Caml-flage Design Document

Arzu Mammadova — am2692
Shea Murphy — sm967
Chris Umeki — ctu3
Mena Wang — mw749

March 27, 2018

1 System Description

Caml-flage is a social networking platform that, as its title suggests, allows users to interact with each other anonymously. People start out as anonymous users and are able to post text, links, and photos, and interact with each other on a public dashboard. Users tag posts with different topics to categorize discussions. Each post and reply will have three camel-shaped buttons - up-camel (like), down-camel (dislike), and double-camel (direct reply) - thereby enabling users to react to one anothers posts.

Each account has a points accumulator initially set to 0. Once a direct conversation starts between two users, they will both have an interaction score that increases with each reply. Once the score reaches a certain number, the two users will no longer be anonymous to each other. That is, both of the users identities will be exposed and each user will now be able to see the others personal profile, i.e. text, photos, etc.

2 System Design

2.1 server.ml

The server module stores all the data that is necessary for the platform to operate including data on each user, posts, comments, and tags. Each individual data group will have another module of their own, detailed below.

We are still investigating the uses of our external libraries and how to connect the data and the server. Therefore, we will be adding more functions to server.mli later on.

2.2 users.ml

The users module will be responsible for keeping track of data related to the user– such as the username, their posts, as well as a list of users which theyve surpassed the interaction score threshold. In addition, the users module will store a hashtable of interaction scores between specific users.

This information will be kept in a record for each individual user, whereas the data of all the users on the server are the individual records stored in a tree.

2.3 entry.ml

Entry has two types: posts and comments. Posts are top level entries posted by the users whereas comments are replies to posts or other comments.

The type defined for a post will include the list of comments associated with a post, which will be implemented as a list of linked lists, a relevance score, determined by the number of up-camels and down-camels, content(text, images,etc.), id of the post, and the date and time the post was published.

The type defined for a comment will be similar to posts. Major differences between the two types would be that comments are linked back to the parent entry, and posts have a title as well as content.

Similar to posts, comments will also store the list of comments/replies, a relevance score determined by the number of up-camels and down-camels, as well as the date and time that the comment was posted.

2.4 tags.ml

Tags are a way for users to categorize their posts as well as to find posts pertaining to a certain topic. Each tag is defined as a string which is the name of the tag as well as the posts that are linked to the tag.

3 Data

Most of our data will be stored in records and lists of records. In addition, we will be using a hash table to store the interaction score between users.

We are using Yojson to store all of our information in order to maintain the server.

In terms of communication, we will be using HTTP GET and POST requests. GET requests will be used to return the contents of the webpage– it is functional because it doesnt change the state of the server.

4 External Dependencies

4.1 Opium

Opium will allow us to maintain the web application, handle HTTP requests, and interact with databases that contain all the users and posts information.

$ opam install opium

4.2 ReasonReact

Our goal is to display all the posts on the server onto an actual webpage. We will use ReasonReact to build the frontend (display of the webpage and user interface) of this project.

4.3 Yojson

We will use Yojson to store the data of users, posts, comments, and tags– this will ensure that data already stored on the server will not be lost.

5 Testing

Because we have many records and record lists that are used to store data, we will be unit testing all the functions that change the records to make sure data is being stored correctly. Module tests will be written for each of the module listed above. In addition, we will be testing the server requests to ensure that data is being processed correctly and that the webpage functions the way we want. These tests will be more interactive instead of in modules and unit testing.

Depending on how we split up the modules and coding, each person will be responsible for unit testing their own modules since theyll be most familiar with how their code should function (this mostly includes the functions that will not be in the .mli). Once we begin implementing the front end portion of the project, everybody is responsible for testing the interactive elements on the webpage.