TENSE MED DYNAMICS

Software Test Plan

April 4th, 2025

Project Sponsor: Jesslynn Armstrong
Ryan Lucero, Cathy Ly, Zach Trusso, Marco Castrita

Introduction

Tensegrity Medical's LightDose app is a mobile health monitoring application made for dog owners. The LightDose app provides precise, real-time data tracking and allows pet owners to monitor canine health metrics by utilizing Bluetooth technology to collect biometric data. LightDose allows owners to administer and track their pet's light therapy treatment all in one application. Focusing on ease of use and secure health information management for the owners, the application plays a key role in pet's lives.

Software testing evaluates and verifies LightDose's functionality. The purpose of software testing is to make sure that the application is reliable, identify and prevent any potential failures, validate user experience, and maintain high-quality software standards. Its primary purpose is to help pets in cardiovascular recovery which reinforces the importance of software testing.

TenseMed Dynamic's testing plan includes: Unit Testing, Integration Testing, and Usability Testing where the focus is to test authentication and user data security, bluetooth device connectivity, data accuracy and transmission, and user interface navigation. The listed testing approaches are required for high reliability and precision keeping in mind that the users are pet owners. Having the potential of unreliable software can lead to inaccurate health readings, data privacy breaches and loss of user trust.

Unit Testing

Unit testing is a software testing approach in which individual units or components of a program are tested in isolation to ensure each component functions correctly. The primary goal is to detect and fix defects early in the development cycle.

Login Testing

Using Flutter's test package (flutter_test) and mocking tools such as mockito to simulate database and authentication interactions, this tracks test coverage to ensure that all critical functions are adequately tested.

The most critical components to be tested within the login process are email format validation, checking password requirements, and checking user credentials. For comprehensive testing, test cases are defined in terms of equivalence partitions and boundary values. Good examples of valid cases for email format validation include properly formatted email addresses such as test@example.com and user.name@domain.co.uk. Invalid examples include patterns like test@.com (no domain name), user@domain,com (comma instead of a period), @example.com (no local part), and user@@example.com (two '@' symbols).

The password verification test verifies whether a password meets security standards. Proper cases are passwords like Passw0rd! that include an uppercase character, a lower case character, a digit, and a special character. Poor cases are passwords like password (not containing uppercase characters, digits, and special characters), PASSWORD123 (not containing lower case characters and special characters), and pass12! (too short). User credential verification ensures the system verifies attempts correctly. A good case should have a valid email-password pair correctly returning a user ID. Bad cases are when a bad password or non-existent email should return an error. To facilitate testing, flutter_test is used for unit tests, and mockito to mock Firebase Authentication and Firestore interactions. Analysis of code coverage also makes sure that all the significant pieces are adequately tested.

User Registration

Prioritizing complex, extensive unit testing, this ensures that the sign up process functions as intended. User registration is the entry point for new users, and it directly affects the overall users experience, data quality and account security. Making sure that this feature works under any and all conditions is essential for protecting sensitive information and supporting a very smooth signup and onboarding experience.

This testing approach leverages Flutter's flutter_test package, which is supported by mockito for simulating FireBase Authentication and Firestore interactions. Giving ability to test key functions without relying on real-time backend response, which also improves overall repeatability and control during testing.

Three Main Focused Area:

- Validation of required fields
- Ensuring email verification
- Password strength enforcement

Key Units Tested:

Function	Test Case	Expected Result
validateEmail()	john.doe@example.com	Valid
validateEmail()	john@ or @example	Error message
validatePassword()	Passw)rd!	Pass
validatePassword()	1234, password	Failed due to poor strength
registerUser()	Valid inputs	Creates firebase Auth user and Firestore doc
registerUser()	Duplicate email	Returns Firebase Auth error
Firestore write	Network failure	Triggers retry logic and user feedback

validateEmail() -confirms the structure and formatting of submitted emails.ValidatePassword() - checks whether or not the requirements were met.registerUser() -manages user creation and database write operations. Firestore helper functions -manage profile data persistence and user document creation.

Test Design Strategy:

Utilizing equivalence partitioning and boundary value analysis is to capture both typical and edge cases. Tests had both valid and invalid scenarios to verify correctness and robustness as well as proper error handling. In order to maintain quality, monitoring test coverage to ensure all essential logic branches are checked and verified. This is crucial for error cases like invalid input combinations or authentication failures, which can result in user frustration or data integrity issues. Ultimately, unit testing on the registration component aims to deliver a stable and secure experience from the very beginning of the user's interaction.

Database Interactions

Database operations are at the core of the application's functionality, managing the storage and retrieval of user information, pet profiles, and health data. Proper testing of these interactions is critical to ensure data integrity and application reliability.

For database interactions, isolated unit tests will be implemented using Flutter's flutter_test package combined with mockito to simulate Firestore interactions. This approach allows testing of database methods without requiring actual connections to the Firebase backend during testing. Testing focuses on three key database operations: User Creation and Storage, Data Retrieval Operations, and Error Handling and Edge Cases.

These tests verify proper formatting, validation, and storage of user data in Firestore. Test scenarios include successful user creation with valid data, handling of missing required fields, duplicate user creation attempts, and network failure

conditions. The implementation mocks Firestore collection references, verifies document creation parameters, tests error handling, and confirms proper ID assignment.

This testing ensures the application correctly queries and retrieves user information. Scenarios include retrieving existing users, handling non-existent user queries, field filtering, collection limits, and connection timeout management. Tests verify query construction, data mapping from Firestore documents to model objects, empty result handling, and performance under various network conditions.

These tests examine the database layer's management of unexpected conditions. Testing includes simulation of server-side errors like permission denial, handling of malformed data, maintaining transaction integrity during concurrent operations, and proper schema migration. The focus is on exception handling, error message clarity, recovery mechanisms, and error logging for troubleshooting.

Testing utilizes mock Firestore instances to simulate database behavior without network connections. For example, user creation tests will verify that documents are created with correct fields and IDs in the mock database. This approach enables thorough testing of normal operations and edge cases while avoiding the instability of live services.

Testing tracks code coverage (targeting >90% for critical operations), edge case coverage, and performance under load. These metrics are integrated into the continuous integration process to identify regressions quickly. Regular metric review ensures maintained database interaction quality throughout development.
This comprehensive database testing approach helps ensure the LightDose App maintains data integrity and reliability under both normal and exceptional circumstances, contributing to a trustworthy health monitoring application for pet owners and veterinarians.

Integration Testing

Authentication-Database Integration

Integration testing for the authentication-database connection primarily focuses on verifying that user credentials and profile information flow correctly between Firebase Authentication and Firestore database services. Unlike unit testing, this testing ensures that these two distinct systems work harmoniously together through all user management operations.

The goal is to ensure that authentication events properly trigger corresponding database operations and that the user experience remains seamless across this technical boundary. Testing employs Flutter's integration_test package combined with Firebase local emulators to create controlled, reproducible test environments.

Testing verifies that when a new user successfully registers through Firebase Authentication, a corresponding user profile document is properly created in Firestore. Test scenarios include successful registration with all required fields, which should create both an authentication record and a matching database profile with the same user ID. The test validates that user attributes are consistent between both systems and that role-based permissions are correctly established.

When a user verifies their email through Firebase Authentication's verification process, the database record should be updated to reflect this status change. Tests simulate the verification process and validate that the application correctly identifies verified users and provides appropriate access levels based on verification status.

When users update profile information (name, contact details, pet information), these changes should be correctly stored in Firestore while maintaining consistency with authentication records where applicable. Tests include scenarios where updates occur during poor connectivity to ensure the application handles synchronization delays appropriately.

This testing verifies that authentication state changes (login, logout, password reset) trigger appropriate database access modifications. The test harness simulates

various authentication states and verifies that database access reflects these states correctly, preventing unauthorized data access while maintaining session continuity for authenticated users.

User Flow Integration

Working on implementing and testing the entire user journey- from signing up to logging in and transitioning into the main features of LightDose, this method connects multiple critical systems, including screen navigation, Firebase Authentication, Firestore data management and session persistence. Integration makes sure that each of these components communicates and works smoothly and allows for an error-free experience to the end user.

The approach uses Flutter's integration_test package in order to simulate real-world user flows in a testing environment. Firebase's testing tools allow us to work with temporary user accounts as well as verify that both authentication and database operations work as intended.

Integration Points Covered:

- Registration -> Email Verification -> Login -> Dashboard.
- Firebase Auth <-> Firestore sync
- Screen transitions and state preservation
- Profile completion prompts and redirects
- Session management

Scenarios and Conditions Tested:

1. Successful Registration Flow

   The user registers with valid data, then receives a verification email, clicks the verification email link, logs in, and is redirected to the home page. The app then pulls data from Firestore and makes sure that the profile is indeed associated with the authenticated user.

2. Unverified Email Handling

   A user who tries to login without verifying their email will receive an

appropriate message. After completing verification, they are able to login successfully. We tested email re-check mechanisms that used user.reload() to make sure that real time verification status works accurately.

3. Incomplete Profile Flow

   After login, users who have not completed their profile are then routed to the profile setup screen. Once required information is entered and submitted, the data is then stored and users are directed to the dashboard where all their information is displayed properly.

4. Screen Navigation and Session Persistence

   Navigation is tested amongst all user flow routes - sign up, login, verification check, profile completion, and dashboard access. We also tested to make sure the app restarts to confirm that authenticated sessions stay on and take the user to the right screen.

5. Error and Exception Handling

   Testing:

   - Backend communication failures
   - Firestore sync delays
   - Unresponsive navigation links
   - Attempts to access protected content without authentication

Each test makes sure that the system “hand-offs” between modules work effortlessly. An example being, Firebase Auth generates a UID which has to match the document key in Firestore. These identifiers are used all throughout the app, and any sort of mismatch can result in user data not loading properly, failed navigation, or even accessing the wrong profile.

Bluetooth Device Connection

The LightDose device can log activity to a serial monitor, displaying real-time information. To test the Bluetooth connection, the device will log each time a device connects or disconnects, recording the device name and connection status. This

includes logging when a device manually disconnects or is disconnected for not meeting the necessary requirements; specifically, if the connection is not from the LightDose app. The logging system will also capture errors and trigger automatic disconnections to protect the system.

For data validation, the device will generate and log test data before transmitting it. The LightDose app will receive, graph, and store the data, enabling interaction for detailed insights. Validation involves comparing logged data from the device and app, using numbered data points to ensure accurate matching.

To evaluate error handling, both the device and app will be fed incorrect data. This break testing approach involves intentionally causing software errors, identifying breakpoints, and refining the software until incorrect data no longer disrupts functionality. This iterative process ensures robust error handling.

Usability Testing

User Interface

In order to evaluate the user interface, manual testing will be conducted by providing testers with the Lightdose app without instructions. Observations will be made on how testers navigate the app, focusing on any challenges or confusion encountered. Afterward, testers will provide feedback on what they liked, disliked, found confusing, and any questions left unanswered from exploring the app.

This approach effectively tests navigation clarity. If testers struggle to move from one page to the next, navigation clarity is considered low. Conversely, if testers can easily navigate, the navigation clarity is deemed high. This goes hand in hand with screen flow logic, as the testing method assesses whether testers can navigate and accomplish tasks without guidance.

Button and interaction intuitiveness will also be evaluated by observing whether testers can identify clickable elements and understand their functionality. This includes assessing the visibility, labeling, and placement of buttons.

To ensure comprehensive testing, testers will be asked to perform common tasks, such as accessing settings or completing an action. Notes, video recordings, or checklists will be used to document observations, and feedback will be analyzed to identify areas for improvement. Success criteria will be established, such as 90% of testers navigating without confusion, to measure the effectiveness of the interface.

2. Information Accessibility

The primary users of this project are dog owners managing the pain and vascular health of their pet, healthcare workers monitoring biometric information, and potential stakeholders in terms of accessibility and security. The focus will be the ease of accessing key features, presenting biometric information clearly, and mapping the user journey to identify and address usability limitations. Usability testing will be

conducted through targeted expert testing and small-scale user testing with an emphasis on rapid feedback loops and iterative refinement.

Expert reviews aim to catch significant usability and accessibility issues early. One UX/UI designer will join for layout and accessibility input, one healthcare provider to determine patient interaction usability, and one security reviewer to ensure data handling and authentication ease-of-use. Each of the experts will perform a hands-on review with a checklist followed by discussion to integrate feedback and prioritize fixes. The data will be collected through evaluation notes and identified usability issues. The process of analysis will focus on prioritizing issues that have the potential to be fixed with the project timeframe.

Small-scale user testing will be conducted to confirm usability and find friction points in key functionality. Participants will consist of 2-4 pet owners at various levels of tech literacy. Methodology will include task-based activities such as setting up the device, tracking progress during therapy, and checking biometrics. A think-aloud protocol will capture live user feedback, and a post-session usability questionnaire, derived from a shortened System Usability Scale (SUS), will yield additional information. Observations, error logging, and user ratings will be the foundations of data, with analysis focusing on determining repeat usability issues and prioritizing quick fixes.

Final acceptance testing will ensure that the product meets a minimum level of usability and functionality. Three to four representative users from earlier test phases will be involved in this stage. The approach will include final usability checklist validation, regression checks for previously reported issues, and performance checks of critical metrics such as app loading time and Bluetooth stability. The analysis will lead to final adjustments to improve overall usability before project submission. Observation notes will capture significant issues and findings from test sessions, user feedback forms will provide quick usability ratings, and iterative fixes will be made immediately after each round of testing to allow continuous improvement.

Conclusion

The testing plan for Tensegrity Medical's LightDose App establishes a robust framework to ensure quality, reliability, and security. Through a strategic combination of unit testing, integration testing, and usability testing, the plan addresses critical aspects of the application's functionality.

Unit testing validates core components like authentication, registration, and database operations, while integration testing ensures seamless communication between Firebase Authentication and Firestore, smooth user flows, and reliable Bluetooth connectivity. Usability testing brings valuable perspective from both experts and actual pet owners to verify that the application is intuitive and accessible.

This comprehensive approach directly mitigates the key risks identified: inaccurate health readings, data privacy vulnerabilities, and potential loss of user trust. By implementing this test plan, the LightDose App will provide pet owners and veterinarians with a dependable, secure tool for monitoring canine health metrics, ultimately delivering on its promise of precision health tracking and secure information management.