Boundless - Design Document

System Interaction

Boundless will be using the following technologies:

- Server and Database: Firebase
- Front-end: React.js, React Native (for mobile experience), Redux (state management), JavaScript, CSS

The initial version of Boundless will be a web application, but we are planning on building a mobile experience as well. We expect most of our users to be using modern web browsers (Firefox, Chrome, Microsoft Edge), so there will be no issues with making requests to our Firebase server.

Requests will be made through user interaction with the GUI and go through Redux, a library for state management.

Below are some definitions to help in understanding the role of Redux and its functionality.

Redux Store:

- Holds application state
- Accessible from all components within the application
- i.e. a global state container

Actions:

- Payloads of information that send data from the application to the Redux Store
- In the context of our application:
  - Actions will be defined to interact with the Firebase DB, obtaining the requested data that will be fed to the reducer, which in turn triggers a state change within our component

Reducers:

- Specifies how application state changes in response to actions sent to the Redux Store

Dispatcher:

- Dispatches an action, the only way to trigger a state change within our component
- A part of the Redux Store

CRC Cards

User	
Parent Class	Subclasses
None	None
Responsibilities	Collaborates
- Set up online profile to introduce themselves - Communicates with other users	- UserProfile - Register - Chatroom

UserProfile	
Parent Class	Subclasses
None	None
Responsibilities	Collaborates
- Maintains preferences to a specific account for a personalized experience - Allows customization of settings including opting our of certain chatrooms	- User - Database - Register - Chatroom

Register	
Parent Class	Subclasses
None	None
Responsibilities	Collaborates
- Offers account creation and user login authentication	- User - UserProfile - Database

Database	
Parent Class	Subclasses
None	None
Responsibilities	Collaborates
- Store all data of all accounts registered - Allows for inserts, updates and deletions of accounts from the application	- UserProfile - Chatroom - Register

Chatroom	
Parent Class	Subclasses
None	None
Responsibilities	Collaborates
- Front-end UI for users to communicate with one another - Stores the user's messages into the database to be displayed to other accounts	- User - UserProfile - Database

A Concrete Example of Updating State

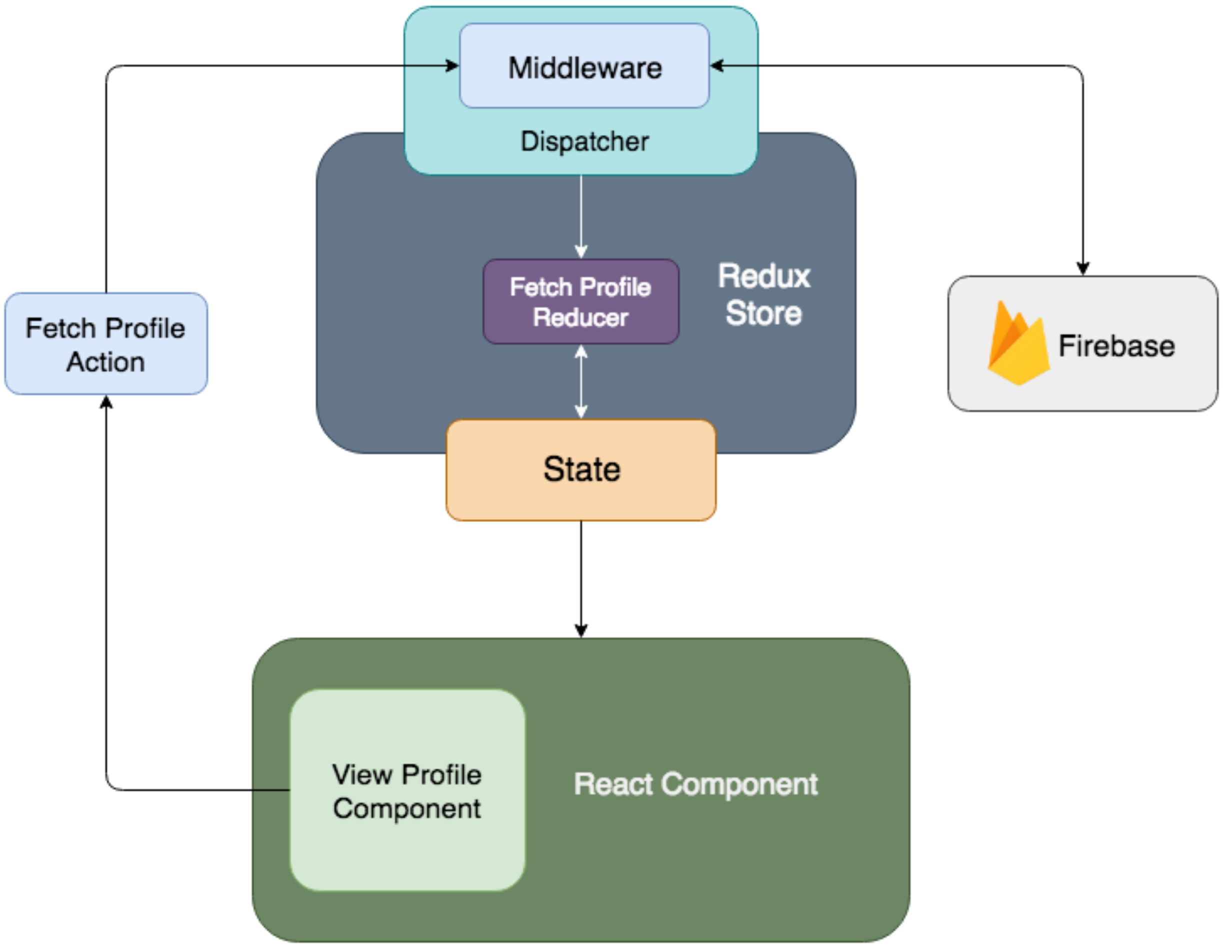
User interaction with a "View Profile" Component:

- There could be a "View Profile" button within the component
- On pressing this button, a "Fetch profile" action will be dispatched
- Code will be written to specify the behaviour associated with dispatching this action
- i.e. We want to pull information about this user from the FirebaseDB, and return this data in a format we specify (JSON)

- An example JSON:

```
user = {
    “user_id”: 1,
    “name”: “Test User”,
    “email”: “test@test.com",
    “major”: “Computer Science”
}
```

- The reducer will take this data and update the Redux Store in a manner we specify:
    - e.g. Update the field the Redux Store called “user_requested” by replacing its current value with the “user” JSON.
- The component from which the action was dispatched will have code to respond to changes in the Redux Store
    - e.g. The View Profile Component can display attributes within the “user_requested” entry in the Redux Store



The diagram on the next page gives a visual representation of these interactions:

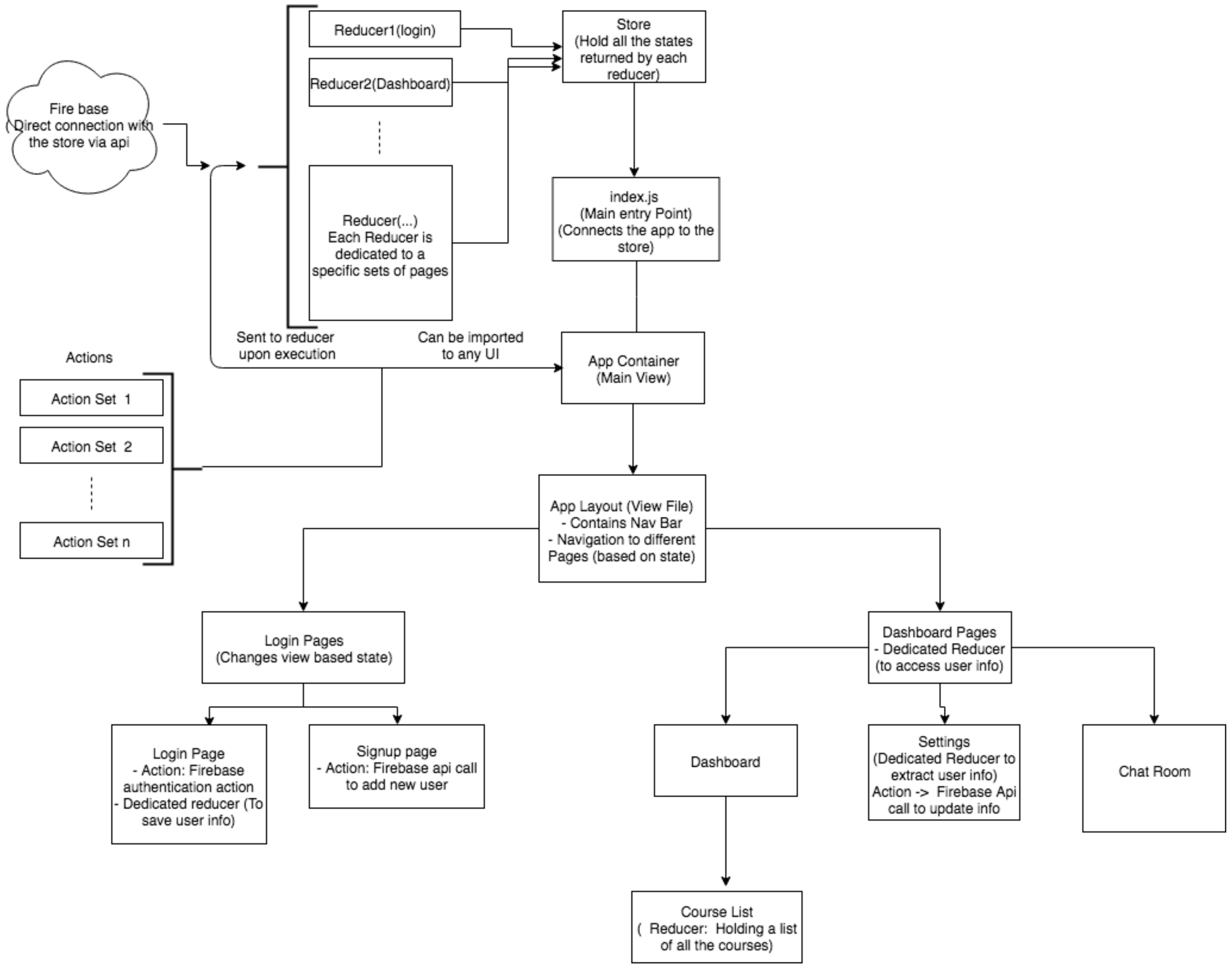
Software Architecture Diagram

System Decomposition

To handle any sort of error through out the program we will be using promises, which will allow us to execute different sets of code based on whether the promise returned successfully or with an error.

Each view will also hold authentication protocol that will be used to check the validation of the users input.

Basic Architecture


Reducer1(login)
Reducer2(Dashboard)
Reducer(...)
Each Reducer is dedicated to a specific sets of pages
Store
(Hold all the states returned by each reducer)
Fire base
(Direct connection with the store via api
index.js
(Main entry Point)
(Connects the app to the store)
Sent to reducer upon execution
Can be imported to any UI
App Container
(Main View)
Actions
Action Set 1
Action Set 2
Action Set n
App Layout (View File)
- Contains Nav Bar
- Navigation to different Pages (based on state)
Login Pages
(Changes view based state)
Dashboard Pages
- Dedicated Reducer
(to access user info)
Login Page
- Action: Firebase authentication action
- Dedicated reducer (To save user info)
Signup page
- Action: Firebase api call to add new user
Dashboard
Settings
(Dedicated Reducer to extract user info)
Action -> Firebase Api call to update info
Chat Room
Course List
(Reducer: Holding a list of all the courses)