U-Post Capstone Midterm Report

Date: June 17th, 2019

Team Members:

- Willis Cheung
- Julian Mulia
- Edward Song

Academic Advisor: Dr. Mohammad Moshirpour
Project Industry Sponsor: Naser Arda, Hunter Hub - Social Enterprise Portfolio
Project Owner: Megan Leslie

Introduction

As a member of the University of Calgary community, it is hard to ignore the cluttered bulletin boards, filled with posters displaying a wide variety of events, services, and advertisements that few people actually take the time to read. This outdated method of communicating events and services not only limits the effectiveness of event awareness and attendance; it also creates a large amount of paper waste. In addition, this method does not allow for real-time updating, data analysis, forecasting capabilities, and the numerous advantages of a digital platform for events. In addition, food waste after events is an increasing concern especially for large communities, such as universities.

In the 2019 Winter semester at the University of Calgary, we tracked the evidence to support these problems and found that there were approximately 25 university-related events per day, the average capacity reached was 60%, and this resulted in nearly 2.5 garbage bags filled with food waste per day from events alone. Our U-Post team believes that through the help of an effective software platform we can help tackle these problems simultaneously to connect our community while reducing waste[1].

Objectives & Motivation

Two main questions motivate our team to develop U-Post. First, is there a one-stop-shop to see everything that is happening in your community/campus right now? Second, can excess food become part of an incentive strategy to improve event attendance, community awareness, and waste reduction?

Objective 1: Online Digital Bulletin Board – A one-stop-shop for all community/campus events, services, and advertisement. Similar to how YouTube contains videos or how Netflix contains movies and series; U-Post aims to contain all community-related events and services.

Objective 2: Waste Reduction Strategy – A community of hungry students and throwing delicious food away doesn't make sense, right? Printing 100 posters in the digital age doesn't make sense, right? Planning an event for 100 people and only 20 show up doesn't make sense, right? What do you do with all the extra food, supplies, space, and paper that is not being utilized? We want to make sense of sharing extra food, we want to make sense of paper-less advertising, and we want to make sense of reaching capacity at events/services.

U-Post will provide a one-stop shop for event/service communication, allowing providers to post event/service details while increasing event awareness and providing a social and environmental impact.

General Scope

The scope of the project is to develop a web-based platform and mobile website. This includes a user-friendly front-end and a cloud-based back-end to ensure U-Post is scalable. The software-developing framework will be discussed with our technical supervisor, Dr. Moshirpour, and will allow our team to develop an efficient, scalable, and user-centric product. Our team will be using agile development principles, which will include constant iterative development and possible integration of new technologies throughout the project. Our initial features in scope are the following:

1. Proposal based on U-Post Project Proposal ENSF 619-06, Winter, 2019, Mulia, Woods, Cheung.

- Content Personalization Strategy – exploiting visitor data to deliver relevant content based on their interests and preferences.
- User-centric Website – easy to use and intuitive for users based on the designs of existing platforms such as Netflix and YouTube.
- Real-Time Updates – allows service/event providers to send updates and posts to individuals interested and/or already registered for their event/service.
- Waste Reduction – allows service/event providers to send an announcement to U-Post users to let them know that there is free food/items leftover from an event. This tool can also be used if an event has unanticipated low attendance to draw in new attendees on the spot.
- Analytic Reports – provides service providers with key metrics regarding the effectiveness of their post, events, etc.
- After Event Content Access – allows service/event providers to release content to individuals who have successfully attended the event.

Methodology & Approach

Our approach towards developing the U-post system will be the following:

Front End – React Framework (JavaScript)
JavaScript is selected for the development of our frontend of our website due to the many advantages offered by JavaScript. These advantages include speed (easily runs with in the client side browser), popularity (resources are abundant and plentiful), dynamic nature (ability to download libraries at runtime), and web browser support [1], [2]. Our selection of JavaScript stems largely from the flexibility offered by JavaScript and the support offered by modern web browsers [2].

React is a JavaScript library used for building user interfaces and is structured to allow for easy development a client side MVC architecture [3], [4]. React is selected for this project as it provides an efficient event-based system and promotes the creation of testable and reusable components [5].

Backend – Django Framework (Python)
Django follows a Model-View-Template (MVT) pattern based off the MVC pattern. Django's model utilizes an Object-Relational-Mapping (ORM) layer which simplifies database and data operations [6]. Django also provides a flexible framework capable of building deep and dynamic websites in a short amount of time, to do so Django offers high-level abstractions of common Web-development patterns [7]. These benefits coupled with the ORM make Django suitable for our project as we aim to complete our project in the short timespan of four months.

Rest API
Rest API will be used for communication between the frontend and the backend of our proposed website as Django offers a built-in rest framework which helps to simplify communication.

Infrastructure – Amazon Web Services (AWS)
With the three major cloud service providers (AWS, GCS, Azure), choice of a cloud service provider relies on a few key factors such as service offerings, pricing, and customers. All three offer similar services and pricing, but AWS is a clear leader in customers serving big names such as Netflix, Dow Jones, and Airbnb [8], [9]. For these reasons there are a plethora of online resources resulting in our choice of AWS to host both our backend and our database.

System Requirements

Our functional and non-functional requirements for the web application were reviewed and updated. As well as, a new System Architecture for the redesign of the web application was developed.

System Use Cases

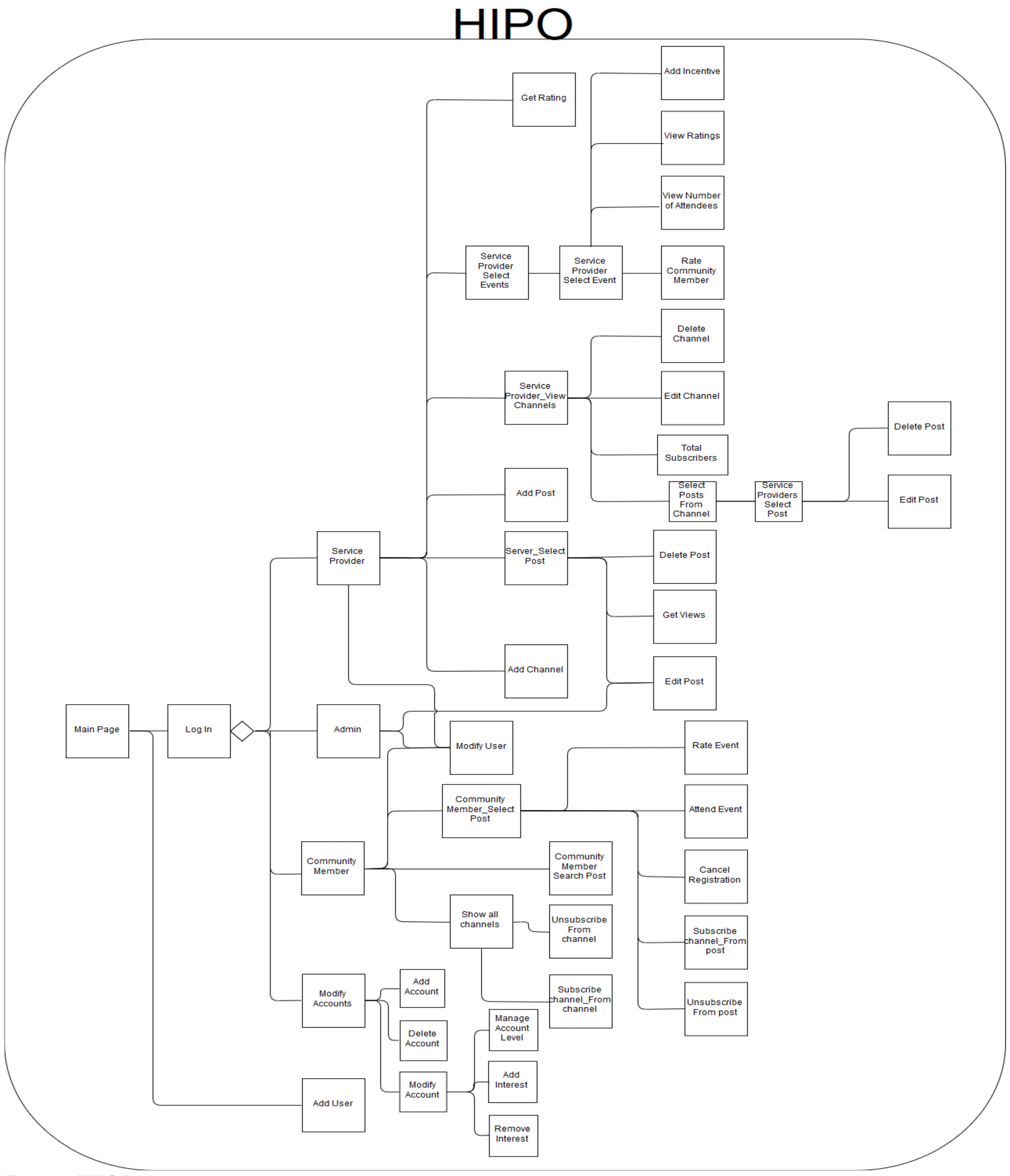


Figure 1 HIPO Diagram

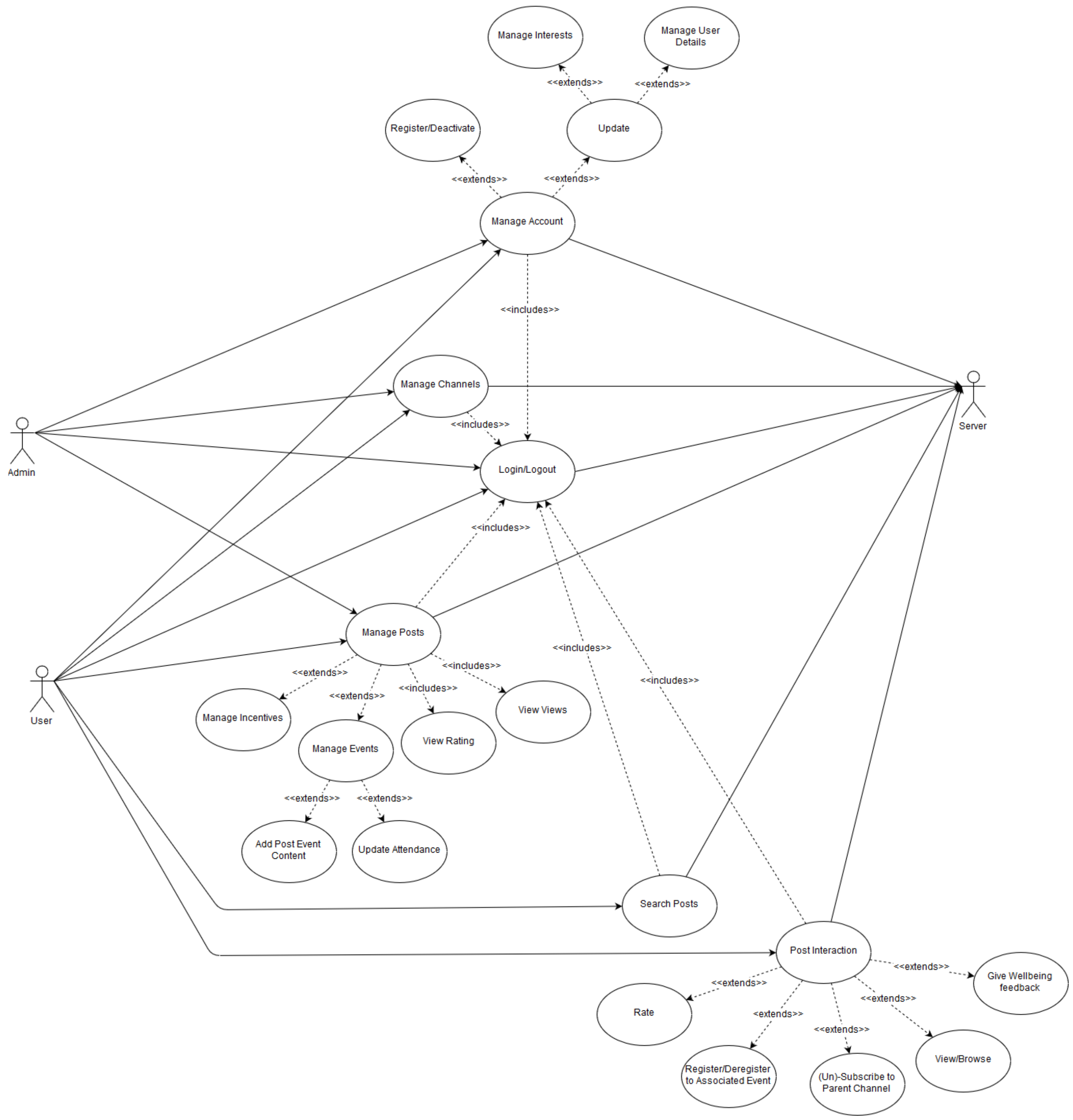


Figure 2 Use-Case Diagram

As seen from the use-case diagram (figure 2) we expect two major groups of users to be interacting with our system, administrators and users. Users and administrators should be able to login to our system; manage their accounts, channels, and posts. In addition to these actions, users should be able to search for posts and interact with posts belonging to other users.

Use Case 1 – Manage Account

Administrators and users should be able to register, deactivate, or update their account. When our users update their account, the user should be able to manage their interests or change their user details such as email, and password. The fields updated by the user should be checked for validity (e.g. valid email and meeting password requirements).

Use Case 2 – Manage Channels

Administrators and users should be able to manage their own channels. This includes actions such as adding a new channel or editing a channel.

Use Case 3 – Login/Logout

Administrators and users should be able to create a login session with a registered account or logout from the session.

To start the login process the user will input their username and password. The username and password are then checked against the customer database and authorization token is returned.

To start the logout, process the user will click a logout button which would terminate the login session and refuse to display sensitive information to the user until a login is provided.

Use Case 4 – Manage Posts

Administrators and users should be able to manage their own posts. A user should be able to assign and manage incentives towards their event, see the total amount of views and user rating towards their posts, add content towards their post, and update the attendance of their post based on the amount of users registered for the event.

Use Case 5 – Search Posts

Users should be able to search for current, passed, and upcoming posts. The user should be able to filter posts by keywords, timeframe, and interests to access the most relevant post to them, their respective community and interests.

Use Case 6 – Post Interaction

Users should be able to interact with posts such as writing feedback, providing a rating, register for an event, managing subscriptions to the parent channel, and viewing different posts.

Non-functional System Requirements

Non-functional Requirement 1 – Speed of System

A fast system would help to improve the user experience. This can be achieved by implementing compression and content delivery network technologies to speed up the loading of static assets.

Non-functional Requirement 2 – Reliability of System

A reliable system would help to improve administrator workflow and increase the user experience. This requirement can be met by developing a robust testing suite and using a container-based deployment strategy to help increase reliability.

Non-functional Requirement 3 – Downtime Response Time

A faster downtime response time would result in higher user satisfaction and reduction in administrator frustration. Additionally, having a system with a fast start up time could quickly reduce downtime of the system.

Non-functional Requirement 4 – Security

Due to handling passwords and possibly other sensitive information of users, security can help prevent data breaches and protect the parties involved. Security should be implemented by hashing passwords and having strict get and post definitions of APIs. It is important as well to achieve an HTTPS secure website to give users piece of mind and trust when using Upost.

Non-functional Requirement 5 – User Friendly UI

Having an intuitive web design helps to reduce user frustration and increases user adoption. This could be implemented by using a similar navigation and layout based on familiar high traffic websites such as Netflix and YouTube. Additionally, having an effective and robust event and channel discovery system. Finally, a streamlined process for user website interaction such as registration, login, etc. would help to achieve this requirement.

System Architecture

The existing UPOST web application created in ENSF 619-6 was based on the simple Django MVT architecture as shown in the following figure 3.

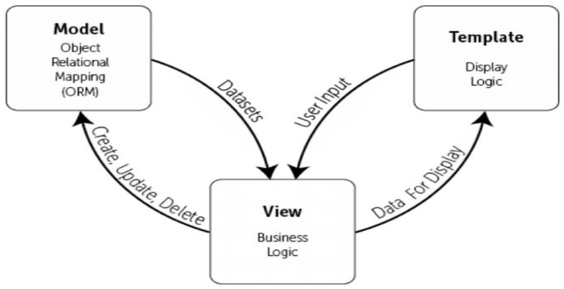
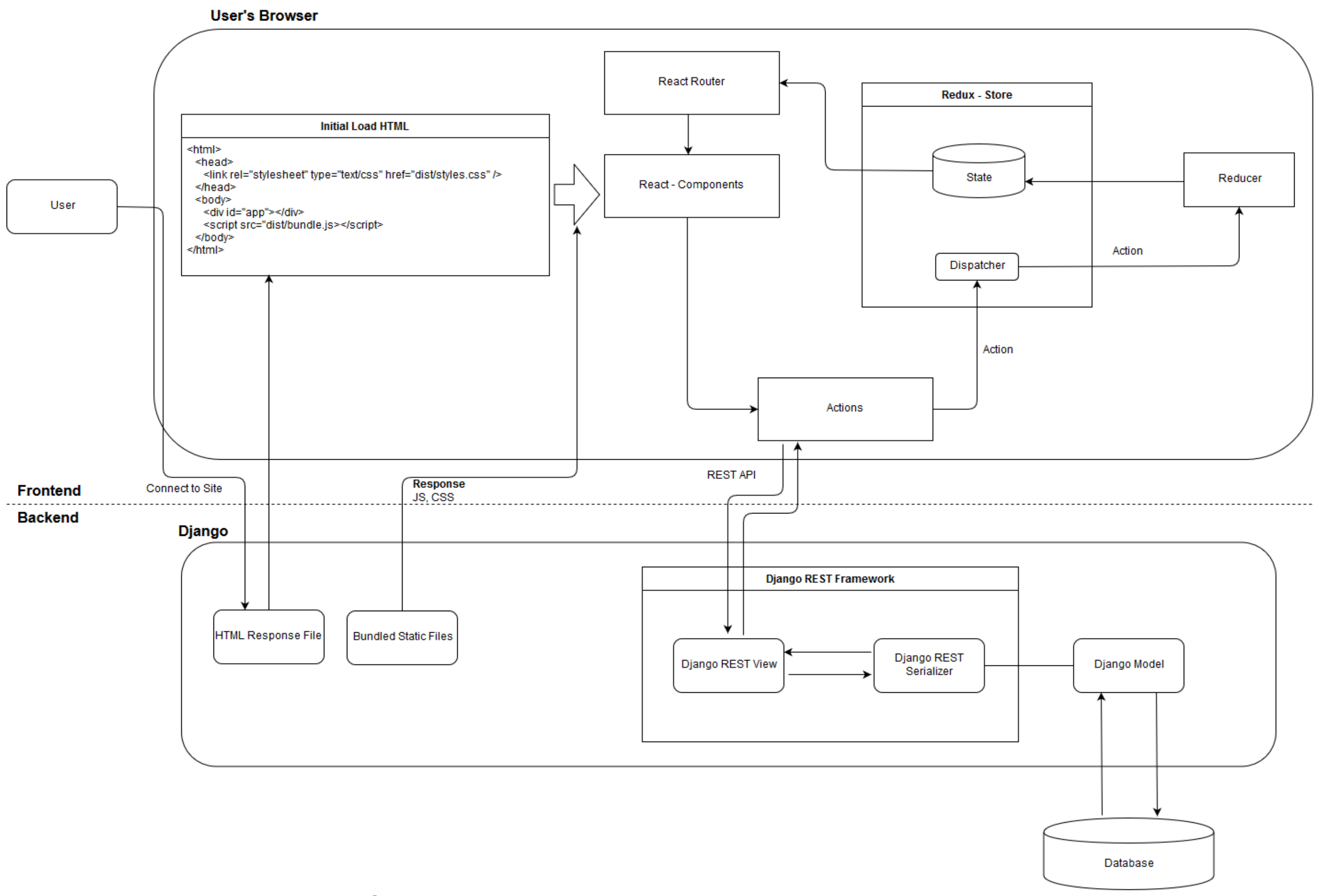


Figure 3 Django MVT architecture

One of the main objectives of the project was to redesign the system architecture to decouple the front and back ends, which could allow for increased flexibility and the ability for different views to connect to the database. The following figure 4 showcases the newly designed architecture of the system:



```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="dist/styles.css" />
  </head>
  <body>
    <div id="app"></div>
    <script src="dist/bundle.js"></script>
  </body>
</html>
```

Figure 4 Current System Architecture

C4 Models

In order to fully describe each level of our system a C4 model approach was used to construct documentation at each different component level.

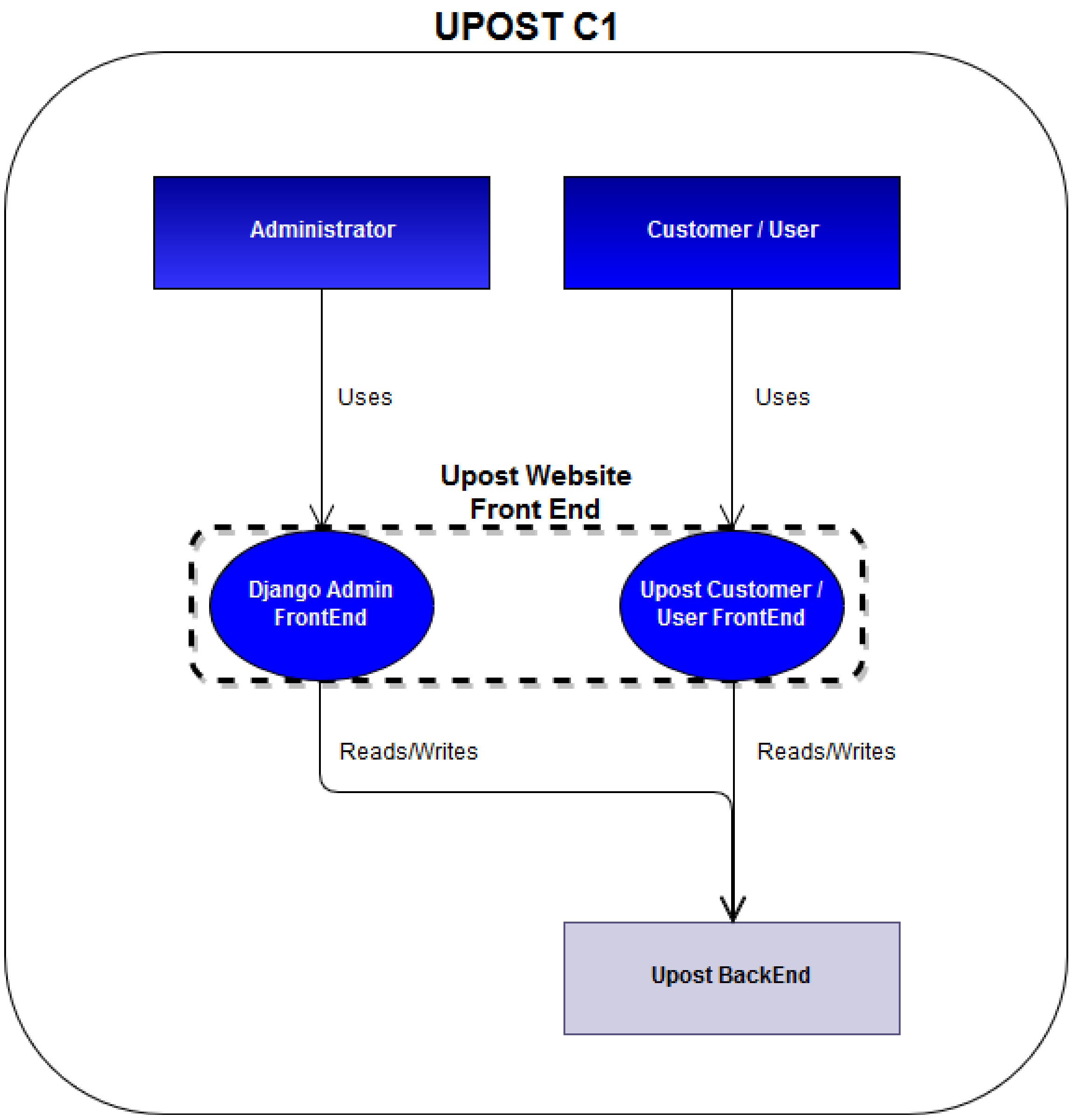


Figure 5 Upost C1

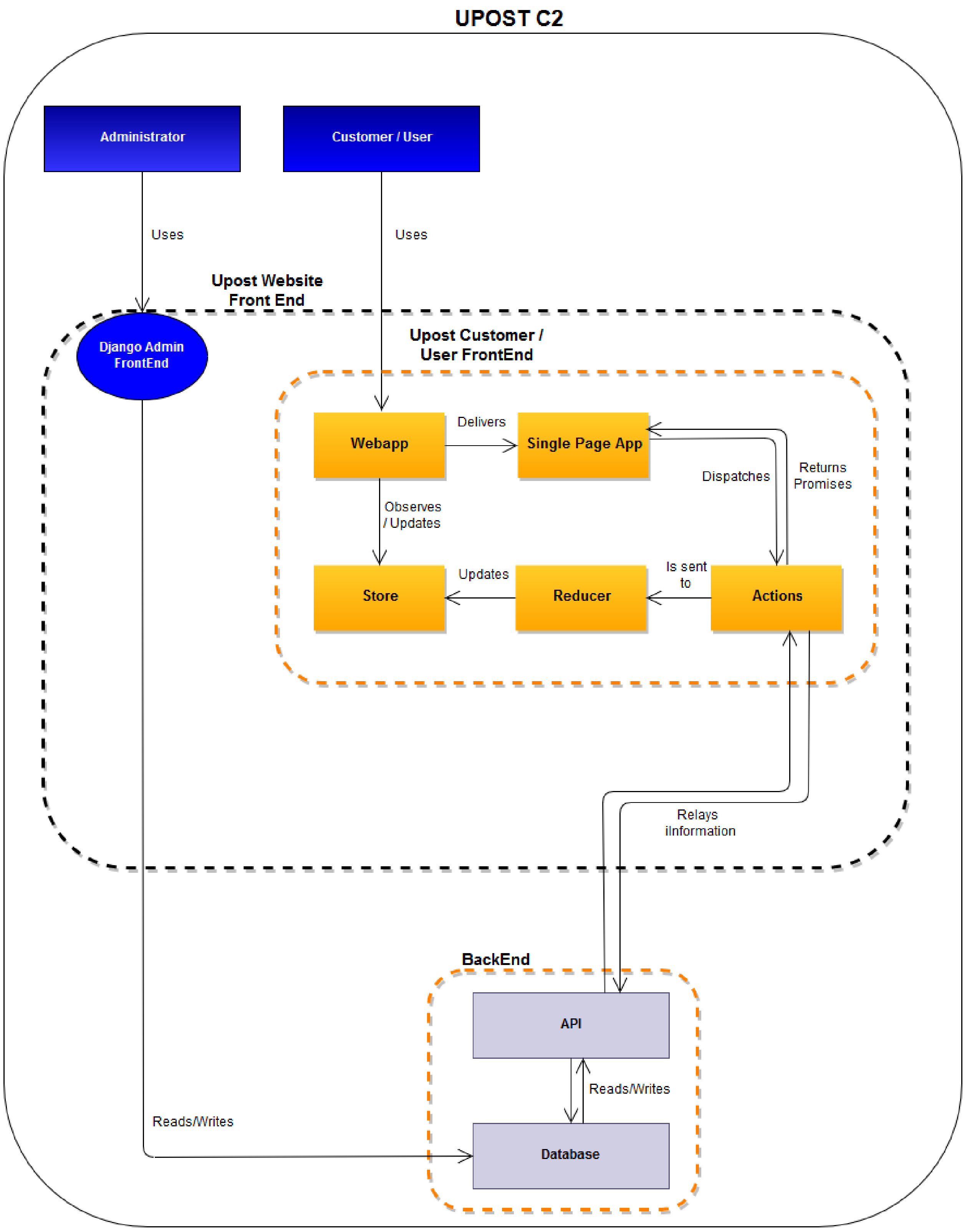


Figure 6 Upost C2 Container Diagram

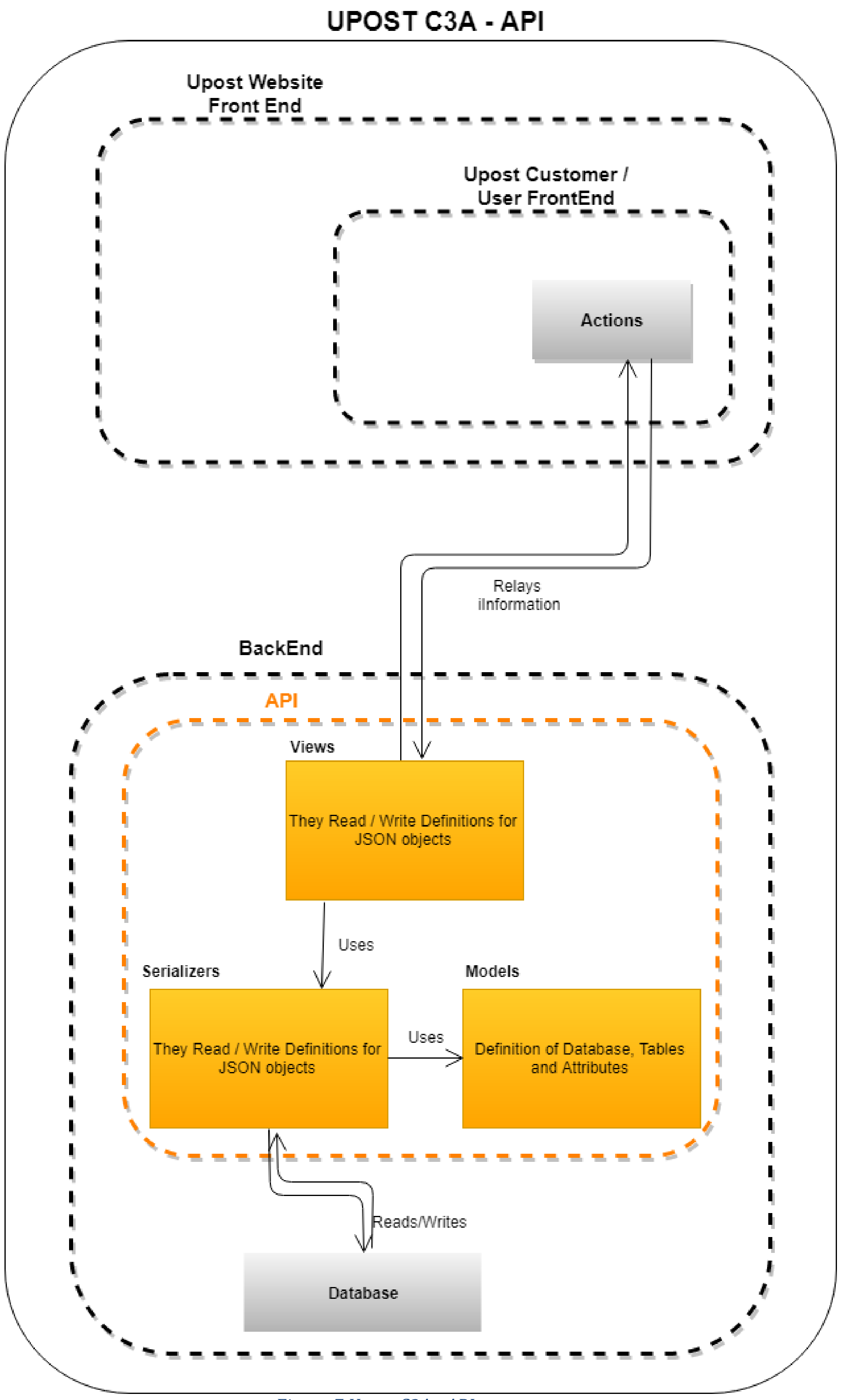

UPOST C3A - API
Upost Website Front End
Upost Customer / User FrontEnd
Actions
Relays iInformation
BackEnd
API
Views
They Read / Write Definitions for JSON objects
Uses
Serializers
They Read / Write Definitions for JSON objects
Uses
Models
Definition of Database, Tables and Attributes
Reads/Writes
Database

Figure 7 Upost C3A - API

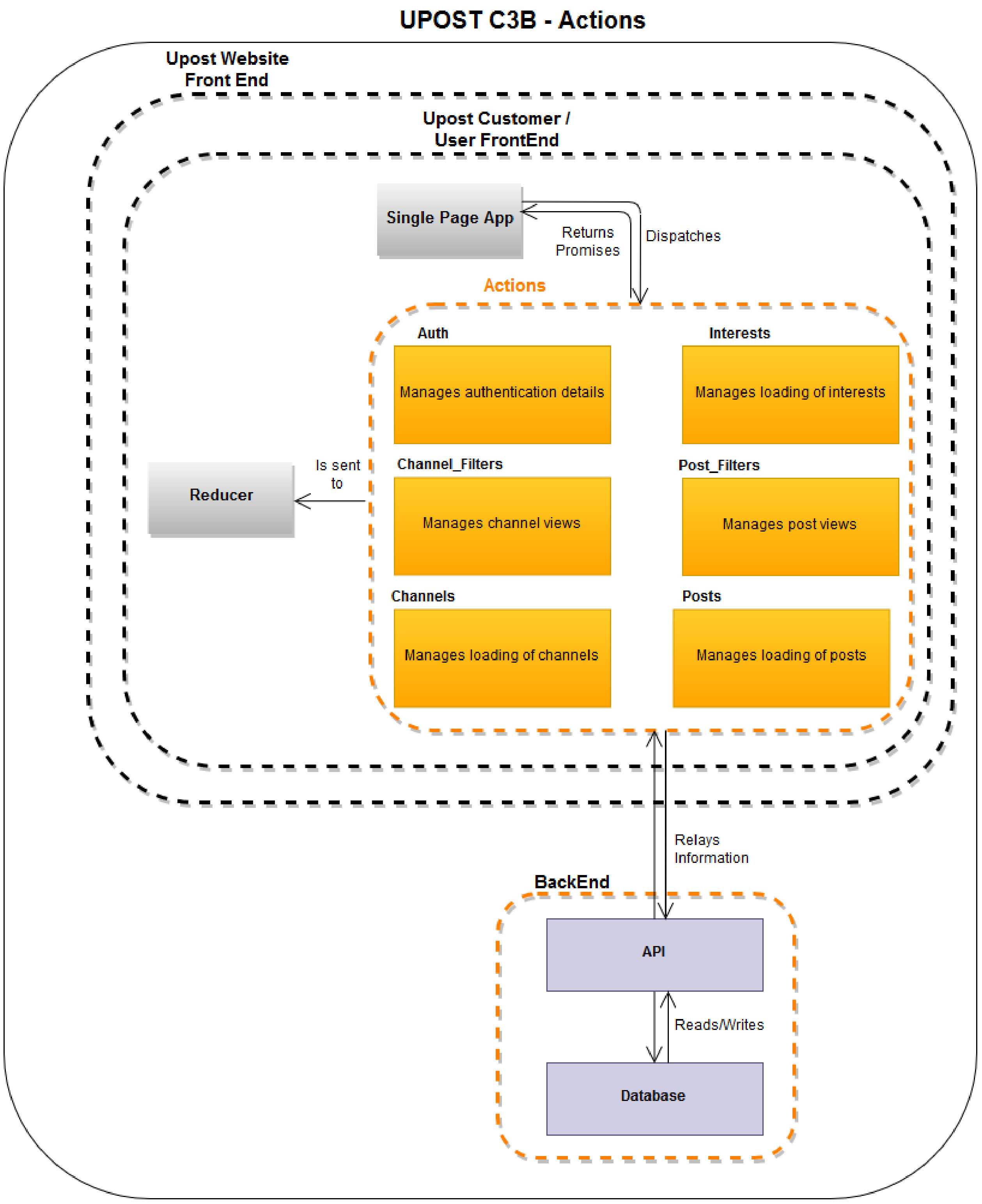


Figure 8 Component Diagram of Frontend Actions

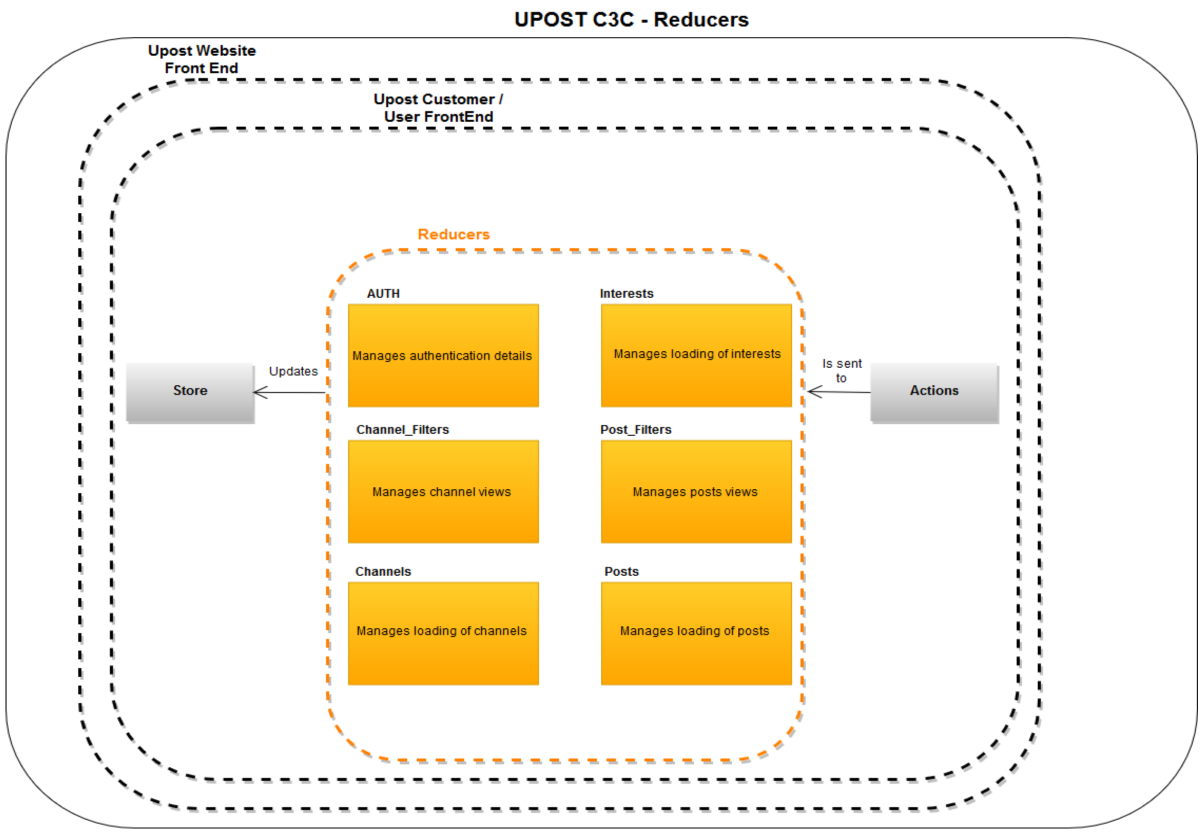


Figure 9 Component Diagram of Frontend Reducers

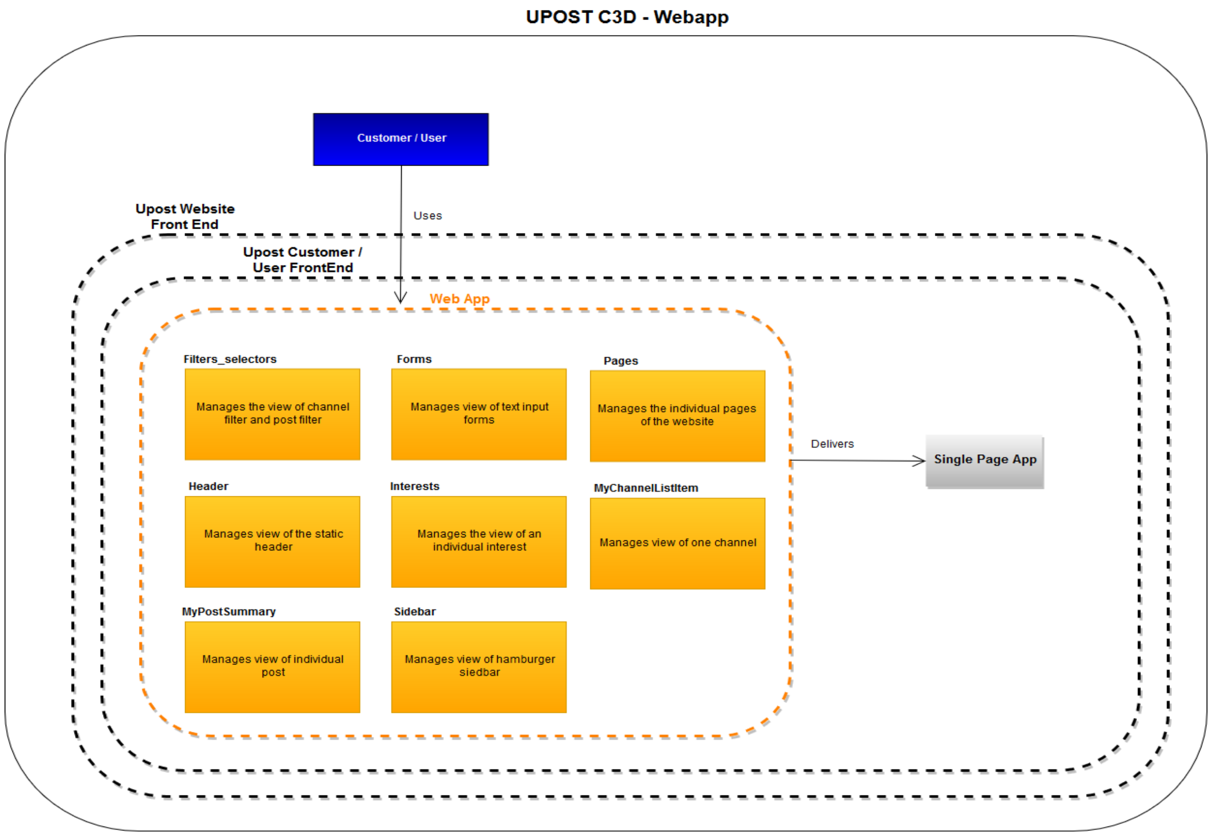

UPOST C3D - Webapp
Customer / User
Uses
Upost Website Front End
Upost Customer / User FrontEnd
Web App
Filters_selectors
Manages the view of channel filter and post filter
Forms
Manages view of text input forms
Pages
Manages the individual pages of the website
Header
Manages view of the static header
Interests
Manages the view of an individual interest
MyChannelListItem
Manages view of one channel
MyPostSummary
Manages view of individual post
Sidebar
Manages view of hamburger siedbar
Delivers
Single Page App

Figure 10 Component Diagram of Frontend Webapp

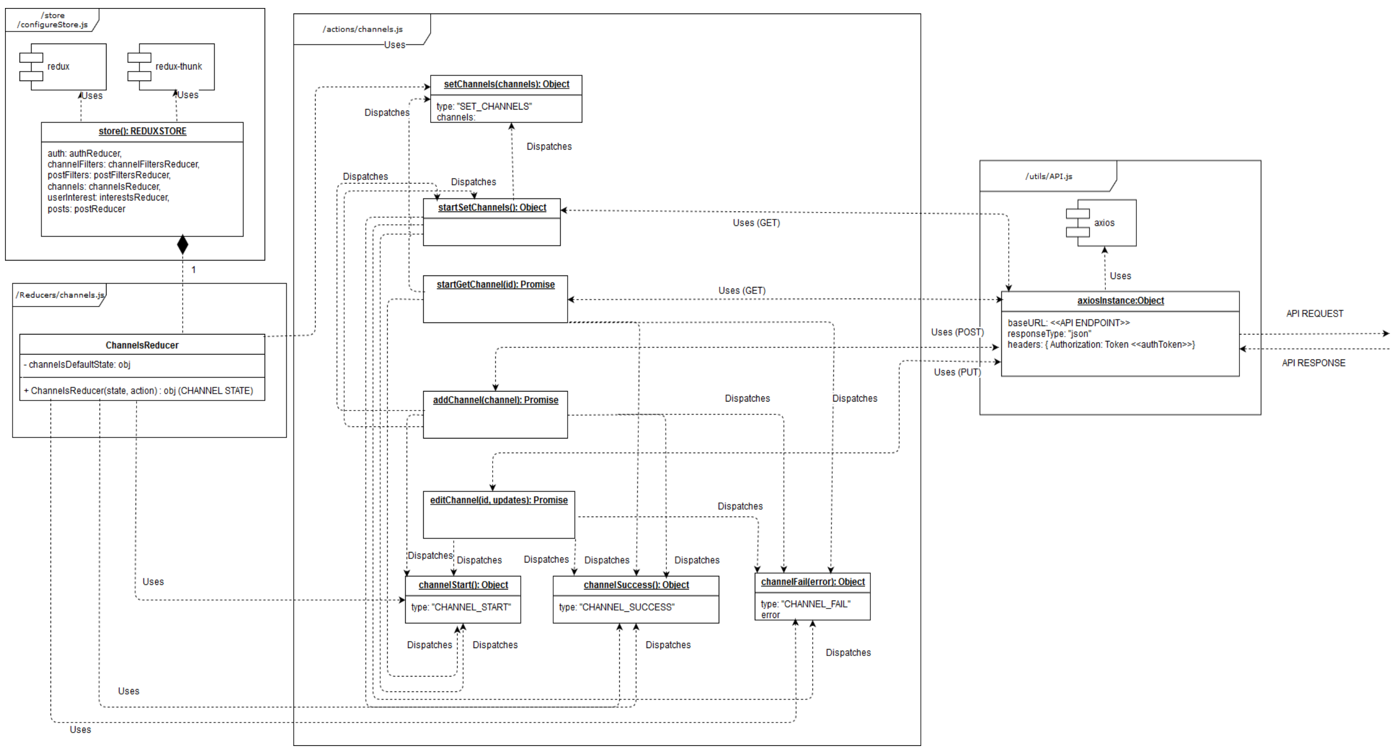

/store /configureStore.js
redux
redux-thunk
Uses
Uses
store(): REDUXSTORE
auth: authReducer,
channelFilters: channelFiltersReducer,
postFilters: postFiltersReducer,
channels: channelsReducer,
userInterest: interestsReducer,
posts: postReducer
1
/Reducers/channels.js
ChannelsReducer
- channelsDefaultState: obj
+ ChannelsReducer(state, action) : obj (CHANNEL STATE)
/actions/channels.js
Uses
setChannels(channels): Object
type: "SET_CHANNELS"
channels:
Dispatches
startSetChannels(): Object
startGetChannel(id): Promise
addChannel(channel): Promise
editChannel(id, updates): Promise
channelStart(): Object
type: "CHANNEL_START"
channelSuccess(): Object
type: "CHANNEL_SUCCESS"
channelFail(error): Object
type: "CHANNEL_FAIL"
error
Uses (GET)
Uses (POST)
Uses (PUT)
/utils/API.js
axios
axiosInstance:Object
baseURL: <<API ENDPOINT>>
responseType: "json"
headers: { Authorization: Token <<authToken>>}
API REQUEST
API RESPONSE

Figure 11 C4 Code Diagram of Channel Actions, Reducers and Store

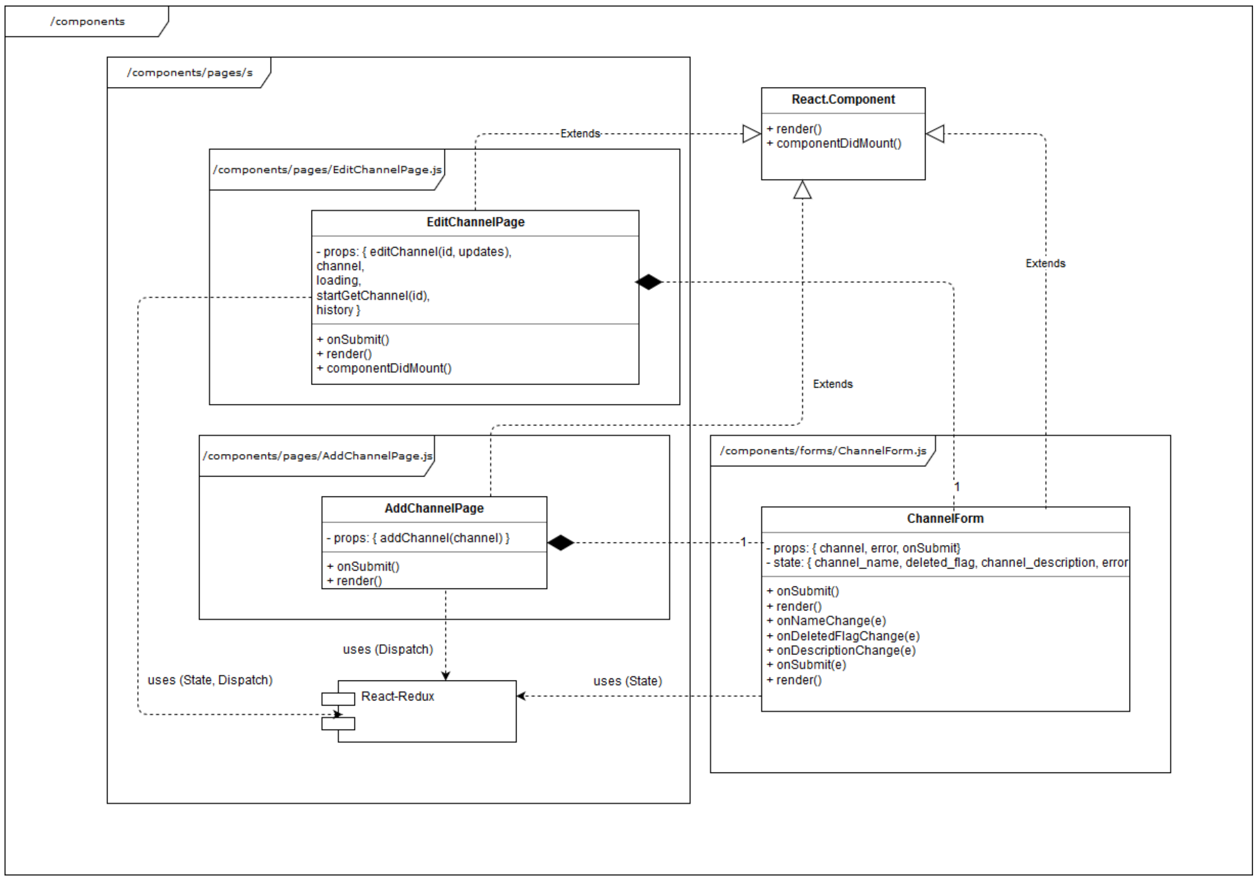


Figure 12 C4 Code Diagram of Channel Components

Prototyping

Prototyping of the U-post system was based on a two-step process prototyping. First an initial pre-totype was developed sketched by hand. This ensured that quick fixes could get done, that each screen had no missing UI elements and an updated interaction was achieved from our previous U-post deployment. The second step was developing a high fidelity prototype using Invision an interactive prototyping platform.

Due to the agile nature of the project a high-fidelity prototype was used as the team's Northstar. Allowing us to compare where we were currently at, versus were we were supposed to be heading without convoluted diagrams. This interactive prototype allowed a common vision for the team in terms of design, types of screen, user interactions, placement of UI elements, etc. which are hard to describe in developing diagrams. The high fidelity prototyping also allows for a learning curve on desired UI elements and interactions, against our current knowledge of REACT. Allowing us to create an effective backlog with Stories (to-do items) for a continuous progress.

The interactive prototypes can also be simulated on an actual device and tested for user likeability; which is one of our main concerns with U-post. The prototypes used and viewed from the following link:

https://projects.invisionapp.com/prototype/UPost-cjw40ag900057sp019lad3k4o

The following are screenshots of the high fidelity prototype screens they are a complete redesign from what the previous Upost website developed solely in Django (Appendix A) looked before. The new design enhanced a better UI interaction and usability, which are key for the success of the Upost Website.

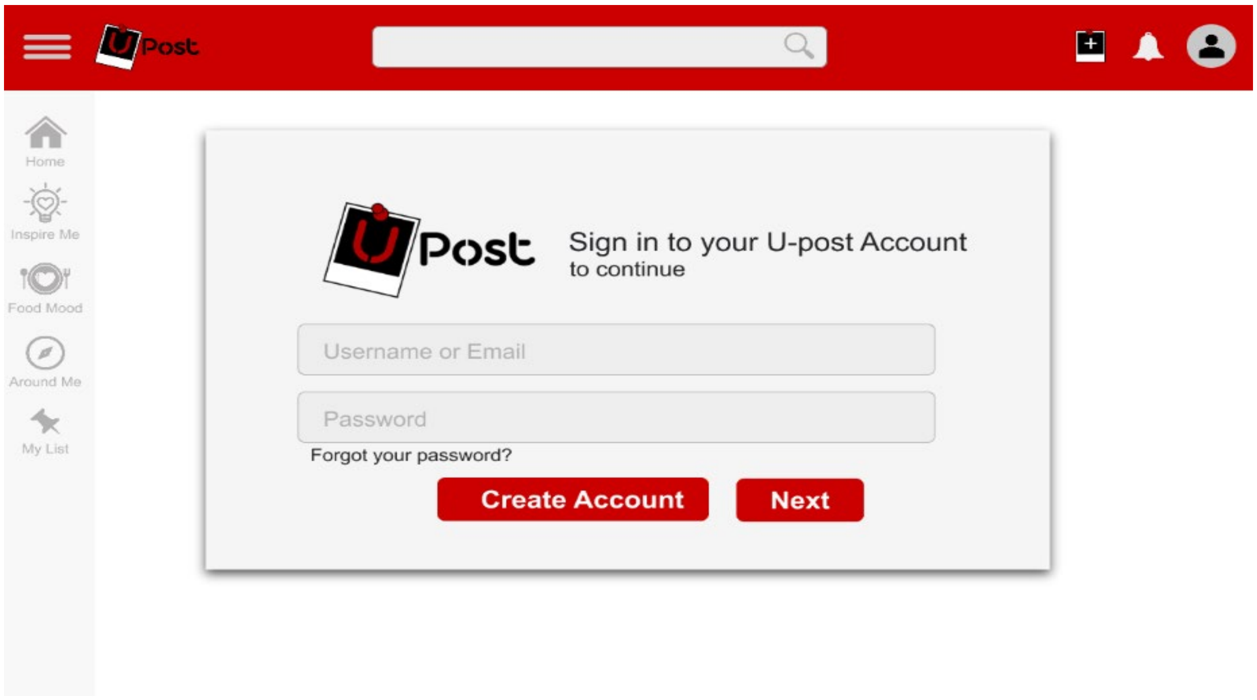
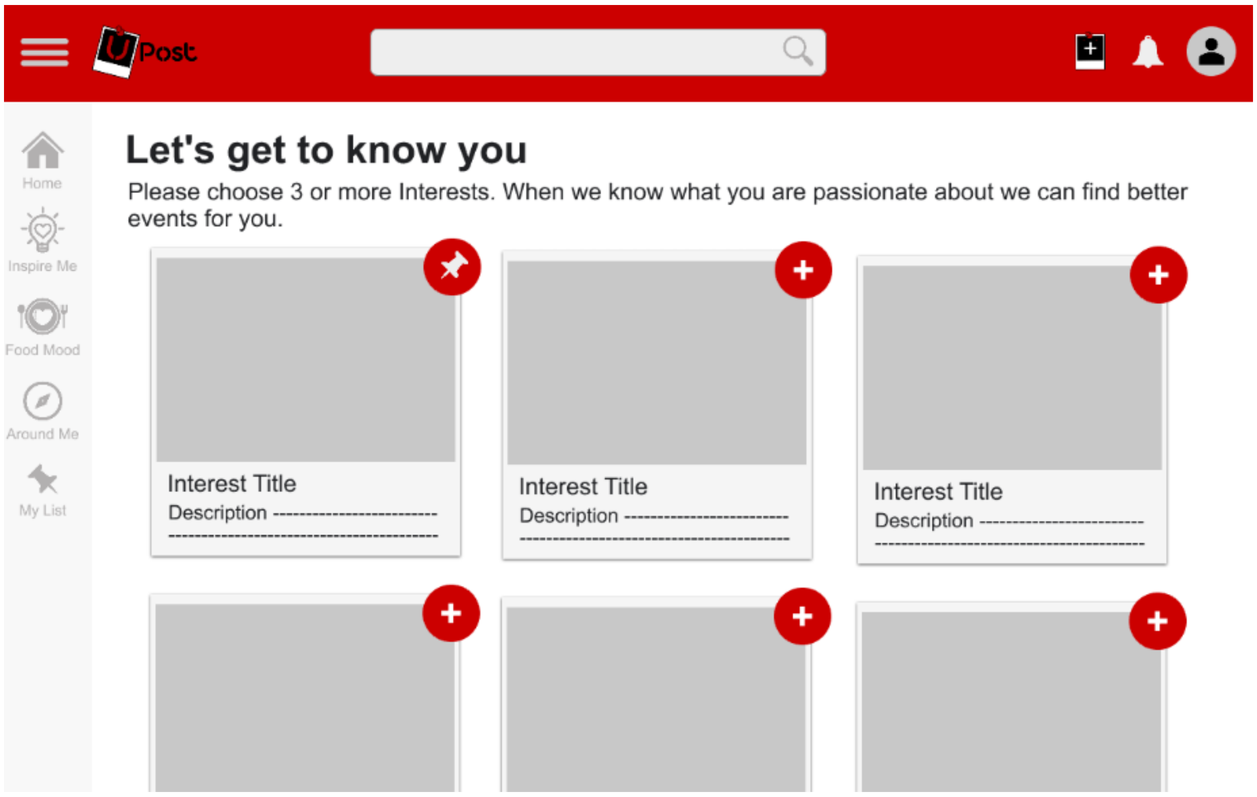


Figure 13 Prototype Login Page



Let's get to know you

Please choose 3 or more Interests. When we know what you are passionate about we can find better events for you.

Figure 14 Prototype Interest Page

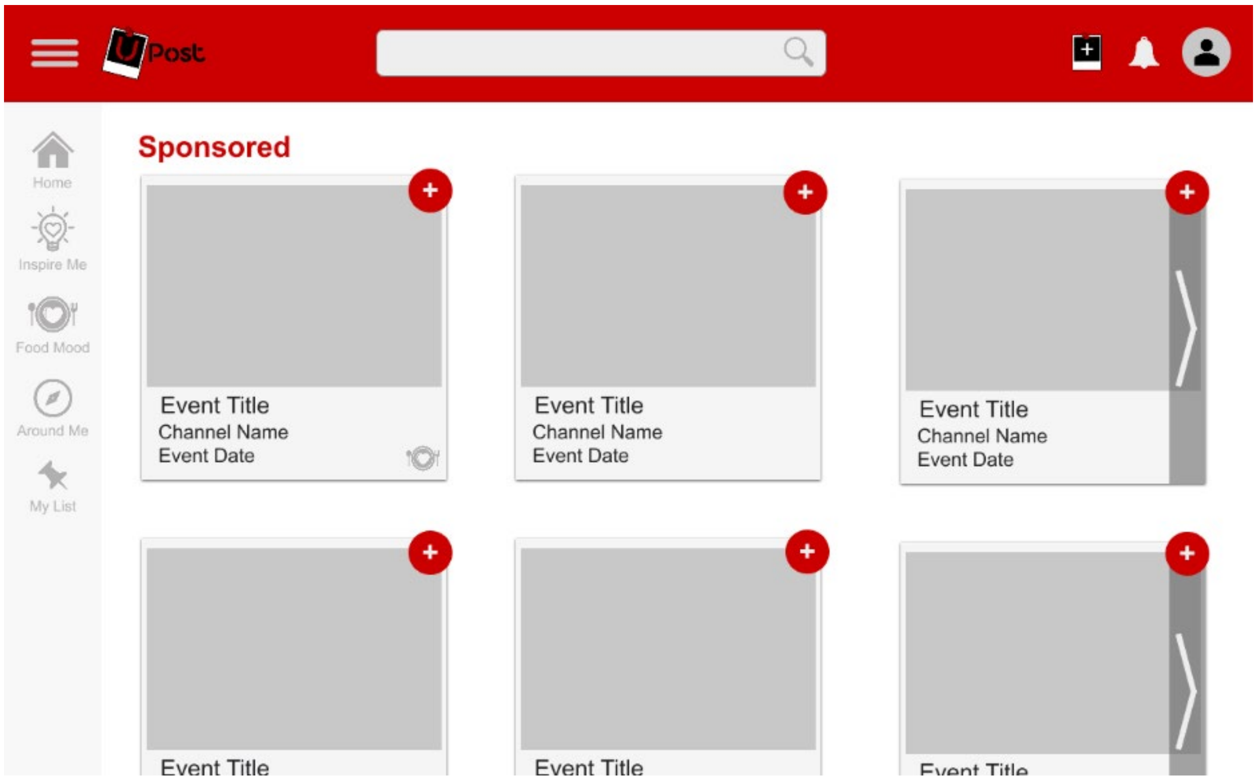


Figure 15 Prototype Home Page

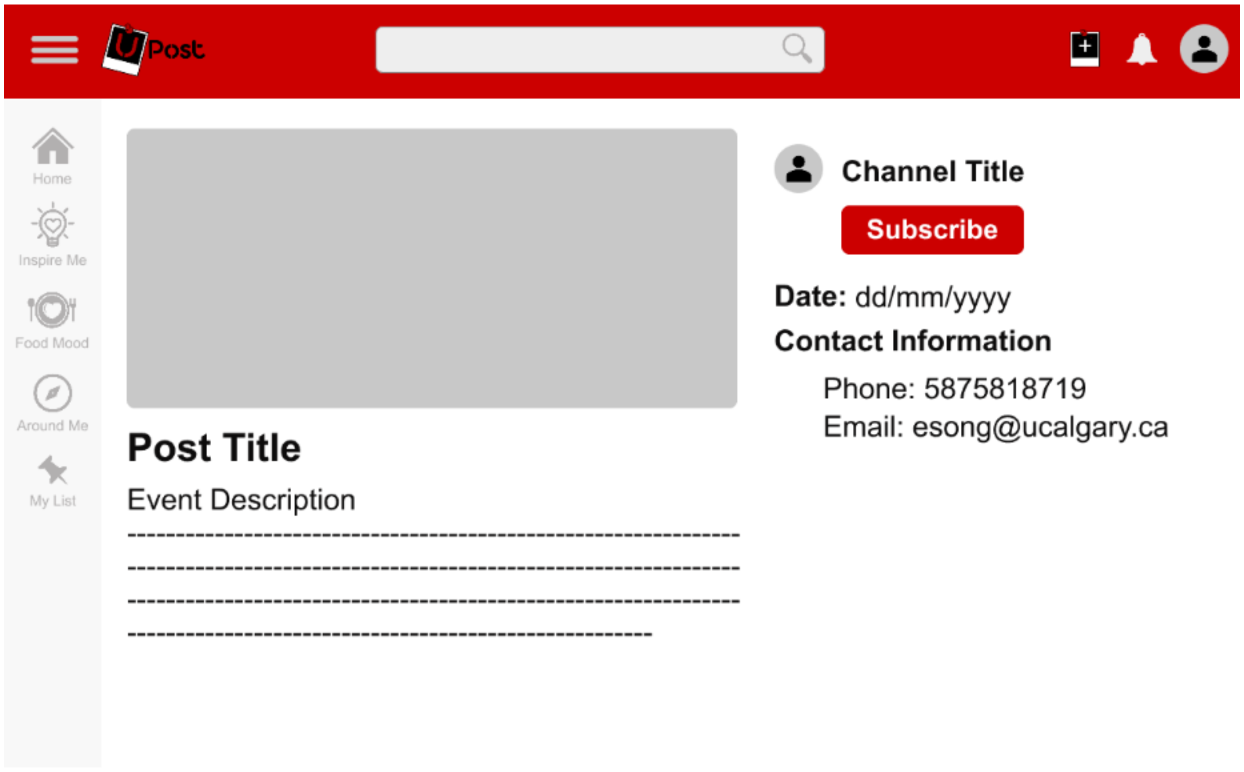


Figure 16 Prototype Post Page

Project Development

Learning Frameworks

Having no prior exposure to React.js or Django REST Framework, the first three weeks of the project were spent on learning the two frameworks. A sample application using React and Firebase was built and deployed on Heroku (See https://wc-expensify.herokuapp.com/) using a Udemy tutorial (https://www.udemy.com/react-2nd-edition/) to build the skills necessary to build our own application. From the tutorials we were able to construct a boilerplate for our own application. The tutorials provided exposure to the core functionality of React, including:

1. Components
    a. ES6 classes
    b. Stateless functional components
    c. Third party components
2. Webpack configuration
    a. Babel
    b. Source maps
3. Styling
    a. SCSS
4. Routing
    a. Server vs client routing

    b. React router
    c. HTML5 history API
5. Redux
    a. Actions
    b. Reducers
6. Testing
7. Deployment
    a. Development vs production
8. Interaction with Database backends (Firebase)
    a. Promises
    b. Updating, and fetching data
    c. Asynchronous redux actions

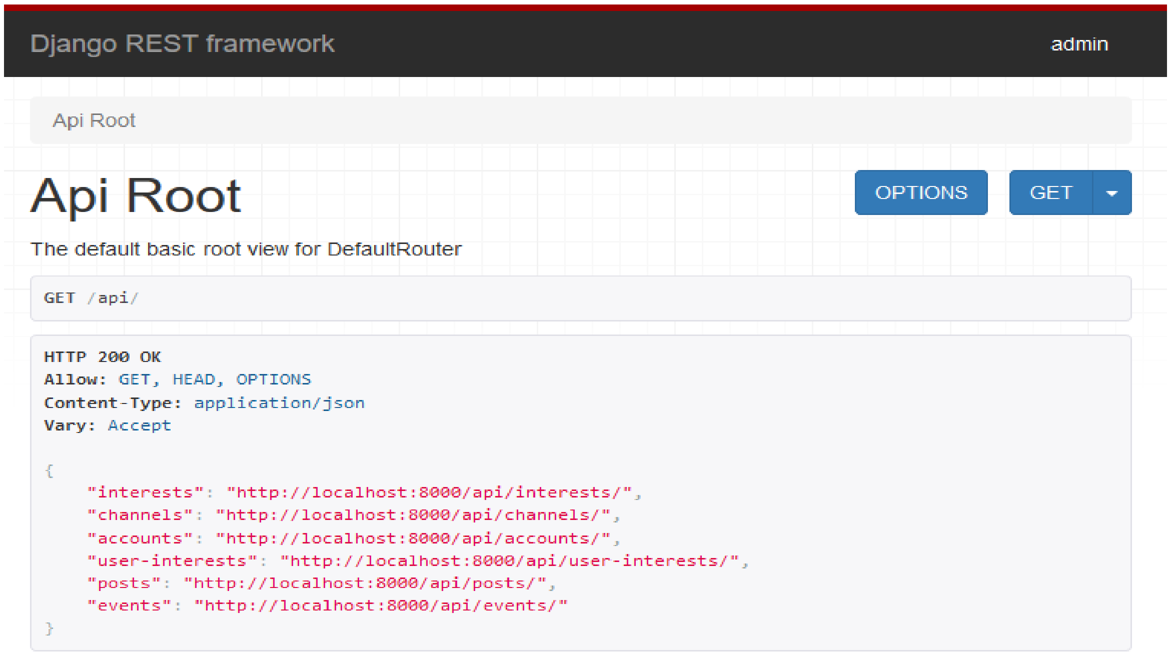
All of which were used and applied to our current progress of the Upost project.

Upost Feature Development

The following features and components of the UPost web application have been developed:

1. Login (with token authorization)
2. Registration
3. My Channels (add, edit)
4. My Posts (add, edit)
5. My Interests (add, edit)

For the features built in the backend, the API endpoints were configured with the associated views and serializers. All view sets constructed were assigned appropriate permission classes. And an updated version of the backend models were built enhancing the models in ENSF 619-6. The following are screenshots of examples of the achieved progress:



Api Root

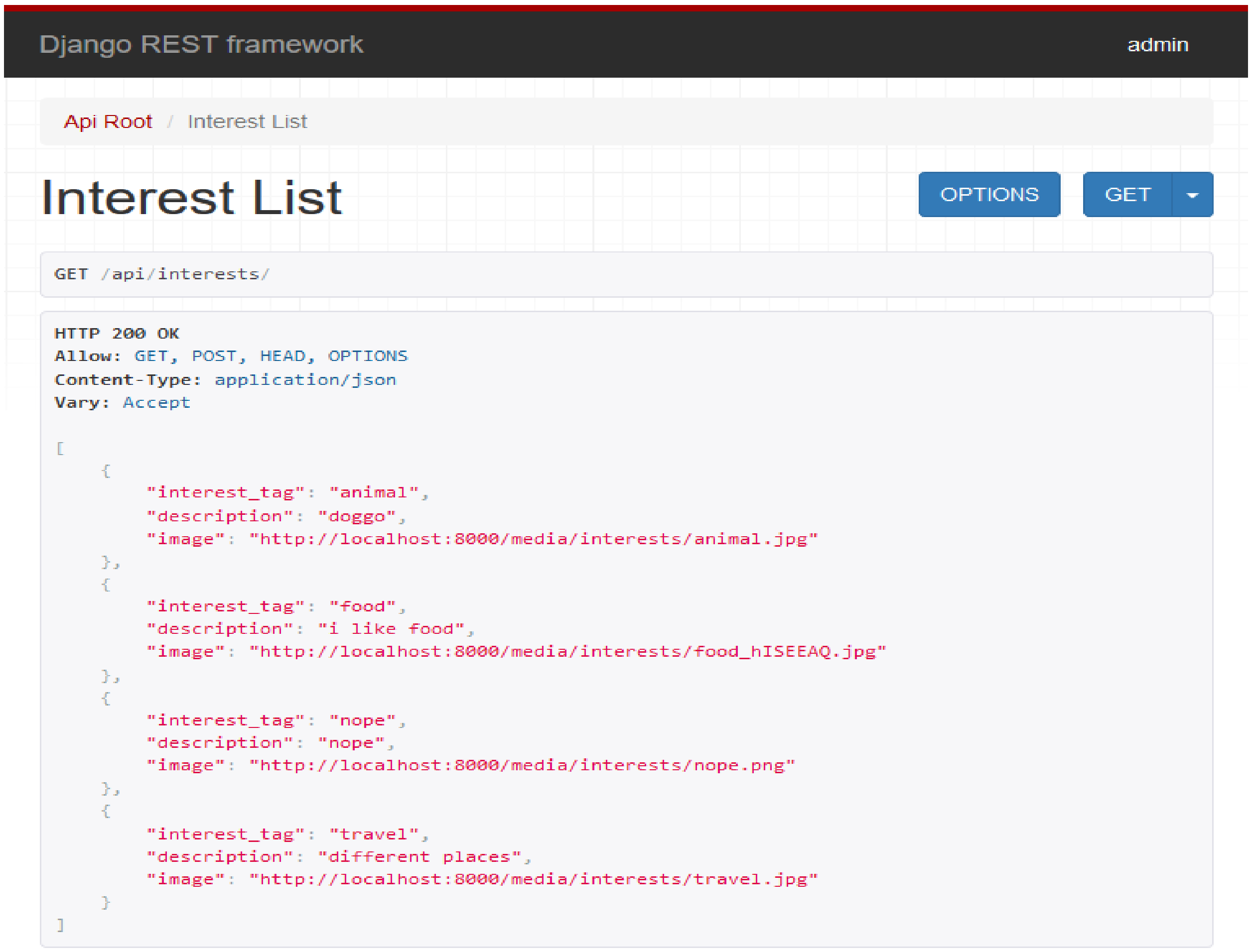
The default basic root view for DefaultRouter

```
GET /api/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "interests": "http://localhost:8000/api/interests/",
    "channels": "http://localhost:8000/api/channels/",
    "accounts": "http://localhost:8000/api/accounts/",
    "user-interests": "http://localhost:8000/api/user-interests/",
    "posts": "http://localhost:8000/api/posts/",
    "events": "http://localhost:8000/api/events/"
}
```

Figure 17 Django REST framework API root



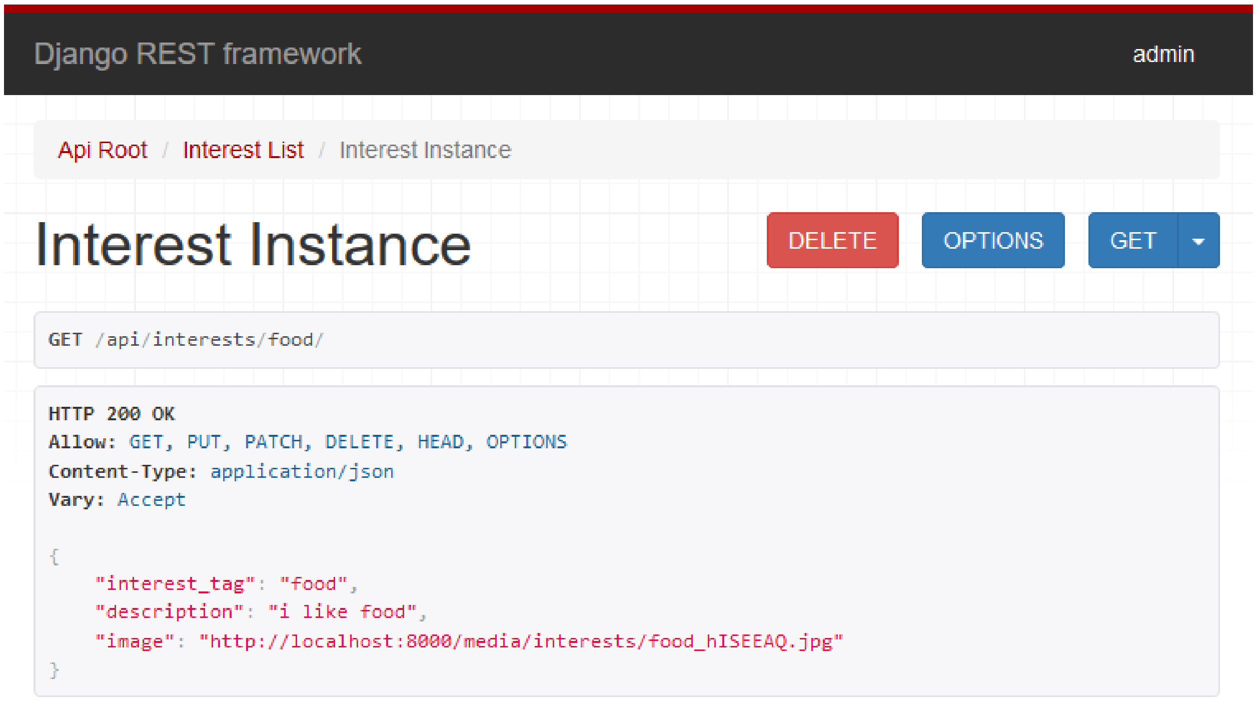
Interest List

```
GET /api/interests/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "interest_tag": "animal",
        "description": "doggo",
        "image": "http://localhost:8000/media/interests/animal.jpg"
    },
    {
        "interest_tag": "food",
        "description": "i like food",
        "image": "http://localhost:8000/media/interests/food_hISEEAQ.jpg"
    },
    {
        "interest_tag": "nope",
        "description": "nope",
        "image": "http://localhost:8000/media/interests/nope.png"
    },
    {
        "interest_tag": "travel",
        "description": "different places",
        "image": "http://localhost:8000/media/interests/travel.jpg"
    }
]
```

Figure 18 Interest List API endpoint



Interest Instance

```
GET /api/interests/food/
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "interest_tag": "food",
    "description": "i like food",
    "image": "http://localhost:8000/media/interests/food_hISEEAQ.jpg"
}
```

Figure 19 Food Interest Instance API endpoint

For the features built in to the frontend, all pages and forms were built with reusable React components and assigned a public or private route, with appropriate actions and reducers setup to make API calls and manage local component states and the Redux store.

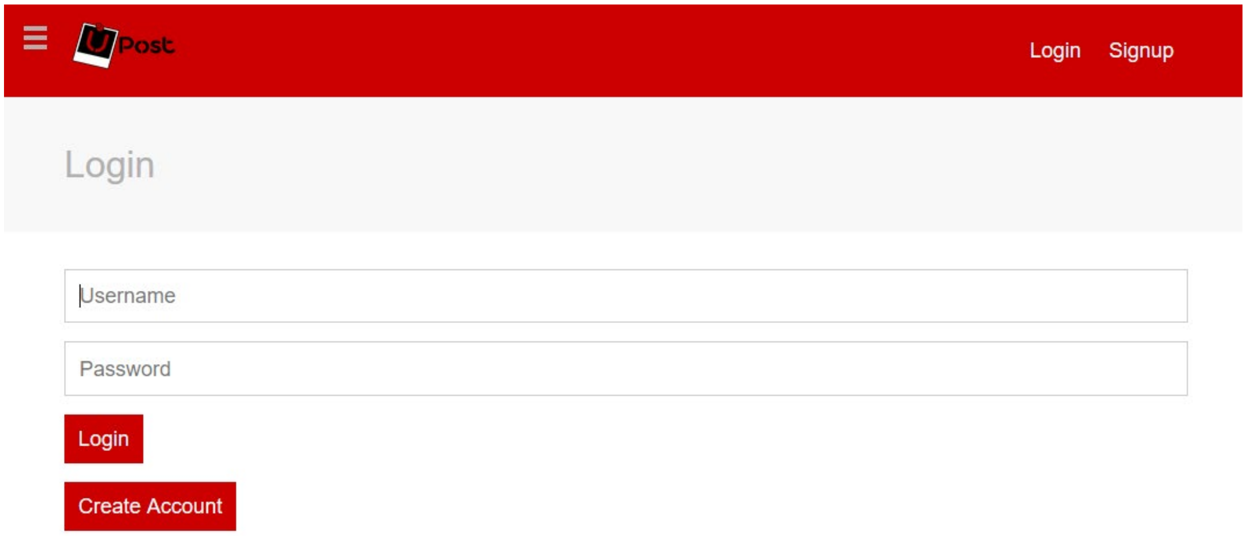
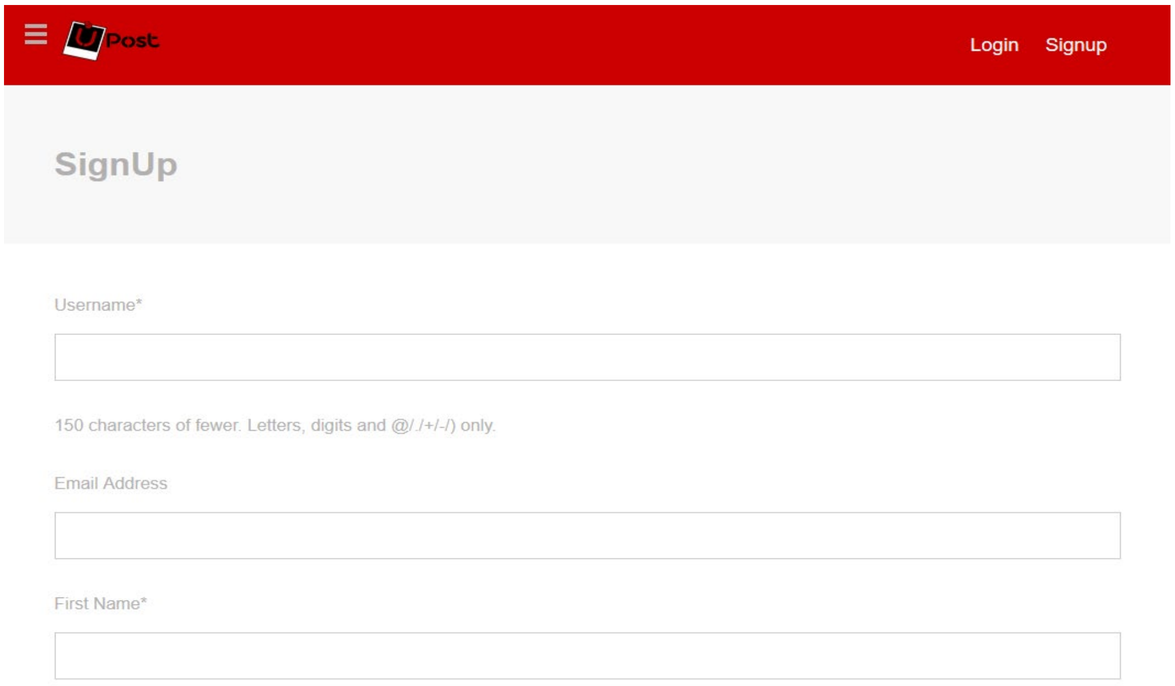
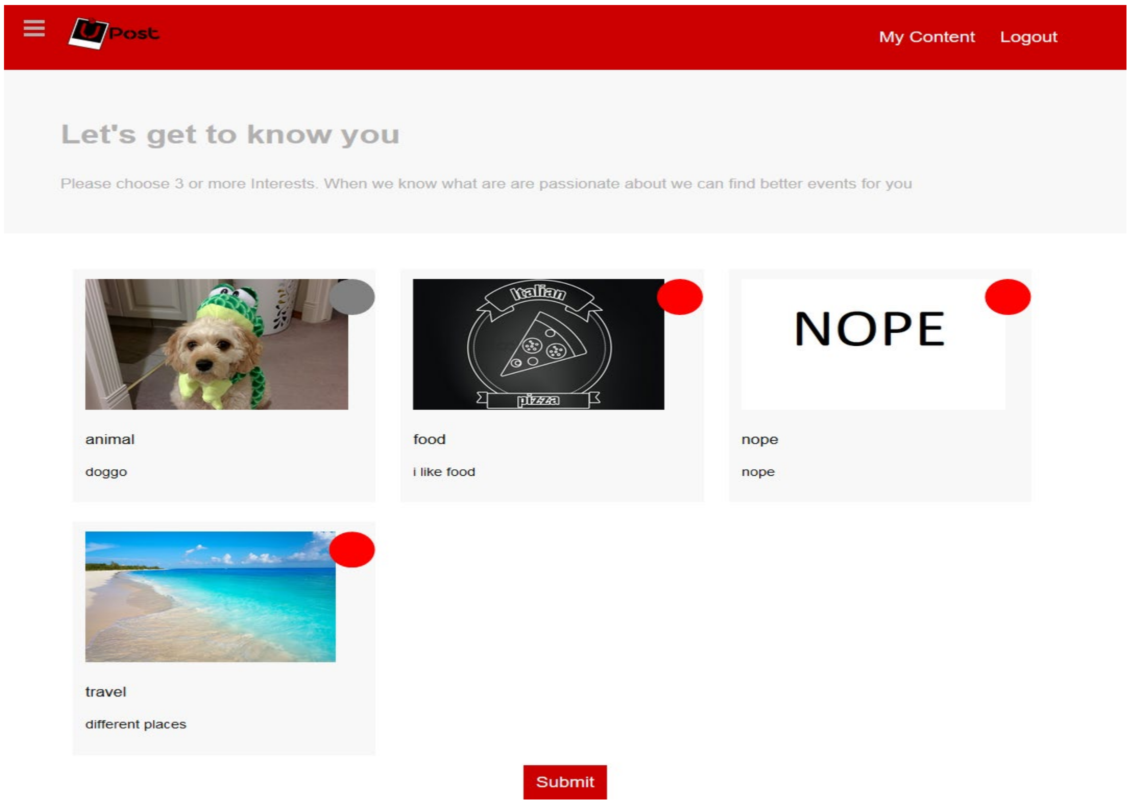


Figure 20 Deployed Login Page



150 characters of fewer. Letters, digits and @/./+/-/) only.

Figure 21 Deployed Signup Page



Let's get to know you

Please choose 3 or more Interests. When we know what are are passionate about we can find better events for you

animal

doggo

food

i like food

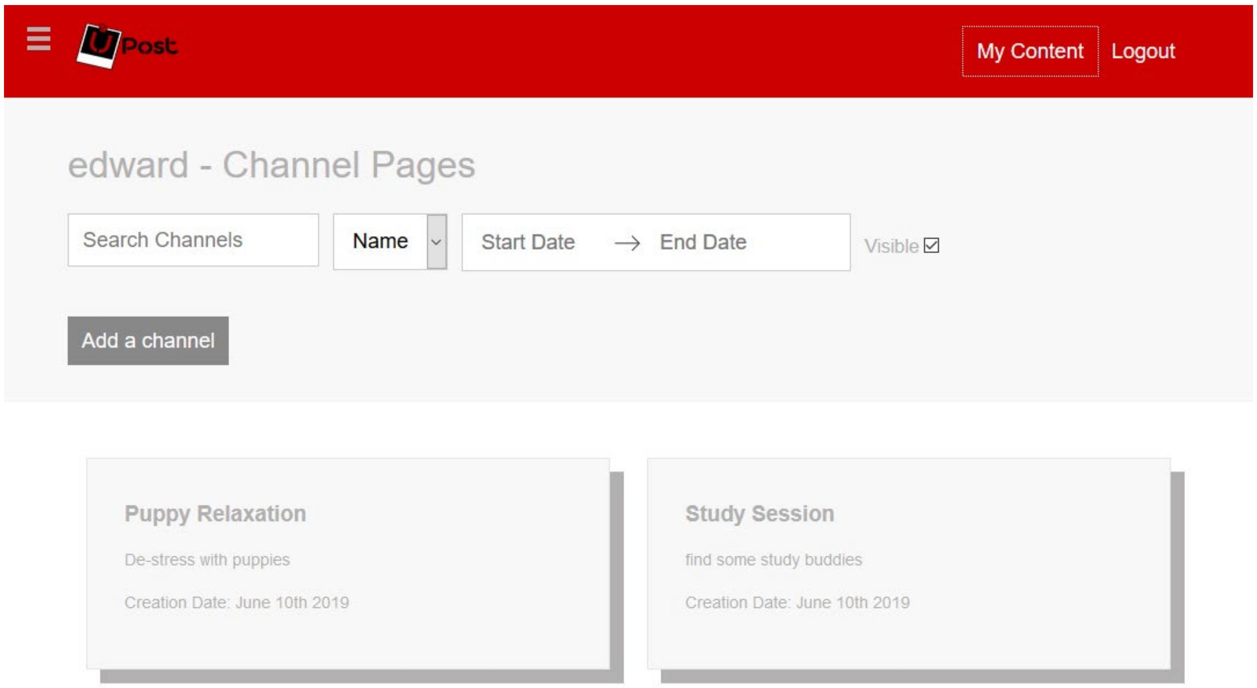
nope

nope

travel

different places

Figure 22 Deployed Interest Selection Page



edward - Channel Pages

Puppy Relaxation

De-stress with puppies

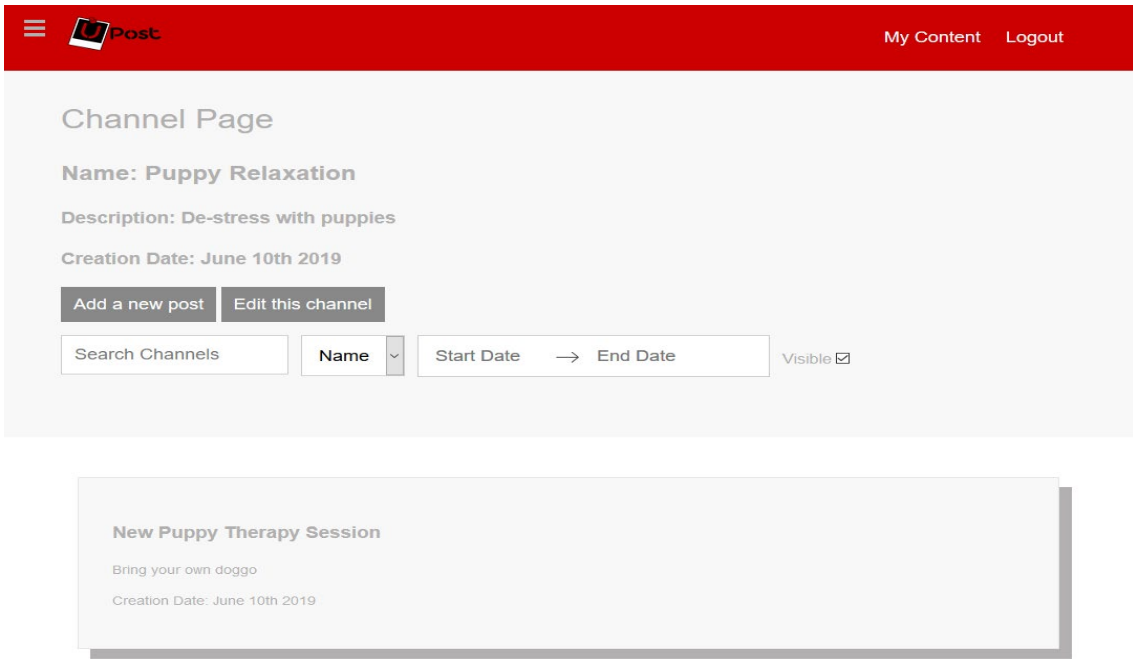
Creation Date: June 10th 2019

Study Session

find some study buddies

Creation Date: June 10th 2019

Figure 23 Deployed Channels Page


Post

My Content Logout

Channel Page

Name: Puppy Relaxation

Description: De-stress with puppies

Creation Date: June 10th 2019

Add a new post | Edit this channel

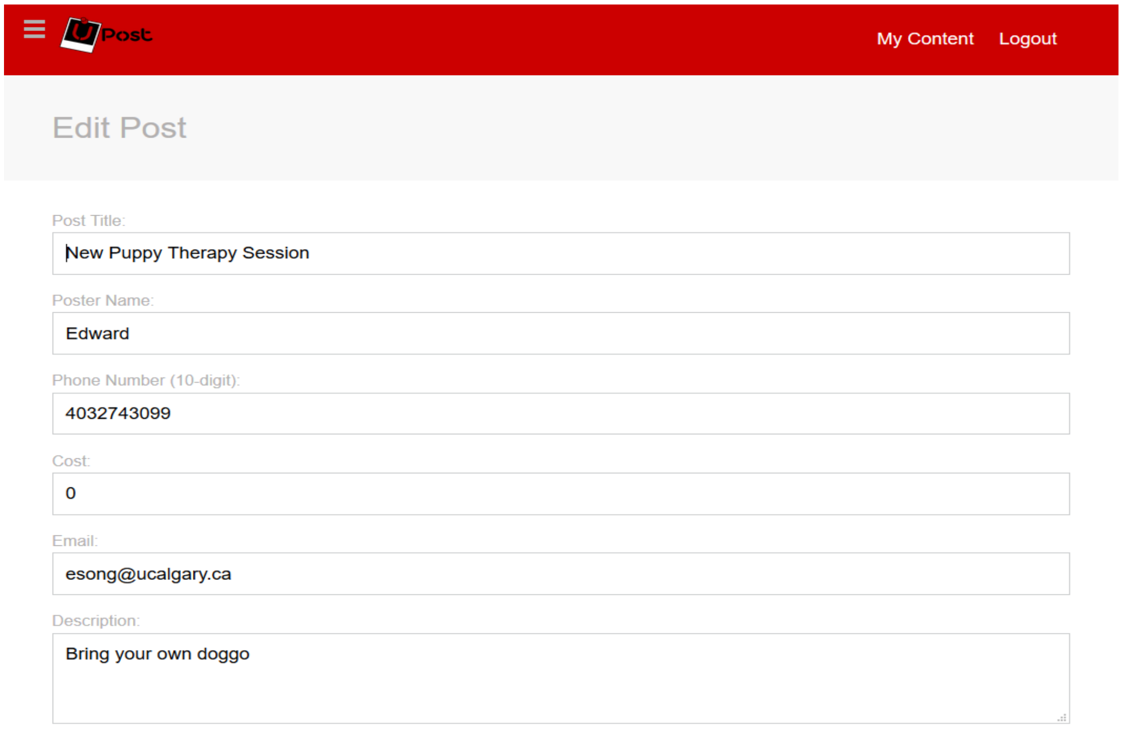
Search Channels | Name | Start Date → End Date | Visible ☑

New Puppy Therapy Session

Bring your own doggo

Creation Date: June 10th 2019

Figure 24 Deployed Channel Management Page


Post

My Content Logout

Edit Post

Post Title:
New Puppy Therapy Session

Poster Name:
Edward

Phone Number (10-digit):
4032743099

Cost:
0

Email:
esong@ucalgary.ca

Description:
Bring your own doggo

Figure 25 Deployed Edit Post Page

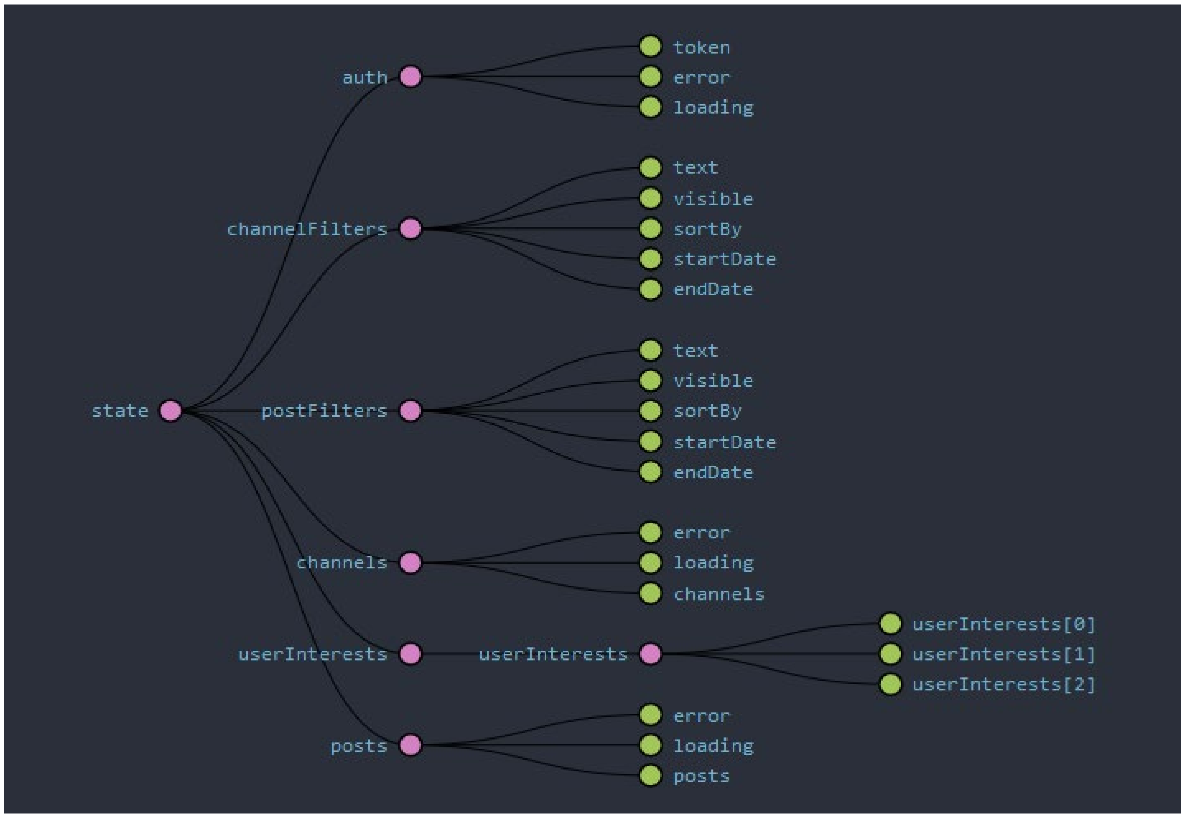


Figure 26 Developed Redux State Shown by Redux DevTools Addon

Upost Deployment

Amazon Web Services (AWS) was used for an initial deployment to host both the backend and database. The following steps were followed for deployment:

1. Development of a frontend app in Django to deliver static assets (javascript, css, images)
    a. Options to consider include: using Apache HTTP server, NGINX, or WhiteNoise
    b. WhiteNoise was ultimately selected due to ease of setup and compatibility with a Content Delivery Network (CDN)
2. Configuration of webpack to bundle our React and Styling components to javascript and css files within the Django frontend app directory
3. Configuration of a Python Web Server Gateway Interface (WSGI) HTTP server for local deployment
    a. Options to consider included: Apache HTTP server and mod_wsgi, Gunicorn and NGINX, or Waitress
    b. Waitress was selected due ease in setup compared to Apache and mod_wsgi and compatibility with Windows based computers over Gunicorn and NGINX which only support Unix based operating systems
4. Deployment of web app to AWS Elastic Beanstalk
5. Creation and configuration of AWS Relational Database Service (RDS) for a MySQL database

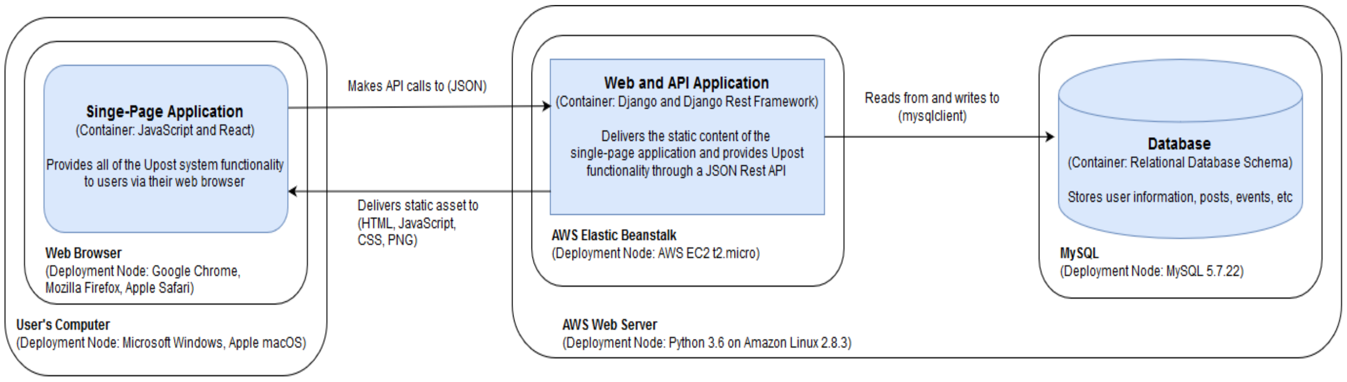

Singe-Page Application
(Container: JavaScript and React)
Provides all of the Upost system functionality to users via their web browser
Web Browser
(Deployment Node: Google Chrome, Mozilla Firefox, Apple Safari)
User's Computer
(Deployment Node: Microsoft Windows, Apple macOS)
Makes API calls to (JSON)
Delivers static asset to (HTML, JavaScript, CSS, PNG)
Web and API Application
(Container: Django and Django Rest Framework)
Delivers the static content of the single-page application and provides Upost functionality through a JSON Rest API
AWS Elastic Beanstalk
(Deployment Node: AWS EC2 t2.micro)
AWS Web Server
(Deployment Node: Python 3.6 on Amazon Linux 2.8.3)
Reads from and writes to (mysqlclient)
Database
(Container: Relational Database Schema)
Stores user information, posts, events, etc
MySQL
(Deployment Node: MySQL 5.7.22)

Figure 27 Upost Deployment Diagram

From our currently deployment steps above a resultant architecture shown in figure 27 was obtained. Once a connection is made from the user's web browser to the backend web application, static assets used to form the single-page applications are sent to the user's computer. Based on user interaction, the single-page application communicates with the backend API application with API calls. The API application then communicates with the database to read and write information. The current deployment of our web app can currently be viewed using the following link:

http://upost-env.z9ame8dp78.us-west-2.elasticbeanstalk.com/

Testing

An automated testing suite was developed for our current implementation of our project. This includes testing of actions, reducers, and reusable components (ie. header, sidebar, etc). It should be noted that we have implemented testing only on the client side, with results from API calls being mocked using Jest to prevent unnecessary API queries every time the test suite is run.

Testing of actions ensures that the correct actions objects are dispatched when the methods are called, and that chained actions involving API calls are being dispatched correctly and are returning the desired results.

Testing of reducers ensures that the state changes accordingly when a specific action is dispatched from the Redux store.

Page testing involves a mix of snapshot testing, which will alert the user if the rendering of any of the components changes over time, and component function testing using mock functions to test the correct functionality of buttons. Testing of redirects is also performed to see if successful submission of child forms performs the desired results.

Form testing involves snapshot testing and the testing of local state changes to ensure that form input changes correctly manipulate the local state of the component.

To date, a total of 34 test suites have been created, with a total of 132 tests written. The implementation of these tests should help with future debugging when new features are added.

Our team took an Agile approach to the development of the Upost Website. Due to the lack o knowledge of all the tasks (stories) needed to complete to finalize the project a Burn up chart was used. This type of chart is used to progressively track the status of each story needed for the project and the appearance of new discovered stories along the way. The following terminology is essential to understand the Burnup Chart:

Story: Identified tasks to be accomplished. All stories take around the same effort and time but in some cases some stories take longer than others causing a lag between the backlog and the Done status.
Backlog: Collection of known + discovered Stories needed to be completed to keep advancing the project.
In Progress: Collection of tasks that after a sprint are still in progress and have not been completed. In our case learning tasks took more than just one sprint.
Merging/Debugging: Completed stories devoted to fix something broken after new code was introduced. Completed stories also devoted to git hub merges during development that require a lot of revision time or create bugs. Our team minimized this type of tasks after starting to detect them with implementation of testing.
Done: Story that has been fully completed.
Average throughput: Average amount of stories that the team has completed during the previous Sprint. This allows a realistic evaluation of the performance of the team. Creating real expectation with the client and time required for the following Sprint.

Overall the project is on track with the desired development of the project and has achieved a total of 140 completed stories for the midterm point.

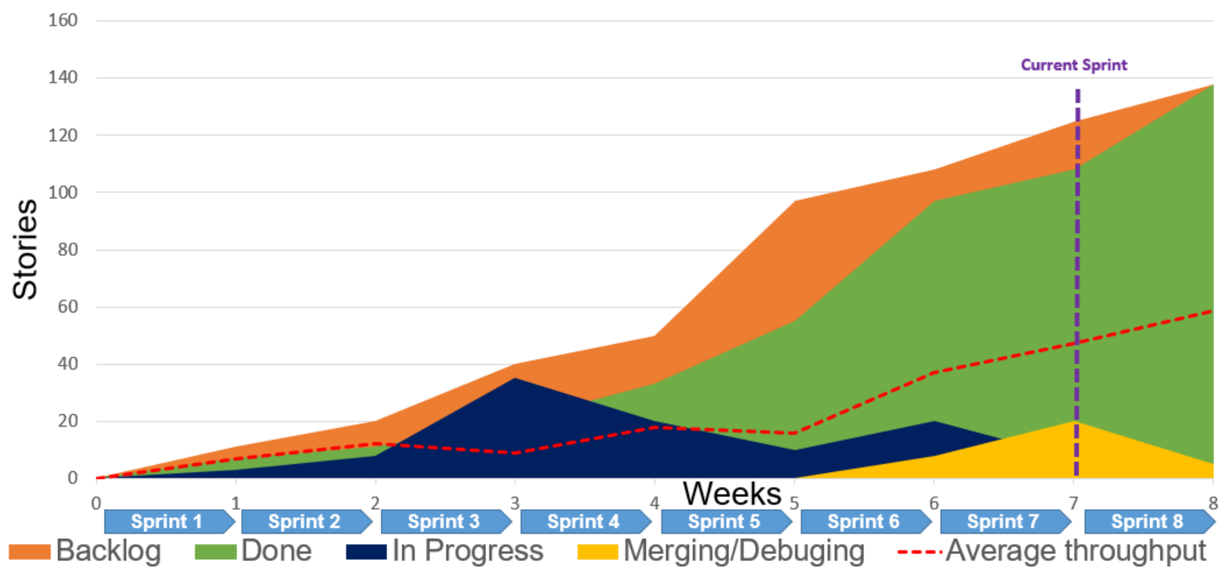


Figure 28 Project Progress Sprint BurnUp Chart

Future Work & Challenges

Future work in the months of July and August will focus on three main areas: development, deployment, security.

Further development is required to meet the planned vision for the web app as outlined by the Invision prototype. Continued development of the test suit should occur concurrently with the development of the web app. Additionally, the deployment should focus on setting up a Content Delivery Network (CDN) using AWS CloudFront to handle static assets, a CDN should be utilized to increase scalability and optimize speed. Containers using a platform such as docker should be implemented into our deployment process to simplify configurations and increase maintainability.

Security should be expanded to consider improving authentication of users and limiting usage rates of users to prevent spam attacks. Improvements for authentication could be performed through integration with the UofC login system where all users are confirmed to be students, implementing features such as security captchas, posting limits, etc. Additionally security should consider removal and reporting of offensive content and possibly integration of disciplinary measures. A focus to consider a finalized Minimum Viable Product were community members can start using some of the features of the Upost website will be prioritized. We expect to allow the UCalgary student community to start adopting of the Upost key features to connect a community member with relevant events of their interest. Finally, we will continue collaborating with our University Project Industry Sponsor to plan an effective lunch plan at UofC.

References

[1] freeCodeCamp, “Advantages and Disadvantages of JavaScript,” freeCodeCamp Guide. [Online]. Available: https://guide.freecodecamp.org/javascript/advantages-and-disadvantages-of-javascript/. [Accessed: 08-May-2019].

[2] R. Chugh, J. A. Meister, R. Jhala, and S. Lerner, “Staged information flow for javascript,” ACM SIGPLAN Notices, vol. 44, no. 6, p. 50, 2009.

[3] T. Khuat, “Developing a frontend application using ReactJS and Redux,” thesis, 2018.

[4] S. A. Robbestad, Reactjs blueprints. Brimingham: Packt Publishing Limited, 2016.

[5] M. K. Caspers, “React and Redux,” pp. 11–14, Feb. 2017.

[6] Y. Nader, “What is Django? Advantages and Disadvantages of using Django,” Hackr.io Blog, 03-Sep-2018. [Online]. Available: https://hackr.io/blog/what-is-django-advantages-and-disadvantages-of-using-django. [Accessed: 02-May-2019].

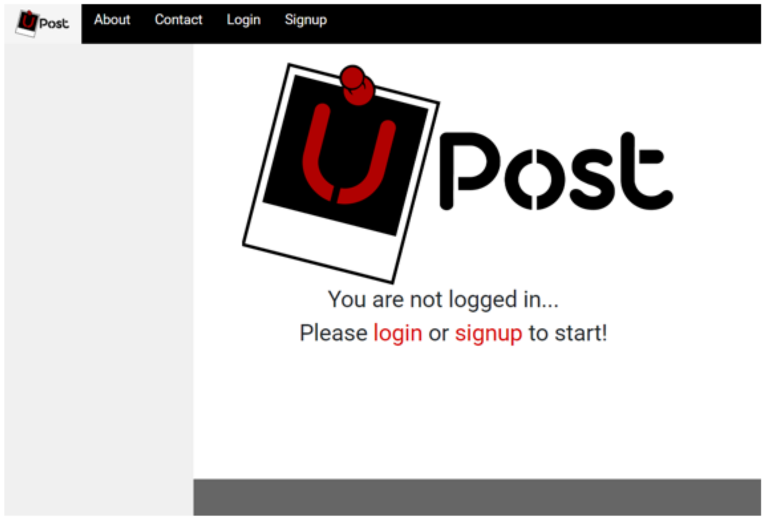
[7] A. Holovaty and J. Kaplan-Moss, The definitive guide to Django: Web development done right. Berkeley: Apress, 2009.

[8] “AWS Free Tier,” Amazon. [Online]. Available: https://aws.amazon.com/free/?awsf.Free Tier Types=categories#featured. [Accessed: 08-May-2019].

[9] W. Manager, “Amazon AWS vs Microsoft Azure public cloud - which is best for enterprises,” Cloud Computing, 07-Mar-2017. [Online]. Available: https://www.newgenapps.com/blog/amazon-aws-vs-microsoft-azure-best-for-enterprises. [Accessed: 08-May-2019].

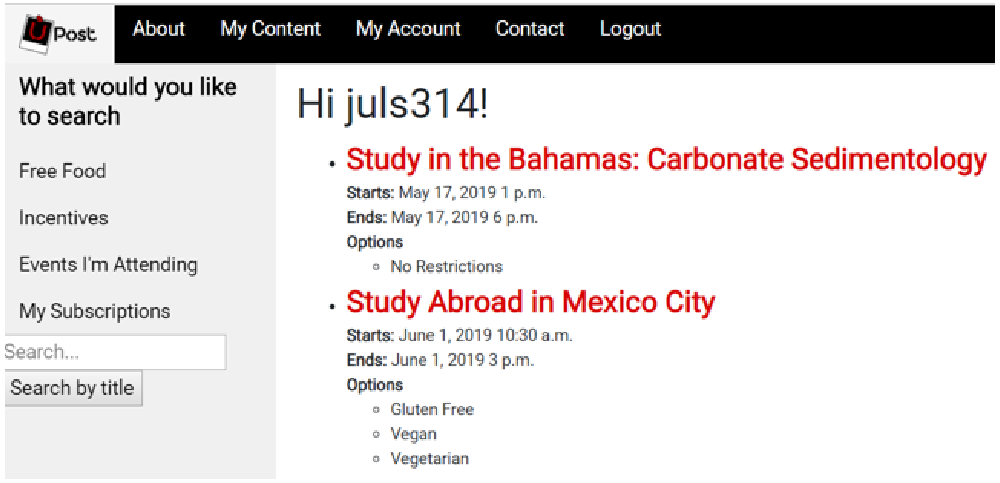
[10] D. Tech, “Comparing AWS vs Azure vs Google Cloud Platforms For Enterprise App Development,” Medium, 14-Aug-2018. [Online]. Available: https://medium.com/@distillerytech/comparing-aws-vs-azure-vs-google-cloud-platforms-for-enterprise-app-development-28ccf827381e. [Accessed: 08-May-2019].

Appendix A: Previous Upost Django Design Examples



You are not logged in...
Please login or signup to start!

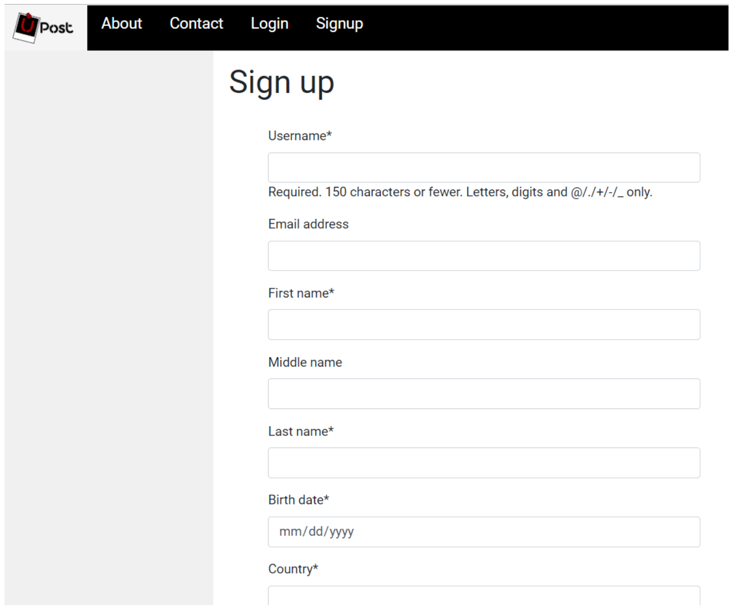
Figure 29 Deprecated Upost Main Login/Signup Page



Hi juls314!

- Study in the Bahamas: Carbonate Sedimentology
  Starts: May 17, 2019 1 p.m.
  Ends: May 17, 2019 6 p.m.
  Options
  - No Restrictions
- Study Abroad in Mexico City
  Starts: June 1, 2019 10:30 a.m.
  Ends: June 1, 2019 3 p.m.
  Options
  - Gluten Free
  - Vegan
  - Vegetarian

Figure 31 Deprecated Event Search Page



Sign up

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address

First name*

Middle name

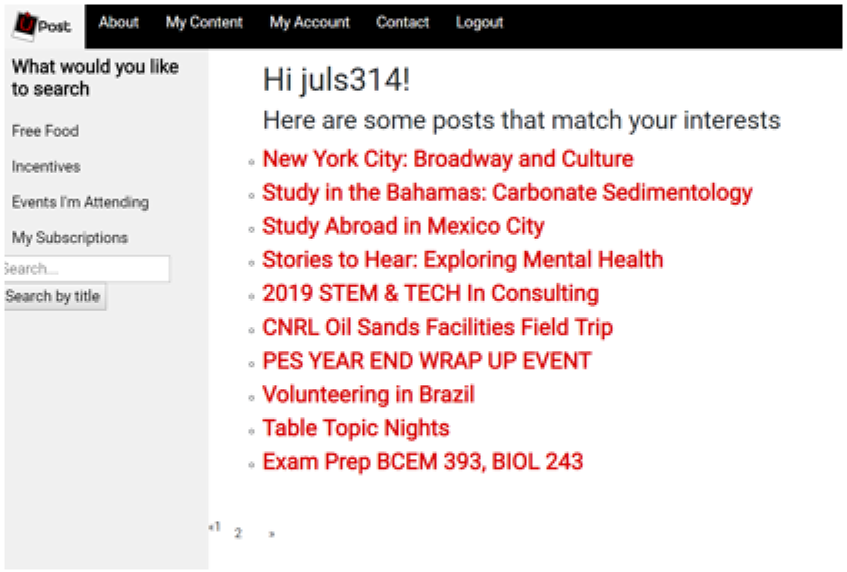
Last name*

Birth date*

mm/dd/yyyy

Country*

Figure 30 Deprecated SignUp Page



Hi juls314!

Here are some posts that match your interests

- New York City: Broadway and Culture
- Study in the Bahamas: Carbonate Sedimentology
- Study Abroad in Mexico City
- Stories to Hear: Exploring Mental Health
- 2019 STEM & TECH In Consulting
- CNRL Oil Sands Facilities Field Trip
- PES YEAR END WRAP UP EVENT
- Volunteering in Brazil
- Table Topic Nights
- Exam Prep BCEM 393, BIOL 243

Figure 32 Deprecated Event Interest Match Page