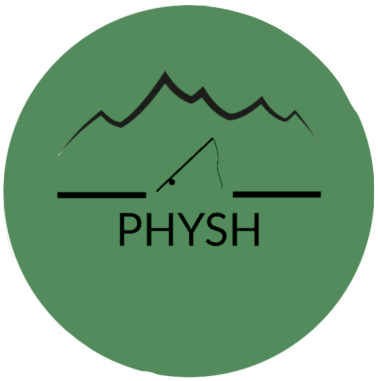
FISH - Fish Identification Search History



Software Design Document

Version 1.0

Prepared by the members of Team Physh:

Ryan Mason, Scott Austin, Shelby Hagemann, Eduardo Martinez, and Jack Normand

Sponsored by David Rogowski, Ph.D.

Mentored by Vahid Nikoonejad Fard

Table of Contents

Introduction

Data collection is a critical part of the operations of many organizations, such as the Arizona Game and Fish Department. Researchers at the Arizona Game and Fish Department, such as our Sponsor Dr. David Rogowski, have been working alongside the Grand Canyon Research and Monitoring Center. Together, these organizations have been collecting data on the rainbow trout, brown trout, and other minor species of fish populations found in the waters below the Glenn Canyon Dam. This is a fifteen-mile stretch of tailwaters called the Lees Ferry Fishery. These organizations want to know everything to do with these fish populations. For this reason, in 2011 they began tagging these fish populations using PIT tags, which are almost identical to the tags typically found in household pets. Over one hundred thousand fish have been tagged over the past twelve years but the issue that these researchers are running into is that the scientists tasked with going out and regularly catching these fish to update the data can only visit the fishery two to three times a year. Considering the amount of tagged fish that are likely still alive, this is not a sufficient amount of data collection to see any major trends within the data as well as the effects the Glenn Canyon Dam may be facing.

On top of this issue, the actual process the angler must go through to get the new information uploaded to the database is extremely time-consuming. When an angler catches a fish they scan it using a PIT scanner. Then, they manually record all of the necessary information on paper, because a cellular or internet connection at the fishery is not likely. Once they can make a phone call, they will contact the data collection scientist at the AZGFD and verbally provide each piece of information for the respective fish. The scientist will then manually type all this information into the database—this process is repeated for every fish. This process is reversed for viewing the caught fish's old information: the scientist would need to verbally relay this information to the angler in the fishery. There are some obvious flaws to this system which is why the development of this application will not only completely overhaul the volume of data coming in but also will make it significantly easier for a scientist from AZGFD or the everyday angler to view and collect data.

The mobile app we have envisioned has three major components: the app itself, which allows the angler to view and record data, a global database that holds all current fish data, and a

locally stored database that will have a copy of the global database as well as a cache for all updates made by an angler. The process of using the app starts with catching and scanning the fish to retrieve its respective PIT tag. Once retrieved, a Bluetooth connection will allow this number to transfer from the PIT scanner to the application. Once the ID is transferred it will locate that fish's data and display it directly on the angler's phone. An option will then be given to update this fish information, opening up fields of entry for each of the fish attributes. Once the update is complete it will be sent to the local database on the user's phone. A background system will be put in place for the app to automatically listen for a cellular/internet connection. Once connected the app will send the locally cached updates to the global database. For future plans, we hope to be able to introduce a profile system with a form of gamification to help keep the anglers engaged throughout this data collection process. We hope that this system will not only make the current process significantly easier but also encourage more anglers to be scanning and increasing the current data pool.

Implementation Overview

Our mobile application focuses on offline data collection, which requires several tools and functionalities. In this section, we will discuss the necessary development and functional components for achieving this project's minimum viable product.

Hardware Components

PIT Tag: An implantable Passive Integrated Transponder (PIT) used for the unique identification of individual fish in our project. Each tag contains a unique hexadecimal code that will allow us to search for specific fish in our database.

PIT Scanner: A device used to read and identify the previously mentioned PIT tags. The scanner uses Radio Frequency Identification (RFID) to read the unique code contained on the tag. Our specific scanner can read 125kHz, 128kHz, and 134kHz microchips.

Mobile Device: A portable electronic device that can access the internet and run mobile applications. We will make use of the device's Bluetooth capabilities to connect it to the previously mentioned PIT scanner.

Software Components

React Native: A JavaScript framework for building mobile applications that can run on both IOS and Android platforms. It was developed by Facebook and is based on their React.js library for building user interfaces. Using React Native will allow our team to build our mobile application using the same codebase for both IOS and Android.

Node.js: A JavaScript runtime environment that allows developers to run JavaScript on the server. For JavaScript to be usable on both the front and backends of this app, Node.js will be used as our back-end runtime and will allow us to write server-side logic in JavaScript.

Express: This is a web application framework for Node.js that runs on top of the runtime environment and allows developers to leverage its benefits, making it simple to create RESTful APIs. We will be using Express to create a RESTful API to allow our mobile application to communicate with our database.

React Native Bluetooth Classic: This is a package that allows developers to integrate Bluetooth Classic functionality into React Native applications, allowing access to the Bluetooth capabilities of a user's mobile device for connecting to the PIT Scanner. It also provides APIs to perform common Bluetooth functions.

Architectural Overview

Due to our mobile application's need to continuously communicate with users and save user inputs, our architecture extends beyond the mobile phone to outside resources. The features of this architecture rely on communication with external resources and maintaining a stable connection. It consists of a user interface software for input, an online database to save data, and

an application programming interface to handle communication between the user interface and the database.

Key Responsibilities and Features

The key features of our system can be outlined by the minimum requirements needed to deliver a minimum viable product to our sponsor.

Minimum Requirements

The minimum requirements include (1) the ability to work on IOS and Android devices; (2) the ability to display, update, and push user data to the database; (3) the ability to synchronize data while offline; (4) the ability to retrieve data from a pit scanner via Bluetooth.

IOS and Android Devices: This is an important requirement for our product, since it is unknown which devices the users will be using, and developing a cross-platform framework increases the amount of data received by users.

To achieve this we will be utilizing a react native client for mobile devices. Through the client, it will carry the responsibility of providing a user interface and view to handle user interaction.

Display, Update, and Push user data: Another important requirement for our application is its ability to display, update, and create new data for our client to view.

Here an application programming interface (API) will be responsible for establishing a connection between the react client and our database model. This component will be handling all the push, update, and get requests taken from the react client and respond with corresponding data.

Offline Synchronization: Since our application will be used in areas with an unstable internet connection, allowing offline synchronization with the user's data is an important feature to maintain a connection to the main database.

This component will utilize a local SQLite database, a compressed version of the main database, to render into the react client for the user interface.

Bluetooth Connectivity: The unique PIT identification for each fish will be produced through a universal PIT tag scanner. Allowing our application to communicate with the scanner via Bluetooth is an important feature to obtain accurate identification and provide a user-friendly interface.

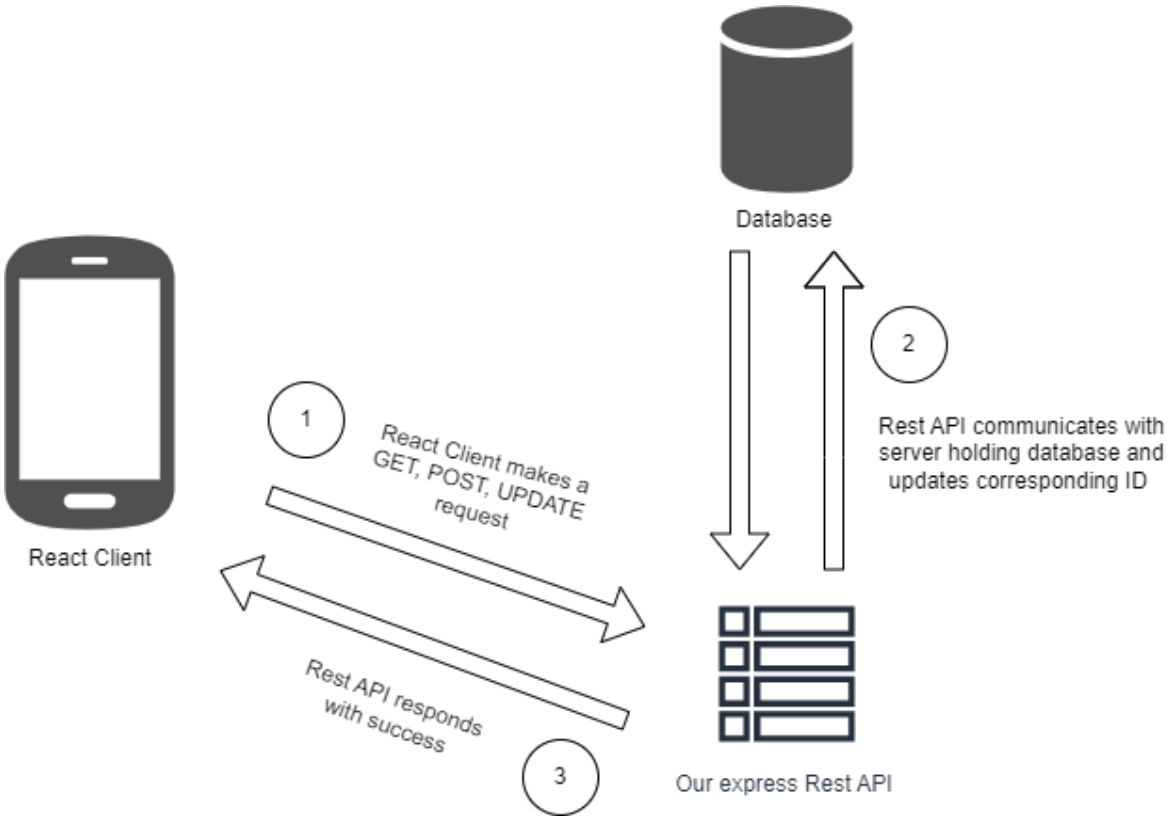
To achieve this we will utilize a supporting library in the React Client which contains modules to allow communication between devices.

Main Communication Mechanisms

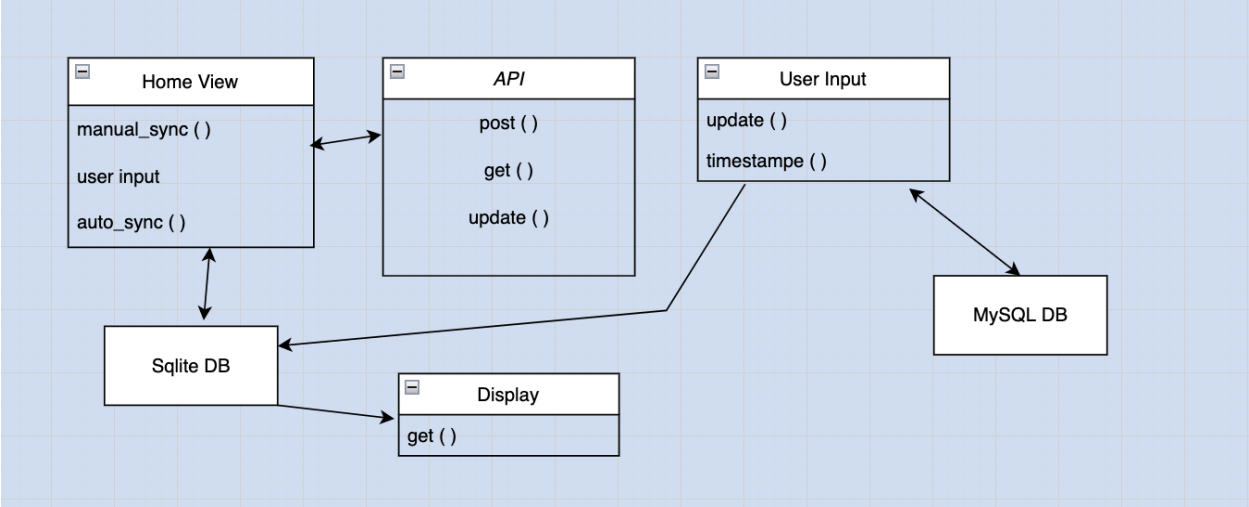
The main communication mechanism in the application involves a react client that will be hosted on the users' devices—followed by an Express Rest API located on our server, and a MySql database. The API will act as a messenger between the front and backend services.

Communication and Control Flows

The flow of communication will begin with the input of user data. This data will be passed either via Bluetooth from the PIT scanner or manually imputed by the user. As shown in the figure below, the React Client, located on the user's device, will pass the data through GET, POST, and UPDATE requests (1). From there Express Rest API will handle these requests and grab the information needed from the MySQL database (2). The API will then respond with the requested information sent from the React Client, which will then display the information on its proper screen. The following graphic displays how these components will work together:

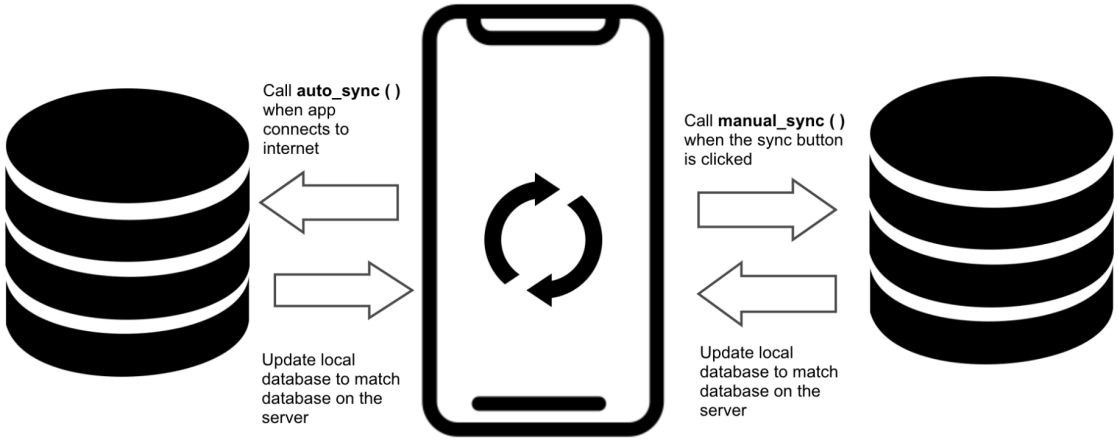


Module and Interface Descriptions



The FISH application design involves extensive backend and frontend development, which is highlighted in this section. Our backend implementation is particularly expansive, as the ability of our application to connect to our servers and merge local and main databases is of utmost importance. The backend modules for this project include the Home View Module, the API Module, the SQLite Database Module, and the MySQL Database Module. Our frontend modules include the User Input Module and the Display Module. In the following sections, we will delve into the requirements and functionalities for each of these modules.

Home View



The Home View module will handle the initial synchronization of the application to the API. This synchronization will happen automatically when the application starts. It will send a request to the API to download a recent copy of the main database.

auto_sync ( )

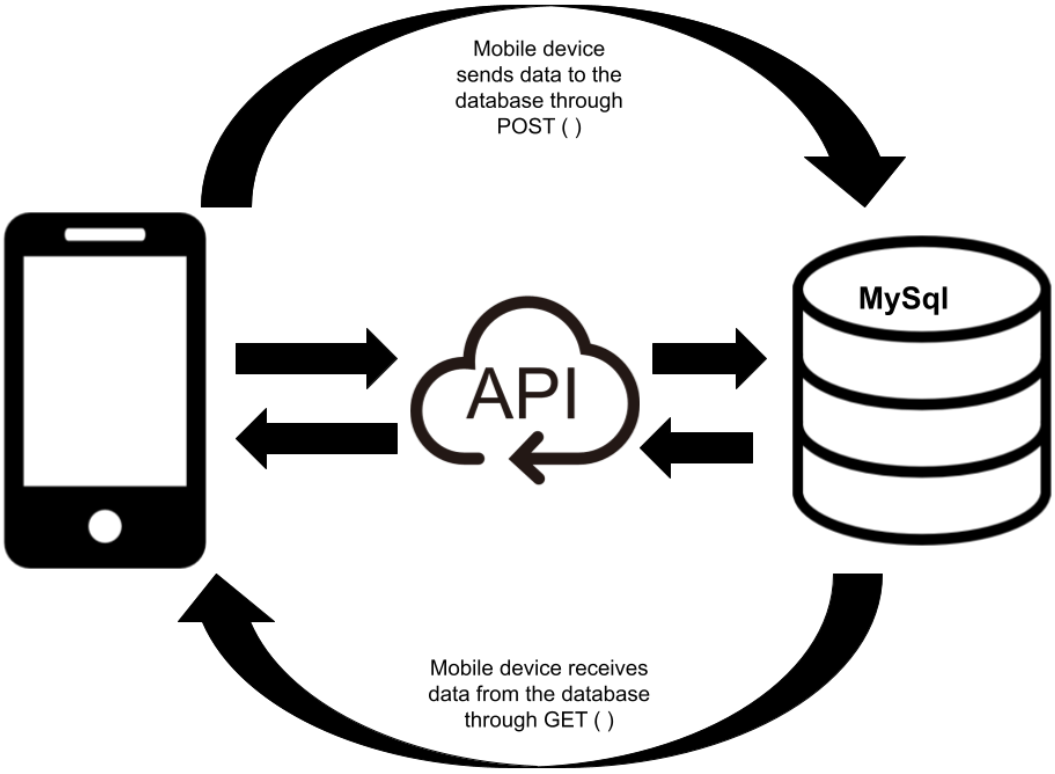
The auto_sync () function will happen on start and takes in no parameters. The purpose of this function is to run in the background when the user starts the application and grab the most recent data needed. The data will be stored for later use when the application goes offline.

manual_sync ( )

The manual_sync () function will be displayed as a button, where the user can send a request to the API and synchronize the most recent data when they have an internet connection.

The following graphic displays how auto_sync ( ) and manual_sync ( ) will work in our final product.

RESTful API



GET()

The GET command from our REST API will be utilized in our front end to fetch server-side fish data from our database for display in our mobile application. The front end will make a GET request to the API endpoint that corresponds to the data we want to retrieve, and the API will then respond with the requested data in a format such as JSON. The retrieved data will then be cached locally on a SQLite database, allowing the mobile application to display the information, even when the device is offline. This provides a seamless user experience and allows for efficient use of resources, as the data is only fetched from the server when it is needed and not continuously in the background.

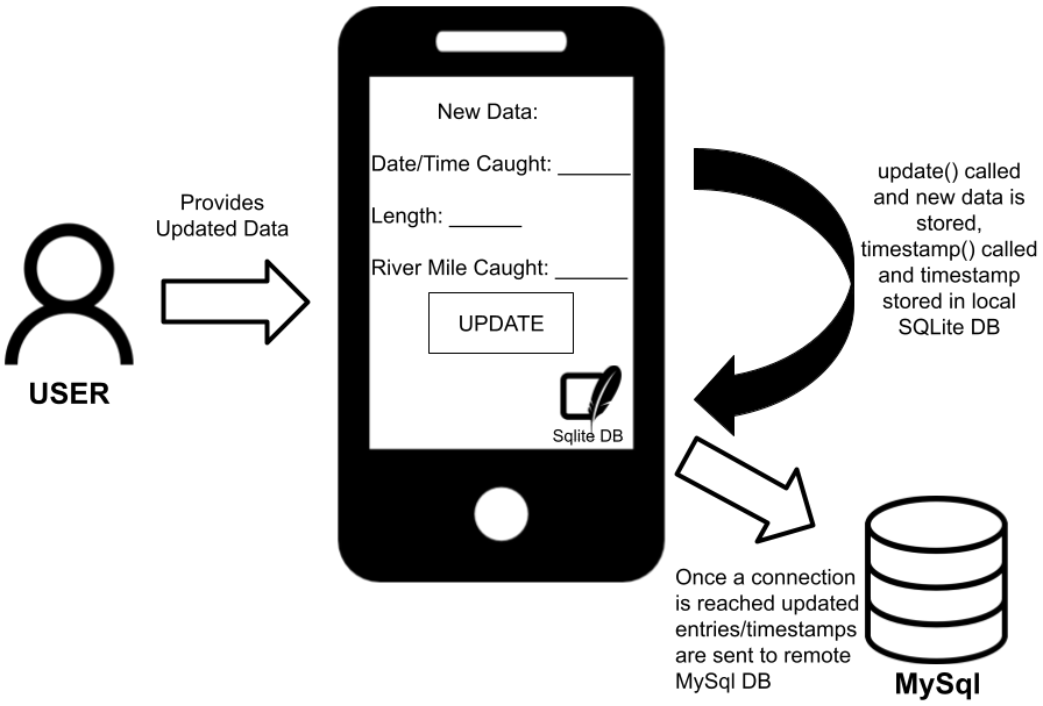
POST()

The POST command from our REST API will play a crucial role in our front end as it allows our mobile application to send data to the server-side database. In our project, the POST request will be used to create new records in the database. The front end will make a POST request to the appropriate API endpoint, including the relevant data in the request body, such as the fish name

and type. The API will then receive the request, process the data, and make necessary updates to the database. This will allow us to send the cached offline catch data to the database once back online, ensuring that no catch information is lost when the device is not connected to the internet.

User Input

The User Input module will focus on retrieving data from the user and ensuring that it is accurate for organizational and analytical purposes. The following functions will take place after the user has inputted their data into the application.



update ( )

The update function will push the newly entered information to the SQLite database, which will be merged with the main MySQL database at a later time when the network connection is stronger. For this function to run to completion, the user must input all of the necessary data on their caught fish, including:

- The fish's PIT tag value
- The date and time of the catch

- The fish species
- The total length of the fish
- The river mile at which the fish was caught

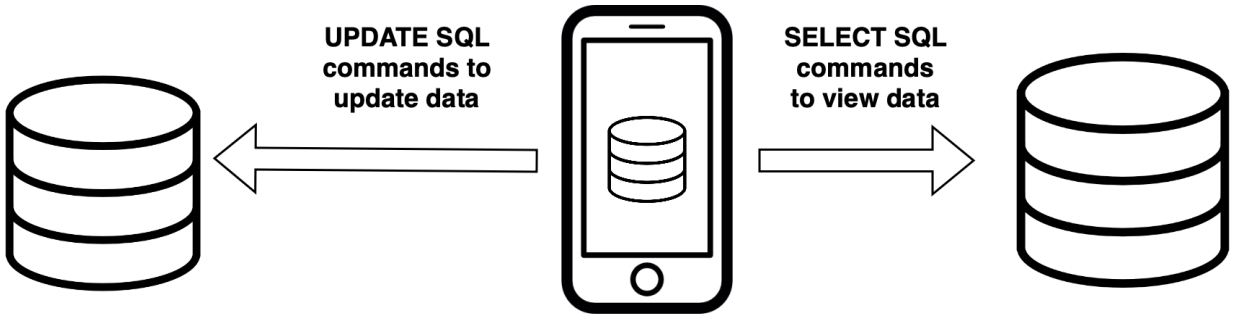
Once the update ( ) function has verified that all necessary fields have been completed, it will store the information in the local database to be merged with the main database at a later time.

timestamp ( )

The timestamp function will log the exact time that a user makes an entry. This function will be automatic; however, users will have the option to edit their time if they need to retroactively upload data at a later time from when they caught their fish. Having an accurate log of what time a fish was caught and recorded is crucial for this project, as it will allow for the fish records to be as precise as possible.

SQLite Database

The SQLite database module will focus on the local storage of fish data. The primary purpose of this database is to allow users to access fish data while offline and have a stored copy of the data on their mobile devices.



The database that is locally stored on the device will be able to be accessed by the application using an SQLite3 library. This library uses a transaction function to access the data.

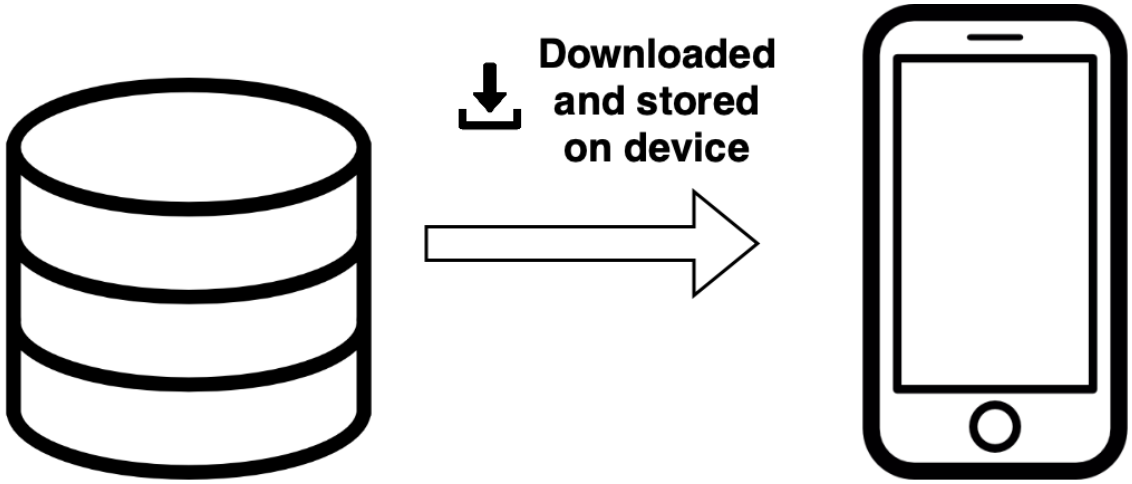
Transaction ()

This function takes no parameters but is run on a database object. It allows the developer to execute SQL commands on the locally referenced database object. These commands can be anything from SELECT commands to REMOVE commands to CREATE commands. Only

UPDATE and SELECT commands will be run for our program, however. Nothing is returned from this function.

MySQL Database

The MySQL database is the main server-side database hosted on a server. This database will be what is downloaded onto the user's device and updated once the user reaches an internet connection. The main focus of this database is to have an up-to-date source of fish data that everyone has access to, and is constantly updated by users.



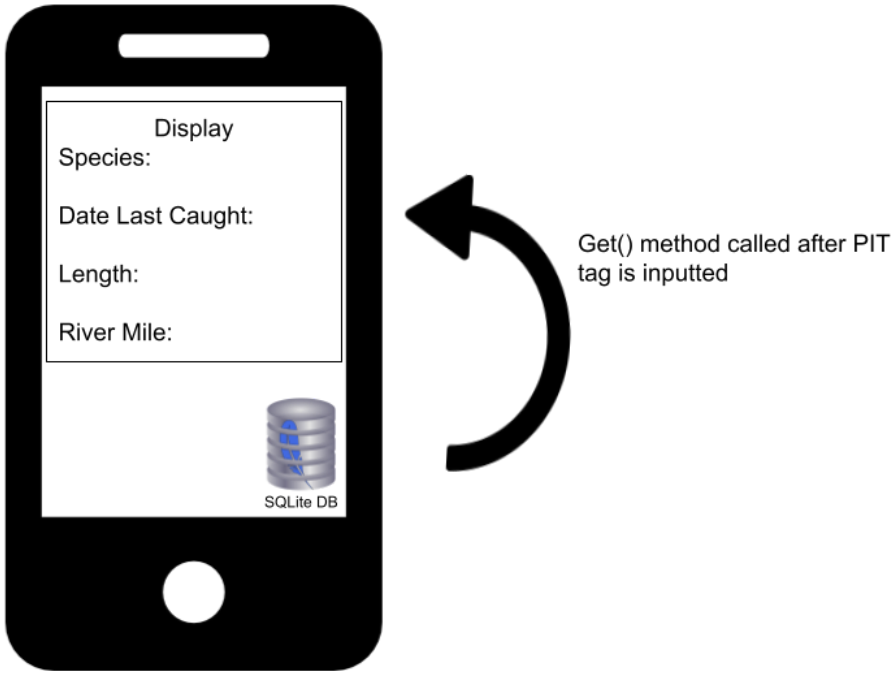
There are no public methods for the database itself, and the data is simply accessed through the API discussed earlier in this section. The data is simply sent to the API through whatever SQL commands are run on it. Whether that be INSERT commands, CREATE commands, SELECT commands, or REMOVE commands. The user will not be able to interact directly with the server-side database and instead must go through whatever commands are allowed in the API.

Display

The display interface module will be working directly for the user by retrieving the entered PIT tag ID's respective fish data. This page will be formatted accordingly to allow for easy interpretation of the fish's data.

get ( )

This method will be used to retrieve a fish's respective data values (date last caught, species, length, river mile) from the locally stored SQLite database.



Implementation Plan

Throughout the Fall semester and before the start of the current semester, our team completed a large amount of planning for this project, began setting up our server and created a basic React Native application to work off of as we develop.

Setting up the Server and Database

We will be using an Ubuntu server to store our project's database. Data uploaded into the app by users will be stored in a temporary, local SQLite database. When the user's device is connected to the internet, their uploads will be pushed to our larger database. Our primary collection of data will be stored in a MySQL database, which will be held on the team Ubuntu server. Our client will have access to the main database and will be able to transfer any necessary data to the government's collective database.

Software Design

We currently have a basic skeleton of our application created with React Native, which is ready to be further developed. As displayed and explained above in this document, we have created several diagrams illustrating our planned architecture and functionality for the application. We also have created a PowerPoint document to be shared among the team members, which contains design/layout ideas and inspiration for how the app should look once finalized.

Developing the Application

We will be using AGILE methods to develop our application. This practice will allow us to focus on the project in smaller, more manageable increments. Our team will meet at least twice per week to discuss our progress on our assigned tasks and delegate new pieces of the project for development. Development for the app will be done in React Native and development for the servers will be done with SQLite and MySQL.

Testing and Refining

To ensure that our project meets its minimum requirements for completion and has the best functionality possible, we will regularly test our product. To avoid software bugs and vulnerabilities, we will be testing the software early and often. As new features are being developed, they will be tested for functionality and completion. Already existing features in the application will also be tested regularly to ensure that they are compatible with newer features.

Once the minimum requirements for this project have been completed, we will assess our project and the remaining deadlines for this course to determine how the application can be further improved and refined. When we reach this point in our development, we will focus on implementing our stretch goals, still adhering to AGILE development methods.

Presenting the Project

Upon completion of this project, we will present our application at the Northern Arizona University Undergraduate Research Symposium. The final software for this project will then be given to our client for the use of research for the Arizona Game and Fish Department.

Conclusion

The collection of information in our data-driven society is an extremely important task, and collecting must be handled at a large scale efficiently. Designing an application that can make this process efficient and straightforward is important, and can make a lot of people's lives easier. All these different technical modules and implementation details outline exactly what our application will look like. The architecture of our program is the most important part on the technical side, and giving an overview of this will give an idea of what it is we are developing. From the way the user interacts with the application, to all the backend calculations and data storage, our application will integrate all of our technologies cohesively and in a streamlined fashion. When everything in the architecture comes together, users will see that collecting data on an organism as small as a fish can have a huge impact on people's lives and tell us a lot about wildlife and the ecosystem they inhabit.