FISH - Fish Identification Search History

Software Testing Plan

Prepared by the members of Team Physh:
Ryan Mason, Scott Austin, Shelby Hagemann, Eduardo Martinez, and Jack Normand

Sponsored by David Rogowski

Mentored by Vahid Nikoonejad Fard

04/07/2023

Table of Contents

Introduction

Throughout the United States, there is an abundance of fish populations living in the many rivers and waterbodies scattered throughout the country. Many of these waterways are actively being blocked by dams affecting these populations in many different ways, and some of these, researchers may not even be aware of yet. Here in Arizona the Arizona Game and Fish Department working with the Grand Canyon Research and Monitoring Center has spent the past couple of decades tagging the many fish populations found in the tailwaters below the Glen Canyon Dam, a 15.5-mile stretch called the Lees Ferry Fishery. By tagging these fish these researchers have given themselves the ability to keep track and view trends of a specific fish's information (Date last caught, Species, Length, and River Mile caught) by catching and updating this information over time. Anglers tasked with collecting and updating this fish data can only go out to the fishery roughly 3-4 times a year, this alone makes it very difficult to collect a sufficient amount of data to conduct a proper study on the native fish populations. On top of this, the process of viewing and updating this data is very inconvenient as when a tagged fish is caught the angler must retrieve the PIT tag ID number using their PIT scanner and when a data connection is made they must contact a scientist with direct access to the database over the phone. From there the scientist will manually locate the caught fish's ID number in the database and then verbally relay that fish's information back to the angler, as for updating the same catch the scientist would need to again manually update each field of data as listed above. Our goal has been to develop a cross-platform mobile application that is accessible by any angler who might be fishing on the Lees Ferry Fishery. This app will give the anglers access to the database information right on their phone. With this data, they will be able to view their caught fish's information by transferring the PIT ID from the scanner to the application. The application will display that fish's data and give the angler the ability to update this information right on their phone. This application will completely remove the need for a data connection to be made and for a scientist to be present for the viewing and updating process.

Our client David Rogowski laid out specific expectations for the application and we plan to conduct a variety of tests to ensure these expectations are met. The first set of tests we plan to conduct will be unit tests. These unit tests will be built to test each major module of our application to ensure each work is developed. The next set of tests that will be conducted are integration tests which focus primarily on the functionality of the interaction of the major modules of our system and again ensure that they are working as designed. The last major set of tests we plan to conduct is the usability tests. These tests will be built to ensure that we have built an intuitive app design that a user will be able to pick up and use immediately.

Unit Testing

Unit testing is a type of process that focuses on testing individual units of code in isolation, typically these units of code are at the function or method level. The goal of unit testing is to ensure that each unit of code, which is the smallest testable component of a software application, is working correctly by producing the expected results

Additionally, the main goals of unit testing are to detect defects early, improve the quality of the code, facilitate code changes, enhance collaboration, and overall support refactoring. In order to deliver a reliable and maintainable application, Team Physh will be utilizing a widely used JavaScript unit testing framework , Jest. Jest provides a well documented api that isolates tests and creates snapshots to keep track of large units with ease. We will be using Jest's easy mocking capabilities to mock the objects inside their units.

Data Entry and Upload

When a user catches a fish and wishes to log its data to the AZGFD database, they will need to use the data entry interface located on the main screen of the FISH app. To use this feature, users will either manually enter their fish's PIT tag or enter it via Bluetooth. Then, they will click the "enter" button, which will navigate them to the data entry page. Here, they will need to enter data on their caught fish into several different entry fields. Then, they will save their entered data from the entry interface. To push their changes to the database, users will need to press the "sync" button, also located on the main page of the FISH app.

To test this unit to ensure that it is working properly, we need to test for potential flaws that could arise as it is being used. One aspect of this feature that we would need to test for is how this unit responds to incorrectly entered data. For example, the first piece of data on a given fish that a user would enter, either manually or via Bluetooth, is its PIT tag value. While this entry field only allows a user to type in numbers, we would need to verify that a user could not paste non-number values into it, and that said action would not result in the user moving on to the data entry screen. Not only is testing for this potential flaw important for the functionality of this application, but it would also be important for preventing potential security risks, such as SQL-Injection attacks.

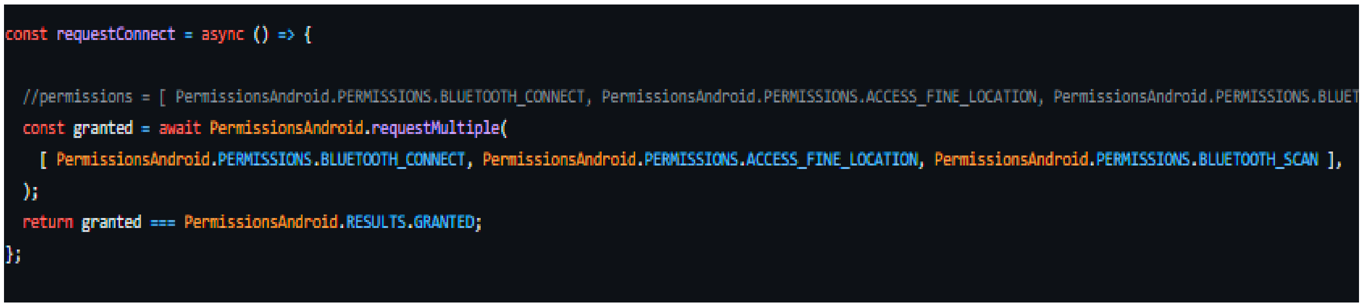
We also would need to account for the possibility of a user inputting an incorrect PIT tag value that is not already located in the database. Since the PIT tag values tend to be fairly long, and manual entry could lead to human error, the app must recognize if a PIT tag value that is not already present in the database is entered. Our client has requested that if this scenario occurs, the user should have the option to create a new entry in the database, which the Quality Assurance team for AZGFD will handle at a later time. During testing, we should verify that the app gives the option for creating a new entry when an unrecognized PIT tag value is entered.

Since this app will be often used in locations that have little-to-no internet connection, we must take a lack of internet access into account for this unit. For example, if a user attempts to click the “sync” button when they do not have access to the internet, the app should not lead the user to believe that their entries are still uploaded to the main database. The app should be able to verify whether or not there is an internet connection and respond to the user accordingly when they attempt to sync with the main database.

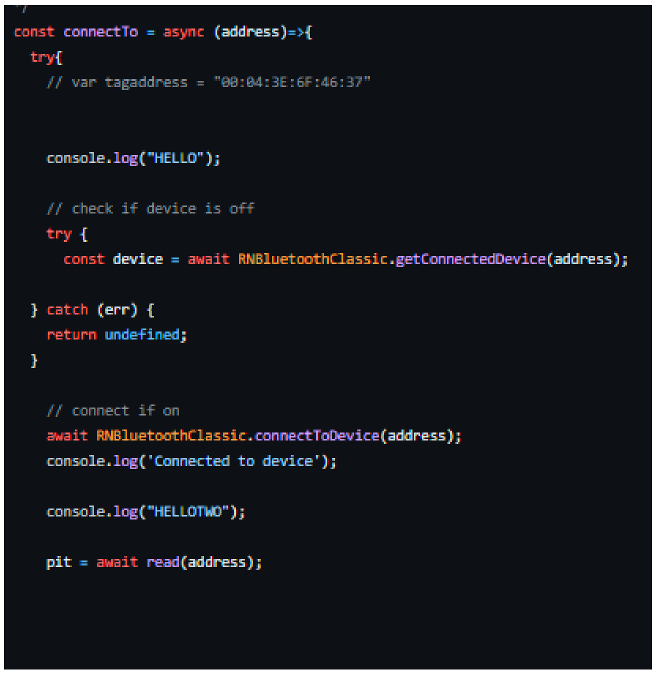
Connecting the phone and scanner via Bluetooth

The Bluetooth component of the application allows the user to have an easy-to-use interface when inputting data into the React native client. In this case, the unit test will include the pair/connection of the Bluetooth device and the user's phone. As well as the data collection from the Bluetooth device and storing the data in the React native client. Overall the unit test will measure the communication between the app and the Bluetooth device. The following will be the main functions and methods we will be testing utilizing Jest to ensure proper Bluetooth functionality:

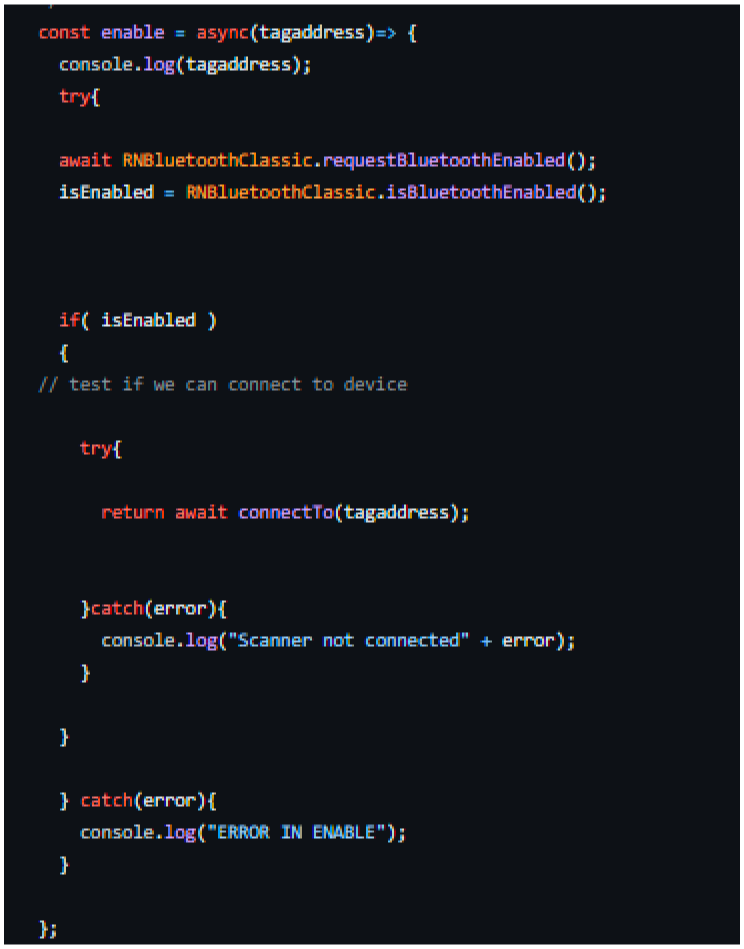
Permissions: In this case, the unit tests are for requesting the necessary Bluetooth permissions for the phone. We will test the number of instances in the app request for the necessary permissions, as well as the proper resolution for an acceptance and denial of the set permissions from the user.


```
const requestConnect = async () => {

  //permissions = [ PermissionsAndroid.PERMISSIONS.BLUETOOTH_CONNECT, PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION, PermissionsAndroid.PERMISSIONS.BLUET
  const granted = await PermissionsAndroid.requestMultiple(
    [ PermissionsAndroid.PERMISSIONS.BLUETOOTH_CONNECT, PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION, PermissionsAndroid.PERMISSIONS.BLUETOOTH_SCAN ],
  );
  return granted === PermissionsAndroid.RESULTS.GRANTED;
};
```

Connection: In this case, the unit test is for establishing a connection to the specified Bluetooth device. We will test that the proper device was retrieved from the specified address, followed by a test of the connection to that address.

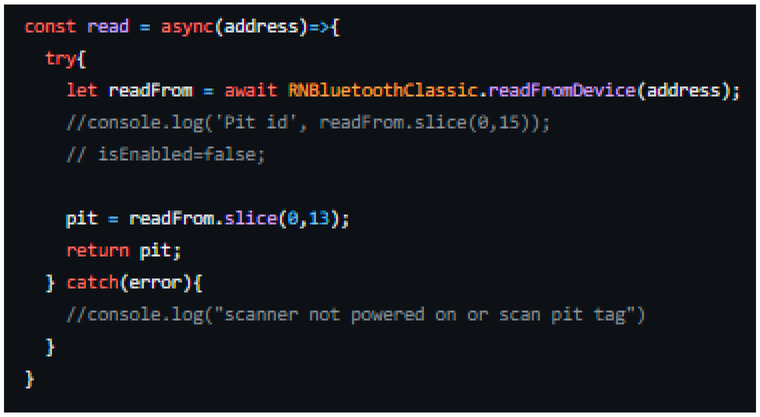

```
const connectTo = async (address)=>{
  try{
    // var tagaddress = "00:04:3E:6F:46:37"


    console.log("HELLO");

    // check if device is off
    try {
      const device = await RNBluetoothClassic.getConnectedDevice(address);

  } catch (err) {
    return undefined;
  }

    // connect if on
    await RNBluetoothClassic.connectToDevice(address);
    console.log('Connected to device');

    console.log("HELLOTWO");

    pit = await read(address);
```

Bluetooth enables: In this case, the unit tests are for detecting if the Bluetooth connection is available for a device. We will conduct a test on the number of instances “Bluetooth enabled” is requested, as well as testing both enabling and disabling Bluetooth functionality.



```
const enable = async(tagaddress)=> {
  console.log(tagaddress);
  try{

  await RNBluetoothClassic.requestBluetoothEnabled();
  isEnabled = RNBluetoothClassic.isBluetoothEnabled();



  if( isEnabled )
  {
// test if we can connect to device

    try{

      return await connectTo(tagaddress);


    }catch(error){
      console.log("Scanner not connected" + error);
    }

  }

  } catch(error){
    console.log("ERROR IN ENABLE");
  }

};
```

Data Collection: In this case, the unit tests are for when requested from the phone to retrieve data from the Bluetooth-connected device. We will test for the information received from the connected device to be in the same syntax as other data in the database.



```
const read = async(address)=>{
  try{
    let readFrom = await RNBluetoothClassic.readFromDevice(address);
    //console.log('Pit id', readFrom.slice(0,15));
    // isEnabled=false;

    pit = readFrom.slice(0,13);
    return pit;
  } catch(error){
    //console.log("scanner not powered on or scan pit tag")
  }
}
```

Offline syncing

The offline syncing component is essential for the user to be able to effectively utilize the application outside of a network. In this unit, we will conduct a test verifying the accuracy of the data retrieval. These tests will include getting the most updated information from the database. Using Jest, we will simulate an offline device by mocking online (connected) and offline (disconnected) scenarios. During these simulations, we will be able to pass queries to the database and verify the information retrieved.

Navigating between screens

Since there are several different screens that users will need to interact with when using the FISH app, it is important that we verify that the navigation between said screens works smoothly and without error. Testing this feature will be relatively simple, as it only entails members of our team running the app on a variety of devices and making notes of any issues or errors that arise when navigating between screens. We would need to ensure that when an action is taken to change screens, the new screen that appears in the app is the one that is anticipated. We would also want to verify that the navigation between screens would be smooth and visually appealing, as slow or lagged navigation animation could make the app appear less professional or appealing to use.

Integration Testing

Integration testing is a critical phase in the software development lifecycle that focuses on ensuring that the interactions and data exchanges between different modules and components of a system take place correctly. Unlike unit testing, which focuses on testing individual units of code, integration testing examines the interfaces between different modules and the overall system behavior. In the context of our mobile application, integration testing will verify that the frontend and backend components are properly integrated and function correctly online and offline. The primary integration points to focus on are the modules responsible for pushing new data entered offline and caching the database when online.

A critical testing point is ensuring that the frontend components correctly interact with the SQLite database. We will write test cases to verify that the front end can read and write data to the SQLite database and display it correctly to the user. Our first set of test cases will focus on verifying that the user can successfully create a new entry in the database and that the frontend can retrieve the data from the database and display it correctly. We will then test that the frontend can delete an entry from the database by simulating a user adding a new entry to the database and then pushing the information to the server. When this happens, the data should be deleted from the SQLite database, and we can visually confirm that the entries were deleted if they are no longer displayed on the frontend.

Once it is ensured that the frontend properly writes to the SQLite database, we will write test cases to verify that the new data entered is properly pushed from the SQLite database to the server-side database. This is important to ensure the data is correctly transmitted to the backend API once the device is online. First, we will simulate an offline scenario by disconnecting the device from the internet, adding new data, and then reconnecting to the internet and pushing the new entries to verify that the correct data is being sent to the server-side database. We will then also test entering new data while online, following this same process. We will then verify that the database on the AWS EC2 instance is correctly updated with the new data that was entered on the mobile application.

Finally, to verify that the caching modules interact properly we will write test cases to verify that the SQLite database instance on the mobile device is correctly updated with any new data from the server-side database that was entered since the user last synced. We will test this scenario by simulating other users adding new data to the server-side database, calling our syncing function on the frontend, and then verifying that the data pushed by the other users are correctly cached on the SQLite database.

In summary, our approach to integration testing will focus on the boundaries between the important modules of our system. By thoroughly testing the modules of database access and data exchange, we will verify that they correctly integrate and ensure that our system functions correctly both online and offline.

Usability Testing

Usability testing is an excellent way to measure how the average user will interact with a piece of software. Generally, this form of testing is done using focus groups, which consist of people that belong to the same demographics as the intended software users. For a project such as this, it is important that usability testing focuses on ensuring that software is intuitive and its instructions are not overly technical. These procedures involve viewing the app from the perspective of a user, evaluating components necessary for the app to function, and what can be understood about its functionality without help or further explanations. For example, a goal of usability testing is to determine how many tasks within an application a user can complete without aid from people running their focus group.

As explained by our mentor, Dr. David Rogowski, many of the people who will be using the FISH app are a part of demographics that generally have less experience working with technology. Therefore, this app must be intuitive to use and not overly complicated. While there will be a help screen and instructions interface in the final version of the app, users should be able to navigate through it on their own. Ideally, there should not be any confusion over what different buttons within the app represent or what users should do first upon opening the app.

As stated before, to ensure that the FISH app is usable and intuitive, it must be used by people in its primary demographic. Usability testing should take place in three separate stages–the first being in a monitored environment where users are given direct instructions on how to use the application. The second stage would also take place in a monitored environment, where testers would not be given instructions and simply be asked to attempt to complete tasks within the app to the best of their abilities. The third stage would involve people using the app in the canyons in which the Arizona Game and Fish Department’s research will take place. In this second stage, users would be able to get a “real-world” experience using the app, rather than in a simulated environment, which would give our team a clearer understanding of how the app must be improved.

In the first stage of testing, focus group participants would be walked through the process of downloading the FISH app onto their cell phones and either given PIT Scanners or asked to bring their own if applicable. During this testing session, we would also provide them with sample PIT tags to test Bluetooth functionality. For this stage of testing, participants would be walked through the process of using the FISH app’s key features. First, they would be asked to start the application, then they would be asked to pair their phone to their PIT Scanner via Bluetooth (if they have an Android device, otherwise they would skip this step), then they would be instructed to enter their scanned PIT tag value and press the enter button to navigate to the data entry page. Next, they would be prompted to enter “dummy data" into the necessary fields and save their log. After this, users would be instructed to press the “sync” button in the app, pushing their newly entered data to the main database on our project’s server. The first stage of usability testing is important to this project in that it would give users the ability to give feedback on the app without concerns regarding confusion about its functionalities.

The second stage of testing, which would entail testers using the app without any guidance, is equally as important as the first stage. This form of usability testing will allow us to measure how intuitive our app is and gain a better understanding of what must be included in the help and instructions interface. Similarly to the first stage of testing, users will use PIT Scanners and the FISH app on their own mobile devices to scan a given PIT tag and enter "dummy data" on a fictitious fish. However, rather than providing testers with step-by-step instructions on how to complete each of these tasks, they will simply be asked to complete them on their own. Once testers have completed the tasks to the best of their abilities, they will be interviewed on their experiences with the app.

After testing participants have gone through the basic procedures of using the FISH app, they would be asked questions about their experience. This would involve asking the testers what they found to be easy to use/understand, what they found difficult to use/understand, and how they believe the app could improve. For example, participants would be asked to rank each of the tasks they were asked to complete from most to least difficult. Then, they would be asked why they found certain tasks difficult and if they had an idea of how they could be made easier. Any tasks that could not be completed by participants without guidance would also be noted and improved upon in later development to increase the app's intuitiveness.

The third stage of testing focuses on how usable the app is in the environment in which it is intended to be used. This stage of testing would take place after our Minimum Viable Product is complete and the app is deployed. People already researching fish populations in Northern Arizona will be asked to download, use, and submit feedback on the FISH app. The primary target of this testing stage is to determine how the app functions in situations where the internet and cell service are not accessible. Testers will inform us about how effective the offline data storage and syncing features are and how intuitive they are to use. In addition, just as we would for the first two stages of testing, we would ask testers for general feedback on the overall usability of the app and the app's appearance.

Another important aspect of our project that we will need to consider during usability testing is the administrative portal. While this is a stretch goal for this project, this module being tested is just as important as the other modules discussed in this paper. The administrative portal, which will allow Arizona Game and Fish Department researchers to view fish data and entries, should be tested by field researchers themselves. AZGFD researchers should test this feature by working through the following tasks and giving their feedback: Searching for a specific PIT tag, sorting the fish data entries by date of entry, and selecting and modifying data entries.

Conclusion

The Arizona Game and Fish Department will be using the FISH application when researching fish populations across Northern Arizona, particularly in the Glen Canyon Dam. The goal of this project is to make AZGFD's research easier to collect and more accurate, eliminating the previous method of data collection, which was highly time-consuming and had vast potential for human error. Testing this application is highly important. Not only to ensure that the app functions as intended but also to make sure that it is intuitive and easy to follow for our key demographic, which typically consists of anglers of older age.