Diverse Makers

Software Testing Plan

Version 1.0

11-08-2024

Project Sponsor: Dr. Jared Duval

Faculty Member: Isaac Shaffer

Project Mentor: Vahid Nikoonejad Fard

Team members:
Daniel Minichetti (Team Lead)

Kane Davidson

Eduardo De La Rosa

Elleana Negrelli

Aaron Ramirez

Overview

The purpose of this document is to outline the testing plan activities for our application, detailing the unit, integration, and usability testing procedures that will ensure our application will meet its goal of providing an accessible platform for individuals with disabilities.

Table of Contents

1.0 INTRODUCTION

Our Diverse Makers application aims to bridge a critical gap in STEM education by connecting individuals with disabilities to maker spaces and STEM learning resources. With over 40 million Americans having a disability and only 3% representation in the STEM workforce, our mobile application serves as a centralized platform for sharing accessible STEM content and fostering connections between users and maker spaces. Our primary goals include providing an accessible user interface for multiple disabilities, secure hosting of STEM learning resources, and facilitating connections with local maker spaces through features such as location-based maker space discovery. The application will be built using React Native for the front end implementation and Firebase for back end services, ensuring cross-platform compatibility and real-time data synchronization. It will also enable users to create an account, access and contribute to guides for STEM activities using a block-based interface, and customize their experience with intuitive accessibility settings.

Software testing is vital in ensuring that applications meet their intended functionality while maintaining reliability and usability standards. This is particularly crucial for our application, where accessibility and ease of use are not just features but fundamental requirements. Software testing allows us to systematically verify that each module functions correctly in isolation, works properly when integrated with other modules, and provides a positive user experience. Through comprehensive testing, we can not only identify issues but address them early in development, reducing inefficiencies while ensuring a high-quality final product. Additionally, through systematic testing, we can verify that our implementation meets both functional requirements (i.e. user authentication and content management) and non-functional requirements (i.e. accessibility and performance).

Our testing plan encompasses three main testing approaches, each targeting different aspects of the application and distribution of effort:

1. Unit testing (50% of testing effort) will focus on validating individual components and functions, particularly around our core features:
    a. User authentication and profile management

b. STEM activity creation and block-based content management
c. Search and filtering functionality
d. Accessibility settings management

2. Integration testing (25% of testing effort) will verify the correct interaction between major system components:
    a. front end and back end communication via Firebase
    b. Data flow between different application screens
    c. Block manipulation and content updates
    d. Settings persistence across sessions

3. Usability testing (25% of testing effort) will ensure the application meets accessibility goals:
    a. Interface navigation and content discovery
    b. Block-based content creation workflow
    c. Accessibility features effectiveness (font size adjustment and high contrast mode)
    d. Overall user experience for individuals with various disabilities

The structure and intensity of our testing plan have been carefully designed to meet our application's unique requirements and user needs. Given our focus on accessibility and increasing STEM learning outcomes, we've allocated the largest portion (50%) to unit testing to ensure a solid foundation for our application's accessibility-related features and core functionality. Our equal distribution between integration testing (25%) and usability testing (25%) reflects the careful consideration of our application's architecture. Integration testing will ensure reliable data flow, state management, and smooth operation between our front end and back end across both IOS and Android platforms. Usability testing will validate our accessibility features, allowing us to create an interface that truly works for users with multiple disabilities. This includes in-depth testing with screen readers, different contrast settings, and various font sizes to promote universal accessibility.

This testing strategy is distributed based on component criticality and our user's needs. For example, the block-based content system receives intensive unit testing due to its central role in content creation and collaboration. Similarly, accessibility settings will undergo rigorous

testing to ensure reliable functionality across different screen sizes and conditions. This approach ensures we allocate testing resources where they will have the greatest impact on user experience and application reliability.

In the following sections, we detail our testing methodologies, success criteria, and specific test cases for each testing type. This comprehensive testing plan will help ensure our Diverse Makers application is validated and successfully serves its mission of making STEM education more accessible to individuals with disabilities.

2.0 UNIT TESTING

Unit testing is a foundational software testing methodology that involves testing individual components, functions, and methods in isolation to verify their behavior and ensure they function properly on their own.

The primary goals of our unit testing are to:

- Verify that each component functions correctly and independently
- Identify and fix bugs early in the development process
- Facilitate safe refactoring or code changes of any kind
- Provide valid documentation of component behavior

For our Diverse Makers application, unit testing is crucial due to our emphasis on accessibility and the need for reliable core functionality. With this in mind, we defined a unit as a distinct piece of functionality that can be tested independently. This includes individual methods, functions, and small components that serve specific purposes within our application. For example, a user authentication method, an activity manipulation function, or a settings management operation would each constitute a unit.

While comprehensive unit testing of each method might seem ideal, we've strategically chosen to focus our unit testing efforts on critical components that directly impact user experience and accessibility features. This decision was primarily driven by time constraints, but

also because most React Native and Firebase functions are already well-tested by their respective frameworks.

2.1 Testing Tools and Metrics

- Primary Framework: Jest (standard testing framework for React Native applications)
- Component Testing: React Native Testing Library
- Coverage Metrics: Aiming for 80% coverage code across critical paths
- Test Environment: Node.js-based testing environment with appropriate mocks for React Native and Firebase functionalities

2.1.1 Testing Processes

Our testing process will follow these steps:

1. Write test specifications based on component requirements
2. Implement test cases using Jest and supporting libraries
3. Run tests automatically on each commit using GitHub Actions
4. Generate and review coverage reports
5. Address any failed test or coverage gaps

2.2 Core Testing Areas

2.2.1 Authentication Module

- Description: Validates user registration, login, and profile management functionality.

Key Test Cases:

1. User Registration
    a. Equivalence partitions:
        i. Valid registration data (i.e. well-formed email, strong password)
        ii. Invalid email formats
        iii. Missing required fields

b. Boundary Values:
    i. Minimum/Maximum password lengths
    ii. Username character limits
    iii. Email length constraints
c. Error Cases:
    i. Duplicate email addresses
    ii. Password complexity requirements not met
    iii. Invalid character types in fields

2. Login Authentication
    a. Equivalence Partitions:
        i. Valid credentials
        ii. Invalid email/password combinations
        iii. Non-existent accounts
    b. Error Conditions:
        i. Network timeout scenarios
        ii. Invalid session states

2.2.2 STEM Activity Management System

- Description: Tests the creation, modification, and retrieval of STEM learning content.

Key Test Cases:

1. Block Creation
    a. Type Validation:
        i. Text block content limits
        ii. Image URL validation
        iii. Video URL format verification
    b. Insertion Points:
        i. Between existing blocks
        ii. End of activity
        iii. Beginning of activity
    c. Error Cases:

i. Invalid URL formats
ii. Empty content
iii. Maximum block count limits

2. Block Manipulation
    a. Sequence Operations:
        i. Block deletion
        ii. Block insertion between existing blocks
        iii. Block reordering (if implementation is complete)
    b. Content Validation:
        i. Maximum content length
        ii. supported media types
        iii. Content sanitization

2.2.3 Search and Filter System

- Description: Ensures accurate content discovery and filtering functionality.

Key Test Cases:

1. Search Operations
    a. Input Variations:
        i. Single keywords
        ii. Multiple word phrases
        iii. Special characters
        iv. Empty searches
    b. Result Validation:
        i. Relevance ranking
        ii. Result limit handling (i.e. pagination)
        iii. No match scenarios

2. Filters
    a. Filter combinations:
        i. Single filter selection
        ii. Multiple filter selection

iii. Filter removal

b. Edge Cases:

i. All filters selected

ii. No filters selected

iii. Invalid filter combinations

2.2.4 Accessibility Settings System

- Description: Validates the configuration and persistence of accessibility features.

Key Test Cases:

1. Font Size Management
    a. Size ranges:
        i. Minimum font size
        ii. Maximum font size
        iii. Intermediate increments
    b. Persistence:
        i. Settings storage
        ii. Cross-session retention
        iii. Default value handling

2. High Contrast Mode
    a. Toggle States:
        i. State persistence
        ii. Default state handling
        iii. Enable / Disable functionality
    b. Visual Verification:
        i. Color contrast ratios
        ii. UI element visibility
        iii. Text readability

2.3 Testing Implementation Details

For each test area, we will implement the following specific test cases:

1. Positive Tests: Verify expected behavior with valid inputs
2. Negative Tests: Ensure proper error handling with invalid inputs
3. Edge Cases: Test boundary conditions and extreme scenarios
4. Performance Aspects: Validate response times and resource usage

2.5 Mocking Strategy Details

To ensure isolated testing, we will utilize Jest's built-in mocking capabilities for the following:

- Firebase Authentication: Mock user states and authentication responses
- Network Requests: Mock timeout scenarios and connection drops
- File System Operations: Mock successful file uploads and downloads
- Navigation Functions: Mock screen transitions and routing
- External Service Calls: Mock location service responses

2.4 Error Handling Coverage

Each test suite will include specific error scenarios for the following:

- Network Connectivity Issues: Sudden connection loss during operations
- Invalid Input Handling: Incorrect data types in activity submissions
- Resource Unavailability: Database connection failures
- State Management Errors: Inconsistent UI states and state transitions
- Authentication Failures: Invalid credentials handling and session expiration

This comprehensive unit testing approach ensures thorough validation of each component while maintaining isolation and reproducibility of test cases. With this strategy we can ensure a thorough validation of our application's core functionality while maintaining a focus on

accessibility and user experience. The test cases and coverage metrics will be regularly reviewed and updated based on user feedback we receive.

3.0 INTEGRATION TESTING

Integration testing represents a vital phase of our testing strategy, focusing on verifying the correct interactions and data exchanges between major components and modules within our application. While unit testing ensures individual components work correctly in isolation, integration testing validates that these components work together as intended, particularly at their interfaces and connection points. This is key for our accessibility-focused application, where smooth component interaction directly influences the user experience.

The primary goals of our integration testing are to:

- Verify correct data flow between front end and back end systems
- Ensure consistent accessibility feature implementation across component boundaries
- Validate real-time synchronization of content and user interactions
- Confirm proper error handling and recovery across system boundaries

For our Diverse Makers application, this testing phase is important due to our accessibility requirements. Even if individual components meet accessibility standards, the transitions and interactions between them must maintain accessibility without degradation. For example, a screen reader must seamlessly navigate between components, and font size changes must propagate consistently across all interface elements.

Our application's architecture presents several critical integration challenges. The combination of the Expo and React Native front end with the Firebase back end requires careful testing of asynchronous operations. Moreover, our block-based content system demands thorough verification of real-time updates and collaborative features. Most importantly, accessibility features must work flawlessly across component boundaries to provide a consistent experience for users with various disabilities. The following sections will delve into the specific integration points we'll test, our testing methodologies, and our approach for ensuring seamless

communication between all system components, with an emphasis on maintaining accessibility features across module boundaries.

3.1 Major Integration Points and Testing Approach

3.1.1 Front end to Back end Integration

- Description: This integration point ensures our React Native components correctly communicate with Firebase services.

What We'll Test:

- User authentication flow works between front end and Firebase Auth
- Activity data is correctly saved to and retrieved from Firebase
- Real-time updates flow properly from Firebase to the UI
- Error situations (like network issues) are handled appropriately

How We'll Test:

- Create test sequences that track data from UI input to Firebase storage
- Verify data retrieved from Firebase displays correctly in the UI
- Confirm error messages from Firebase reach the user interface
- Test offline capabilities and data syncing

3.1.2 Block System Integration

- Description: This covers how our block-based content system works together with data storage and user interface.

What We'll Test:

- Blocks can be created, moved, and deleted with proper data persistence
- Block content (text, images, videos) loads and displays correctly
- Multiple users can interact with blocks simultaneously
- Block order is maintained correctly in storage and display

How We'll Test:

- Create test scenarios that manipulate blocks and verify storage state

- Check that block changes for one user appear for others
- Verify block ordering remains consistent between saves
- Test block content and loading across different devices

3.1.3 Settings and Navigation Flow

- Description: This ensures app navigation and settings work consistently across all screens

What We'll Test:

- User settings persist across different screens
- Navigation between screens maintains proper state
- Accessibility settings remain consistent throughout the app

How We'll Test:

- Follow common user paths through the app
- Check settings persistence across app restarts

3.2 Test Environment and Tools

To run our integration tests, we'll use the following tools:

- Firebase Emulator Suite for back end service simulation
- Expo test environment for front end
- Network condition simulation for testing connectivity issues
- Test accounts with various permission levels

Implementation Details:

1. Start with basic connectivity tests between front end and back end
2. Add tests for data flow and state management
3. Verify error handling and edge cases
4. Test accessibility feature consistency

This approach will ensure our application components work together reliably while maintaining accessibility and usability for all users. Our testing strategy is designed to evolve alongside our application, with regular reviews and updates to test cases based on user feedback and newly identified edge cases. Through rigorous integration testing across all system boundaries, we aim to build a platform that connects individuals with disabilities to STEM activities while maintaining high standards of reliability and accessibility.

4.0 USABILITY TESTING

Usability testing is the critical evaluation method that focuses on how real users interact with our application, measuring their ability to accomplish intended tasks and their overall user experience completing workflows. For the Diverse Makers application, usability testing is especially important as our primary users include individuals with multiple disabilities who require an accessible and intuitive interface for engaging with STEM learning resources.

Unlike unit and integration testing, which focus on technical functionality, usability testing delves into the human factors of software interaction. This includes evaluating key aspects such as ease of learning, error tolerance, and user satisfaction. These factors are vital for our application given that users may have varying levels of technical proficiency and different accessibility needs.

The main goals for our usability testing are to:

- Verify that users with different disabilities can effectively navigate and use the application
- Ensure that STEM activities and learning resources are both accessible and comprehensible
- Validate that accessibility features not just meet but exceed user needs
- Identity and eliminate any barriers that might interfere with usage
- Measure overall user satisfaction and engagement with the application

These goals are especially important because poor usability could discourage users from engaging with STEM learning opportunities. This extends to accessibility barriers which could completely prevent some users from accessing content or limit the discovery of valuable makerspace resources. With this in mind, our testing strategy will emphasize real-word usage scenarios and incorporate feedback from multiple stakeholders, including individuals with various types of disabilities, Maker space representatives, STEM educators, and accessibility specialists. The following section will detail our specific testing methodologies, participant selection criteria, and evolution metrics.

4.1 Testing Considerations

Our usability testing strategy is shaped by several factors specific to our user base, these include:

4.1.1 User Demographics

- Description: Defines and categorizes our target user groups to ensure usability testing represents the range of individuals who will interact with our application.

Primary Users:

- Demographic: Individuals with various disabilities (i.e. visual, auditory, motor, cognitive)
- Age Range: 14 to 65 years
- Technology Familiarity: Varying levels of familiarity with technology
- Interests: Varying interest in multiple STEM areas

Secondary Users:

- Maker space Representatives
- Support Networks (i.e. caregivers)
- STEM Educators

4.1.2 Critical Usage Factors

- Description: Identifies and analyzes the essential accessibility requirements, environmental conditions and technical considerations that must be accounted for during usability testing.

Accessibility Requirements:

- Screen reader compatibility is essential for visually impaired users
- High contrast mode for low vision users
- Touch targets must be suitable for users with motor impairments
- Clear and consistent navigation for users with cognitive disabilities

Learning Context:

- Users may access content in many environments (i.e. home, educational settings)
- Different devices and screen sizes must be supported natively
- Internet connectivity may vary with application usage

4.2 Testing Methodology

Our testing approach will encompass three complementary methods, these include:

4.2.1 Expert Reviews

- Frequency: Will occur over the period of a week throughout testing development.

Participants:

- An Accessibility Specialist
- A STEM Education Professional

Process:

1. Experts conduct systematic reviews using standard protocols
2. Document findings using standardized evaluation forms
3. Participate in debrief sessions with the Diverse Makers team
4. Provide a written recommendation for application improvements

Focus Areas:

- WCAG 2.1 Compliance
- Navigation Patterns
- Content Presentation

- Error Handling and Feedback

4.2.2 User Studies

- Frequency: Will occur over the period of a session with two to four participants per group.

Test Groups:

1. Visual Impairment Group
    a. Screen reader user
    b. Low vision user
    c. Testing Focus: Navigation and content comprehension
2. Motor Impairment Group
    a. User with fine motor challenges
    b. User with mobility devices
    c. Testing Focus: Touch interaction and navigation efficiency
3. Cognitive Disability Group
    a. User with learning disability
    b. User with attention difficulties
    c. Testing Focus: Content clarity and task completion

Session Structure:

1. Pre-test Questionnaire (5 minutes)
2. Guided Tasks (10 minutes)
3. Free Exploration (10 minutes)
4. Post-test Interview (5 minutes)

4.3 Analysis and Reporting

Our analysis and reporting approach incorporates a comprehensive system for collecting, analyzing, and documenting test results to drive continuous improvements, encompassing the following elements:

Data and Metrics Collected:

- Task completion and success rates
- Time-on-task measurements
- Error rates and patterns
- Overall satisfaction scores
- Accessibility compliance levels

Analysis Methods:

- Quantitative analysis of task completion data
- Qualitative analysis of user feedback
- Comparative analysis across different user groups

This usability testing plan ensures that our application truly serves its purpose of making STEM learning resources and content more accessible to individuals who previously had a difficult time accessing materials due to their disabilities. Through our multi-faceted approach of expert reviews and user studies, we will gather both qualitative metrics and quantitative insights that will drive continuous improvements to the user experience. The involvement of diverse stakeholders will help ensure we address the full range of user needs and challenges. By maintaining a detailed analysis of results, we are committed to creating not just an accessible application but one that empowers users to engage with STEM learning resources.

5.0 CONCLUSION

The Diverse Makers testing plan establishes a strategic framework to validate our application's core mission: making STEM education accessible to individuals with disabilities. By allocating our testing efforts across unit testing (50%), integration testing (25%), and usability testing (25%), we ensure thorough coverage while prioritizing the components most critical to user success.

Our unit testing approach targets the foundation of user experience: authentication, activity management, search capabilities, and accessibility features. Using Jest and the React Native Testing Library, we'll validate each component against clearly defined success criteria, ensuring robust performance even in edge cases. This methodical approach to testing individual components will catch issues early, reducing development bottlenecks and ensuring reliability where it matters most.

Integration testing bridges the gap between components, focusing on the critical pathways between our React Native front end and Firebase back end. By verifying data flow, state management, and accessibility feature persistence across components, we ensure these essential connections function reliably. This testing is particularly vital for features like our block-based content system, where seamless interaction between components directly impacts user experience.

The usability testing strategy directly engages with our target users, incorporating both expert evaluation and structured testing sessions with individuals having different accessibility needs. This hands-on approach will provide concrete feedback about real-world usage, helping us refine the interface and interaction patterns to better serve our diverse user base.

The strength of this testing plan lies in its layered approach, from foundational components to real-world usage. Each testing phase builds upon the previous one, creating multiple checkpoints to catch potential issues. With clear success metrics at each level and a consistent focus on accessibility, this testing strategy will help us deliver an application that not only functions reliably but truly serves its purpose of expanding STEM education access. Through this systematic approach, we can confidently work toward our goal of creating a platform that empowers individuals with disabilities to engage with STEM learning opportunities. The implementation of this testing plan will result in an application that not only meets technical requirements but is also genuinely accessible and user-friendly, helping bridge the representation gap in STEM fields for individuals with disabilities.