I've already completed the development of the project, my testing schedule is as follows:

* **Component Testing:** I'll start this immediately to evaluate each individual component's functionality.
* **Integration Testing:** This will kick off as soon as I've completed component testing, to ensure different parts of the project work together as expected.
* **System Testing:** After integration testing is successful, I'll move on to system testing to validate the application as a whole.
* **Cypress for Front-end Testing:** To verify the front-end components, I'll use Cypress to automate this process.

If any unexpected issues come up during testing, I'll evaluate their impact. For minor setbacks, I'll adjust my schedule. For significant delays, I'll focus on testing the most crucial features to ensure they are stable and reliable.

All my testing activities are dependent on their respective development milestones. If a delay occurs, the corresponding testing activity will also be delayed, eliminating any perception that the testing process is the cause of the hold-up.

**17) PLANNING RISKS AND CONTINGENCIES**

There are several potential risks to this project, especially concerning the testing process. As the sole person responsible for this project, the risks are primarily:

* **Time constraints:** As I am the only one working on the project, with only 2 weeks left to complete the testing project, delays in one area could affect the whole schedule.
* **Software/Hardware Availability:** Lack of access to necessary hardware or software can delay testing activities, such as databases and api urls from an older project.
* **Late Changes:** Any late changes to requirements or designs could necessitate re-testing, adding time and complexity.

**In case of these events, the following actions will be taken:**

If there are any changes in the requirements or designs after they have been finalized, I will re-evaluate the testing priorities and reschedule as necessary.

In case of delays due to external factors like hardware/software availability, I will focus on testing that can be done with available resources while waiting for the missing pieces.

If I encounter unexpected delays, I'll prioritize the most critical testing tasks to ensure the core functionalities are robust.

I may decide to work extra hours to keep the project on track, while being mindful of how this could affect my morale and overall efficiency.

It's crucial to note that some of these actions could potentially lower the overall quality of the final product, such as reducing the number of tests performed or increasing the number of acceptable defects. These will be last-resort measures.

**18) APPROVALS**

|  |  |  |
| --- | --- | --- |
| Project Manager | Crystal Sawers | 1/11/2023 |
| Development Lead | Crystal Sawers | 1/11/2023 |
| Test Lead | Crystal Sawers | 1/11/2023 |
| Stakeholder (if any) | [Name] | 1/11/2023 |