Favour Trader
Descriptive overview and Justifications of System Architecture
Comp 4350, Group 8, Winter 2018

Our system utilizes a 3 tier architecture that is very common when implementing the “MERN” stack which stands for Mongodb, Express, React and Nodejs. If you look up MERN stack, you will see many example online of very similar architectures to ours. We ultimately chose this setup for a few reasons: familiarity with some member, desire to by members to learn about these technologies and the abundance of good documentation and resources relating to all of its components.

We use Trello for Iteration planning and work tracking because we like the card and bucket format, a few of us are used to the same format using products like JIRA on our work terms.

For our branching model we use the gitflow model because again several of us have experience with it and it is very easy to learn and keep track of.

We plan to create a domain model diagram soon, our domain is fairly simple though with three distinct models that can be viewed in the /favour-trader/models directory.

The Data Layer

Mongodb: This nosql database is commonly paired with node apps and very easy to use. We host our database on a free tier account of MongoDB’s own “Atlas” database as a service.

Mongoose: This library makes creating schemas and manipulating your data much simpler. It makes validation, casting much easier and allows us to not waste time writing business logic boiler plate. It is an implementation of the ODM (nosql version of ORM, the D is document) kind of similar to rails’ Active record.

The Business Layer

Node Js: The runtime on which our whole system is built, it is very easy to setup and configure simple web server in Node and we access bottom level information such as our environment variable through Node’s api.

ExpressJs: This is a very important piece of our puzzle, Express is a solid and battle tested framework for implementing a nodejs web server, It implements many patterns but most importantly is the “middleware” pattern. This gives an easy to control and manipulate chain of action taken when the server receives a request defining the path a request takes once on our server. It allows us to define and implement our api routes/endpoints and importantly acts as the

base to which our different modules like mongoose (access to db layer) and passport (authentication/authorization) are plugged in to.

PassportJs: This is another battle tested and well documented node library that makes setting up and carrying out authentication/authorization very simple. There are hundreds of auth ‘strategies’ available. We chose to go with a token based jwt (json web token) strategy since we will have a decoupled api that several different clients will be accessing.

The Presentation Layer

React: React is a popular new framework to create modern, ‘reactive’ and performative single page web applications. As opposed to traditional rendering and sending of html from the server, react defines “components” in a mix of javascript and a sort of hybrid templating spec called JSX that allows html to be embedded in javascript and rendered client side. We chose it because it is a very popular framework that we had interest in learning more about and because we wanted to make a single page application.

Webpack: Since React is just Javascript is can be served to the browser as a static asset. Webpack is configured to build our react app and it’s dependencies into a neat bundle to serve to the browser. We used the Create-React-App boilerplate for our frontend app which provides a nice webpack and babel configuration as well as a local development server so that we can see any changes we make in real time rather than manually running a webpack build after each change.

React Native: React native is essentially the mobile counterpart of React. It uses the same ideologies and patterns as react but you are building native app components with javascript as opposed to web components (html,css,js). We chose it because it is another popular and well supported framework and it will be easier for us to stick to a single way of writing frontend code.