Remote

Name: Tomas Perez-Rodriguez

I. Introduction

Remote is an experiment: It delivers on a simple end use case, while experimenting with techniques useful for future projects. The main goal of Remote is to replace the universal remote with a phone. By approaching a display and scanning a bar code, the user can start interacting with that display. Some scenarios that could be enabled by this application are:

- An office manager could control multiple displays across a facility from their phone.
- An event manager could control displays at an event and manage their media remotely.
- A home user could control multiple picture frame or TV displays.

The use case selected also aligns with other future projects I am pursuing. In the agriculture business, we often work on scouting: walking through fields to collect information, such as the presence of rocks or water, or the health of the plants. As scouting is digitized, sensor data collection needs to evolve. A possible use case would be for the farmer to walk their field with their phone or tablet device, scanning sensor QR codes scattered through the field. Later the grower could use that app to interact with the sensors remotely.

This project helps to prove this kind of technology in a simpler way: The TVs replace the sensors, and provide a quick way to experiment with interactivity and location scenarios relevant to the sensor application.

II. Problem Statement

Given the eventual goals of Remote, this project starts with a limited focus: enabling a tech enthusiast home user to control multiple media devices. This use case serves as a starting point from which different applications could be developed in the future. This includes both the remote control media scenarios discussed earlier, as well as evolving into remote sensing applications.

Since we are not producing hardware media devices, or dedicated media center software, Remote will interact with a simple media center application. This application will stand in for a future media server application or plugin for remote connectivity. The main focus of Remote will be to create the React Native application that runs on the phone and remotely controls these media servers.

The detailed user story for Remote is as follows:

- A user loads and installs the media server application on their computer. Starting the application, they see a QR Code on their screen
- The user installs the Remote application on their phone.
- After launching the Remote application, the user uses the QR code scanner to connect to the server.
- The user can start and stop media from the phone.
- The user can connect to multiple servers on multiple systems.
- When the application launches, it shows the user a list of nearby server, and hides servers that are out of reach.

Given the experimental nature of this project, there are a few additional requirements limiting the design choices:

- In the future we may want to work with offline or unconfigured sensors, so an offline binding mechanism (such as QR codes) is preferred.
- In the future, sensors may interact with each other, so use an API mechanism that makes it easier to develop multiple clients.
- Whether a connection to the device can be established or not, it is more important to let the user know the device was once seen in their proximity. This restricts the nearby device filtering to technologies that can remember the location of a previously detected device.

To achieve these goals, these key features were selected:

- GPS: to filter displays near to the user
- Camera and QR Code detection: to discover hosts
- Remote APIs: to interact with the hosts
- Async Storage to persist host information when the application is closed.

III. Design and Implementation

The project is split in two parts: server and remote. The server is a simple NodeJS express application serving a media API. The API functions leverage the node-mpv library to interact with the MPV media player on the host computer.

The main focus of the project is on building the React Native based Remote application. The Remote app is split into the main App module, Page modules and Component modules. Data types and additional helper functions are in the Types module.

The main application 'App.tsx' tracks the currently selected server state. If a server is selected, it shows the media control page. If no server is selected or available, the main app shows the server selection page. The main application hosts two pages:

- PageServerSelection: This is the default startup page, shown when no server state is selected. It hosts a Tab selection and one of QR Code scanner, Nearby Servers or All Servers lists. The Tab selection buttons set the "tab" state to choose which one is shown. All three of the tabs allow the user to select a state and set the server state in response.
- PageServerControl: This page hosts the options, playlist and playback controls for a specific server. The page has an Options button that enables the options state to show the control. It also has a back button that clears the server state, returning the user to the PageServerSelection page. The Playlist and and PlaybackControls components handle the server playback methods.

The biggest challenge in developing this application was coming up with support for SSL on the server. While it is possible to support SSL with private CA signed certificates, iOS won't accept such certificates. On the other hand, since we are working with dynamically created servers on private hosts, it is unlikely the hosts will have TLD host names that can get public CA signed certificates to work with iOS. For this application we get around this by using HTTP, but this problem would have to be resolved to build an effective communication story. In the sensor use case considered earlier, we might be able to use BTLE for communication, but in the server case that's not a likely possibility as some servers may not have the bluetooth connectivity needed.

The second problem to solve was how to configure enough information in a usable QR code, and making the app resilient to unrelated QR codes. In the end, the simplest solution was to encode a simple string (MEDIASERVER) before the json information with the server information. By adding a url field to the json structure, we could also add a link to a app download page for users that don't have the application yet.

The final challenge is in the GPS distance calculation. For the local testing area, the app uses simple cartesian coordinates which is good enough for an experiment. In a full application where distance might need to be specified in meters, the app will need a geodesic formula to calculate the distance correctly around the world.

IV. Minimum UI Requirements

Remote uses a simplified workflow with separate pages for each user task. When a page or component does not have the opportunity to interact with the user, it provides guidance on what to do next. For example, when the app first loads, the server selection page tells them to go ahead and scan a QR code.

Responsiveness in the UI is achieved by separating the UI rendering from the processing. Async operations update the UI when the system is ready for more input, allowing the user to continue interacting with the UI while waiting. For example when a movie is selected, the movie playback controls won't load until the movie actually begins playing on the server.

To maintain a appealing design, Remote uses a consistent language, grouping related controls in a shaded bounding box that allows the user what's happening in logical steps. The server selection and QR code controls are wrapped in a shaded box. So are the tab buttons wrapped in a separate box. When the user changes tabs, the tab box remains constant, but the server selection boxes change in place. The same logic is used in the playback page.

Additional unity is achieved through consistent use of fonts, colors and element design throughout the UI. Control spacing and arrangement are also used to help the user follow the flow of the application.

V. Testing and Evaluation

Describe the testing process used to evaluate the application. Discuss the results of testing, including any issues encountered and how they were resolved. Explain any changes made to the application based on testing and evaluation.

The testing process was limited to manual testing of the Remote application. A javascript script was also used to prototype and verify server functionality and simulate the mobile app before development.

The manual scenarios covered were:

- Discovering a server via QR code on fresh install, app switches to playback and server switches to playback mode
- Discovering a second server via QR code
- Forgetting a server and reconnecting it via QR code
- Walking 100 ft to verify server is no longer in nearby list. (server is no longer on nearby list)
- Configure a server with an empty playlist and discover it (app shows no movies avaialble message)
- Play movies on separate servers at the same time (state remains consistent)
- Attempt to open a server that is out of range (app shows no movies available message)
- Attempt to scan an invalid QR code (nothing happens)

VI. Conclusion

The Remote application allows an user to connect to multiple servers, and play different videos on each server. It enables the user to quickly view the state of each controlled device, and start and stop the media from a single control.

The project has served as an interesting proving ground for many application concepts I've been pursuing over the years. In the past, when trying to solve a similar media server setup, I had thought to host the application as a react web application hosted on the server itself. By building a React Native application, I can now decouple the control from the server and add support for multiple servers in the same app!

On the other hand, I learnt about unexpected challenges, such as the SSL limitations in iOS and the challenges with creating a secure connection to the host in question. Future work may want to establish a token mechanism where the QR code includes a secret token that can be used to validate the user is in the physical presence of the host in question. Additional research is needed into SSL options within iOS to allow connectivity without exceptions.

Another model to explore might be to have the servers connect to a centralized index server on the cloud. Each media server could show a QR code directing the phone to that index server, and the server could then serve as an intermediary with the phone. This would even benefit the case where a phone is not connected to the same network as the media servers. Such a scheme may be even more necessary when working with remote cloud services operating in the internet via IP for Wireless Sensor Networks (IP-WSN).

VII. Figma

The Figma wireframe for the application is available on [figma.com: Remote Wireframe](https://www.figma.com/file/nd4dnUy5mikk9D8X9gGQsH/Remote-Wireframe?type=design&node-id=0%3A1&t=BwToQVPOOuOx1j7L-1).

```
https://www.figma.com/file/nd4dnUy5mikk9D8X9gGQsH/Remote-
Wireframe?type=design&node-id=0%3A1&t=BwToQVPOOuOx1j7L-1
```

VIII. Demo Video

Youtube Video: https://youtu.be/D8zBdg8mkZM

IX. References

- React Native Camera documentation: https://react-native-camera.github.io/react-native-camera/docs/rncamera
- React Native Slider by Michael Blanchard component: https://github.com/miblanchard/react-native-slider
- React Native Community Geolocation component: https://www.npmjs.com/package/@react-native-community/geolocation
- "Is Your Data Secure in React Native? Why You Need SSL Pinning" by The Whitespectre React Native Team: https://www.whitespectre.com/ideas/is-your-data-secure-in-react-native-why-you-need-ssl-pinning/
- Trust manually installed certificate profiles in iOS and iPadOS: https://support.apple.com/en-us/HT204477