Multi-platform Mobile App using Flux-like Architecture

This goal of this project is to implement a mobile client for StoryTime, a tech-ed platform that delivers high-quality, relevant, and free illustrated children's stories to families that don't have books at home. I will be using the React Native framework so that the app can support both Android and iOS using a single codebase. The app’s state and data will be managed using a single Redux store, which is a library that enforces Flux-like architecture, which will help to decouple the app's data from its views.

Introduction

I will summarize the tools of academic interest that I will be using to build this project.

React Native (a JavaScript framework for writing multi-platform native mobile applications)

React Native is a JavaScript (JS) framework that was released by Facebook in 2015 for developing mobile applications. Neither Android nor iOS apps are natively written in JS. However, components written in JS with React Native are rendered natively on the phone because the React native interfaces with the phone’s native APIs. The JS is interpreted by the host system on a dedicated thread, and the host system receives the API calls from the JS to to render native visual components and user interaction.

In contrast with React Native, other multi-platform ("hybrid") app frameworks simply render the app within a web browser-view on the phone. The browser-view technique simplifies app development because the app is really just a rendering a web app designed to run on any web browser. However, browser-views offer poorer UI performance when compared to native apps.¹

I want to use React Native for this project because the app will able to reach more low-income families by supporting both Android and iOS. I currently have no experience developing mobile apps, but React Native will allow me to iterate on the app quickly because I won’t have to learn two separate development paradigms for each platform. Using React Native will also ensure a great user experience on both platforms because it renders views natively, as opposed to doing everything in a browser-view.

Redux (a general, Flux-like architecture for writing JavaScript applications)

Many traditional apps are build with Model-View-Controller (MVC) architecture, whereby displayed elements (Views) derive their content from data generated by the Models and

¹ https://medium.com/@satya164/react-native-solution-to-hybrid-apps-5ff4d5696061#.2vdsw78vq
Controllers. I will be building this project with a Flux-like architecture, which is an alternative model for controlling the flow of data to the Views. Redux is a library that implements many features of Flux architecture that I will be using.

In MVC projects, application logic on both the client-side and backend are responsible for directly changing the state of the application. For example, clicking a button on the client-side may cause the view to render a loading indicator (a state change). However, the click may also signal the backend to do computations/retrieve data, which when returned to the client will cause more visual (and state) changes, depending on the data returned. As applications grow, this often leads to multiple sources changing app state, which can obfuscate the logic for updating the app's state, making the code hard to maintain and debug.

By contrast, Flux and Flux-like models enforce unidirectional data-flow, propagating all state in the app from a single source, called a Store. In Redux, the application logic never directly mutates the Store. Instead, all possible state-mutating interactions must be specified by describing 'actions', which when dispatched will modify the Store, and these changes will then propagate down to the view layer, which then causes the visual change in app state.

Retrieved from https://iweave.com/assets/blog/mvc_v_flux.png, although these diagrams were directly taken from this 2014 conference talk by Facebook: https://youtu.be/nYkdrAPrdcw?t=10m21s
Implementing the app with Redux will expose me to a novel approach to application design. It will also ensure that I follow the strict, safe conventions that the paradigm requires. Because Redux keeps track of all application state in a single Store, it is easy to correlate bugs and errors with specific, recreatable states. This ensures ease of testability, which will allow me to create a good experience for the user.

**Deliverables and The StoryTime Platform**

StoryTime is an ed-tech startup that has developed a research-based curriculum and platform to improve literacy and promote early-childhood cognitive development. The StoryTime app aims to provide the following:

A. Intuitive, easy access to illustrated children's stories  
B. A 'gamified'/feedback-based approach to reading progress within the curriculum  
C. Communication channels between educators and parents  
D. Methods for collecting usage information  
E. A process for evaluating literacy gains

These five interests are the basis for the deliverables of this app. The deliverables will be presented as the source code for the app as well as compiled binaries. The source code will implement the following deliverables:

1. **A User Interface that allows users to receive, collect, and read stories.**
   This will contain at least two main views, one for selecting and reading stories, the other for communicating with the teacher ('chat'/messaging'). This serves the interests of A and C. I may also be able to 'gamify' the process of collecting/reading books by providing constant feedback in either or both views. This serves the interest of B.

2. **A tie-in to the backend of the platform that will allow collection of usage information.**
   StoryTime hopes to use this information to improve the usability of the app and provide analysis for teachers. This portion of the app serves the interest of C, D, E and partially B.  
   **Note:** The analysis of the information collected by this deliverable is an entirely different project. This project only aims to develop a mechanism for collecting data to be analyzed by a black box. See appendix 1 for an overview of what type of information I will collect.

While these two deliverable seem like simple goals on paper, implementing them will be far from trivial. The app will need to cater to user experience while also connecting many asynchronous sources of data (e.g. the database, the story store, the StoryTime curriculum scheduler). I will also need to invent and implement data-collection techniques that are specific to the app’s use case.
Appendix 1- Information Collection in the App

The StoryTime app is essentially a medium to administer of a reading curriculum based on child development research. The Android App is simply one way to administer it, and we need to know if it is an effective administration. In particular, we have the following questions:

- How long do parents stay on the program?
- What type of content to parents care about most?
- Do parents want to engage with teachers in chat?
- Is our research-based curriculum viable as a literacy intervention?

To validate the curriculum and to create a user experience that appeals to our users (low-income parents), I will implement a system for collecting data on usage in the app. I will simply provide a means to collect the data in this project. (The analysis is a separate project.)

Most usage information that app collects on the user will be measures of engagement, including but not limited to:

1. Time spent reading on the app per week
2. Time spent reading on each particular story
3. Number of times opening each book
4. Number of messages between parent and teacher
5. Frequency of app use

In addition to helping us answer the questions listed above, the analysis of these data will also help us to refine the user experience, allowing us know what features are not used. I don't know yet which of these metrics will be most important.