liftVectr Alpha Project Spec and Test Plan

PURPOSE

This test plan outlines key features present for the liftVectr Alpha build, and includes plans and provisions in place to test for acceptable functionality. The liftVectr project includes a hardware portion, consisting of the IMU and Bluetooth Low Energy (BLE) functionalities within a microcontroller chip, and a software portion which consists of an Android app.

OVERVIEW

The microcontroller used for the project originally was the Arduino Nano 33 BLE, but it is in the process of being replaced with the Seeed Studio XIAO nRF52840 Sense, which has a much smaller chip size than the Arduino. Regardless of hardware chip, the hardware functions involve idling until paired to a BLE receiver, and continually reading the IMU and publishing the measurements (X,Y,Z dimensions of gyroscope in degrees/second and accelerometer in gs/second) while connected. Case housing for the chip and other hardware features such as rechargeable battery will be implemented for the Beta build when the parts arrive and a means for 3D printing is secured.

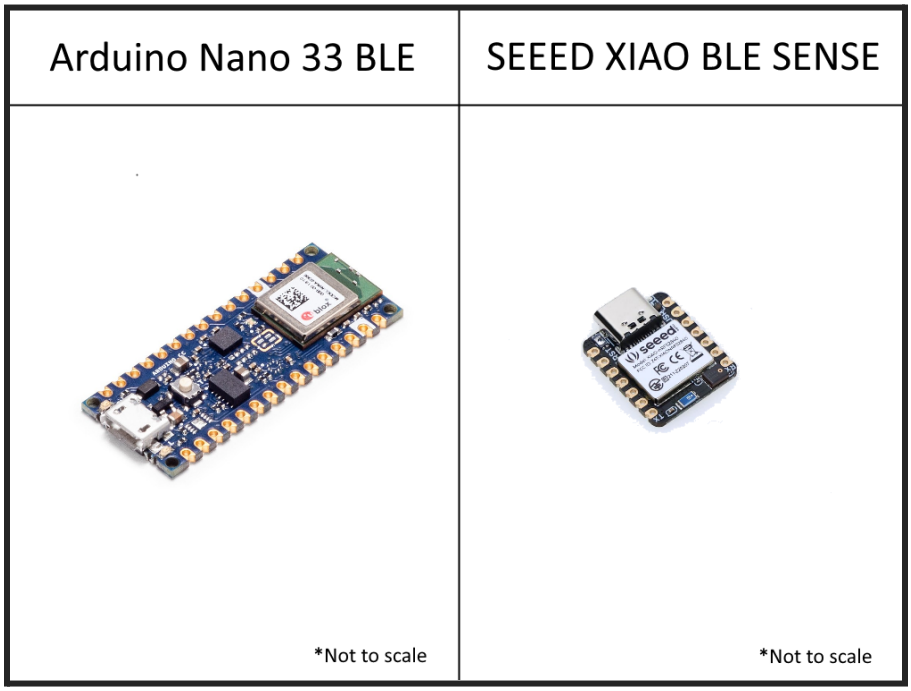

Figure 1.1: Microcontrollers used in development.

The Android app provides a multi-page UI for recording a new exercise session, listing previously recorded exercise sessions, and displaying selected exercise sessions with charted data. The backend functions involve handling communicating with the hardware over BLE, reading and writing exercise data with a local Room database, and performing processing and display of exercise data.

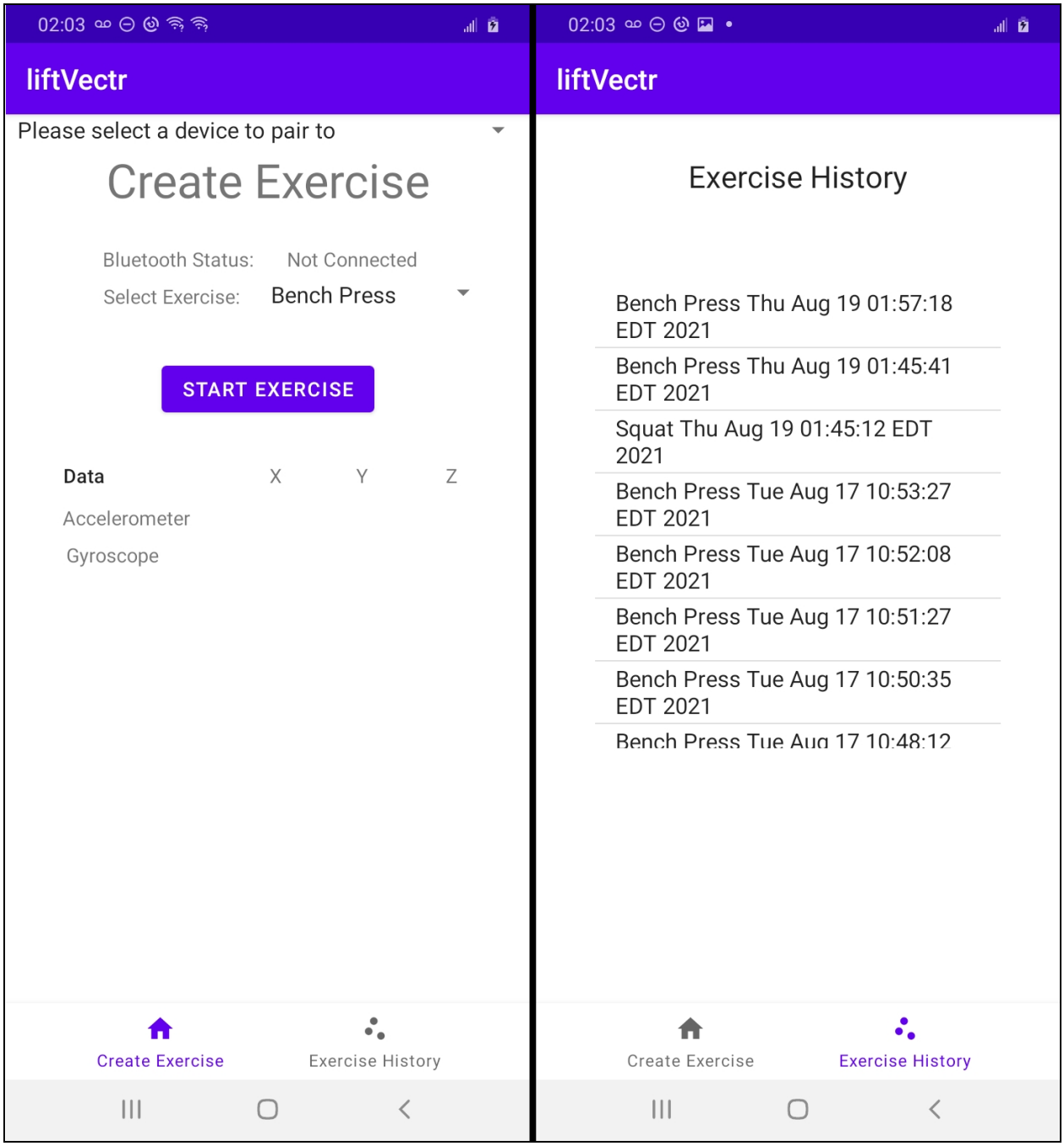

Figure 1.2: Main app screen navigation via NavBar.

KEY FEATURES AND TESTING PLANS

Software: Android BLE Communication

Key Component: BluetoothController Utility Class

- Includes Functions:
  - Initialize Bluetooth helper
  - Scan for open Bluetooth devices
  - Read published Bluetooth data
  - Pair with Bluetooth device by name
  - Disconnect with paired device

Key Component: PermissionsHandler Utility Class (static)

- Includes Function:
  - Check for ungranted permissions
  - Ask user to grant ungranted permissions

Testing Plan:

1. Black-box user test using an Android phone with the Arduino Nano 33 BLE as a testbench for several use cases outlined in the diagram below. No crashes, unreasonable delay times, or unexpected black-box behavior should be exhibited.

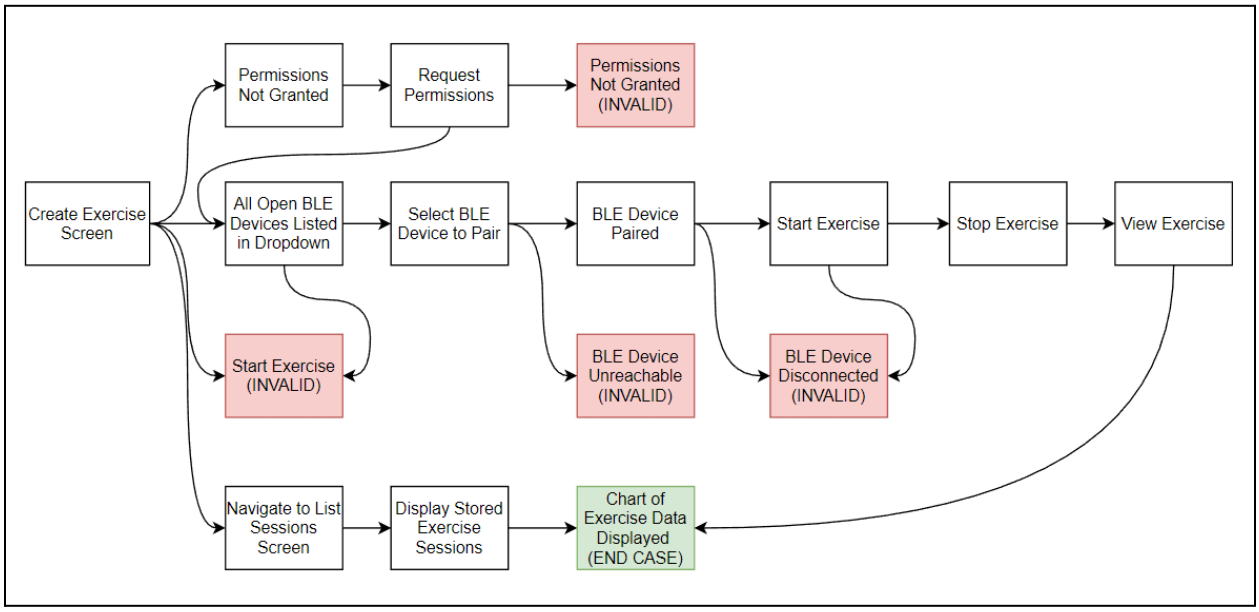

Figure 1.3: Use-case flowchart with valid and invalid BLE interactions that must be handled properly.

2. White-box analysis via extensive logging to identify BluetoothController and PermissionsHandler activity. Each function call logs a corresponding message which should only appear when expected to be called.
3. Automated JUnit test cases to perform basic code coverage checking using Mockito to simulate complex interactions between BLE devices and the user's device (to be implemented).

Hardware: SEEED XIAO BLE SENSE Data Collection/Transmission

Key Task: Modify hardware code for compatibility with Seeed XIAO BLE Sense

- Action Steps
    - Imported board package and LSM6DS3 IMU library for the Seeed XIAO BLE Sense
    - Refactored IMU accelerometer/gyroscope data gathering function names with those corresponding to new IMU library
    - Code for checking if IMU data is available
        - Replaced data available functions with reading status register and checking if the appropriate bit is set
    - Code for setting accelerometer range to +/- 16G
        - Added functionality for reading control register and and writing a new value after initializing the IMU

Testing Plan:

1. Completed testing to check code compatibility with the Seeed XIAO BLE Sense chip by connecting via bluetooth-enabled smartphone through the LightBlue app and reading appropriate IMU data values.
2. Proceed with an integration test alongside the Android Studio application to determine if appropriate IMU data values are reliably received.

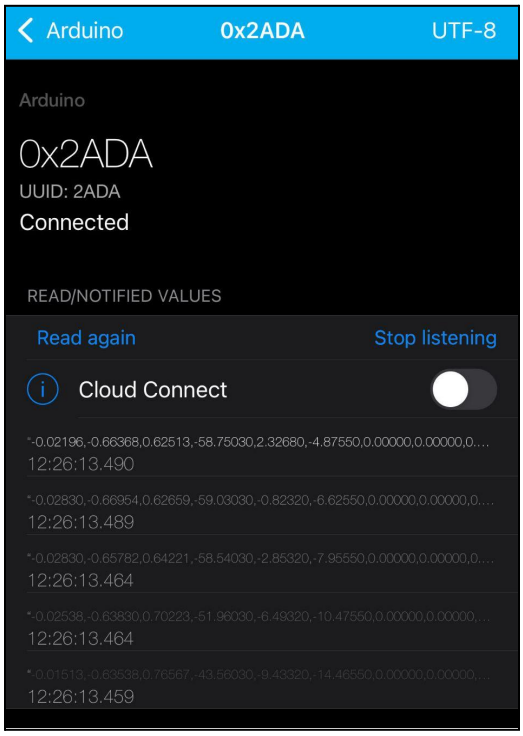

Figure 1.4: Reading IMU Data from Seeed XIAO BLE Sense through LightBlue app

Software: Basic NavBar Implementation

Key Component: Navigation Bar displayed on the bottom of app pages

- Components
  - FrameLayout and BottomNavigationView objects added to .xml files
    - To each app page for a navigation bar to appear, a FrameLayout and BottomNavigationView object has been placed in their respective .xml file
    - Handles the size and appearance of the navigation bar
  - Menu object created with its own .xml file
    - Menu object contains the title and ID's of the various menu options for the navigation bar
    - Drawable clip art .xml files also added to add small clip art images of the different navigation bar options
  - BottomNavigationView object instantiated and ItemSelectedListener set up on MainActivity
    - In MainActivity and any subsequent activities that the navigation bar is to appear, a BottomNavigationView object is initialized at the beginning of the onCreate.
    - This object is set to an initial ID, initially to the activity ID in which this navigation bar appears
    - An ItemSelectedListener then follows, which listens for a change in what menu item the user selects, and navigates to the appropriate app page

Testing Plan:

3. Black-box diagram representing intended user behavior for working with the navigation bar which was used as the criteria and steps to perform testing. No crashing, delays, or other unintended behaviors were observed, the navigation bar allowed the user to smoothly navigate between the app's pages and illuminates which menu item has been selected.

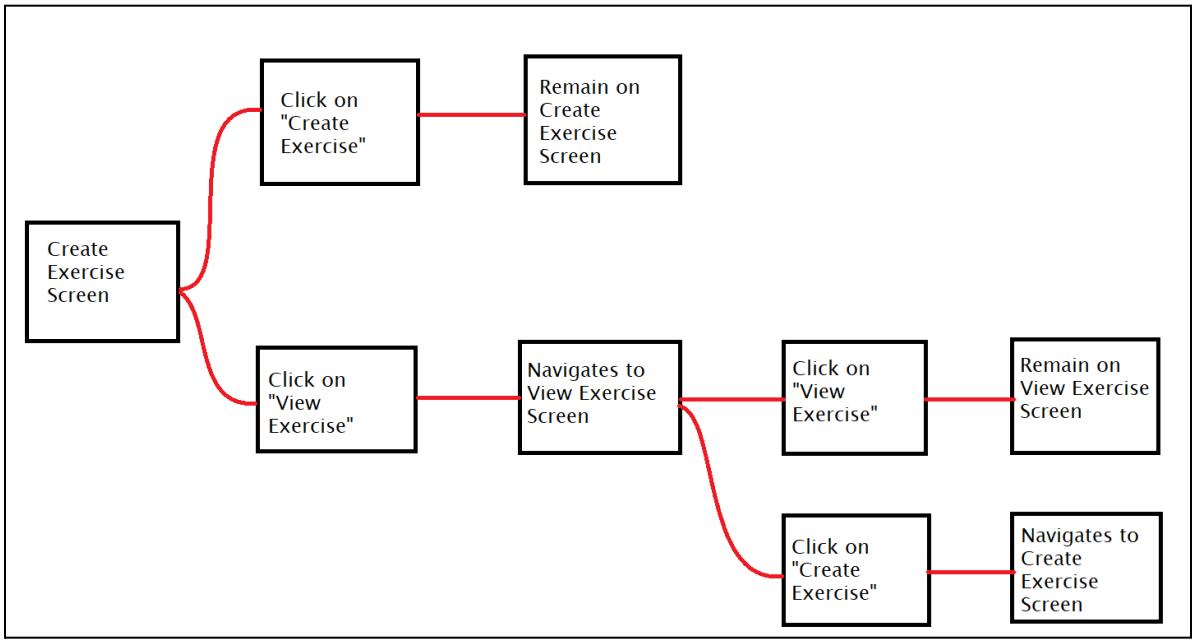

Figure 1.5: Use-case flowchart with intended user behavior for using the navigation bar to navigate through the app

Software: Exercise History Page

Key Component: Exercise History Activity

- Components:
    - Displays a list of previous exercises
        - Pulls previous exercise entries from the database
        - Each exercise entry contains the type of workout (bench, squat, or bench press) and the date of the entry
        - A selected exercise entry transitions to the ChartDisplay page
    - Implemented the navbar
        - Users can transition back to the home page by pressing the corresponding button on the nav bar
    - Added a ListView and BottomNavigationView to .xml files
        - Incorporated the BottomNavigationView for the page and the ListView contains all the exercise entries

Testing Plan:

1. Black-box user test on the exercise entries. A selected entry should transition to the ChartDisplay page and the page should remain the same if no item is selected.

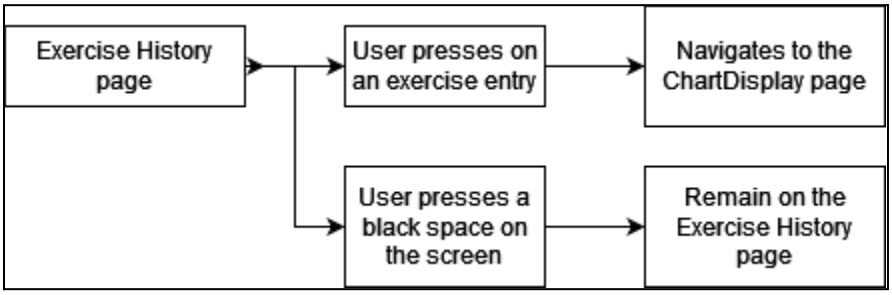

Figure 1.6: Use-case flowchart with intended user behavior for using the exercise history page.

Software: Room Local Database

Key Component: Exercise Database Class

- Includes Functions:
    - Get Singleton Database Instance (main access point to the database)

Key Component: Exercise DAO Class

- Includes Functions:
    - Insert exercise to the database
    - Delete exercise from the database
    - Delete all exercises from the database
    - Get all exercises (in reverse-chronological order) from the database

Key Component: Exercise Repository Class

- Includes Functions:
    - Asynchronous insert exercise
    - Asynchronous delete exercise
    - Asynchronous delete all exercises

        - Asynchronous get all exercises

Key Component: Exercise View Model Class
- Includes Functions:
    - Wraps existing exercise repository functions
- Components:
    - Extends AndroidViewModel class
    - Contains a LiveData list of exercises

Key Component: Data Type Converter Class
- Includes Functions:
    - Convert between Date and long types
    - Convert between List<IMUData> and String types

Testing Plan:
1. Instrumented unit tests for the Exercise DAO class have been implemented. Tests exist for each database CRUD function (testCreateExercise, testDeleteExercise, testDeleteAllExercises, testGetAllExercises) and validate that the functions are performing correctly on a newly generated “in Memory Database”. The ExerciseViewModel, which calls these DAO functions via the ExerciseRepository, has been implemented within our app and no unexpected behavior has been identified.

<u>Software: Continuous Integration Pipeline</u>
Key Component: Github Action YAML File
- Components:
    - Triggers on the creation/update of a PR, or direct pushes to main branch
    - Calls build job, which builds the Android app with Gradle (useful for detecting compilation issues)
    - Then calls test job, which runs local unit tests and instrumented tests, and uploads the test results
    - Then calls report job, which generates a report including both the unit tests and instrumented test results

Testing Plan:
1. The Continuous Integration pipeline is a vital automated testing tool, helping the team to prevent non-compiling or misbehaving code from being merged to and breaking the main branch. The CI itself will not require any testing, and future modifications will only be for optimizing pipeline speed (likely with caching). Given that the CI detects and runs all tests in the Android app, it will continue to improve as our team writes more unit/instrumented tests.

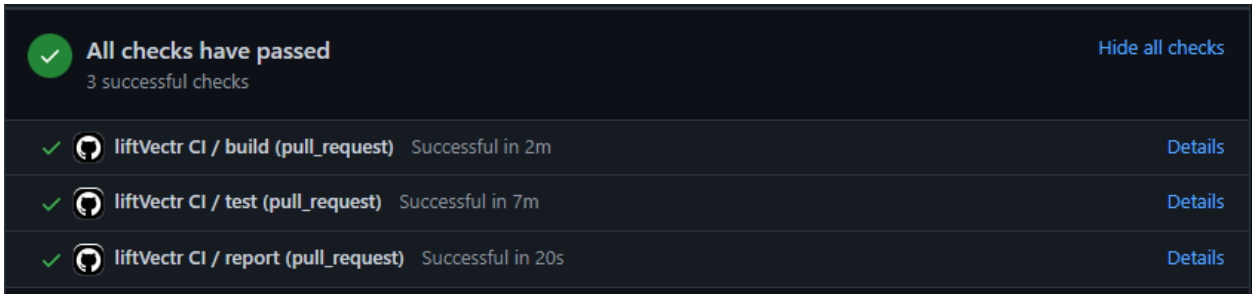

Figure 1.7 CI Jobs running successfully on a Pull Request

Figure 1.8 Build or test failures are easily identifiable with console output

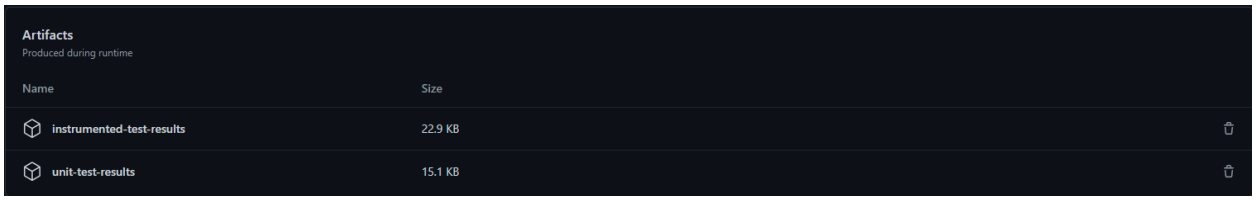

Figure 1.9 Test reports are generated for further debugging