Aquarium Management

Alexandre Correia (53298) & Hugo Rodrigues (53309)

June 2022

1 Introduction & Goals

Taking care of an aquarium is a hard task when it comes to responsibility and amount of work. In order to keep fish alive, it is necessary to manage the aquarium's environment properly, i.e. according to the fish's living conditions, specially regarding the characteristics of the water - type (salted or fresh), temperature and pH - as well as the light conditions - darker or lighter.

People need to measure manually those parameters very often to make sure that the fish's living conditions are being satisfied. So, it is useful to have a mechanism that gives some sort of alert when those conditions are not being respected, e.g. if the water's pH is not within the appropriate range for the fish, then the user could be notified with that information, instead of manually measuring it consistently.

In the context of the course of Mobile and Pervasive Computing Systems, we aim to develop a system that not only shares the real-time conditions of aquariums with people, but also allows them to set, in real-time, some of the aquariums' characteristics either locally (same network) or remotely (different networks).

2 Functionalities

In order to achieve the required fish living conditions on the aquariums, the system developed enables the users to both check and set various characteristics of multiple aquariums through an Android application:

- Temperature and PH:
  - Check the current value;
  - Set the allowed ranges of values;
  - Check if the value is, or not, withing the range of values allowed.
- Brightness:
  - Check the current level;
  - Set the mode: either manual or automatic;
  - When on manual mode, increment or decrement the level.

As far as the aquariums are concerned, it is possible to physically simulate the changing of these characteristics' values. They notify through their hardware when the temperature and/or the pH are not within the range of values allowed. Also, when working on automatic mode, it is capable of adjusting the brightness level inside the aquarium, depending on the ambient light from outside.

3 Architecture

For reaching these goals and functionalities, the system developed in this project consists of the design and implementation of three components, as shown in Figure 1: a physical simulator, a mobile application and a database.

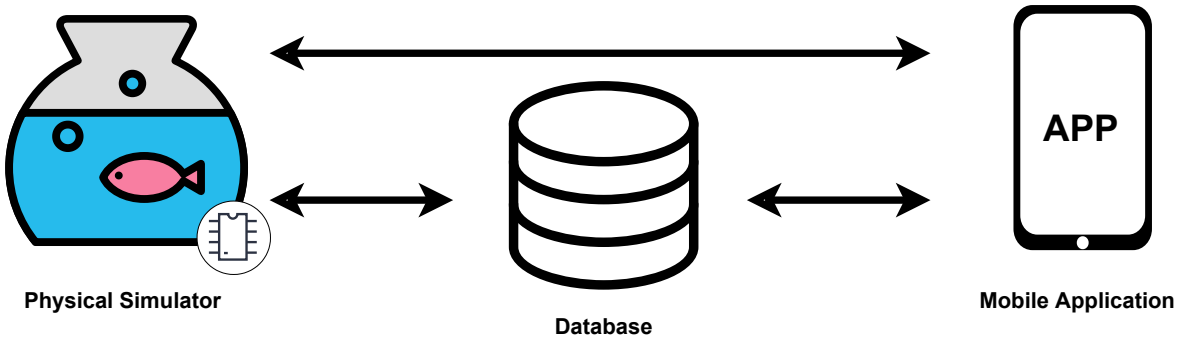

Figure 1: Architecture

The physical simulator senses and controls the various hardware components, which are accessible and controllable via a single-board microcontroller.

The mobile application is the main way of interaction with the user, allowing him to access and modify some characteristics, and define some alerts for each of the aquariums.

For persistently storing every measured characteristic from each aquarium, there is a database which stores that information. Both the physical simulator and the mobile application can communicate directly with each other and with the database. The physical simulator needs to inform, periodically, the database with the measured values so that the real-time state of the aquarium can be stored, as well as to obtain the remote updates from the database. Also, the mobile application needs to get the real-time conditions of the aquarium, periodically, from the database to display them. However, if both of them happen to be in the same network, then they can communicate directly with each other, but the final state of the physical simulator must be stored in the database.

4 Physical Simulator

4.1 Overview

The physical simulator consists of assembling some hardware components, as displayed in Figure 2:

- Microcontroller: 1 NodeMCU ESP32
- Sensors: *2 Potentiometers* and *1 Photoresistor*
  - 1 Potentiometer (7): for simulating the pH level
  - 1 Potentiometer (8): for simulating the temperature level
  - 1 Photoresistor (9): for measuring the ambient light
- Actuators: *4 White LEDs*, *1 Yellow LED* and *1 Red LED*
  - 4 White LEDs (1-4): for simulating the internal light with multiple intensities (fixed)
  - 1 Yellow LED (6): for warning the user that the pH level is not within the correct range (intermittent)
  - 1 Red LED (5): for warning the user that the temperature level is not within the correct range (intermittent)

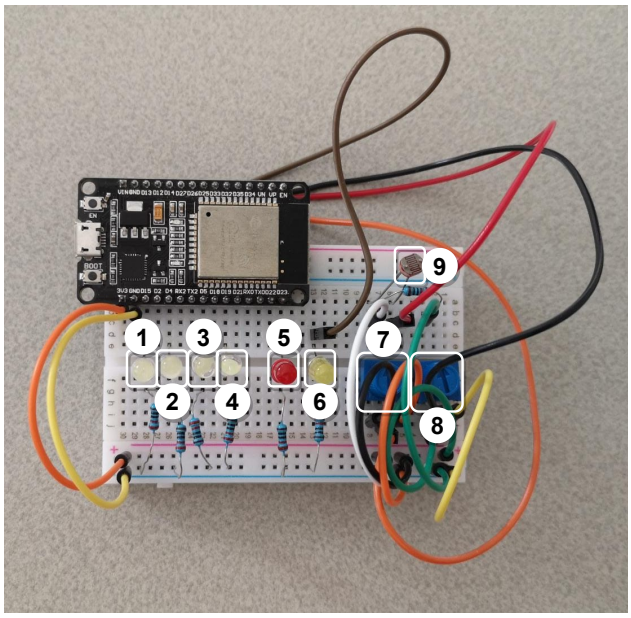

Figure 2: Physical Simulator

4.2 Implementation

As far as the software development of this component is concerned, we developed it using the PlatformIO extension [4] of Visual Studio Code [5], which is a platform that manages every detail of this component (boards, libraries, build, upload, monitor, among others).

4.2.1 Setup

When the aquarium is powered on, it connects itself to the internet (the WiFi details are on the `secrets.h` file) and also to the Firebase Realtime Database (the connections details are also on the `secrets.h` file), using the Firebase ESP32 Client library [2]. Then initializes the local web server, which receives the local requests asynchronously, and finally creates the aquarium object on the database with every characteristic, including its *ssid* and *localIP*.

Listing 1 illustrates the pattern of the `secrets.h` file, that needs to present in the `src/` folder.

4.2.2 Execution with current and old variables per characteristic

Every second, to update the various values (temperature and its range, pH and its range, brightness and mode), the aquarium starts by getting the updated current values from the database.

To efficiently recognize updates on these values and avoid updating them every time they are read, the aquarium checks if the current value for each of the characteristics has changed since the last read or updated value.

As mentioned before, there is the possibility of having both remote and local updates. In this scenario, if there are only these two variables, then there is a collision when updating the current value since the local updates can be overlapped with the remote ones and vice-versa. So, we assumed that the local updates should have priority over the remote ones, i.e. in case of conflict, the local update is stored as the current value.

4.2.3 Execution with current, old, remote and local variables per characteristic

To handle this collision and priority situation, the aquarium stores these received updates separately and then chooses the current value from these additional variables, where local overlaps the remote value.

4.2.4 General Execution

With these collisions treated, then the aquarium updates the values of the current temperature and pH, and updates every updated value in the database.

To finish this execution (which is repeated every second), the LEDs that simulate the aquarium's light are turned on or off, according to the needed extra brightness.

4.2.5 Local Web Server

For receiving the local requests asynchronously while the aquarium is powered on, there are some defined endpoints to access and update the characteristics of the aquarium:

GET / (Info): shows the values of every characteristic

PUT *`/brightness/increase`* (Brightness Increase): increases the brightness

PUT *`/brightness/decrease`* (Brightness Decrease): decreases the brightness

PUT *`/mode/auto`* (Automatic Mode): changes the brightness mode to automatic

PUT *`/mode/manual`* (Manual Mode): changes the brightness mode to manual

PUT *`/ranges`* (Ranges): changes the ranges of values of the temperature and pH, given the request's body. Listing 2 shows the JSON pattern of the request's body.

5 Database

The database we chose relies on the Firebase Realtime Database [1] for the following reasons:

- There is no need for advanced querying, sorting or transactions;
- Both the physical simulator and mobile app will be sending a stream of tiny updates;
- Simple data is easy to store.

The database stores data as one large JSON tree, each field being saved as a key/value pair. The aquariums are stored as JSON objects, directly at the root of the JSON tree, where the key corresponds to its id. The remaining data, such as the properties of each aquarium, is kept as a key/value under the aquarium's JSON, where the key is the name of the property.

An example of the data stored in the database is shown in Listing 3, consisting of a JSON tree that has one aquarium with the id *-N2Qy6IYbzSgi–YvBun*.

6 Mobile Application

6.1 Overview

The mobile application is composed by three main screens, which are illustrated in Figure 3:

- Aquariums List: displays two lists with the aquariums' names:
  - Local: showing only the local aquariums;
  - Remote: showing all the remaining aquariums;
- Aquarium Details: displays the environment conditions of the chosen aquarium: temperature, pH and brightness levels, which might be obtained directly from physical simulator (if local) or from the database (if remote). The user can also choose between manually setting the illumination, or letting the system to manage it automatically;

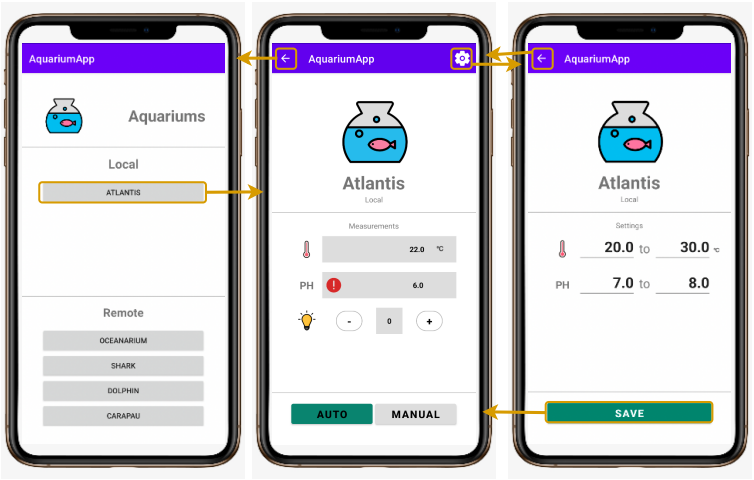

Figure 3: Mobile Application

- Aquarium Settings: allows the user to specify the range of values that are allowed for the temperature and pH levels.

When the users open the application, they observe the Aquariums List screen, where they choose the aquarium they intend to control and keep up with.

After selecting an aquarium, they are redirected to the Aquarium Details screen that allows them to observe the environment characteristics of the aquarium selected, as well as controlling the brightness, or simply having it automatic (according to what the photoresistor sensor is capturing).

Finally, when the users tap on the settings button of the aquarium, they can configure the range of values allowed for the temperature and the pH levels of the water.

If the value of any of these parameters (temperature and pH levels) happens to be outside of the specified range, then an alert on the Aquarium Details screen is shown, e.g in Figure 3, we can observe that the measured value for the pH is outside of the defined range of values, so a "danger" icon is shown in the pH measurement section, and the same could happen for the temperature.

6.2 Implementation

This component was developed in Kotlin using the Android Studio IDE.

In order to have three different interfaces for the user, we created three Activities (*AquariumList.kt*, *AquariumDetails.kt*, *AquariumRanges.kt*).

The data that the application consumes and forwards may come/go from/to two different targets: Firebase Realtime Database (remote) and the ESP32 itself (local). To send/query data to/from the Firebase Realtime Database, we used the Firebase Realtime Database for Android library [3]. In order to locally send requests to the ESP32, the Google's Volley library [6] was used.

6.2.1 Aquarium List

In order to display a button for each existent aquarium, it is necessary to query the database. As mentioned already, there are two lists of aquariums: local and remote. For each one of the received aquariums, we create a button with its name, and if the *ssid* of the aquarium matches the one the Android's device is connected to, then we place the button in the *local* list, otherwise we place it in the *remote* list.

We took advantage of the fact that the Firebase Realtime Database library [3] offers callbacks that are triggered whenever something changes in the database. So, when any aquarium changes, specially their *ssid*, the activity performs the necessary changes automatically, e.g. if an aquarium used to have a different *ssid* from the Android device's, and it changed to the same of the Android device's, then its correspondent button moves from the *remote* list to the *local* list.

6.2.2 Aquarium Details

Whenever this activity is initialized, it receives the *id* of the aquarium, whether it is remote or local to the Android device, and its *localIP*.

If the aquarium is remote, a callback is created to query its data (using the *id* property) on the creation of the activity. Whenever the remote data changes (database), this callback is called in order to display the most recent updates. Otherwise, we submit isolated requests to the aquarium locally every second.

When the user increments/decrements the brightness (when on manual mode) or changes modes (from manual to automatic or vice-versa) we perform isolated requests to the database when dealing with a remote aquarium, and then the main callback gets triggered with this change. However, when dealing with a local aquarium, these isolated requests are submitted directly to the aquarium.

For example, Listing 4 shows the pattern of a response from an aquarium, when requesting its data locally (*GET* request to the / endpoint of the async web server of the ESP32).

6.2.3 Aquarium Ranges

Whenever this activity is initialized, it also receives the *id* of the aquarium, whether it is remote or local to the Android device, and its *localIP*.

If the aquarium is remote, we request the ranges of the temperature and pH in an isolated manner, i.e. no callbacks, using the *id* property. Otherwise, we submit a local request to obtain them. Either of these operations is done only on the creation of the activity, since users may update the ranges on the *EditText* components.

When hitting save, isolated requests are performed to the target. The target is the database when we are dealing with a remote aquarium. However, when the aquarium is local to the Android's device, it sends the ranges data directly to the ESP32 (*PUT* request to the */ranges* endpoint of the async web server of the ESP32). After receiving a 200 response, we forward the user to the Aquarium Details interface.

6.2.4 Interface Design

Having every detail working on each of the activities screens, it is important to have a nice and intuitive interface to help the navigation.

We started by creating some margins between the components and the edges of the screen, with the use of some guidelines (e.g. at 5% and 95% vertically from the top, and at 10% and 90% horizontally from the left). With these various guidelines, the elements can then be better placed from each other as well as adjust automatically to most of the devices screen sizes.

7 Conclusion

This project is another example of a smart IoT system. It reveals to be very useful for people who do not have the time to keep up with the aquarium's proper characteristics.

In terms of scaling the system for having many more aquariums, and for having more security, it would be necessary to store more aquariums on the database. In order to restrict the access to the aquariums, it would be necessary to implement an account system, where each user would have access to a restricted set of aquariums. To do that, a database is needed to store the accounts, and their respective aquariums.

What we take the most out of this project is that we were introduced to two new technologies: development of Android applications and microcontrollers programming, where third-party applications are able to access them. Also, we have noticed that debugging issues related to hardware (ESP32) are more difficult to detect and solve, whereas on the Android side, we found it easier.

References

[1] *Firebase Realtime Database*. URL: `https://firebase.google.com/docs/database`. (accessed: 19.05.2022).

[2] *Firebase Realtime Database Arduino Library for ESP32*. URL: `https://github.com/mobizt/Firebase-ESP32`. (accessed: 19.05.2022).

[3] *Firebase Realtime Database Library for Android Documentation*. URL: `https://firebase.google.com/docs/database/android/start`. (accessed: 09.06.2022).

[4] *PlatformIO Website*. URL: `https://platformio.org/`. (accessed: 09.06.2022).

[5] *Visual Studio Code Website*. URL: `https://code.visualstudio.com/`. (accessed: 09.06.2022).

[6] *Volley Documentation*. URL: `https://google.github.io/volley/`. (accessed: 09.06.2022).


```
// Replace with your network credentials
const char *WIFI_SSID = "";
const char *WIFI_PASSWORD = "";

// Replace with your Firebase API key
const char *API_KEY = "";

// Replace with your Firebase Realtime Database URL
String DATABASE_URL = "";
```

Listing 1: Pattern of the *secrets.h* file

```
{
    "temperatureMin": 20
    "temperatureMax": 30,
    "phMin": 6.5,
    "phMax": 7.5
}
```

Listing 2: JSON pattern of the request body for the */ranges* endpoint

```json
{
  "-N2Qy6IYbzSgi--YvBun": {
    "brightness": 1,
    "localIP": "172.20.10.11",
    "manualMode": true,
    "name": "Atlantis",
    "ph": 7,
    "phMax": 8,
    "phMin": 6,
    "ssid": "MEO-A8B113",
    "temperature": 26,
    "temperatureMax": 30,
    "temperatureMin": 20
  }
}
```

Listing 3: Firebase Realtime Database JSON tree with one aquarium stored example

```json
{
    "name": "Atlantis",
    "manualMode": false,
    "brightness": 0,
    "temperature": 22,
    "temperatureMin": 20,
    "temperatureMax": 30,
    "ph": 6,
    "phMin": 6,
    "phMax": 8
}
```

Listing 4: Pattern of a JSON response from an aquarium (ESP32) when requesting its data