

CS419

Dev Team: Moonwalk

Members: Wesley Jinks, Alex Rappa, Andrew Brown

Introduction

Moonwalk is a web based platform that integrates major social media networks to provide the user with one all encompassing user experience. Our website enables users to connect to their social media profiles, such as Twitter, and aggregates the user's social media feeds into one unified user stream. User's can seamlessly read, post, and share content across their social media profiles as well as customize their unified stream, enabling users the ability to curate their news feeds. Moonwalk's platform is built on the React JavaScript framework developed by Facebook, with the backend API being built on Hapi, an open-source node.js web framework developed by Wal-Mart's lead engineer Eran Hammer.

User Perspective Description

1. Users can register for a Moonwalk account with any valid email account.
 - 1.1. Users can pick a unique username
2. Users can login for access & adjust their saved settings under Profile Settings
 - 2.1. Users can link social media profiles to their Moonwalk Account
 - a. Twitter (implemented)
 - b. Facebook (not implemented)
 - c. Instagram (not implemented)
 - 2.2. Users can change their email address
 - 2.3 Users can change their password
 - 2.4 Users can delete their Moonwalk account
3. Users can view news feed in Moonwalk's unified Stream
 - 3.1. Users can view trending/top content without logging in
 - 3.2 Users can view their news feeds after logging in and linking their social media profiles
4. User can post images, videos, texts, and links
5. User can post a text description/summary and title

6. User can post, repost and like content on other social media sites directly

7. User can search through the news feed using keywords

7.1. User's can create a table sentiment analysis terms, compare sentiment analysis of different search terms, and view the individual tweets and sentiment

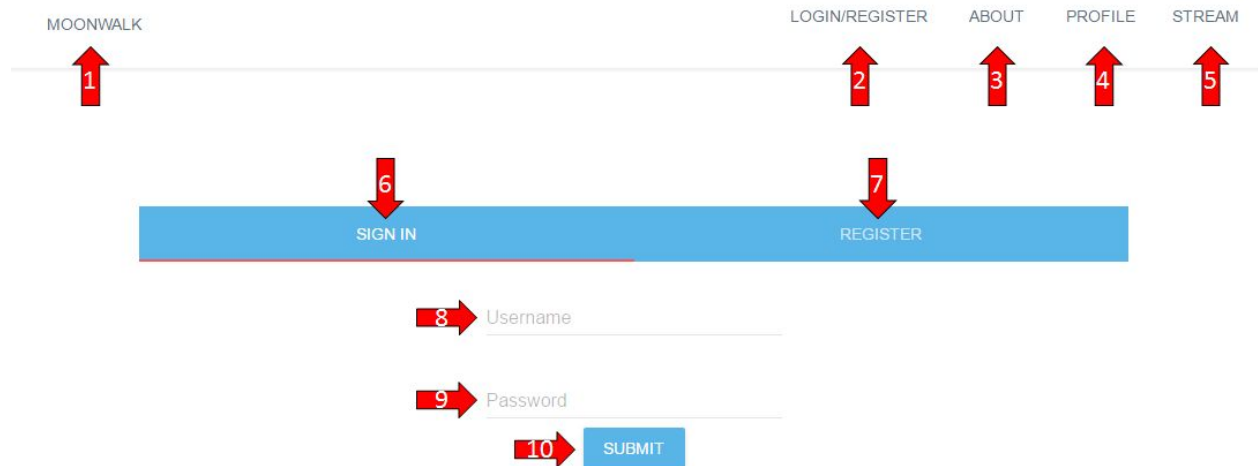
7.2. Users can quickly find specific content by searching

7.3. Users can view real-time trending topics across a variety of cities

Usage Instructions

Homepage: <http://54.212.196.159:3000/>

The Moonwalk landing page is where users can sign in or register for an account. New users can click the "Register" tab and fill out the required fields



The App Bar at the top of the screen serves as the primary means of site navigation

1. Moonwalk: button will take users back to this homepage from anywhere in the site
2. Login/Register: links to this sign in page
3. About: link to the sites About page with a brief description of our project
4. Profile: link to the user's Profile settings; must be logged in to access
5. Stream: link to the Stream page where the social feed is displayed
6. Sign In: this tab is set as default to allow users to quickly sign into their account
7. Register: clicking this tab will display the new user registration fields (see below)
8. Username: required field for an existing user to sign in
9. Password: required field for an existing user to sign in

10. Submit: this button will log the user in

User Registration: <http://54.212.196.159:3000/>

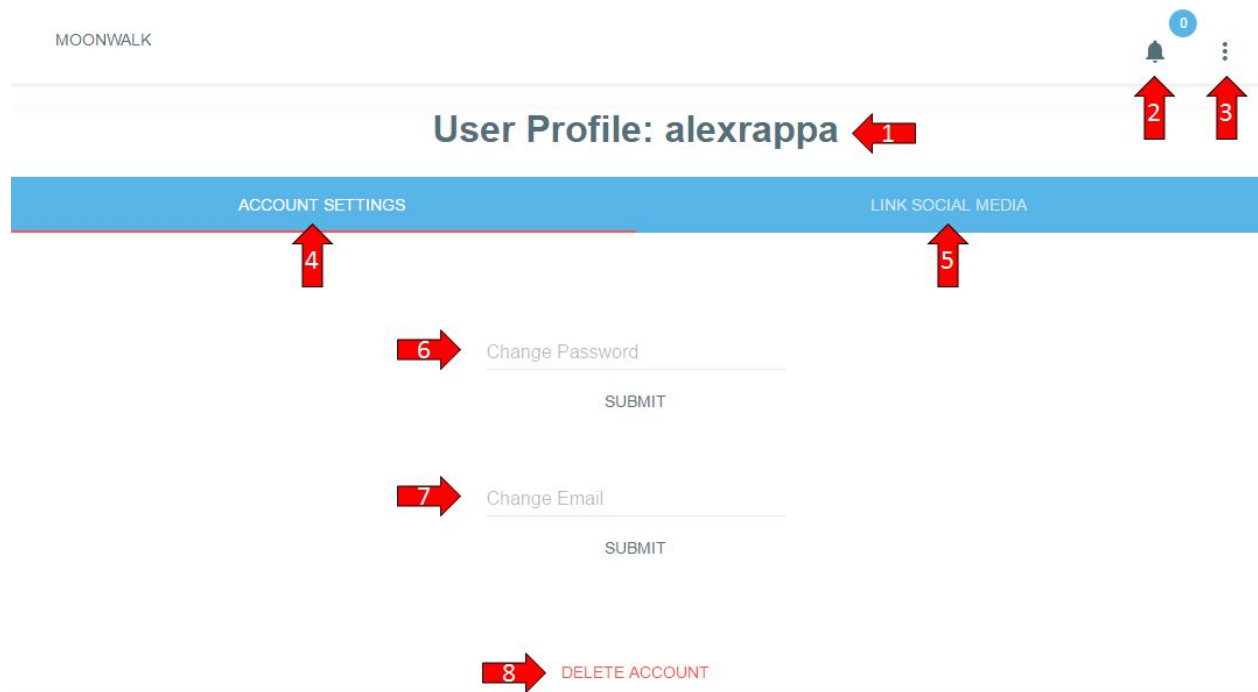
Selecting the Registration tab will display the form a new user must filled out to get an account.

The screenshot shows the Moonwalk user registration form. At the top, there is a navigation bar with the text "MOONWALK" on the left and "LOGIN/REGISTER", "ABOUT", "PROFILE", and "STREAM" on the right. Below the navigation bar is a blue bar with two tabs: "SIGN IN" and "REGISTER". The "REGISTER" tab is selected. Below the tabs is a registration form with five numbered red arrows pointing to the input fields and the submit button. The form consists of four text input fields labeled "Username", "Email", "Password", and "Password", and a blue "SUBMIT" button.

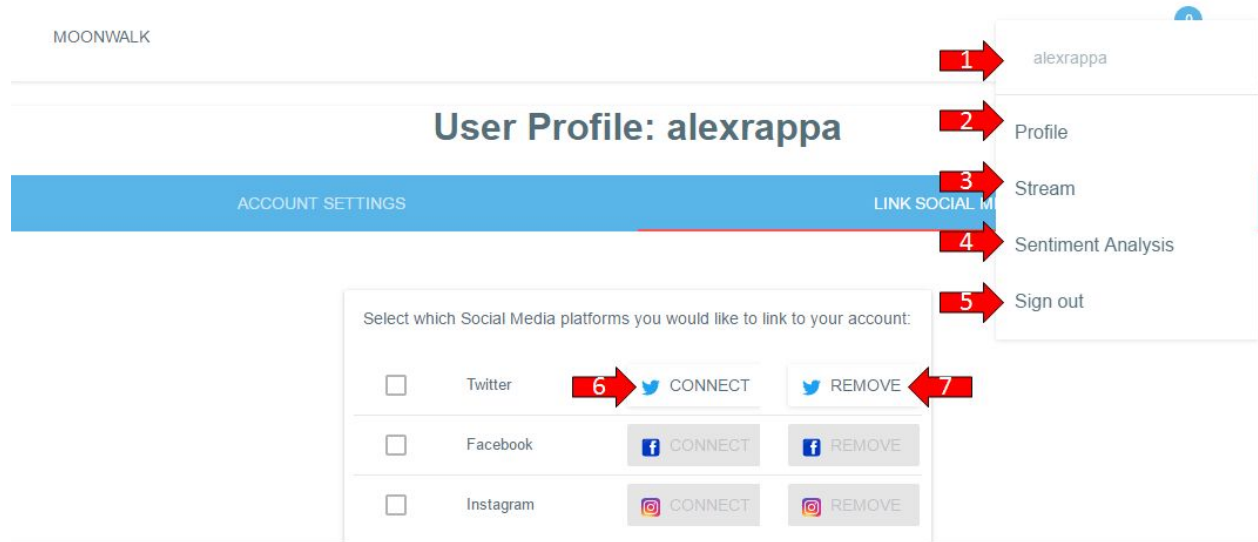
1. Username: display name for the user's Moonwalk account
2. Email: user email
- 3,4. Password for user's account (must match, at least 6 characters)
5. Submit button will register the new user and redirect to the new user walkthrough

Profile Settings: <http://54.212.196.159:3000/profile/>

New users are automatically redirected to the Profile page. Here users can edit account settings, delete their account, and link their social media profiles.



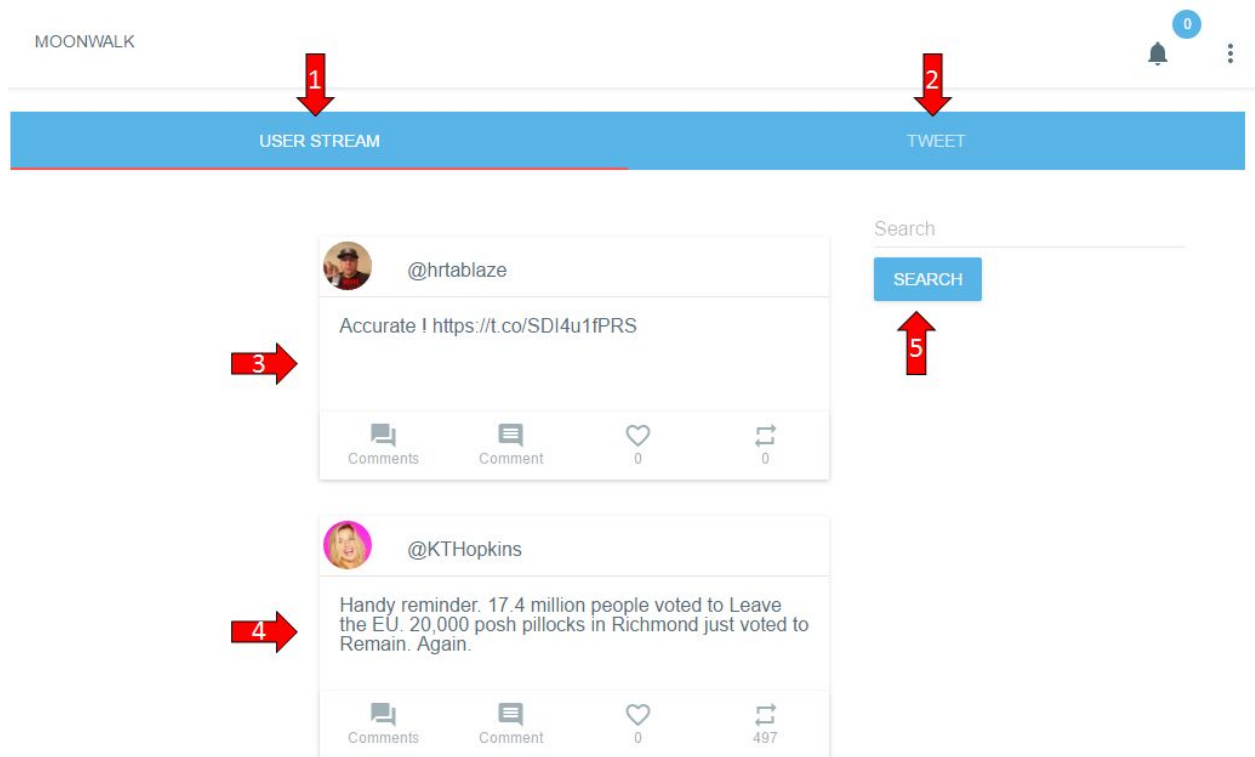
1. Current Username is displayed at the top of the page
2. Notifications Alert (not fully implemented) displays number of currently notifications received from the user's linked social media account.
3. User Menu Bar when selected displays a dropdown menu used to navigate through the site (see below)
4. Account Settings tab displays the settings a user can change on their account
5. Link Social Media is where the user can link to their social media profiles (see below)
6. Change Password is available to the user
7. Change Email is available to the user
8. Delete Account when clicked will prompt the user if they are sure they want to permanently delete their Moonwalk account.



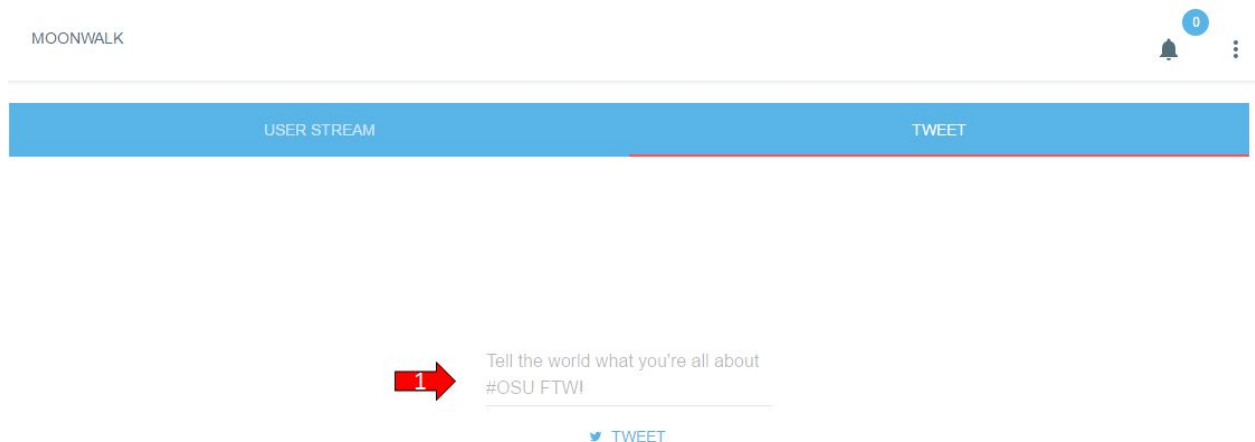
1. Username is displayed at the top of the menu
2. Link to Profile page
3. Link to Stream Page
4. Link to Sentiment Analysis page
5. Sign out button will end the current session
6. Connect when clicked will redirect the user to Twitter's page to authenticate the user and link the accounts.
7. Remove when clicked will remove and current Twitter information saved to the user's Moonwalk account.

User Stream: <http://54.212.196.159:3000/stream>

The Stream page displays the users linked social media feed all on one page.



1. User Stream tab displays the Social Content
2. Tweet tab displays the textfield used to post a new tweet
- 3, 4. Each post is it's own element and are displayed in a single list
5. Search feature allows users to search posts by key words



1. Under the Tweet tab there is a text field where users can submit a tweet that will be posted on their account.

Sentiment Analysis: <http://54.212.196.159:3000/sentiment>

This page allows the user to search social media posts by key words, run the content through a sentiment analysis, and display the results.

Term	Score	Average	Polarity	# Pos	# Neg	Avg Pos	Avg Neg	High	Low
New York	-4	-0.04	13						
Texas	55	0.55	14						
california	-5	-0.05	11						
florida	88	0.88	12						

1. Textfield where the user can enter keywords to search

2. Each search result is displayed as a single row in the results table alongside previously searched terms. Clicking on a row will open the sidebar (see below)

Average	Polarity	# Pos	# Neg	Avg Pos	Avg Neg	High	Low
-0.04	13						
0.55	14						
-0.05	11						
0.88	12						

1. Clicking on a row opens the sidebar with the result from that search

2. The sidebar contains the posts that were found in the keyword search. Each post has its own Sentiment Analysis rating displayed above it.

Trends: <http://54.212.196.159:3000/trends>


1. This page allows the user to select a city or country and see the real-time trending tweets for that area.
2. After clicking on a location. The user can select a trending topic by clicking on it.
3. After selecting a topic, a sidebar will open on the right with the trending tweets for that topic.
4. The user can then choose different topics to view or can select a different area.

MOONWALK

- WORLDWIDE
- WINNIPEG
- OTTAWA
- QUEBEC
- MONTREAL
- TORONTO
- EDMONTON
- CALGARY
- VANCOUVER
- BIRMINGHAM
- BLACKPOOL
- BOURNEMOUTH
- BRIGHTON
- BRISTOL
- CARDIFF
- COVENTRY
- DERBY
- EDINBURGH
- GLASGOW
- HULL
- LEEDS
- LEICESTER


TRENDS

- EVERYONE FREE ALL NIGHT TWEETS: NULL
- #SANDIEGO TWEETS: NULL
- #PADRES TWEETS: NULL
- #CHRISTMAS TWEETS: 142963
- TYGA TWEETS: 153711
- #PAC12CHAMPIONSHIP TWEETS: 16917
- 21 SAVAGE TWEETS: 124936
- #KIIJINGLEBALL TWEETS: 143508
- TAIWAN TWEETS: 304104
- ROSE BOWL TWEETS: NULL
- BUFFS TWEETS: 14619
- WESTERN MICHIGAN TWEETS: 15767
- PAC 12 TWEETS: 25164
- SEFO LIUFAU TWEETS: NULL
- PJ FLECK TWEETS: NULL
- JEREMIH TWEETS: 18686
- BOSCO TJAN TWEETS: NULL
- MYLES GASKIN TWEETS: NULL
- O'DEA TWEETS: NULL
- FAZE TWEETS: 25331
- LARRY SCOTT TWEETS: NULL




@ChinoPapasito


FANTASTIC TONIGHT EVERYONE FREE ALL NIGHT !!!! @FantasticRest_lounge
<https://t.co/SXvrtJFGGp>



0




0




@CSmoothnyc


TONIGHT FRIDAY FOLLOW ME BABY
🔴🔴🔴🔴🔴🔴 Everyone free all night 100\$...
<https://t.co/EjJmLRZq15>



0




0




@CSmoothnyc


TONIGHT FRIDAY FOLLOW ME BABY
🔴🔴🔴🔴🔴🔴 Everyone free all night 100\$...
<https://t.co/lelQrU6O1v>



0




0




@DJCinco

TONIGHT 🔴🔴🔴🔴🔴🔴🔴🔴 Everyone free all night 100\$ bottles all night Group of 5 ladies free belaire...
<https://t.co/adWJp2W1C3>



0

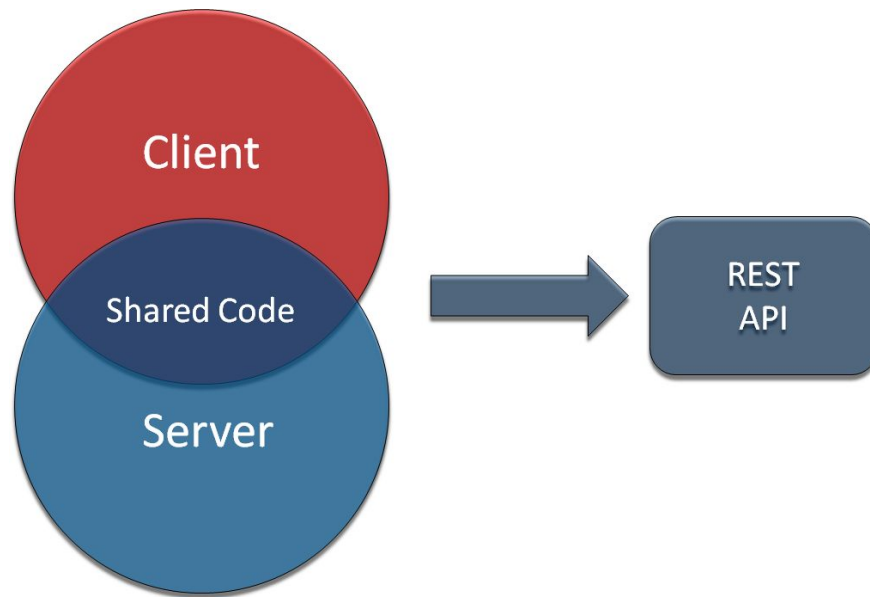


0

Software and System Functionality

Server Side Rendering:

Our goal was to implement server-side rendering because larger professional companies like Twitter, AirBnB, and Facebook among many other utilize this technique to improve page load time. We thought it would be good to get experience with the concepts and requirements of rendering on the server vs client-rendering that we have done. There are many articles discussing the benefits and methods of server-side rendering so we will focus on our implementation.



Server ("server/index.js"):

- We created an Express.js server for the page render. It has routes for the different views we implement like a typical Express backend would. For a route that matches a page to be rendered, a function is called.

Rendering ("server/routes/handleRender.js"):

- The render function calls match which is a react-router method for routing on the server. If there are not errors and the route is a valid route, we call handle render.
- handleRender() sets the user-agent for Material-UI, initializes the store on the server for our application state, and calls a React method renderToString to render our top level component to a string. We then initialize and stringify our initial app state. We then read our index.html file, appending the parent component to the root div. We also append the initial state, then send the rendered html.
- Our index.html file has a reference to the client bundle to serve those assets to the client. It also includes a reference to the styles.css file that we extracted from the client bundle during our build. We ran into a lot of difficulty getting css stylesheets to work with server-side rendering. Inline styles require no configuration, but external stylesheets need to be extracted to prevent errors on the server.

Other Routes:

- We also have an API server implemented with hapi.js that handles connections to our mongo database and making calls to external apis like twitter.

- We have begun implementing proxying to call the API server routes from our rendering server to have a layer between the client and the API server as well as to keep client HTTP methods consistent. We currently have the proxying implemented for working with twitter data, specifically getting a stream of tweets from twitter.

API Server

- We connected our API Server to a MongoDB instance to store users and user data.
- We also connected our API server to twitter to display tweets.
- We use json-web-tokens for auth state for users and on login or register requests that are successful send the client a token for authentication.
- We implemented OAuth on the API server to connect with twitter and other API's so that we don't need user's account information to connect our App to data from those API's.
- Structure:
 - /oauth: oauth handlers, database models, database and route controllers
 - /social: social handlers, database models, database and route controllers
 - /user: user handlers, database models, database and route controllers
 - Index.js => main server script

Client

We are implementing the client with React as our view framework and redux to manage the state.

Folder Structure Summary:

- /actions => Redux action handlers for state management
- /components => Smaller, children components
- /Containers => Larger, parent components
- /lib => Middleware for managing the state
- /reducers => When an action takes place these take the results of the action and the previous state to create a new state for the application
- /utils => Api utilities for handling http requests and responses
- Client.js => Entry point for the client.
- configureStore.js => Configuration for the store which holds the state tree of the application

- Routes.js => Routing structure and route definitions where we define which components map to which routes

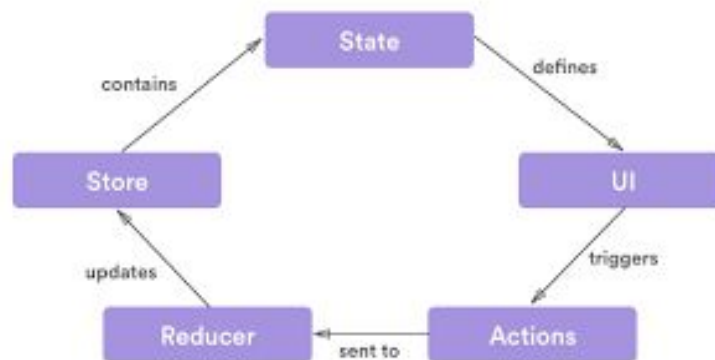
App Functionality:

The rendering of a page goes through Client.js, to routes.js, to Containers/App.js, to each page component

- Client.js:
 - Get the history from the browser
 - Initialize state from the server
 - Set the store with the state
 - We wrap our parent element in the Material UI theme
 - We pass the store to our provider which makes the store available to components
 - We pass the history and routes to our parent router component
- ConfigureStore.js:
 - We first pass thunk as middleware which handle async actions to our store.
 - Create the store with the combined reducer of all the reducers, the initial state, and the middleware we added.
- Containers/App/:
 - This container wraps the Navbar and the component pages.
 - It is what our pages are rendered in through the router

Async Handling with Redux:

This diagram shows state management with redux. For our asynchronous actions like getting data from the server, we defined functions in the utils folder. Then for our asynchronous actions we are able to easily create a start action that initializes the asynchronous flow, and a success and failure action. We wrap all three in the async action and call the function like GET, POST, etc that we want to implement.



Example with Tweet Stream:

1. utils/api.js -> defines fetch request and response handling
2. actions/streamActions.js -> defines async actions
3. reducers/streamReducers.js -> defines state modification from async actions
4. lib/selectors.js -> small helper
5. components/StreamList/stream.js -> Access stream props and actions in component on load

Software Libraries, APIs, Tools, Servers, and Systems

1. Libraries:
 - 1.1. React: user interface
 - 1.2. Redux: app state management
 - 1.3. ExpressJS: View Routing
 - 1.4. Hapi: API Calls
2. Languages
 - 2.1. Javascript: front-end & NodeJS server-side routing
 - 2.2. Python: Server-side microservice, API for sentiment analysis
3. API's
 - 3.1. Social Media
 - 3.1.1. Twitter (for posting and retrieving)
 - 3.2. Sentiment Analysis
 - 3.2.1. <https://github.com/thisandagain/sentiment>
4. Development Tools

- 4.1. Testing
 - 4.1.1. Mocha: unit tests
 - 4.1.2. Enzyme: UI components
 - 4.1.3. Karma: test runner
- 4.2. Build Tools
 - 4.2.1. Webpack: bundling
 - 4.2.2. Babel: javascript compiling: adding es2015/2016 support
- 4.3. Documentation
 - 4.3.1. ESDoc
- 5. Servers
 - 5.1. AWS or Google Cloud
 - 5.2. NodeJS with Hapi for Routing, IO tasks
 - 5.3. Python for API management, data analysis, CPU tasks(since node will block on these)
- 6. Database
 - 6.1. MongoDB
- 7. Node Modules/NPM Packages
 - 7.1. Dependencies
 - 7.1.1. **"aphrodite"**: Inline CSS library
 - 7.1.2. **"axios"**: Promise based HTTP Client
 - 7.1.3. **"Babel-polyfill"**: Provides necessary polyfills for ES2015
 - 7.1.4. **"babel-runtime"**: Babel self-contained run-time
 - 7.1.5. **"express"**: web framework/server
 - 7.1.6. **"history"**: Session history management plugin
 - 7.1.7. **"immutable"**: Immutable data collections for redux patterns
 - 7.1.8. **"object-assign"**: Pony fill for ES2015 that doesn't overwrite the native method
 - 7.1.9. **"react"**: react
 - 7.1.10. **"react-dom"**: react package for working with the dom
 - 7.1.11. **"react-redux"**: react bindings for redux
 - 7.1.12. **"react-router"**: mounting library for react
 - 7.1.13. **"redux"**: state management
 - 7.2. Dev Dependencies

- 7.2.1. **"babel-cli"**: babel command line
- 7.2.2. **"babel-core"**: babel compiler core
- 7.2.3. **"babel-eslint"**: Adds eslint functionality for babel-transpiled code
- 7.2.4. **"babel-loader"**: babel module loader for webpack
- 7.2.5. **"Babel-plugin-transform-require-ignore"**: ignore files by file extension type
- 7.2.6. **"babel-plugin-transform-runtime"**: Externalise references to helpers and builtins, automatically polyfilling your code without polluting globals
- 7.2.7. **"babel-preset-es2015"**: babel preset for es2016
- 7.2.8. **"babel-preset-react"**: babel preset for react
- 7.2.9. **"babel-preset-react-hmre"**: babel preset for Hot Module Replacement
- 7.2.10. **"babel-register"**: babel require hook
- 7.2.11. **"esdoc"**: documentation generator for javascript
- 7.2.12. **"eslint"**: pattern checker for javascript
- 7.2.13. **"eslint-config-airbnb"**: AirBnB's styleguide
- 7.2.14. **"eslint-loader"**: loader for webpack
- 7.2.15. **"Eslint-plugin-import"**: adds support for es2015 import syntax for eslint
- 7.2.16. **"eslint-plugin-jsx-a11y"**: internationalization
- 7.2.17. **"eslint-plugin-react"**: react specific linting
- 7.2.18. **"material-ui"**: material design react components
- 7.2.19. **"react-hot-loader"**: tweak react components in real time
- 7.2.20. **"webpack"**: packs and builds CommonJS/AWD modules, allows bundling, file loading and preprocessing
- 7.2.21. **"webpack-dev-middleware"**: Offers middleware to serve webpack bundle to express or other server
- 7.2.22. **"webpack-dev-server"**: Development server for testing
- 7.2.23. **"webpack-hot-middleware"**: Webpack hot reloading plugin for adding to existing dev server

Team Member Accomplishments

Alex Rappa: I worked on the frontend development for this project. I helped with the initial setup of routing scheme used to connect each React component to a specific url path. I created the frontend framework which houses each separate component (Home, About, Profile, Stream). After the initial structure was in place, we decided to use Material-ui to better handle our visual design and I integrated the library into each of our React components: homepage's sign in form and registration form, new user walkthrough, profile settings page, and stream page's grid list. Most of my effort was focused on the component structure and styling, and minor tweaks to the components action handlers.

Andrew Brown: I did the backend API, using Hapi for the web framework (routes, controllers, models), MongoDB for persisting user information, and AWS for hosting. This functioned as a stand-alone, REST api. The reasoning for this structure was that our frontend is something of a Single Page Application, so it gets all of its data by sending requests to the API. I created endpoints for registering, logging in, updating users, and then many different ones for the social aspect as well, including linking a Twitter account, retrieving Twitter data, running a sentiment analysis, and others. The sentiment analysis is done using an NPM library called 'sentiment.' It parses the text and ranks the words as positive or negative based on the AFINN word list. More information can be found here:

http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010

Since the tweets are an array, I used a map function to run a sentiment analysis on each one, and keep a running tally of totals, polarity, and other metrics. Wesley helped with this aspect as well.

Wesley Jinks: I created the structure and data flow for the frontend using React and Redux. I spent several weeks researching how to make an isomorphic React app, and implemented the server side rendering. In doing so, I developed the build scripts using webpack for the client and server builds. I also built out the code to manage the state using Redux. Redux let us store our application state in one place that is accessible from any component. I chose Redux because it is the defacto standard for managing application state with react. I was also interested in its functional approach, IE immutability. Using Redux, I set up functions to fetch data from the backend API, and map the results to state, and update our React

components to reflect the new state. I also set up dev tools such as hot reloading that helped the team be more productive. I also worked with Andrew's backend API code to connect the backend api to the frontend and implement features like the tweet stream, the sentiment analysis, storing user information in session storage, login and register features, and the trending page. I also worked to abstract components and features to make them easy

Closing

Overall we were happy with the outcome, especially the sentiment analysis page. The end result turned out to be a fun website, even if we did not hit all of our goals, such as adding Facebook and other social accounts. Mainly we wanted to take a deep dive into React and Redux, creating an isomorphic web app, and connect social networks to sentiment analysis. So we did meet our core goals, and gained an incredible amount of knowledge about modern web application structure.