**ShareTaxi**

CS3216 Assignment 3

*Naomi Leow Wen Xin | Yeo Zi Xian, Justin | Ding Xiangfei*

Compulsory Milestones

**Milestone 0: Describe the problem that your application solves**

Our application solves the lack of a quick, mobile platform for computing a route with multiple stops and connecting users who want to share a cab. Currently, Google Maps does not provide for multiple-location routing on mobile, and even on desktop there is no optimisation done - the routes are ordered by the input order. Other platforms are also not as powerful or mobile-friendly as we desire such them to be. We have designed ShareTaxi to overcome these issues.

**Milestone 1: Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application’s objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?**

Our app allows users to plan an optimised shared taxi route with multiple points. First, a user can key in multiple start and end points (usually one for each person with them), and the app optimises the route based on the selected option. Second, a user can share a route they are currently planning, and allow other users to add their own stops to that route via requests. Users can then accept those stops into their route if it makes sense to them. Similarly, users can browse their friends’ routes and send their own requests to other friends.

It makes the most sense to implement the application as a mobile cloud application because of its usefulness on-the-go. Many people decide to use taxis when they are in a rush or because they are trying to save time; be it at home or when outside, being able to quickly access our features on mobile is vital to our application. Mobile cloud computing would also allow us to retrieve route data from the server and save that data on the phone locally, for reference when users have no connection. For long term usage, some users may reuse some routes often, and mobile cloud computing also allows us to transfer their saved, shared and friends’ routes to any device they login to. Most importantly, to carry out our collaborative feature, we need to build a centre storage that syncs with each logged in user.

**Milestone 2: Describe your target users. Explain how you plan to promote your application to attract your target users.**

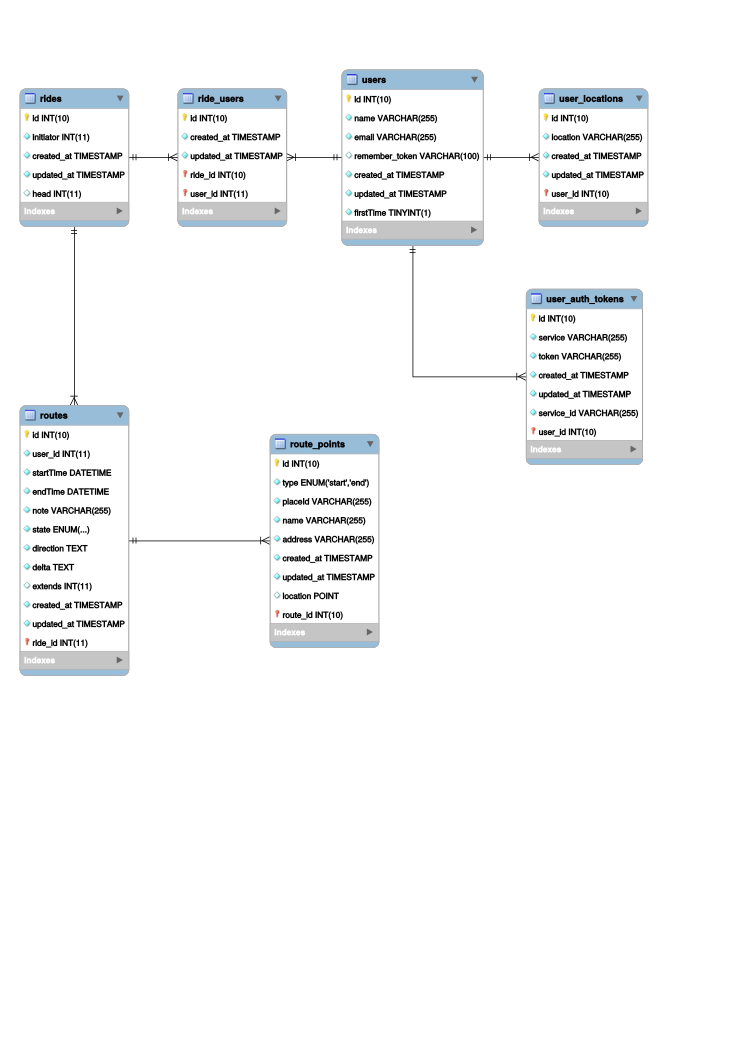
Our target users in general are anyone who may need to use a taxi in future, and wants to share to split cost. We intend to promote our app as a quick-and-easy tool to saving time, effort and money when taking a cab, giving the example of sharing a cab as a group when returning home late at night. Although our app is not something users will use every day, by explaining it in the familiar context of sharing a cab home, we can imprint in our target users’ minds that when such a need arises, they should try using our app. As users begin to use our app, they should naturally realise its other possible uses on their own.

We intend to allow sharing via links and Facebook as well, for our collaborative feature. This will inherently spread knowledge and usage of the app as users attempt to get people to share a taxi with them.

**Milestone 3: Pick a name for your mobile cloud application. (Not graded).**

ShareTaxi.

**Milestone 4: Draw the database schema of your application.**



**Milestone 5: Design and document (at least 3) requests of your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief explanation on the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices (if any.)**

**POST /{provider}/token**

Update current users' access token

Parameter:

provider: Currently only 'facebook' is allowed

Input:

token: new access token

**POST /logout**

Log out of the system

**GET /{provider}/token**

Retrieve current users' access token

Parameter:

provider: Currently only 'facebook' is allowed

Output:

JSON, with field 'data' being the access token

**GET /user/friends**

Retrieve current users' friends in various social network who also uses the system

Output:

JSON, an array of User objects

**GET /user/facebook**

Retrieve current users' facebook profile

Output:

JSON, with format following Facebook Graph API format

**GET /rides**

Retrieve current users' joined rides

Output:

JSON, an array of Ride objects

**GET /rides/{id}**

Retrieve the information of the ride of the given id

Output:

JSON, a Ride object

**POST /rides**

Create a new ride with the given route information

Input:

JSON, a Route object as the initial version of this ride

Output:

JSON, a Ride object with fields:

'route': a Route object

'joinedUsers': an array of User objects, which will only include the ride creator initially

**DELETE /rides/{id}**

Remove the ride of the gien id

**POST /rides/search**

Search rides with some criteria

Input:

longitude: required; Longitude

latitude: required; Latitude

distance: required; distance from the given point of longitude and latitude

startAfter: optional; time point where rides should start after

endBefore: optional; time point where rides should end before

Output:

JSON, an array of Ride objects whose RoutePoint is within the given distance from the given point and sorted in ascending order

**GET /rides/{id}/routes**

Get current version of routes and also the ride's requests

Parameter:

id: the id of the ride

Output:

JSON, an array of Route objects

**GET /user/rides/joined**

Get current user's joined rides

Output:

JSON, an array of Ride objects

**GET /rides/from/own**

Get current user's created rides

Output:

JSON, an array of Rides created by user

**DELETE /rides/{id}/users/{userId}**

Remove a user from the ride

Parameter:

id: the id of the ride

userId: the id of the user to be removed

**POST /rides/{id}/users/{userId}**

Add a user from the ride

Parameter:

id: the id of the ride

userId: the id of the user to be added

**GET /rides/{id}/requests**

Get a list of requests to the ride

Parameter:

id: the id of the ride

Output:

JSON, an array of Route objects as the requests from other users

**GET /rides/from/friends**

Get a list of rides created by the current user's friends

Output:

JSON, an array of Ride objects created by the friends

**GET /routes/{id}**

Get information of the route of the given id

Parameter:

id: the id of the route

Output:

JSON, a Ride object with the fields:

id: the route id

ride\_id: the ride this route describes

user\_id: the creator of this route

state: allowed values are 'requested' (being a request) or 'accepted' (being accepted)

direction: Google Map Direction API data to describe the path of the route

origins: an array of RoutePoint objects for departure points

destination: an array of RoutePoint objects for arrival points

shared\_details: an object with fields

arrival\_time: the latest arrival time of the route

notes: additional notes

**POST /routes**

Post a new version of the route or make a new request to a ride

Input:

JSON, a Route object with a valid ride\_id

Remark:

If the current user is the creator of the target ride, the ride will be updated with this version of route; otherwise the route is treated as a request to the ride

**POST /routes/{id}**

Update the route of the given id

Parameter:

id: the id of the route

Input:

JSON, a Route object

**DELETE /routes/{id}**

Remove the route (being a request or current version)

Parameter:

id: the id of the route

**POST /routes/{id}/accept**

Accept the request and merge the waypoints

Parameter:

id: the id of the route request

**POST /user/routes/requests**

Get a list of requests made by the current user

Output:

JSON, an array of Route objects

**GET /route\_points/{id}**

Get information of the route point

Parameter:

id: the id of the route point

Output

JSON, a RoutePoint object with fields:

type: either 'start' (departure point) or 'end' (arrival point)

name: name of the point

longitude: longitude of the point

latitude: latitude of the point

formatted\_address: address in a human readable format

google\_place\_id: Google Place ID for references and checking against Google Map APIs

**DELETE /route\_points/{id}**

Remove the route point

Parameter:

id: the id of the route point

**POST /route\_points/{id}**

Update the route point information

Parameter:

id: the id of the route point

Input:

JSON, a RidePoint object

**Milestone 6: Tell us some of the more interesting queries (at least 3) in your application that requires database access. Provide the actual SQL queries you used.**

Here is a sample query we did to get closest points around a given origin, which uses some db functions to speed up look-ups:

select \*, x(location), y(location), geo\_distance(location, point(130,2))

from `route\_points` where mbrcontains(linestring(point(126.51739425837,0.55072463768116),point(133.48260574163,3.4492753623188)), route\_points.location)

This one gets the details for a ride a user has joined:

select rt.note, rt.endTime, r.id, r.initiator, rp.placeId, rp.name,

rp.address from `rides` r

join `ride\_users` ru on r.id = ru.ride\_id

join `users` u on ru.user\_id = u.id

join `routes` rt on ru.id = r.id

join `route\_points` rp on `rp.route\_id` = rt.id

where ru.user\_id = 1;

This last one gets the details for a ride a user has created, as well as the user’s details for display:

select rt.note, rt.endTime, r.id, r.initiator, rp.placeId, rp.name, rp.address, u.name

`rides` r join `users` u on r.initiator = u.id

join `routes` rt on rt.id = r.head

join `route\_points` rp on rp.route\_id = rt.id

where u.user\_id = 1;

**Milestone 7: Find out and explain what [QSA,L] means. Tell us about your most interesting rewrite rule.**

QSA: Append the query string at the end of the original url to the rewrite of the original url.

For example,

RewriteRule ^foo/(.\*)$ /index.php?b=$1 [QSA]

applied to /foo/bar?q=something

results in /index.php?b=bar&q=something

instead of /index.php?b=bar without the [QSA] rule

L: Do not apply subsequent rewrites if the rewrite rule is matched.

Our most interesting rewrite rule is RewriteRule ^([^.]+)$ /#/$1 [L], which reroutes all paths that are not files to “/#/”. It does so by prepending “/#/” to any url extension that does not contain a ‘.’ (which represents a file). This allows our urlRouter (an AngularJS construct) to work properly.

**Milestone 8: Create an attractive icon and splash screen for your application. If you did not implement a splash screen, justify your decision with a short paragraph.**

Our icon features a taxi that seems to be at the middle waypoint of a route. This ‘route’ is formed by 3 circles connected by 2 lines -- a warped share icon. Attractive, *and* symbolic.

**Milestone 9: Try adding your application to the home screen to make sure that they are working properly. Make sure at least Safari on iOS and Chrome on Android are supported.**

The application appears to lag a little, but they work properly. Tested on iPad and Note 4.

(Local storage does not work on Safari. This is an issue with indexeddb)

**Milestone 10: Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application.**

Our UI was built using Ionic, whose styles follow the design language of iOS and Android fairly closely, giving it a very native feel while minimising the CSS styling we had to do on our own.

Our main view is the map with 3 buttons at the top: “Plan”, “Share”, and “Save”. We placed “Plan” (which creates a new route) in the main screen, first button on the left because it is our primary feature, the feature we expect users to use the most and want the quickest access to. The “Share” and “Save” features are closely tied to the “Plan” feature, so it makes the sense to have them in the same view. Moreover, the “Share” feature is also one of our key features, so we wanted to make it fairly visible, so the users will notice and remember it as a significant feature.

Our other features were hidden in the side menu. While this obscures some features from the user, we felt that the UI for the side menu was intuitive and familiar enough to most users, and that a user looking for these features would be able to find them within one tap.

Our UI focused a lot on clarity, in particular from the moment a new user opens our app to the first route they plan. We designed the introduction tutorial (which only opens once per browser) to explain the key features (Plan, Share, Find) of our app, and at the same time familiarise the user with UI elements such as button icons and the side menu. For example, in our tutorial, we explained how to “Plan a Route”, and with the icon we would use for the button next to the title. This allowed us to simply put the icon and “Plan” as a button within our app, which was shorter yet clear to the user, because of the link that was formed in the tutorial. We also made use of placeholders and structured our forms as sentences from the user’s point of view, to give the user a clear idea of what they need to fill.

We also attempted to adapt some UI styles from Google Maps, to create a sense of familiarity, given that our app also deals with transport. An example of this can be seen in the route details being displayed at the bottom of the screen, and tapping on the box opens up more details. We felt that this UI design was consistent with users’ current mental models, and at the same time allowed us to maintain a clean and pleasant overall UI.

**Milestone 11: Implement and briefly describe the offline functionality of your application. Explain why the offline functionality of your application fits users’ expectations. State if you have used Web Storage, or any other technology. Explain your choice. Make sure that you are able to run and use the application from the home screen without any internet connection**

We used localStorage to cache the user’s saved routes’ directions. This allows the user to access them when they have no connection. Our app is heavily reliant on Google Maps API, so we are unable to save map data for offline use (it is illegal unless we buy a license) However, we can still provide the directions that we have computed when online. This fits the users’ expectations because being a browser and maps application, the common understanding is that a search for new data requires internet connection, and only manually saved data are retained. While a user may hope that the map will show when offline, it should be accepted that most apps refrain from downloading large sections of maps. For example, Google Maps’ save map feature has size limits as well.

**Milestone 12: Explain how you will keep your client synchronised with the server. Elaborate on the cases you have taken into consideration and how they will be handled.**

The client will attempt to sync with the server by uploading local route data, and caching it if it fails. For example, if the internet connection fails while sharing a route, that route will be cached for upload when the internet connection returns.

**Milestone 13: Compare the advantages and disadvantages of basic access authentication against other schemes such as digest access authentication, cookies, or OAuth. Justify why your choice of authentication scheme is the best for your application.**

In our application, we chose to use Facebook authentication because we wanted to make use of our users’ Facebook friends’ data. Our goal was to eventually let users post a route to the user’s Facebook page for sharing, which would require the user’s Facebook access token. We also considered the possible need to use the friend-list data to implement permissions for shared routes. Since we already intended to use Facebook for its features, it made sense to use its authentication for our application, rather than implement a separate authentication system that may complicate the user experience.

*Advantage of basic access authentication against OAuth:*

* Simple to implement and does not require exchange of information with another server.
* Authentication works as long as the application’s server is up and running.

*Disadvantage of basic access authentication against OAuth:*

* Basic access authentication is vulnerable to third party attacks where an attacker can sniff the user’s credentials sent over the network and thereafter masquerade as the user with the user’s credentials.

**Milestone 14: Describe 3 workflows within your application. Explain why those workflows were chosen over alternatives with regard to improving the user’s overall experience with your application.**

1. Planning a new route:
   1. App initialises to main view, showing map, header bar, and 3 buttons at the top
   2. User taps on “Plan” button >> form pops up
   3. User fills in start locations, end locations, selects “Fastest Route” option
   4. User submits form >> form hides, map shows route and a route details bar appears at the bottom of the screen

An alternative was to initialise the app to the plan route form, which when submitted would show the map with the route. This would reduce the number of steps required by 1, and channel the user’s actions directly to planning a route. While it could make for a shorter workflow, we feel that it does not provide for a complete UX. Showing the user the empty map first gives them a hint that the map will be filled with a route, giving them a visualisation of what to expect and what to do. Also, it obscures the other features that we want the user to know we have, such as the share and save functions, which will most likely be afterthoughts. This could give them the impression that our only feature is routing, causing them to miss the other important features that could benefit them.

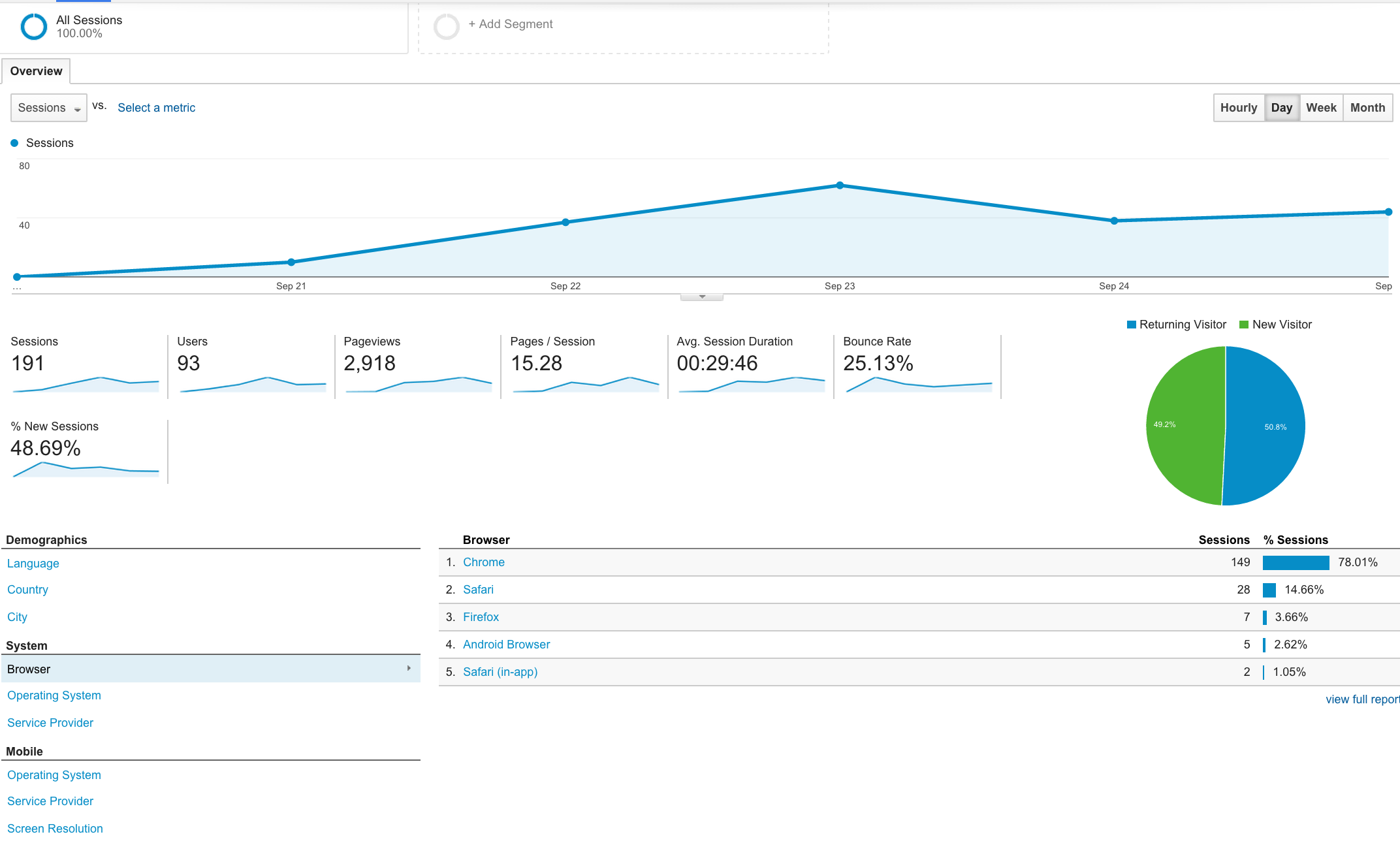
1. Finding a route to join
   1. App initialises to main view, showing map, header bar, and 3 buttons at the top
   2. User taps on side menu button on header bar >> side menu appears
   3. User taps on “Browse Friends’ Routes” >> view switches to friends’ routes list
   4. User taps on one of the friend’s route cards
   5. Success dialogue appears

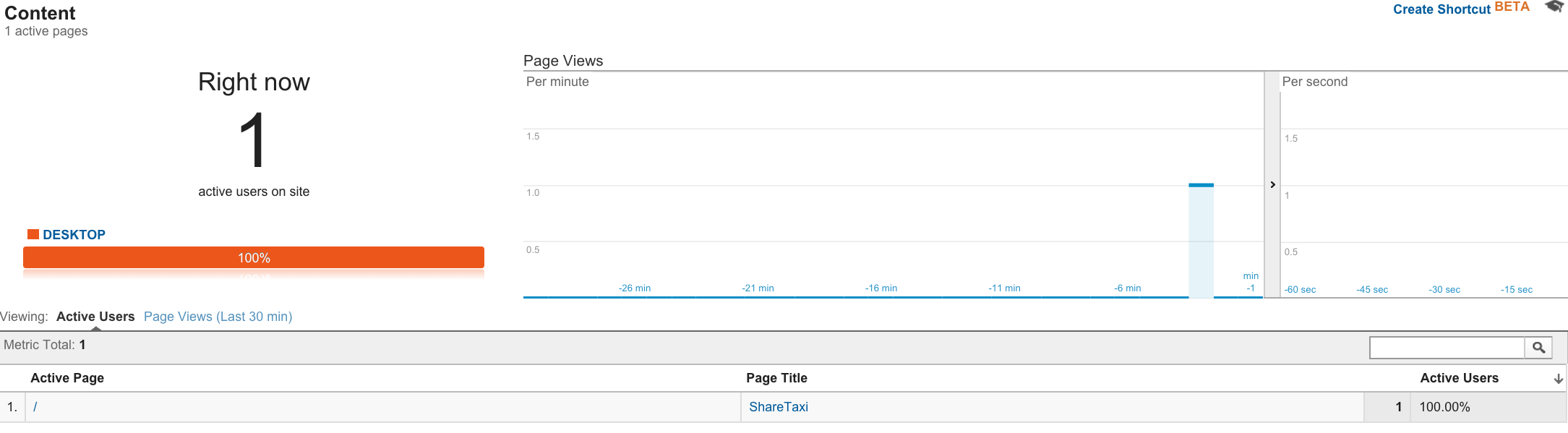
Our initial plan was to place this feature within a button on the main view, beside Plan and Share - along with other buttons. This would minimise steps and maximise visibility of each feature. We very quickly realised 2 issues with this: first, the bar had limited space, and second, using icons only was not enough to convey the purpose of the features of our app. It would defeat the purpose of showing a feature if the users could not understand it. We found that with a single and icon word per button, we could fit 3 buttons on a single bar, which we felt was the compromise that allowed users to understand our app and have the minimal number of steps for the more important features. Upon further inspection, we also realised that the other functions (those in the side menu) tend to be more complex and require multiple views, which we felt would be more clearly represented if they had their own “section” which was navigable to via the side menu. For a consistent user experience we placed all views that contained lists within the side menu.

1. Checking out routes the user has joined
   1. App initialises to main view, showing map, header bar, and 3 buttons at the top
   2. User taps on side menu button on header bar >> side menu appears
   3. User taps on “Routes I’ve Joined” >> view switches to relevant list
   4. User taps on one of the route cards >> view switches to map view, showing the route and route details (similar to in planning a route)
   5. User peruses route information as required

We considered allowing users to set routes to a finalised state, at which they cannot be modified further. All of the finalised routes that the user was in, regardless of whether they shared it, would then be placed in a single section. On one hand, it seemd viable, and could possibly allow for features such as a quick view of the upcoming routes by time. However, we felt that from a user perspective, mixing the routes up could be confusing. Moreover, there was also the possibility that a given route would not be ‘finalised’ by the owner, which meant that the user would not see it in that finalised section. This could lead to some trouble and frustration for the users, which we wanted to avoid at all costs.

**Milestone 15: Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.**





Optional Milestones

**16: Identify and integrate with social network(s) containing users in your target audience. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)**

We integrated with Facebook, because we wanted to allow the user to share their routes there. We chose Facebook because of its familiarity to both us and the users, and we felt that our app fit in with the the privacy settings and general feel of media within the network. We wanted a sort of open-collaboration within all their friends, which would benefit us as well by attracting more users. Its popularity is hence also a big plus point.

**17: Make use of the Geolocation API in your application. Plot it with Bing or Google Maps or even draw out possible routes for the convenience of your user. (Optional)**

Geolocation and Google Maps API were used in our application. We plot the user’s current location with the green marker, and draw out routes by passing input locations to Google Maps API.

**18: Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. (Optional)**

We used AngularJS with Ionic. We felt that this combination could allow us to do the native look and feel we wanted fairly easily, using html-like syntax and injection via directives. On top of being the foundation for Ionic, we felt that the AngularJS was very powerful and useful in ways we needed. Data binding was particularly handy, especially when retrieving data to display, and repeating html elements using ngRepeat was extremely useful. An alternative would have been ReactJS, a supposedly faster but more barebones framework. However, we felt that it did not suit our needs as well. We read that AngularJS had better support for HTML and a more mature community, which would help the front end developer who was new to such frameworks to pick it up faster. Moreover, one of our team members had experience with AngularJS already, and given that our team has one less member, we felt that it was better to choose something that at least one member was familiar with, rather than a framework new to all of us. Overall, we felt that while AngularJS may not be the perfect framework, it was the best choice for this project given the conditions.