Where Should We Eat
CS3216
Assignment 3
Team 1

Core Milestones

Milestone 0 : What Problem Does Our App Solve?

When eating your friends, the question normally props up - Where Should We Eat. Ideally we would like to eat at a place that is convenient for everyone to reach and easy to get back home from. Our application sets out to solve this common everyday problem.

Milestone 1 : Description of the App, and why it's Mobile Cloud

The application intends to use the locations of a group of friends to find the centrally most convenient place for everyone to meet. The idea is to minimise the money spent by everyone on taxis to get to the same location, as well as the distance travelled. The server of our application helps us to find these most suitable locations by finding the most centrally located food eateries using the API provided by Yelp and then providing the costs for each person in the group to get to that location using the Google DistanceMatrix API. It makes most sense to implement this idea as a mobile application as the mobile interface would be the easiest medium to use when with your friends.

Milestone 2 : Target Users

People who like having supper with their friends! Especially for groups where people live in different areas of the city.

We plan on attracting users to the app by using the app to go for food outings along with our friends. For example, in such cases our friends then see, "Hey, this is a really great app" and use the app to find a place to go eat when they next have food outings with friends, like supper.

Milestone 3 : Pick a name for your mobile cloud application

We were somewhat inspired by the website "Where the fuck should I go to eat?" (wtfsigte.com). We were going to go with the name "Where the fuck should we go to eat?", but instead opted for "Where should we eat?". It is the exact same question for groups of friends who want to have a meal together, but disagree or do not know where they can grab a meal. "Where should we eat?" Well, this app solves that for you.

Milestone 4 : Draw the Database Schema

Our Database schema was defined in a file, which we'll copy verbatim:

```
User
    ident Text
    password Text Maybe
    UniqueUser ident
    deriving Typeable
Email
    email Text
    user UserId Maybe
    verkey Text Maybe
    UniqueEmail email
Person
    name String
    postal String
    phone Int Maybe
    location Location
    deriving Show
    deriving Generic
Place
    name String
    location Location
    yelpid String
    imgurl String
    deriving Show
    deriving Generic
Search
    people [PersonId]
    filters [String]
    chosen PlaceId Maybe
    places [PlaceId]
    deriving Show
SearchPlace
    searchid SearchId
    placeid PlaceId
    distance [Int]
    deriving Show
```

Milestone 5 : At least 3 prominent REST API

3 of our more prominent REST API points:

- `POST /doSearch`
  Begins a user's Search for places, accepting the input of an array of people.
  - REQUEST:
    A JSON object with property "people", which is an array of Person objects (with properties "name", "postal", and "phone").
    e.g. `{people: [{name: "", postal:"", name:""}]}`
  - RESPONSE:
    A JSON object with the search ID wrapped in an object.
    e.g. {searchId: 343}
- `GET /search/#SearchId/places`
  Gets the places found for the given search with the given ID.
  - REQUEST:
    #SearchId is the id of the search, as returned earlier by some /doSearch POST request.
  - RESPONSE:
    A JSON object wrapping a property "places", which is an array of Place objects.
    e.g. `{places: [placeName, placeLocation: {lat, lng}, placeYelpip, placeImgurl]}`
- `GET /search/#SearchId/#YelpIdentifier/distance`
  Returns the distances for the given place, for the given search.
  - REQUEST:
    #SearchId refers to the id of the search, as returned earlier by some /doSearch POST request.
  - RESPONSE:
    A JSON object wrapping a property "distances" which is an array of the names of all the people who are going to the given place along with the distances of each of their houses(entered postal code) from the given place `{distances: [distance, personName]}`

Milestone 6 : At least 3 interesting SQL queries

Our most interesting SQL queries have been listed here -

- Retrieve the distances of each person in the party from the given locations

```
distanceTuple <- runDB $ selectFirst [SearchPlaceSearchid ==. searchId,
SearchPlacePlaceid ==. placeIdKey] []
```

- Get the PeopleId for the people who have just been added to the database using their unique phone numbers or unique name + postal code combination:

```
let nameAndPostalFilter p = [ PersonName   ==. (personName p)
                            , PersonPostal ==. (personPostal p)]
      phoneNumberFilter   p = [ PersonPhone  ==. (personPhone p)]
```

```
    selectPersonFilterFor  p = (nameAndPostalFilter p) ||. (phoneNumberFilter p)

peopleIds <- mapM
               (ensureInDBandGetIdWith selectPersonFilterFor)
               inputPeople

-- Function to retrieve/insert people
ensureInDBandGetIdWith selectFilterFor p = do
  maybeEntity <- runDB $ selectFirst (selectFilterFor p) []
  case maybeEntity of
    Just e ->
      -- previously in DB
      return $ entityKey e
    Nothing ->
      -- wasn't previously in DB
      -- So, need to insert it.
      runDB $ insert p
```

- Retrieve the tuple in the database Place where the yelpid is same as the required yelpid

```
placeId <- runDB $ selectFirst [PlaceYelpid ==. yelpId] []
```

Milestone 7 : URL Rewrites

`[QSA,L]` What does it mean?? (It's so intense!).
As per https://httpd.apache.org/docs/current/mod/mod_rewrite.html#rewriterule
`QSA` = "Query String Append", which appends the current GET parameters to the redirected URL. (The default behaviour is to discard these).
`L` = "Last", and no more rewrites are performed. (Comparable to a "break" statement in C-like languages).

Our most interesting rewrite?
This rewriting is useful for PHP, since that's how PHP receives its GET parameters. With the Yesod framework, we specify the REST Routes we would like in this "pure" form, and Yesod (magically) handles the processing to give it to our functions.
The most interesting thing we do is have Nginx serve a reverse-proxy from port 80 to the port yesod runs on.

Milestone 8 : Attractive Icon + Splash Screen

Created an icon and added a Splash Screen that loads the homepage

Milestone 9 : Style the UI; why our UI design is the best possible UI for the App

The jQuery mobile framework has been used to design our application. Since whatsapp is highly popular in our target region and does a very neat job, we decided to inspire our UI designs from whatsapp making changes where required (like displaying the Google Maps and the places). We believe that due to the high popularity of whatsapp, having an application adopt a similar design would give a native feel as well to the application.
The usage of jQuery mobile also minimized the need to write specific CSS for the application, since jQuery themeRoller allowed for online editing and generation of theme CSS. As such, our only CSS was to position the embedded map 'correctly'.

Milestone 10 : Offline Functionality

The offline functionality provided by our application involves being able to add/view/update/delete the contacts of the user on his/her application. By making use of the local storage, the application also stores the information of the Outing that a group of friends (Party) are going to along with information of their tentative taxi fares. This information is hence readily available offline for the users to access in the form of their history. Users will be able to access all their prior Outings along with information about whom they went on the Outing with.

Milestone 11 : Syncing

We felt that syncing features for this app were unnecessary, since all the user's data is saved onto local storage, and the user is the only user who creates content in their user experience. (i.e. no "social" interaction at all).

So, there is no difference for 99% of users for if we were to sync or not, since the server would never provide more information than was already on their phone, since the way our app is designed ensures that server and local information would be in sync most of the time, barring outlier cases.

In the case where the user is logging in from a new device, the information they would be easily reproducable (by just running the search with the same input again); but since our app is more about providing a search utility to find a place for people in different locations to eat, it's more likely a user will "favourite" a location in another Map app (e.g. FourSquare) in addition to our app, so this is no great loss for this user.

Milestone 12 : Authentication

Quoting from the Yesod Book's chapter on Authentication and Authorization:

For many use cases, third-party authentication of email will be sufficient. Occasionally, you'll want users to actual create passwords on your site. The scaffolded site does not include this setup, because:

- *In order to securely accept passwords, you need to be running over SSL. Many users are not serving their sites over SSL.*
- *While the email backend properly salts and hashes passwords, a compromised database could still be problematic. Again, we make no assumptions that Yesod users are following secure deployment practices.*
- *You need to have a working system for sending email. Many web servers these days are not equipped to deal with all of the spam protection measures used by mail servers.*

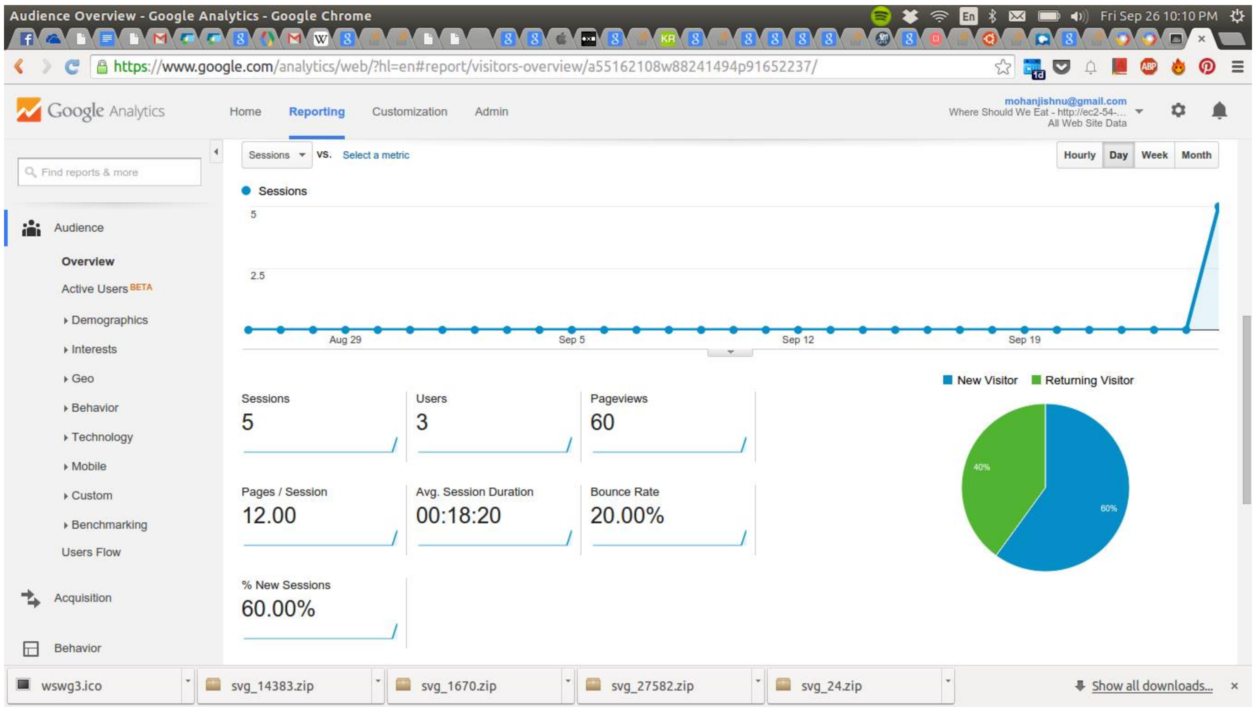
For these reasons, and for reasons of simplicity, we've opted for Yesod Auth's Googleemail auth provider, which makes use of sessions to help verify the user's authentication & authorization.

Milestone 13 : User Experience

User Story 1:
User has several friends, of which many belong to different overlapping social cliques. This means for each outing, each friend will be repeated a few times, but the exact combination of friends will not be identical each time. The user can save each friend and the associated postal code in the Friends list. Then, for each outing, the user can easily choose the friend again instead of retyping every time, yet it has the flexibility to allow for different combination of friends.

User Story 2:
User remembers he went on an outing prior with a group of friends to a restaurant in a great location but can't remember the details. Fortunately, all past outings are shown in the Outings tab, which includes a comprehensive detail page which shows the important information relating to the restaurant chosen, the friends he went with, and even the estimated cost each friend takes to go to their house/specified location from the restaurant.

Milestone 14 : Google Analytics



Sessions 5

Users 3

Pageviews 60

Pages / Session 12.00

Avg. Session Duration 00:18:20

Bounce Rate 20.00%

% New Sessions 60.00%

Cool Milestones

Milestone 15 : Social Network Integ.

We didn't believe that this feature would have helped our application since it was heavily reliant on the users' and their friends' postal code.

Milestone 16 : Geolocation API + Google Maps

Our application makes heavy usage of Google Maps display and APIs as providing estimates of distances and routes is key to our application.
We allow user to input in postal codes representing locations of him/her and friends, and using geocode functionality, plot these on a google map display. Further, we show possible food options, which when clicked on are also plotted on the same map. The different routes for each of user and friends from the highlighted place are also plotted on the map, allowing easy visual examination of the distance and general route. For clarity of information, different colours are used for each users route. This facilitates decisions like which of the friends can perhaps cab together, as well as gives the group a general idea of their path from their location to the food place or vice versa.

Milestone 17 : Why jQuery Mobile was the best choice

We have considered a number of frameworks that boasted the whole “Native Look and Feel” thing, such as ChocolateChip-UI and Junior among others, we decided on jQuery Mobile mainly because of the relative ease of implementing it compared to the others. Also, we felt more comfortable with its API compared to the others, as well as the way it can be implemented with minimal styling effort but still in a presentable, even if “minimalist-zen”, way.

Honorable Mention: Haskell Backend

Much to the derision, amazement, “wow you guys are brave”, and numerous other reactions of others, we implemented our backend in Haskell cause what better time to check it out than to check it out now (thus fulfilling the spirit of CS3216 - learning new things, and daring to try them).