CS 294S Research Project in Software Systems and Security **Spring 2015**

Homework 1

Introduction

In this homework, we will explore the Omlet Chat API for web apps, and how to write a small app that can be launched from Omlet.

You are required to submit a (hopefully working) solution to receive feedback on this assignment. Working in groups up to 3 people is allowed, and only one submission per group is required.

Submission instructions will be posted on the website soon.

The problem: extending QuikPoll

QuikPoll is a simple application that allows a user to create a multiple choice question, and then poll all other members of a group chat for answers. We would like you to extend it to support one of the following:

- Instead of one of the choices listed, a user has the option to write a free form answer of his own. The answer is visible only to the poll creator.
- Instead of one of the choices listed, a user has the option to write a free form answer, and subsequently all other users see that as an additional option

Alternatively, any interesting and non trivial extension to this app will be accepted as a valid submission for the homework, so feel free to explore any new way of using the Omlet API.

The starting code for this assignment can be found at `http://suif.stanford.edu/~courses/cs294s/hw/hw1-quikpoll.zip`. The code itself should be self-explanatory, and while you're not required to make use of any of the starting code in the submission, you are strongly encouraged to read through it.

The Omlet API

The object model

As introduced in the slides in class, the Omlet API for Web Apps is centered around the idea of a RDLs, or rich deep link.

A rich deep link is a "deep link", that is, a link that opens into an app instead of a web browser, with a set of associated metadata, such as a thumbnail, a title and a description.

To create a RDLs, one would use the following API:

```
var movie = { title: "Die Hard",
              year: "1988",
              rating: 10,
              synopsis: "John McClane saves the day.",
```

```
            thumbnail: "http://ia.media-imdb.com/images/M/
MV5BMTY4ODM0OTc2M15BMl5BanBnXkFtZTcwNzE0MTk3OA@@._V1_SX214_AL_.jpg" };

var rdl = Omlet.createRDL({
     noun: "movie",
     displayTitle: movie.title + " | Rotten Tomatoes",
     displayThumbnailUrl: movie.thumbnail,
     displayText: movie.synopsis,
     json: movie,
     webCallback: movie.links.alternate,
     callback: window.location.href,
});
```

Here, we observe that the `json` argument is a free-form JSON object that represents all the state and useful information of the web app. `callback` is the actual link location, and most probably will always be `window.location.href` (that is, the current page). `noun` is a single word that generically describes what is being shared. Here we choose *movie*, so that the app will show *Joe shared a movie* in the notifications and chat list. The other fields should be self-explanatory.

The app flow

The interaction with a web app in Omlet follows two flows, which determine how the app receives the data from the chat and how it shares back.

In the first flow, the app is activated from the drawer, and it receives no data from the chat. In this case, the expectation is that the app will produce some data or state, that will be sent back to the chat.

This happens through the call

```
Omlet.exit(rdl);
```

Where `rdl` is a rich deep link created as previously shown.

As the name of the API implies, this call exits the app and returns to the chat. In other words, apps in Omlet behave similarly to modal dialogs in desktop apps: they are summoned for a specific purpose, they generate some interaction and some data, and they are closed after to let the user go back to his main focus (in this case, the Omlet chat).

A simpler form of sharing is also possible, where the app just generates some text, and shares it as

```
var obj = {
  type: "text",
  data: {
   text: "First post!!"
  }
}
Omlet.exit(obj);
```

An app that called `exit` this way would show *First post!!* in the chat feed, without any additional decoration.

On the other hand, if an RDL is shared, the app can be activated by clicking on it, leading to the second application flow.

In this case, the app receives the data that was part of the RDL through the following API:

```
Omlet.ready(function() {
  var movie = Omlet.getPasteboard();
  if (movie !== undefined) {
     // activated from the RDL!
  } else {
     // activated from the drawer
  }
});
```

Here, the return value of `Omlet.getPasteboard()` is the object that was passed as the `json` argument in the `createRDL` call, and not the RDL itself. Naturally, it is possible to pass data to the web application also by modifying the callback with a target or query part, and then your code in the server or in the client would interpret that data.

The document API

In addition to preserving state through a rich deep link, in Omlet it is possible to store state in documents, which are just named JSON blobs associated with an identity.

```
Omlet.document = {
    create: function(success, error),

    get: function(reference, success, error),

    update: function(reference, func, parameters, success, error),

    watch: function(reference, onUpdate, success, error),

    unwatch: function(reference, success, error)
}
```

Each function is asynchronous and accepts optionally a `success` callback, which is called with the result of the call in case of success (as a JSON object), and an `error` callback which is invoked if errors occur.

The successful result of a `create` call is an object whose `Document` property is a document reference that can be serialized and can be passed to the other calls. The successful result of `get` is the document itself. Other calls don't have a meaningful successful result and for them the argument to `success` should be ignored.

The `func` argument to `update` is called to generate the document or to update it with the new parameters. It is passed the old document as the first argument, and the app specified parameters as the second.

The `onUpdate` argument to `watch` is called every time the document changes, for example because it is being updated by another user. It receives the new document as its only argument.

Testing the app

For this assignment, you should install the current development version of Omlet. The link for Android is `https://rink.hockeyapp.net/apps/fcf1b3ef34ef8c787444d5bc08beb09b`, while the link for iOS is `https://rink.hockeyapp.net/download/5d38b90055954607b0adf23a1260f209`.

Subsequently you need to register your apps in the new Omlet developer portal at `https://idp.omlet.me/apps`. To log in, you need to go to Settings on the new Omlet build and hold the Omlet egg icon at the bottom to activate a QR code scanner and scan the code on the portal page.

More details

More details can be found in the slides that were shown in the presentation in class (`http://suif.stanford.edu/~courses/cs294s/slides/om-api-stanford.pptx`), as well as the official Omlet developer page at `http://www.omlet.me/developers/`.