Max Maybury
Julian Contreras
Yonglin Wu
Shidan Xu

6.170 Software Studio
Fall 2014
Hiproute
Revised Design Document (Changed are identified in blue)

Motivation

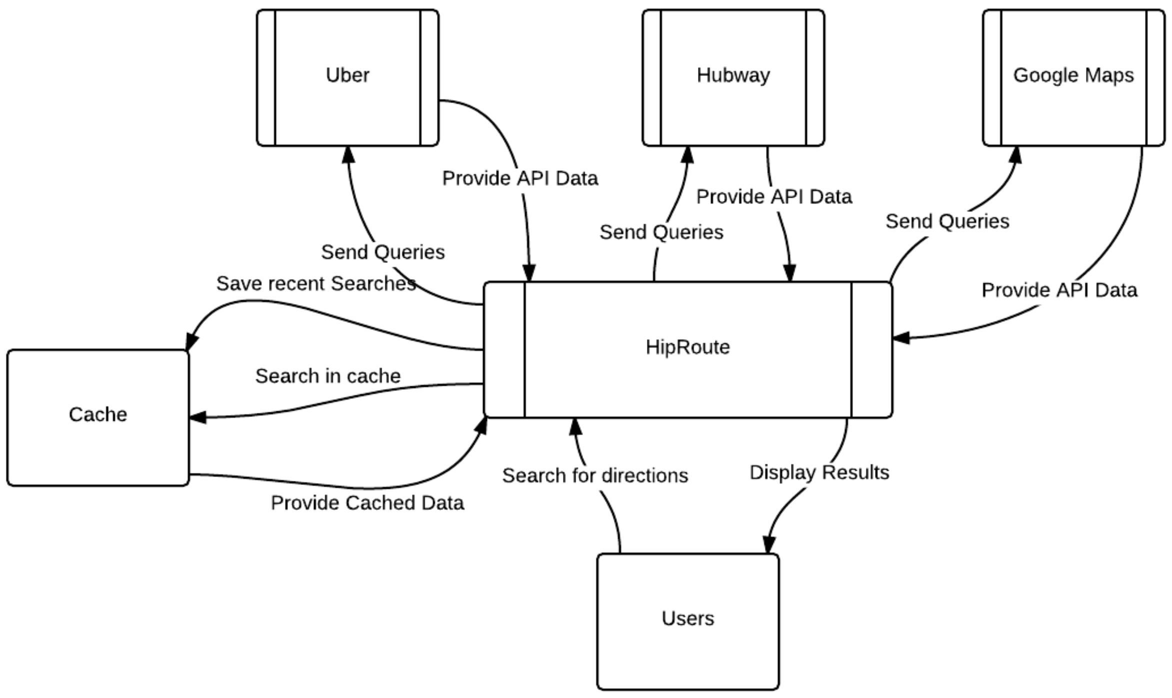
Hiproute is a web platform that aggregates and synthesizes searches for directions across multiple sources and methods of travel. Hiproute estimates the distance, time, and cost associated with each route, and gives a recommendation to the user.

While Google Maps, the unequivocal leader among directions providers, is a very useful tool, it lacks access to information from Hubway, the local bikeshare system. Furthermore, Google Maps does not give users access to cost information. These factors can have a big impact on a user's travel choices.

Purposes:

- **Eliminate the need of users to search on multiple platforms when comparing transportation to go between places in the city.** Many times when users compare ways to go from point A to point B in the city, they have to search on multiple platforms, e.g google maps, uber app, and hubway map. Our app aggregates the search results and display them together in a simple way, which simplifies the search and comparison process for the user.
- **Allow users to take both money and time into account when comparing transportation in the city.** People value their time and money in different ways. Users would likely be more cost-conscious during the peak travel hours of the day while wanting to arrive as quickly as possible during off-peak hours. Each search could potentially be different in terms of preferences.

Context diagram



Concepts

Agony - the weighted evaluation of a particular transportation choice focused on cost/time. This satisfies the second purpose: users have various major concerns when evaluating a transportation choice.

Search - combines multiple searches of different transportation together, and returns a collection of routes that are keyed on time or cost. This satisfies the first purpose: eliminates the user's need to search on multiple platforms.

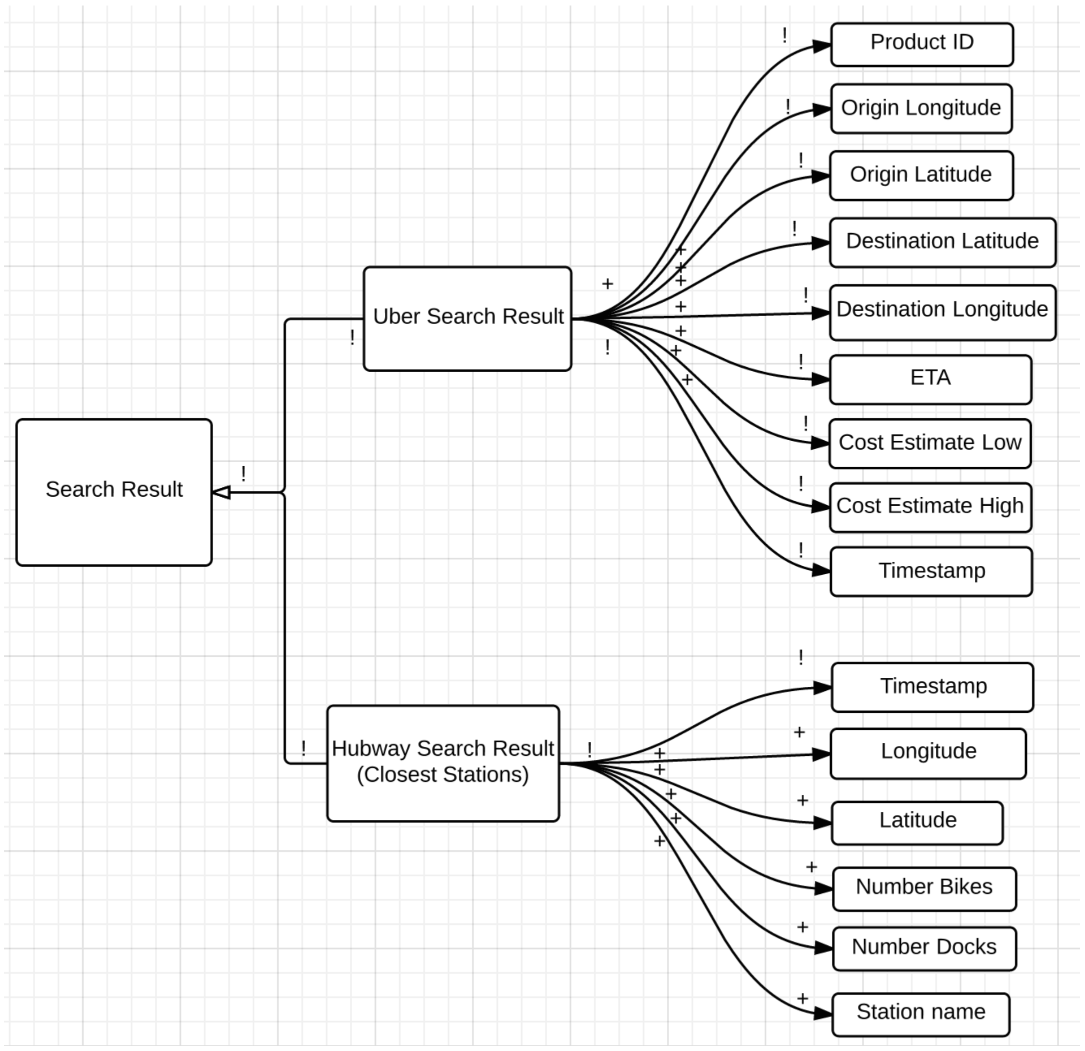
Route - A route is a path going from point A to B using some transportation method. For same point A and B there can be several "routes", each for each type of transportation.

Providers: Ways of getting from point A to point B. In Hiproute there are four providers, public transportation, Uber, hubway and walking.

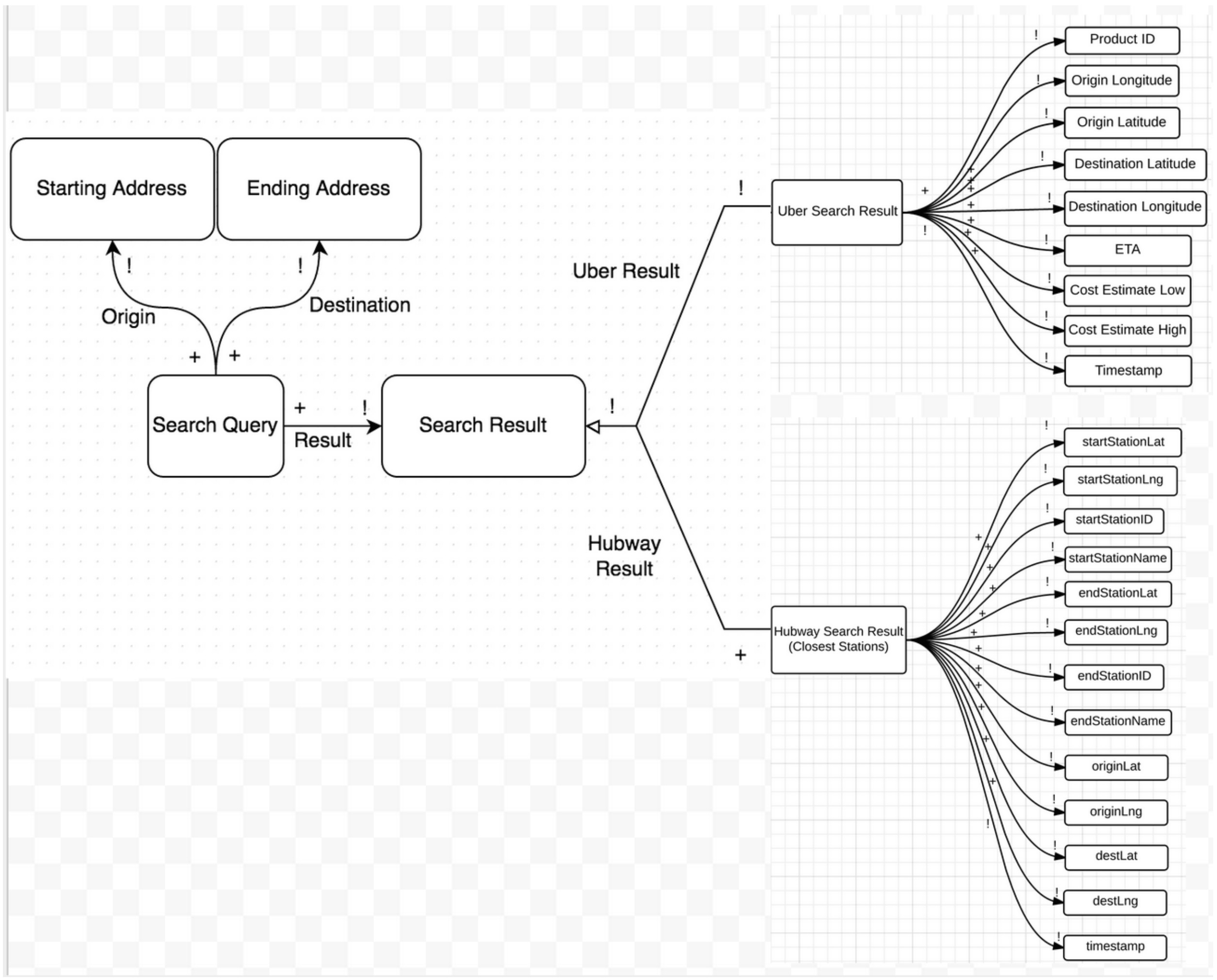
Data model

1. Note that we are only keeping track of things we store in the database.
2. We applied design moves from old data model from 4.1 (Figure 1):

a. Restructured the data stored in the Hubway search result so that now the result specifies the starting and ending location of the query, and of the stations.
b. As per discussion with the TA, we added an element of search query, which takes in a starting address and an ending address, and returns a search result back. (see figure 2)

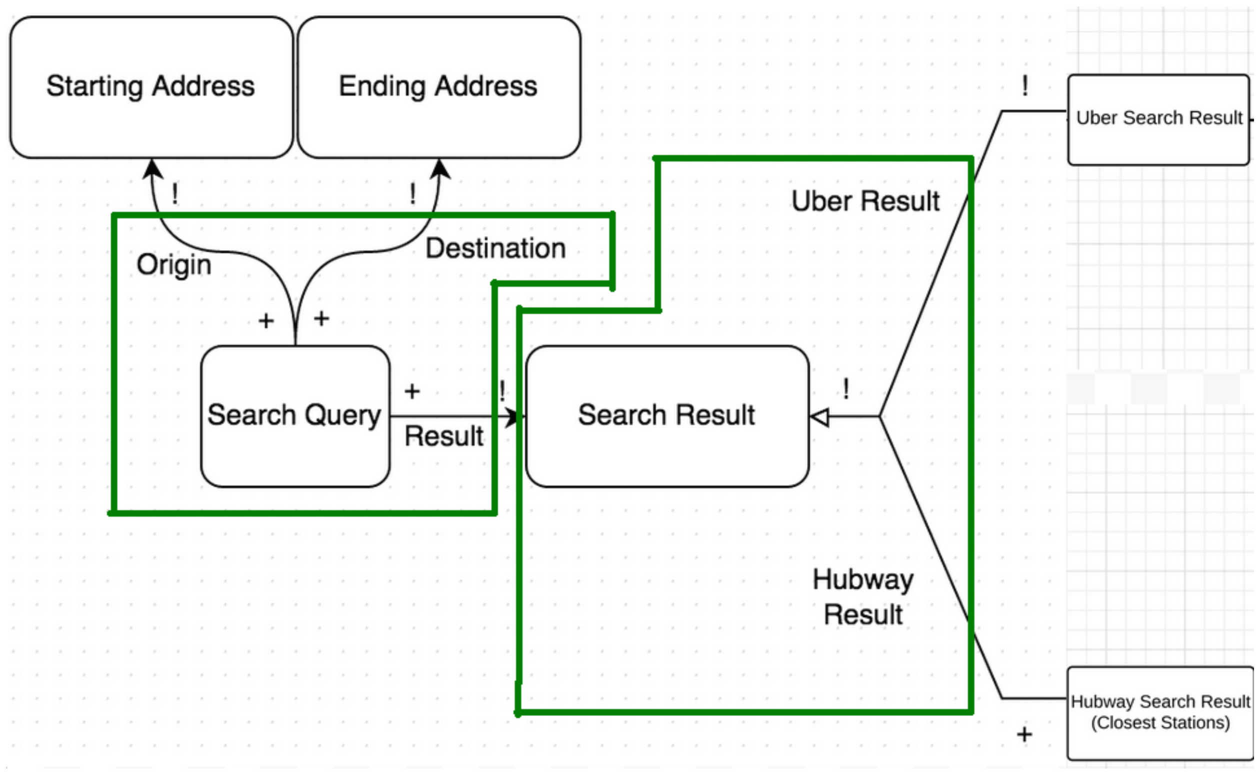


(Figure 1. Old Data Model)



(Figure 2. New Data Model)

- Contours of the Data Model: (Figure 3)
    - A search result contains an Uber search result and multiple Hubway search results (by station).
    - A search query contains a starting address and an ending address, and contains a search result.



Search Query: {Origin, Destination, Search Result}

Search Result: {Uber Result, [Hubway Result]}

(Figure 3: Contours)

Security concerns

In general , for Hiproute, security is not a major concern, because all the search results we display to the user are publicly available.

Summary of key security requirements

- Past search history is stored internally in server and no users have access to it.

Threat Model

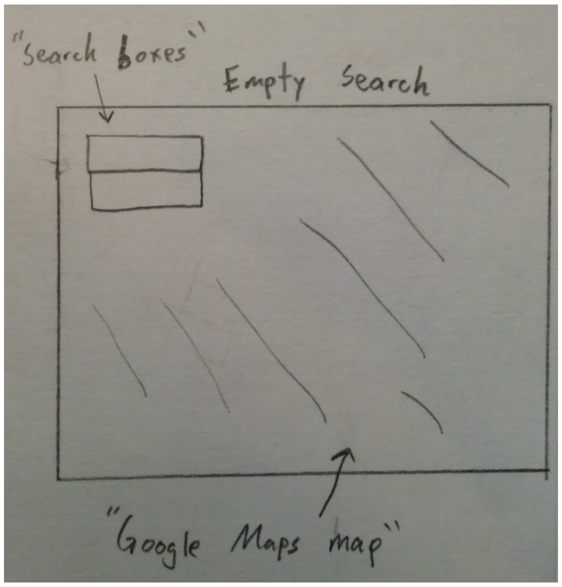
- We assume that google and uber data are reliable over an https connection
- We assume that any vector of attack will originate from a malicious api call or malicious input into the search area.
- Can assume no interest from state actors or criminal syndicates, since no profile information stored (name, email, credit card).

- Since this web service is completely free and there is no money transaction between any parties, there is very little risk of fraud.

Potential risks and mitigations:

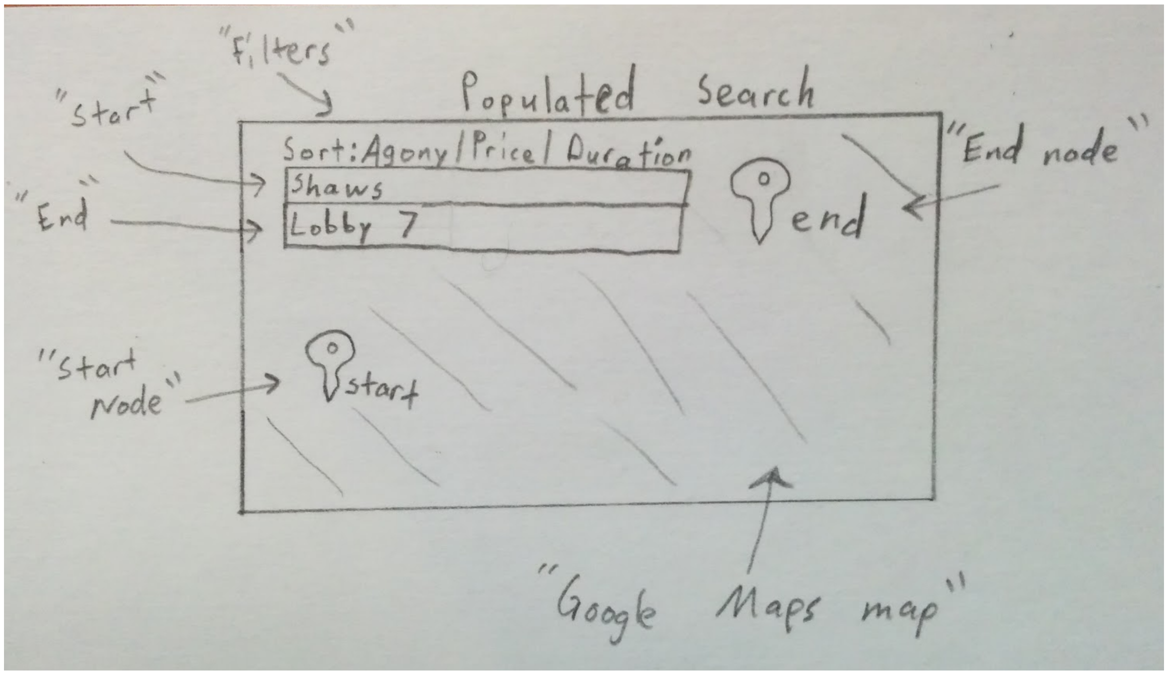
- Hackers break into the system and get access to all past search history on Hiproute. Our resolution is that Hiproute does not expose public API to access past search history, and database is hosted on the server with basic security, which is fairly hard to get through. Also Hiproute deletes all serach history that is older than 30 minutes, which reduces the amount of data stored in database. Even in the event where hackers have access to search history database, it is not a huge threat to our users as Hiproute does not store where the searches are coming from, so it is very difficult to infer the identity of the user who made the search. Also the information the users provide: address of places they are going to, are not usually sensitive information. On top of that, Hiproute only stores search history from the last 30 minutes.
- Another potential hack is the man-in-the-middle attack, where a hacker tries to redirect the user to a place that is different from where the user searched for. Our resolution is to use https to prevent wrong directions returning to users.
- Code injection into search query area. Our resolution is to check the sanity of searches before processing them.

Wireframes

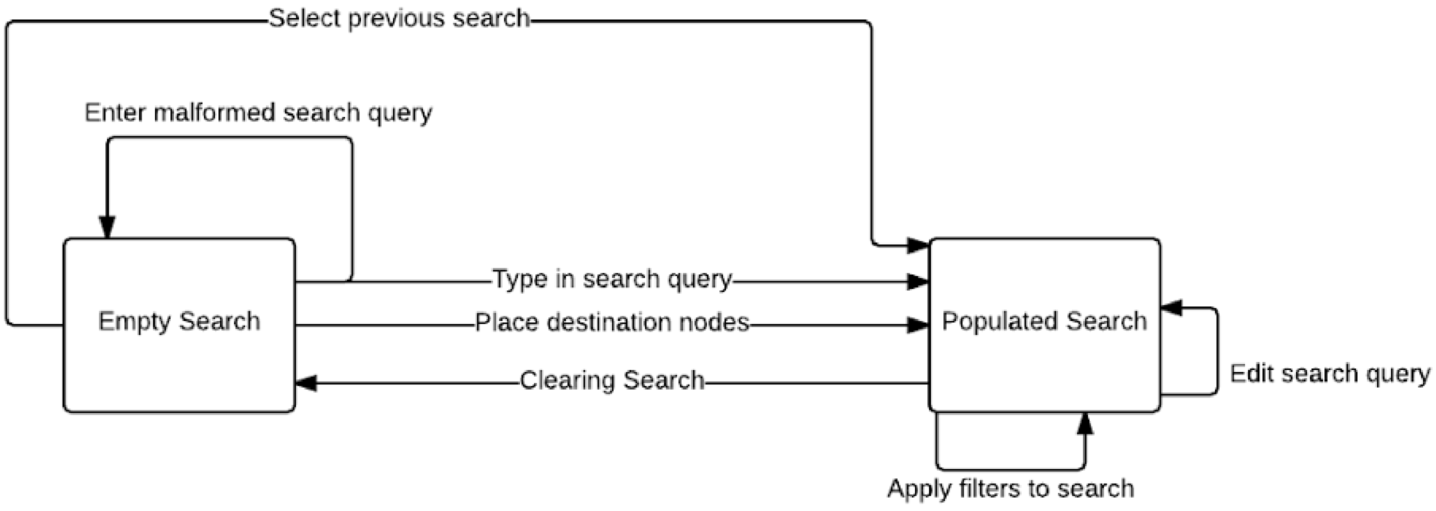


Empty Search

Populated Search



Flow chart



Design challenges

Issue: How much past search history data does Hiproute cache? Hiproute caches all users' search results in database for a short amount of time. If one of the APIs are down and a user makes a search on Hiproute, Hiproute can try to find a similar search in database and returns the cached results. The questions is how much search history data Hiproute saves.

Potential Solutions

- Save all search results, including google API, uber API and hubway API. In this cases, Hiproute still works even when all three API providers are down.
- Only save uber and hubway API data. This reduces the amount of data to store, but it makes Hiproute dependant on Google.

Resolution:
For the cache, we decided on just caching the Uber and Hubway api data instead of the google api data. This was done because Google's api is more complex and has a very high uptime. On the other hand, Uber's api is simpler and changes less often, a better candidate for caching. Another reason for not caching Google api data is that a Google api outage ultimately means our whole site is unusable anyway.

Issue: How does Hiproute let user choose their preference of time and money? Hiproute ranks search results based on combined "agony" of time and money. In user interface, how do users indicate their preferences?

Potential Solutions

- Have two tabs, one for "prefer less money" and one for "prefer less time", and users can choose either one.
- Have a slider from "time" and "money so that user can choose anything in between.

Resolution:
A person may have a preference over time vs. money, but its not one extreme or another. So we plan to include a slider (in the final product) for the user to change the weight they give to each factor. When we calculate our agony, we will adjust the numbers accordingly to the desired weights.

Issue: How should the UI look like?

Potential Solutions

- resemble Google maps, with map overlay on entire page.
- Resemble Hipmunk, only shows the routes info but not the map
- have a 2-column screen, left side for search and right side for map.

Resolution:
We chose to use the Google maps approach, overlaying the search results on top of the map. We first rejected the Hipmunk approach because we are showing the users routes, unlike air travel, the user wants to see how he reaches there. We then chose the Google maps approach because the map is a crucial map to our UI, having a bigger map can let the user understand where he is going to a higher accuracy.

Issue: Should we include a user?

Potential Solutions:

- We should have a user. Having a user enables us to remember the past searches the user makes, which could potentially be cached. There is also potential to make it a social map where users going to the same location can maybe help each other out or go together.
- We should not have a user. Having a user introduces an extra layer of authentication, which is unnecessary for a mapping app. The same user is possible to make two same searches, but it is also very likely that the user does it in short period of time. Hence we can cache by time instead of by user.

Resolution:
We chose not to have a user. Based on the purposes we wrote down, there is no incentive for the website to include any social features. The website should be an easy go-to for people checking out ways to travel; hence the extra layer of authentication slows down the process and is extraneous.

Code Design Challenges

Issue: Should we call Google API, uber API and hubway API from server or from client?

Potential Solutions:

- Call all APIs from server side, server will provide its own API for client, which can be simpler. Server can call google, uber and hubway API, aggregate the results, sort the results and return to client in one package. By putting some computation on server it can improve client side speed and it enables us to save cached results on server in the case when uber, google or Hubway APIs are down.
- Call all APIs from client side. Essentially everything can be done on the client side. Client side can call google, uber and hubway directly, and all the sorting of different options can be done on the client side. This greatly simplifies the architecture of the app by eliminating all server side code. Another advantage is that since all sorting is done on client side, client does not need to call server every time the sorting preference (user's preference of time or cost) changes. On the other side, since everything is on client side, client side can be slow. Also we have no way to save

cached search results, so anyone of Google, Hubway or Uber API is down, the app stops functioning.

Resolution:

We choose a combination of both. Since the client side is using Google map javascript API, it is calling google map API already. Having server receives calls for Google API and then forward it to google API is just an unnecessary step. At the same time displaying routes and markers are more straight forward Also we are doing sorting of the routes because users can change preference of time/money on a slide bar (on client side). If sorting is done on server, every time a user drags the slide bar, client has to call server again to update the ranking of routes, which is potentially very costly.

For Uber and Hubway, client will call server, and server will call uber and hubway API. The reason for that is that we want to store a cache of Uber and Hubway API search results in case Uber or Hubway API is down. The search results cache will be saved for 30 minutes. For uber and hubway API call, server is doing minimal computation other than aggregate several API calls to Hubway and Uber to return to client in one API call, since ranking of different uber options is done on the client side.

Issue: Location lat longs are required to make a uber request. Should we get the lat longs from Google's geocoder, or from a search result?

Potential solutions:

- Use Google's geocoder API, pass in an address and get the lat long back.
- Use a search by Google, use the returned marker's lat longs.

Resolution:
We chose to do a search on the address and return the marker's lat longs. Since the search box has an autocomplete feature, the address we get from a Google search is more accurate, compared to directly calling the Google geocoder API with partially filled location info. We are not making an extra call because we are calling the search to display the origin and destination markers anyways. It's a convenient win-win.

Issue: How should the separate Uber api calls be bundled to avoid "callback hell"?

Potential solutions:

- Nest each callback within the same method and return on the last callback
- Modularize each callback and call each on the same nest level in one method

Resolution:
We choose the obviously preferable second method and this allows fine grain debugging on each uber api call and is much easier to read. Errors in one method are prevented from cascading and effecting the rest of the callback code.

Issue: The route display for Hubway is much more complicated than for other searches.

Potential solutions:
- Ignore walking from location to station, use Hubway terminals as destination and origin
- Display all routes as Google Maps Bicycling
- Overlay 3 legs: walking to origin station, biking to destination station, and walking from destination station to final destination

Resolution:
Each leg of the trip should be shown independently. This will certainly cause for more complexity with map interactions, but will provide a more accurate result for the user.

Issue: Transit cost estimates are not accurate for very long distances.

Potential solutions:
- adjust to pricing by distance
- find hard-coded values
- find distance maximum and do not calculate prices further than that

Resolution:
Transit values for long distances will be provided to the user, but we will notify the user that these estimates may not be accurate, as they are intended to be used within a small radius from Boston.