Locus

Project report

CSE 636: Data integration

Project advisor

Dr. Jan Chomicki

Prepared By

Abhishek Pimple

Adhip Vihan

Introduction

Location aware applications are useful to provide many useful services to the user. Most of the users are familiar about the location they live in. That is they know about what are the best places to eat, what are best things to explore, which events are happening around and how's the current weather like. Using traditional search, user can search the above items of interest about any particular new location using a separate search queries for each of them. There is a need for the service which can gather the required data from different sources about the particular location and combine the results in a single page using unified interface before showing them to the user.

In project Locus, we are strongly focussing on location our users are interested in, hence the name Locus (**Loc**ation foc**us**). This is the mashup project in which we have solved data integration query problem. Here the query from users is the location they are interested in and the result is the relevant things of interest about that location. We are gathering data from disparate sources using API endpoints provided by services from those sources. Data obtained from the above disparate sources is consolidated and then returned as a query result to the user on a single page using unified interface.

In this report, we discuss previous related work in the domain of location search and what motivated us to pick up this problem and solve it. We also discuss about different problems that need to be addressed and how we have solved those problems using our system design. We have demonstrated our system architecture and justification of the design choices we have made to solve the data integration query problem. We then conclude with the experimental evaluation of our system.

Previous related work

Previous such work and their limitations are as follows:

- Google with Google Now: Google now provides the user with the weather information about the place, as well as with the hotels and events information. But there are certain limitations associated with it, which are
    - It doesn't allows the user to search by zipcode, this issue has been taken care of in locus.
    - It doesn't provides a single page coherent solution. For example to look for hotels, it basically opens up a new search page to search for hotels in the area.

- Tripadvisor: it provides the user with places of interest and places to eat, but the one has to navigate through the page & each category has a separate webpage associated with it. It also doesn't provides the weather information.

- Yelp: It provides the user with the places to eat. But like all the other options discussed above, it is domain specific and doesn't provides much information about the place like weather and events etc.

Motivation

In the previous topic, we discussed about previous related work in this domain and their limitations. If we had to define our motivation in one single line, it would be like this.

*"To build a **single application** to provide **relevant data** pertaining to user entered **location** on **good user interface**"*

Above single line epitomizes our motivation behind implementing this project. There wasn't a one stop solution which displays all the relevant data to the users once they enter the location they are interested in. The **single application** denotes it is a mashup project with single unified interface for showing query results. **Relevant data** includes weather forecast, interesting places to explore, best places to eat and events happening around the user entered **location** which can either be city name or zipcode. **Good user interface** is something which is modular and self explanatory to the end user.

Problem to be addressed

Unfamiliarity with the location

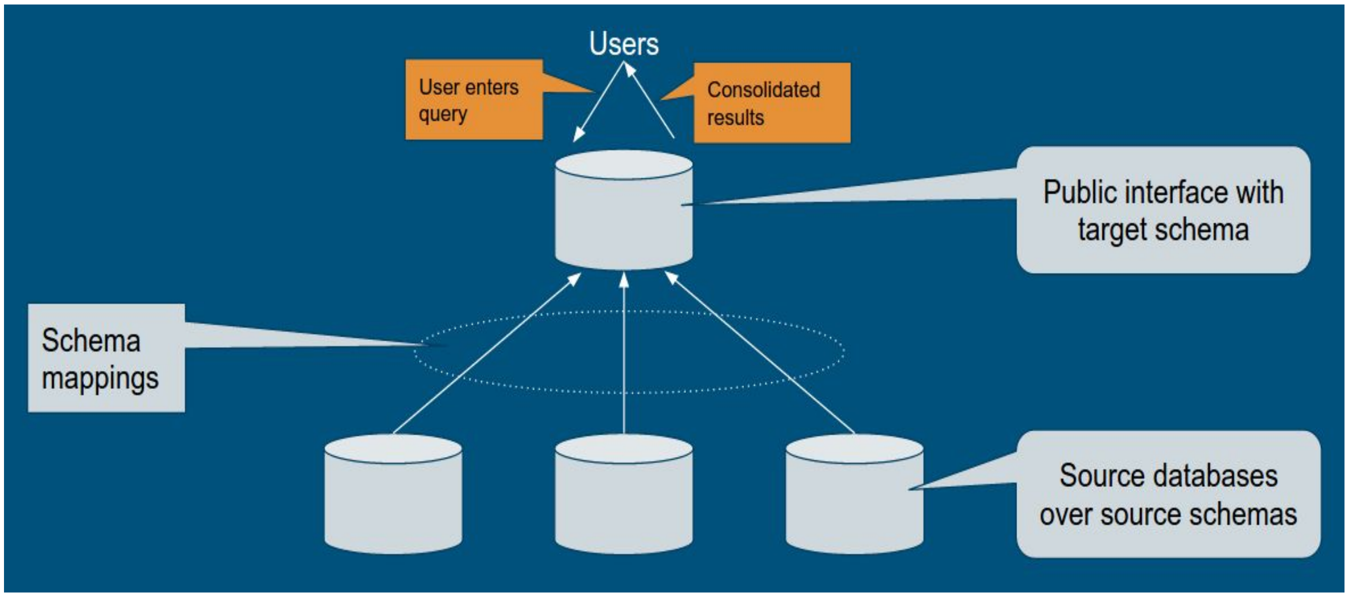
Most of the users are unfamiliar about many locations. Many times users might not know a complete location. For example, user might only know a certain city which starts with 'Buf'. Also some users might only know postal codes about certain location and not the city name or state name. In this case, there is a need to address the problem which demands search from different granularities of the location so that users can search either by using city name or by postal code, whichever they are familiar with.

User wants to explore things of relevance

Even if the user is unfamiliar about the new location, he wants to explore certain relevant things pertaining to that location. This relevant things can have indirect impact on users interest in that particular location. This relevant things includes current weather conditions and weather forecast, interesting things to explore, best places to eat and what are the popular events that are happening near that location. The above set of data can certainly act as an aid to the users interest in that location.

Data integration query problem

Data integration query problem is a very known problem in the domain of data integration.


Users
User enters query
Consolidated results
Public interface with target schema
Schema mappings
Source databases over source schemas

In this problem, user queries a public interface which also holds the target schema. This public interface then gathers data from disparate sources which can be holding source data in different source schemas. Source data from different sources is then mapped to the target schema using schema mapping. Once data is mapped to the target schema, it is returned to the user in the form of consolidated results.

Technical contribution

We are basically integrating data from four different sources here and displaying it to the user. Here is what we have contributed:

Considering Granularity of the location

Lack of granularity of location while searching as a major limitation of the previous works in the field. We have implemented search by the city name as well as the search by the zipcode in our application, providing user with the choice of input and thus making app usage flexible.

Auto-suggest feature

We have also implemented the auto suggest feature which dynamically suggests the possible city names when the user starts typing into the search bar.

Providing the one stop solution

We have provided the one stop solution which sticks together coherently in a single page. Unlike the previous works, all the relevant data which can be used for planning the trip and is within the scope of application is displayed on a single unified interface.

Solving Data Integration Query problem

We have solved a version of data integration query problem through Locus. Data is being fetched from multiple API's and cleaned, formatted and transformed to a presentable format to the end user. Queries entered by user are run against the database and query term is being used to query relevant APIs.

Basic concepts and definitions

MVC Framework

Locus is powered by MVC framework which stands for Model, View and Controller. Model takes cares of the data, View decides how data will be displayed to the user and Controller act as a mediator between Model & View. Controller contains the business logic for the application.

We have used following technologies:

1. Node.js for the backend
2. Angular 2 for the front-end
3. Mongo DB for storing the data.

Node.js uses asynchronous model of programming & hence provides fast response. **Asynchronous programming** is a means of parallel programming in which a unit of work runs separately from the main application thread and notifies the calling thread of its completion, failure or progress.

Node.js provides service such as fetching place information from Mongodb in the backend. It accepts GET/POST requests and provides response to the client.

GET request is for retrieving data. It allows the users to request the same URL over and over harmlessly. **POST** is for writing data. The data is in the body of the request instead of the URL. Locus uses **HTTP GET** request to fetch the data from the API.

Application program interface (API) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact.

Input to the API is the consumer-key and the request parameters. For example the eventful API that Locus uses for fetching the event data has below format

*http://api.eventful.com/json/events/search?**app_key**=XXXX&**q=music**&**l=buffalo***

Here there are three parameters that are being provided for this http request:

app_key: This is the consumer key which is provided by eventful.com when the developer registers for using it’s API services.

q=music: This parameter specifies the type of the event. One can provide it with specific values if we want our events to be filtered by some category. Like here the value is **Music.** This means that the API will only return Music related events.

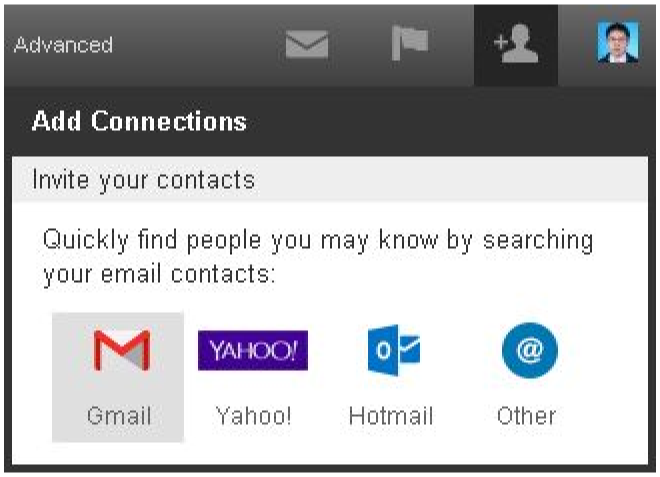
l=buffalo: This parameter specifies the location for which the data is requested. We can provide here with the name of the city as well as the zipcode.

For some component like the Yelp API, things aren’t as simple as providing a Key in the URL and getting the results. They uses more advanced techniques of authorization like OAuth 2.0 for user authentication.

OAuth 2.0

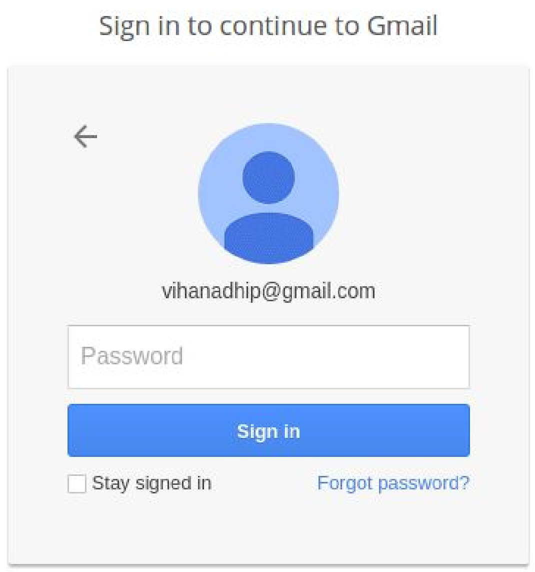
OAuth 2.0 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as yelp. Here is an example of how that works

1. I login to LinkedIn and want to connect some friends who are in my Gmail contacts. LinkedIn supports this, so I click this button:



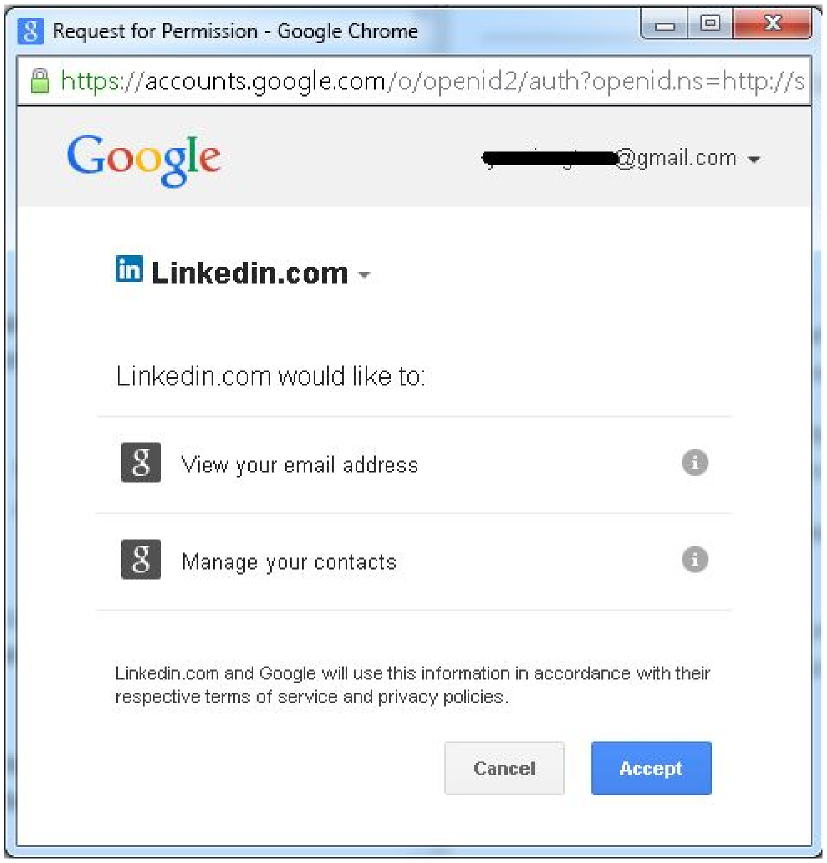
Quickly find people you may know by searching your email contacts:

2. A web page pops up, and it shows the Gmail login page, when I enter my account and password:


Sign in to continue to Gmail

vihanadhip@gmail.com

3. Gmail then shows a consent page where I click "Accept":



Request for Permission - Google Chrome

https://accounts.google.com/o/openid2/auth?openid.ns=http://s

[redacted]@gmail.com

Linkedin.com

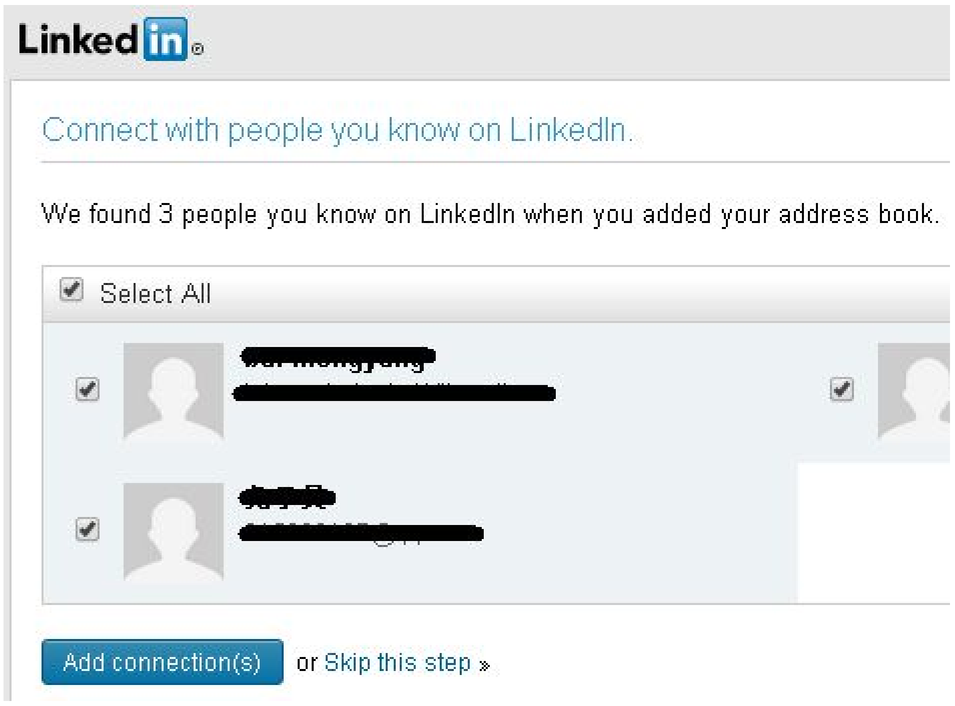
Linkedin.com would like to:

View your email address

Manage your contacts

Linkedin.com and Google will use this information in accordance with their respective terms of service and privacy policies.

4. Now LinkedIn can access my contacts in Gmail:



Connect with people you know on LinkedIn.

We found 3 people you know on LinkedIn when you added your address book.

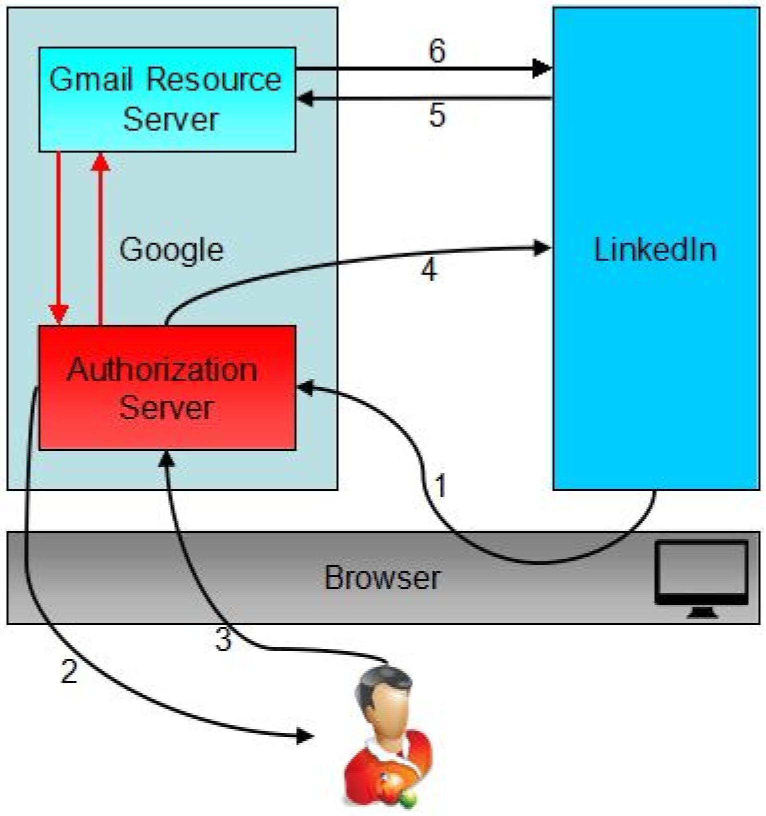
[redacted]

[redacted]

[redacted]

[redacted]

Below is a flowchart of the example above

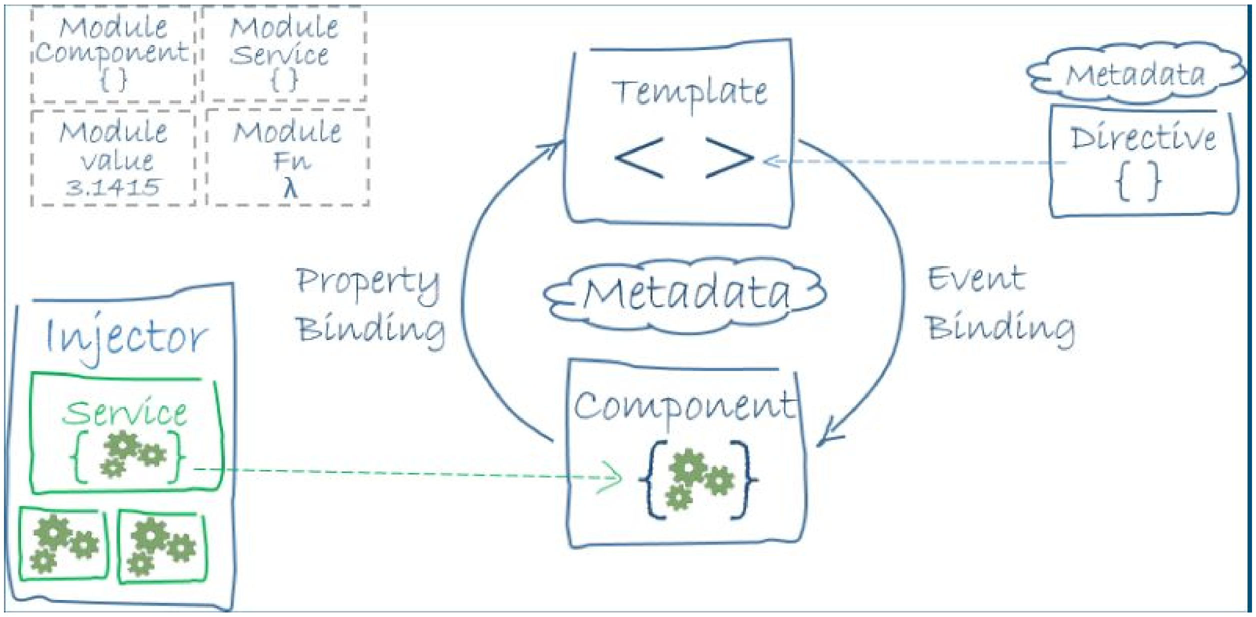


Here is what is happening behind the scene [8]:

1. LinkedIn requests a token from Gmail's Authorization Server.
2. The Gmail authorization server authenticates the resource owner and shows the user the consent page. (the user needs to login to Gmail if they are not already logged-in)
3. User grants the request for LinkedIn to access the Gmail data.
4. the Gmail authorization server responds back with an access token.
5. LinkedIn calls the Gmail API with this access token.
6. The Gmail resource server returns your contacts if the access token is valid. (The token will be verified by the Gmail resource server)

Angular 2.0

Locus is divided into sub-components like weather component, event component, places to eat and places of interest component.



In Angular 2.0, a component is a special kind of directive that uses a simpler configuration which is suitable for a component-based application structure.

Components are integrated into one bigger component which is **app component** using directive. There are 3 types of directives in angular 2

- Components — directives with a template.
- Structural directives — change the DOM layout by adding and removing DOM elements.
- Attribute directives — change the appearance or behavior of an element.

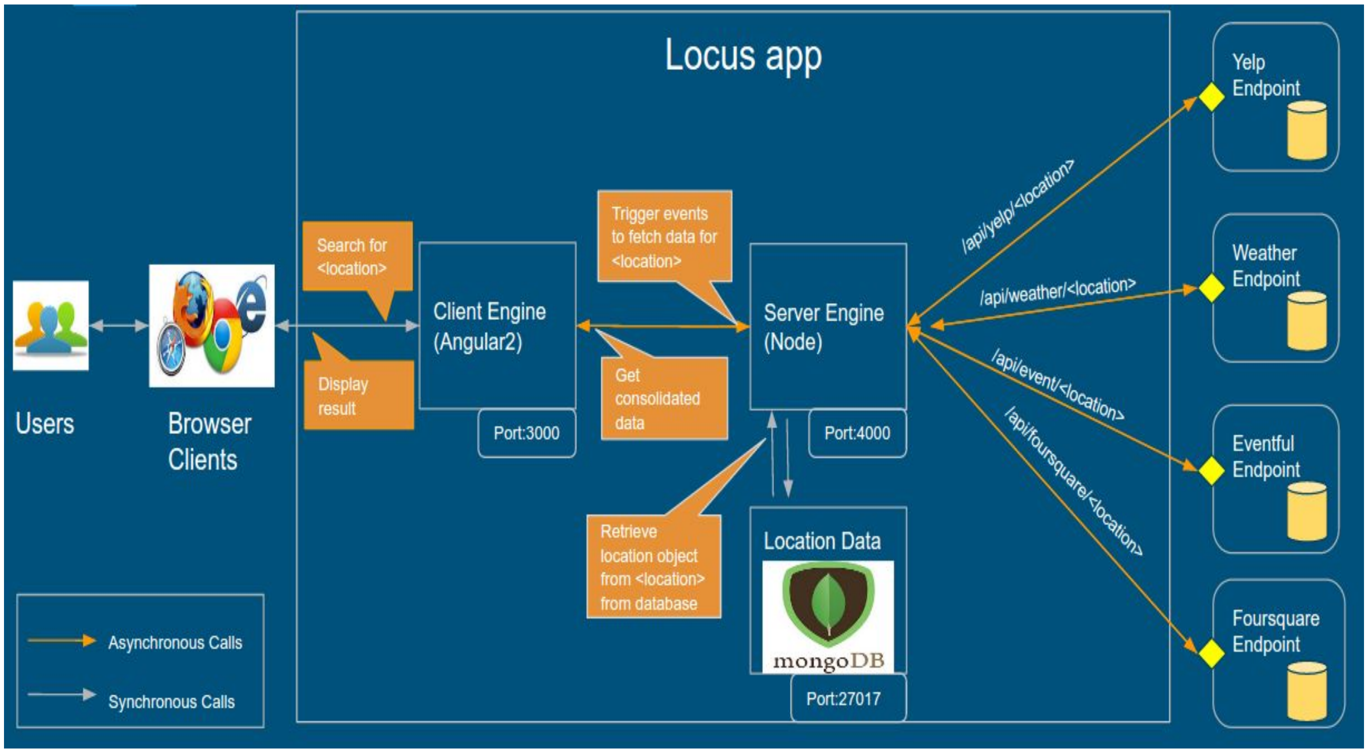
Each HTTP call that is being made is done using **Services.** Services are javaScript functions that are responsible for doing a specific task only. Angular services are injected using Dependency Injection mechanism and include the value, function or feature which is required by the application.

Methodology

System architecture

On a logical level, our system can be divided into three parts viz.

1. Client Engine
2. Server Engine
3. Mongodb database



Once client engine receives query for the user entered location, it triggers different events in server engine to gather relevant data. Server engine asynchronously calls API endpoints of different sources like yelp, weather, eventful and foursquare. Once the server engine gets data from the above disparate sources, it is mapped to the target schema of the result page. This consolidated result is then returned to the client engine which in turn returns the HTML content to the browser. This HTML content is rendered in the browser of the user.

Datasets

1. Data acquisition

We have used the location dataset which we have obtained from simplemaps.com [1]. The dataset is in CSV format like below

zip	lat	lng	city	state	zcta	parent_zcta	county_fips	county_name	military
58341	47.7753	-99.85	Harvey	ND	TRUE		38103	Wells	FALSE
62972	37.537	-88.7913	Ozark	IL	TRUE		17087	Johnson	FALSE

We can see that the above data needs some cleaning and transformation. For example, parent_zcta column is empty, so we need to remove such entries before storing in the database.

Also there exists only state codes and not the entire state names. Therefore we have used state code to state name mapping from Wikipedia for generating state names in each respective entry in the above data. For example, we get “North Dakota” as a state name from state code “ND”.

2. Data cleaning

As discussed in the data acquisition step, data set which we have obtained requires some sort of cleaning.

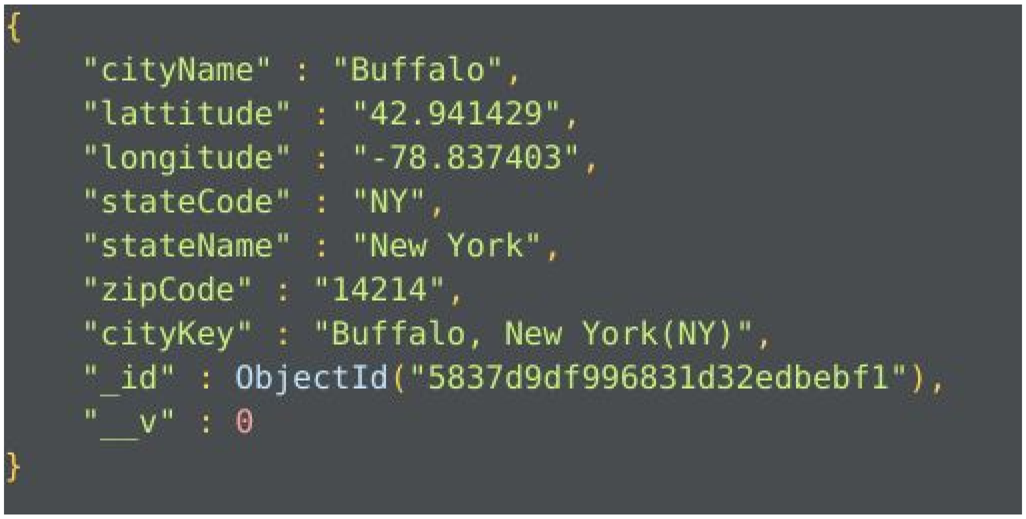
Empty values: In some entries, data was not present in zipcode and parent_zcta columns of the above dataset. To create consistent and reliable database, we implemented Java program to omit such entries with empty values.

Invalid data: Some entries consists of syntactically invalid data which does not makes sense for the particular attribute of the location. For example, latitude entry with value TG35.745. We implemented Java program to remove such invalid entries from the dataset because it could throw runtime error when we try to get some data from an API by providing latitude and longitude pair and one of them is syntactically invalid.

3. Data transformation

Once the data is cleaned in step 2, it need to be transformed from CSV format to JSON format so that we can store it in our Mongo DB database. We implemented Java program to transform each entry in CSV file to JSON object.

Example of a JSON object in Mongo DB is below

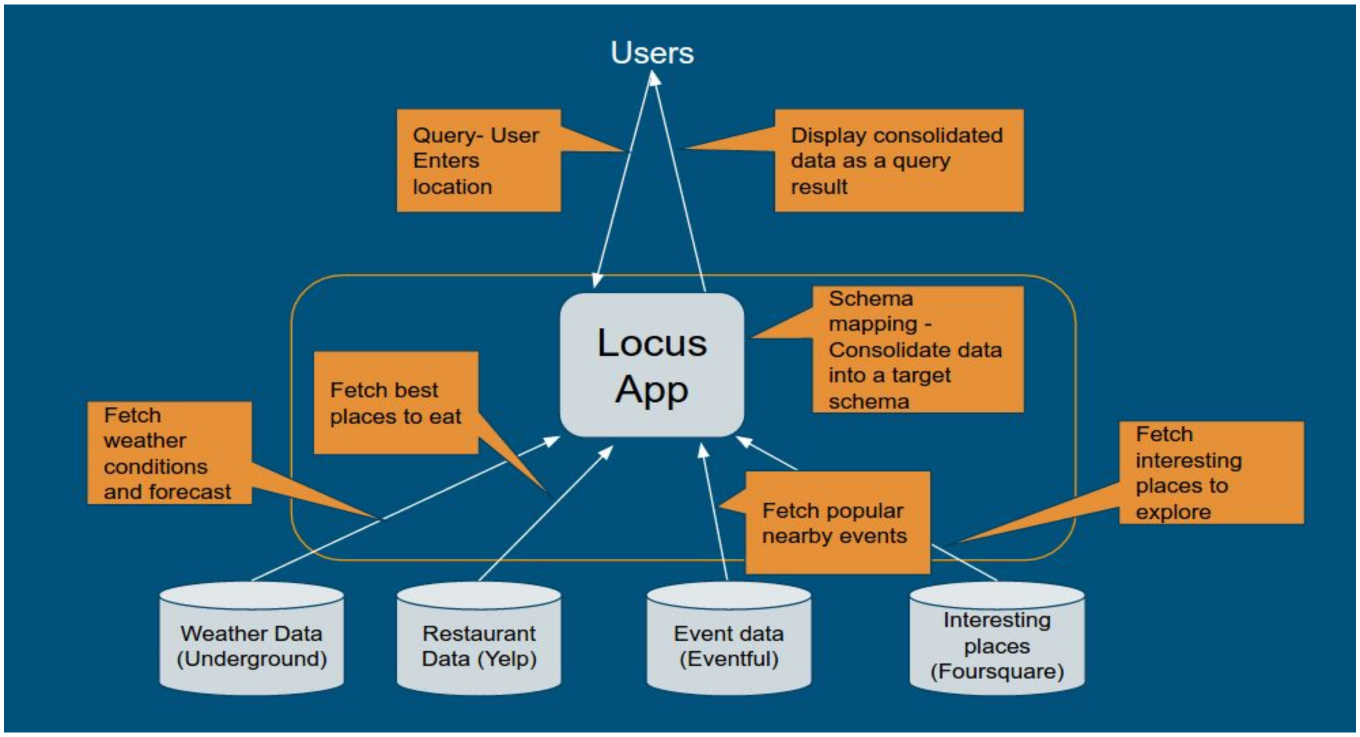


```
{
    "cityName" : "Buffalo",
    "lattitude" : "42.941429",
    "longitude" : "-78.837403",
    "stateCode" : "NY",
    "stateName" : "New York",
    "zipCode" : "14214",
    "cityKey" : "Buffalo, New York(NY)",
    "_id" : ObjectId("5837d9df996831d32edbebf1"),
    "__v" : 0
}
```

In this location object, we get cityName, latitude, longitude, stateCode, zipCode attributes from CSV file. We get stateName attribute from state code to state name mapping from Wikipedia. We have generated cityKey attribute by combining cityName, stateCode and stateName attribute to show it to the user as the user types in the search box. This cityKey is much more helpful for the user as there could be cities with the same name but from different states.

Approach

Solving data integration query problem


Users
Query- User Enters location
Display consolidated data as a query result
Locus App
Schema mapping - Consolidate data into a target schema
Fetch best places to eat
Fetch weather conditions and forecast
Fetch interesting places to explore
Fetch popular nearby events
Weather Data (Underground)
Restaurant Data (Yelp)
Event data (Eventful)
Interesting places (Foursquare)

Locus Algorithm

1. User enters location - either city name or zip code.
2. Server validates the location and retrieve location object from MongoDB belonging to the user entry.
3. Using attributes in location object, we get data from yelp, weather underground, eventful and foursquare asynchronously.
4. Data obtained from the above step is mapped to the target schema i.e. HTML layout of the result page.
5. Results are shown to user by rendering HTML content obtained in step 4.

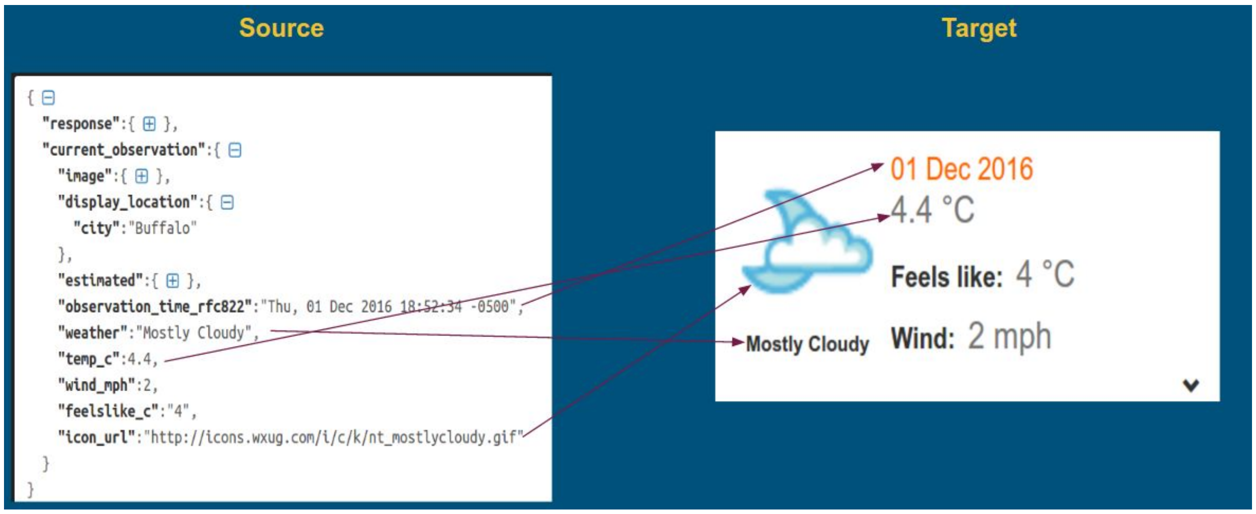
Schema Mappings

1. Weather component

In this component, we show current weather condition and weather forecast for the next 3 days to the user. We have implemented this component using weather underground API [2].

- **Input:** StateCode and City Name attribute of location object
  - Eg. StateCode = NY and CityName = Buffalo
- **Output:** Current weather and weather forecast for next 3 days
- **Output format:** XML or JSON
  - We have chosen JSON because it is easier to deal with in Javascript which is our language of choice.

Source-to-target schema mapping:



```
{ ⊟
  "response":{ ⊞ },
  "current_observation":{ ⊟
    "image":{ ⊞ },
    "display_location":{ ⊟
      "city":"Buffalo"
    },
    "estimated":{ ⊞ },
    "observation_time_rfc822":"Thu, 01 Dec 2016 18:52:34 -0500",
    "weather":"Mostly Cloudy",
    "temp_c":4.4,
    "wind_mph":2,
    "feelslike_c":"4",
    "icon_url":"http://icons.wxug.com/i/c/k/nt_mostlycloudy.gif"
  }
}
```

From the above diagram, we can see how the attributes in source schema are mapped to the specific attributes in target schema. Also attribute like observation time are transformed to the simple date format so that it is readable to the end user.

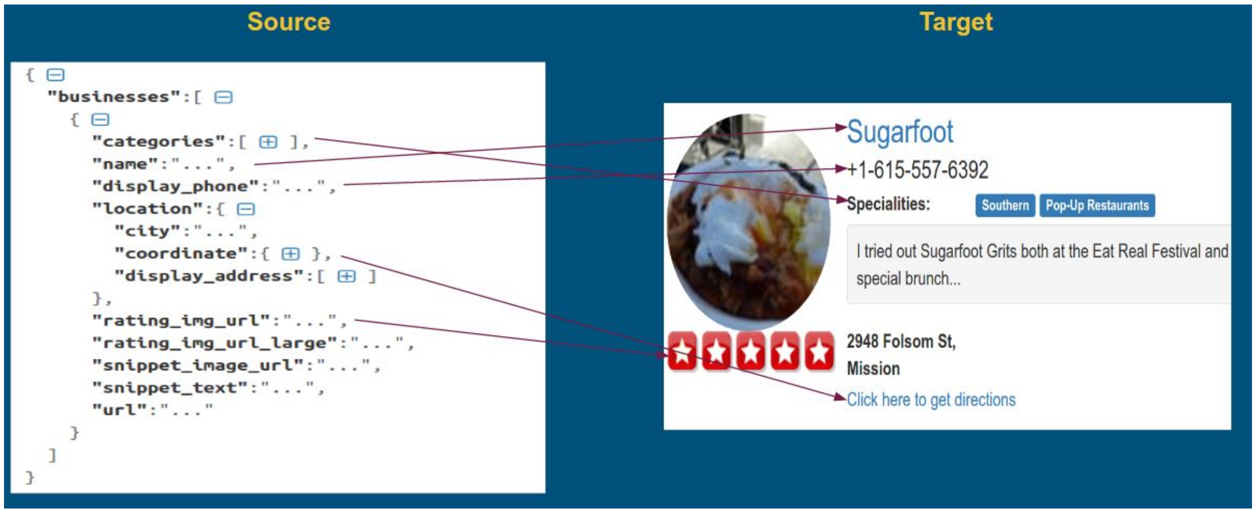
2. Yelp Component

In this component, we show top 20 popular places to eat around the user entered location. We have implemented this component using Yelp search API [3].

- **Input:** Latitude and Longitude of location object. Radius of location is fixed to 20 miles.
  - Eg: latitude = 37.78 and longitude = -122.41
- **Output:** Top 20 best places to eat around 20 miles radius of location entered by user.

- **Output format:** JSON

Source-to-target schema mapping:



```
{ ⊟
  "businesses":[ ⊟
    { ⊟
      "categories":[ ⊞ ],
      "name":"...",
      "display_phone":"...",
      "location":{ ⊟
        "city":"...",
        "coordinate":{ ⊞ },
        "display_address":[ ⊞ ]
      },
      "rating_img_url":"...",
      "rating_img_url_large":"...",
      "snippet_image_url":"...",
      "snippet_text":"...",
      "url":"..."
    }
  ]
}
```

Sugarfoot
+1-615-557-6392
Specialities: Southern Pop-Up Restaurants
I tried out Sugarfoot Grits both at the Eat Real Festival and special brunch...
2948 Folsom St,
Mission
Click here to get directions

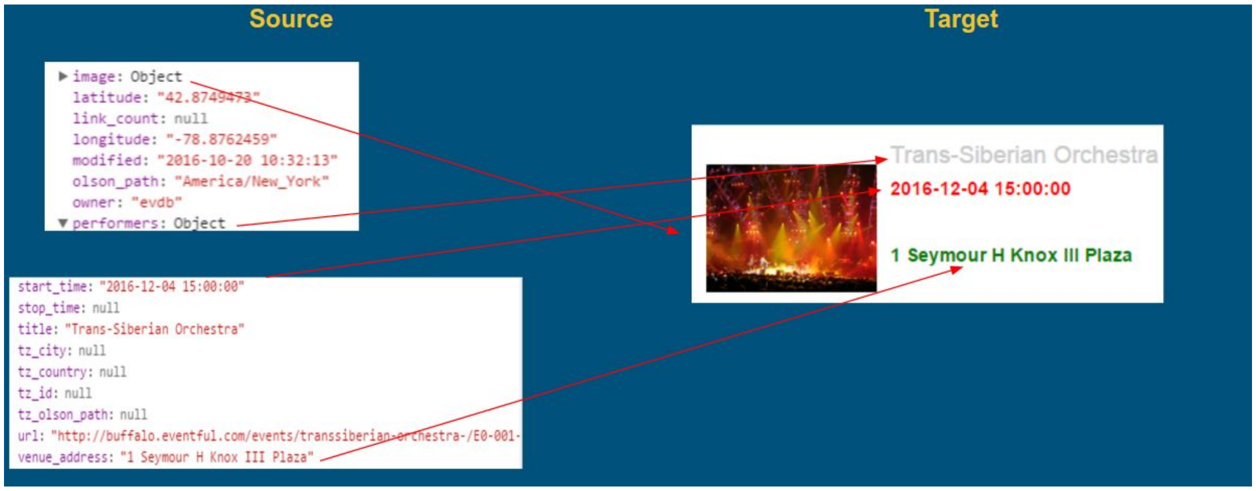
From the above diagram, we can see how the attributes in the source schema are mapped to the attributes in the target schema. We have used latitude and longitude attributes in the source schema to map the location on Google maps. User can click on “Click here to get directions” to open the above place in google maps.

3. Event Component

In this component, we show popular events happening around the user entered location. We have implemented this component using eventful API [4].

- **Input :** Latitude and Longitude of location object
  - Eg: latitude = 37.78 and longitude = -122.41
- **Output:** Events happening in the city sorted by popularity
- **Output format:** XML or JSON
  - We have chosen JSON because it is easier to deal with in Javascript which is our language of choice.
- Output can also be restricted to certain types of events like Music etc.

Source-to-target schema mapping:



```
▶ image: Object
  latitude: "42.8749473"
  link_count: null
  longitude: "-78.8762459"
  modified: "2016-10-20 10:32:13"
  olson_path: "America/New_York"
  owner: "evdb"
▼ performers: Object
```

```
start_time: "2016-12-04 15:00:00"
stop_time: null
title: "Trans-Siberian Orchestra"
tz_city: null
tz_country: null
tz_id: null
tz_olson_path: null
url: "http://buffalo.eventful.com/events/transsiberian-orchestra-/E0-001-
venue_address: "1 Seymour H Knox III Plaza"
```

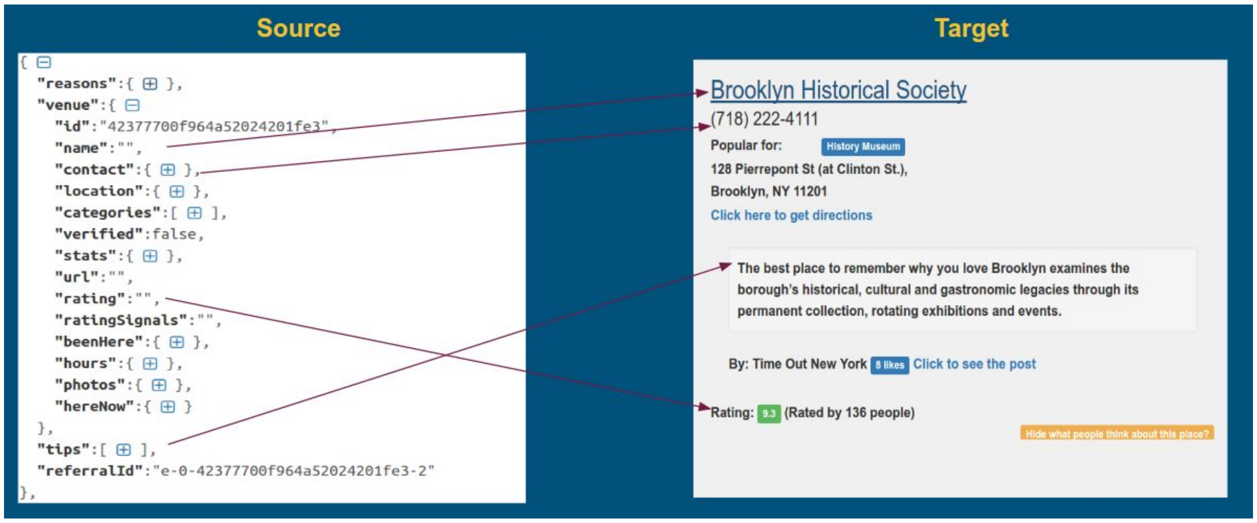
Source attributes from source schema are mapped to the target attributes of the target schema similar to the weather and yelp component.

4. Foursquare Component

In this component, we show interesting things to explore around the user entered location. We have implemented this component using Foursquare explorer API [5].

- **Input:** City Name Or latitude and longitude attribute of location object
  - Eg: CityName = San Francisco **OR**
  - latitude = 37.78 and longitude = -122.41
- **Output:** Must visit places & things to do in the area
- **Output format:** JSON
- If input is Lat-Long, places within 20 miles radius are displayed.

Source-to-target schema mapping:



```
{ ⊟
  "reasons":{ ⊞ },
  "venue":{ ⊟
    "id":"42377700f964a52024201fe3",
    "name":"",
    "contact":{ ⊞ },
    "location":{ ⊞ },
    "categories":[ ⊞ ],
    "verified":false,
    "stats":{ ⊞ },
    "url":"",
    "rating":"",
    "ratingSignals":"",
    "beenHere":{ ⊞ },
    "hours":{ ⊞ },
    "photos":{ ⊞ },
    "hereNow":{ ⊞ }
  },
  "tips":[ ⊞ ],
  "referralId":"e-0-42377700f964a52024201fe3-2"
},
```

Brooklyn Historical Society
(718) 222-4111
Popular for: History Museum
128 Pierrepont St (at Clinton St.),
Brooklyn, NY 11201
Click here to get directions

The best place to remember why you love Brooklyn examines the borough's historical, cultural and gastronomic legacies through its permanent collection, rotating exhibitions and events.

By: Time Out New York 8 likes Click to see the post

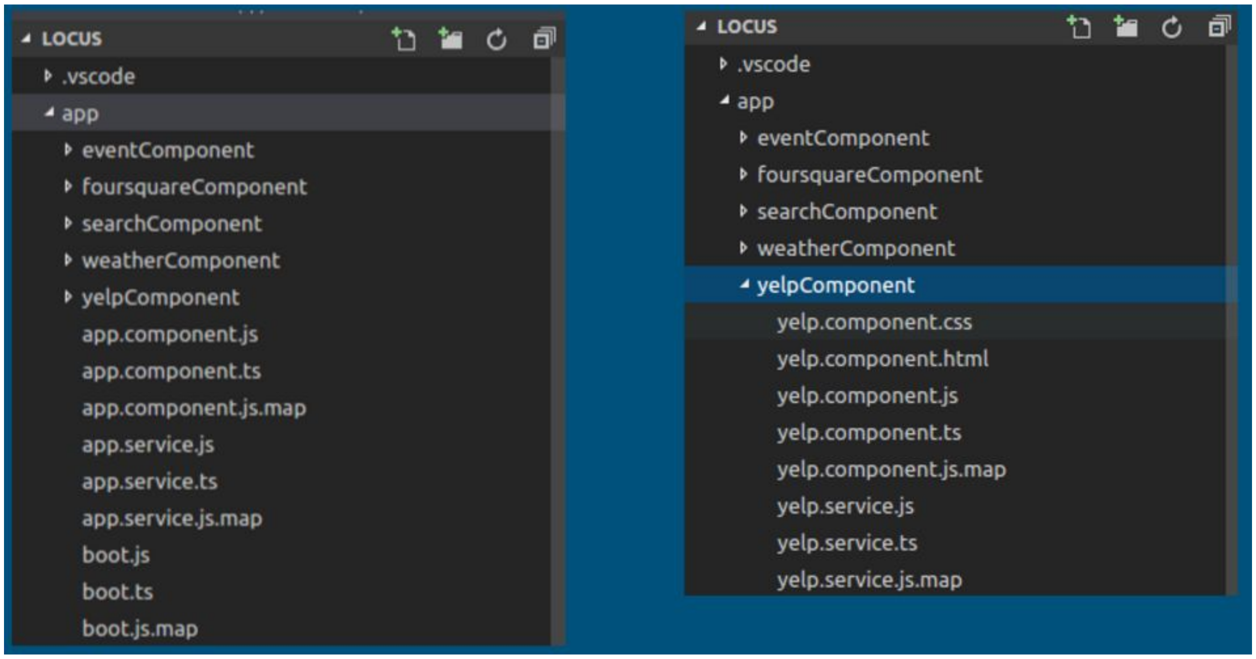
Rating: 9.3 (Rated by 136 people)

Hide what people think about this place?

Source attributes from source schema are mapped to the target attributes of the target schema similar to the weather and yelp component. We have used latitude and longitude attributes of source schema to map the given address of the event on google maps.

Package structure

We have abstracted implementation of each component for better modularity of code. Also it helps in maintainability and unit testing of the individual components.



```
LOCUS
  .vscode
  app
    eventComponent
    foursquareComponent
    searchComponent
    weatherComponent
    yelpComponent
    app.component.js
    app.component.ts
    app.component.js.map
    app.service.js
    app.service.ts
    app.service.js.map
    boot.js
    boot.ts
    boot.js.map
```

```
LOCUS
  .vscode
  app
    eventComponent
    foursquareComponent
    searchComponent
    weatherComponent
    yelpComponent
      yelp.component.css
      yelp.component.html
      yelp.component.js
      yelp.component.ts
      yelp.component.js.map
      yelp.service.js
      yelp.service.ts
      yelp.service.js.map
```

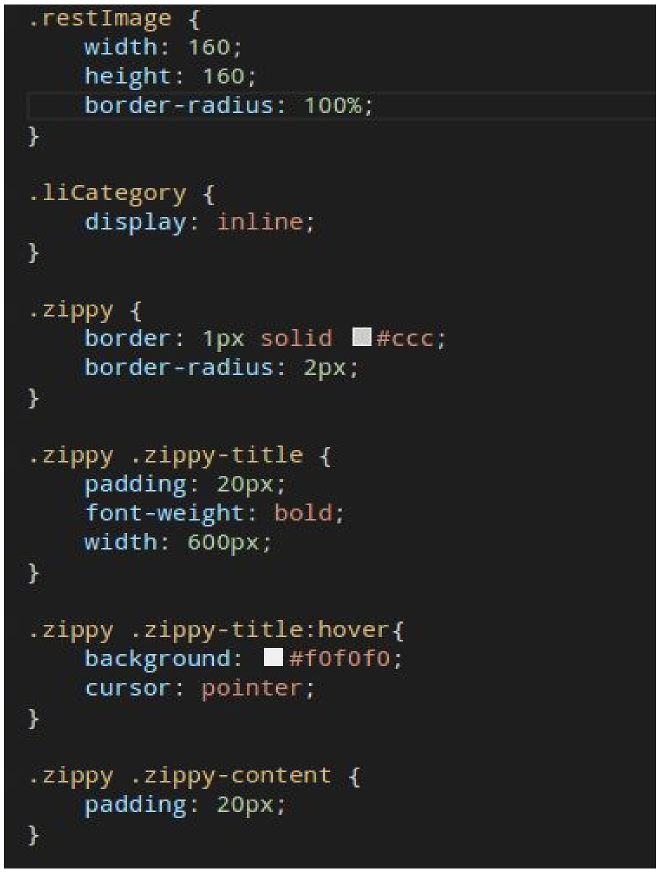
We can see from the above diagram that there are five components in our root folder “app”. Search component is used to get input from the user using simple forms and also display any warnings to the user. The other four components corresponds to the relevant data we want to display to the user based user query.

Each component has its component.css, component.html, component.ts and service.ts files. Their functionality is given below.

component.css: for styling html content
component.html: for displaying the data to the user
component.ts: for binding response data content to html
service.ts: to trigger appropriate events in backend server engine

We explain below how above files are linked with each other by taking Yelp component example.

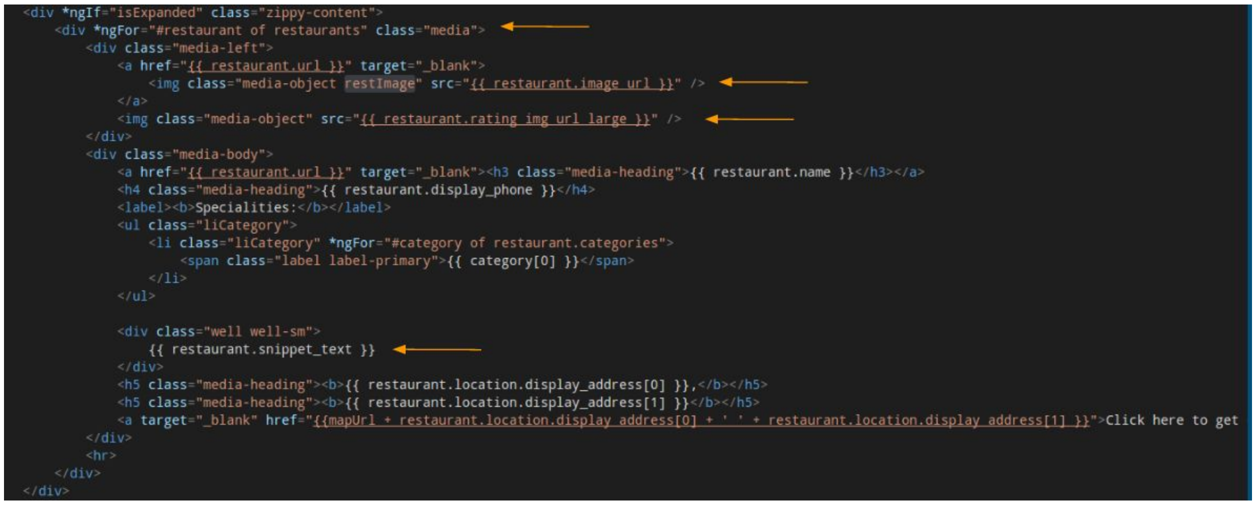
yelp.component.css



```css
.restImage {
    width: 160;
    height: 160;
    border-radius: 100%;
}

.liCategory {
    display: inline;
}

.zippy {
    border: 1px solid ■#ccc;
    border-radius: 2px;
}

.zippy .zippy-title {
    padding: 20px;
    font-weight: bold;
    width: 600px;
}

.zippy .zippy-title:hover{
    background: □#f0f0f0;
    cursor: pointer;
}

.zippy .zippy-content {
    padding: 20px;
}
```

Above file holds CSS classes for their appropriate HTML tags. For example, restImage css class corresponds to “img” HTML tag.

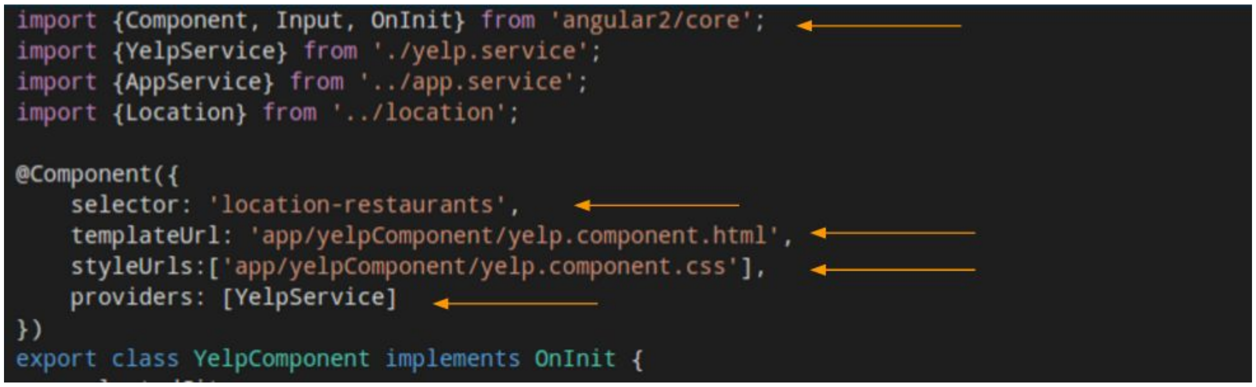
yelp.component.html



```
<div *ngIf="isExpanded" class="zippy-content">
    <div *ngFor="#restaurant of restaurants" class="media">
        <div class="media-left">
            <a href="{{ restaurant.url }}" target="_blank">
                <img class="media-object restImage" src="{{ restaurant.image_url }}" />
            </a>
            <img class="media-object" src="{{ restaurant.rating_img_url_large }}" />
        </div>
        <div class="media-body">
            <a href="{{ restaurant.url }}" target="_blank"><h3 class="media-heading">{{ restaurant.name }}</h3></a>
            <h4 class="media-heading">{{ restaurant.display_phone }}</h4>
            <label><b>Specialities:</b></label>
            <ul class="liCategory">
                <li class="liCategory" *ngFor="#category of restaurant.categories">
                    <span class="label label-primary">{{ category[0] }}</span>
                </li>
            </ul>

            <div class="well well-sm">
                {{ restaurant.snippet_text }}
            </div>
            <h5 class="media-heading"><b>{{ restaurant.location.display_address[0] }},</b></h5>
            <h5 class="media-heading"><b>{{ restaurant.location.display_address[1] }}</b></h5>
            <a target="_blank" href="{{mapUrl + restaurant.location.display_address[0] + ' ' + restaurant.location.display_address[1] }}">Click here to get
        </div>
        <hr>
    </div>
</div>
```

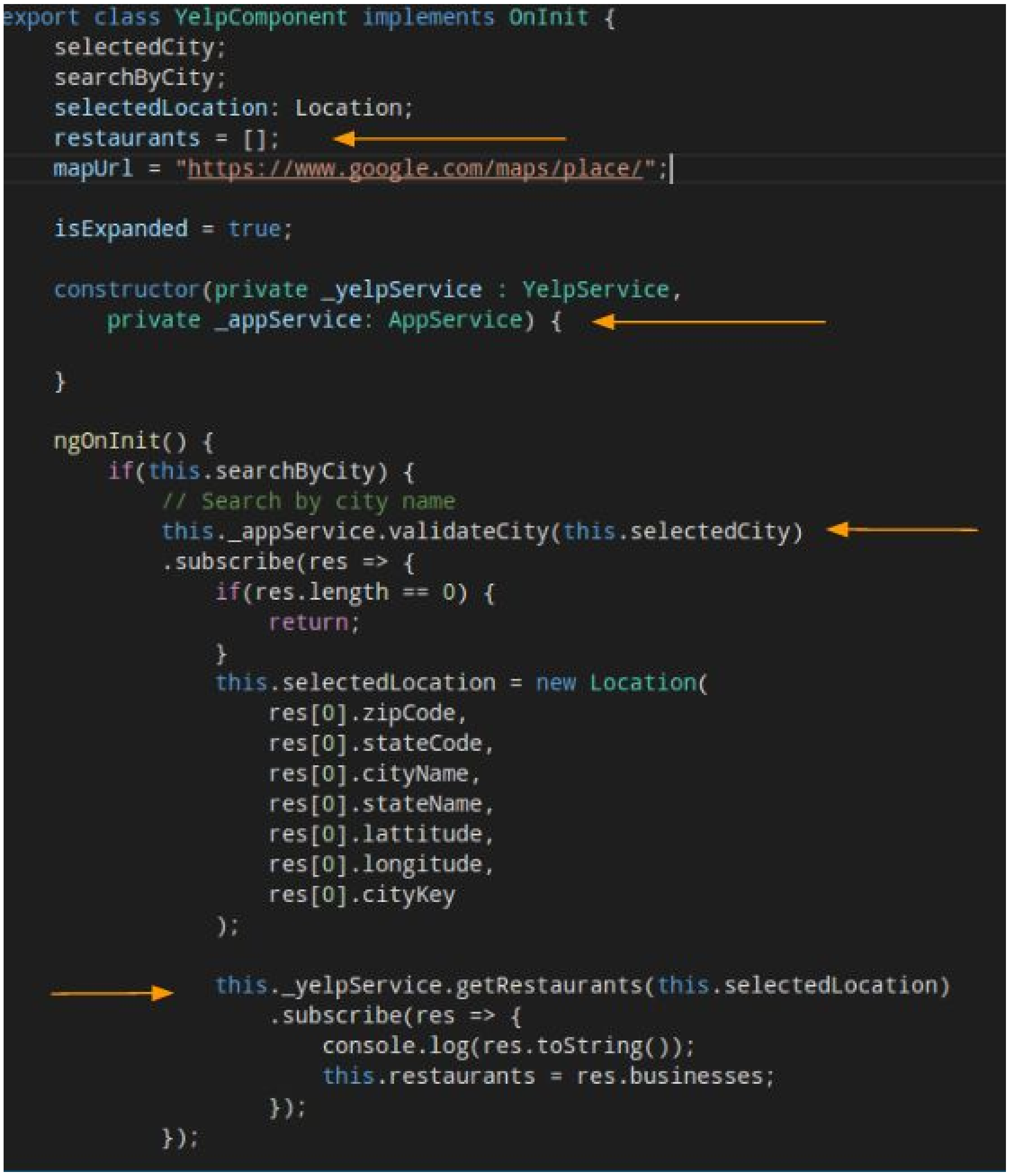
The above file holds the HTML structure which is rendered to the user browser. We iterate over each restaurant of the response content and then access attributes of the response content object in the HTML. For example, we are rendering restaurant.image.url in “src” attribute of “img” tag of HTML.

yelp.component.ts



```
import {Component, Input, OnInit} from 'angular2/core';
import {YelpService} from './yelp.service';
import {AppService} from '../app.service';
import {Location} from '../location';

@Component({
    selector: 'location-restaurants',
    templateUrl: 'app/yelpComponent/yelp.component.html',
    styleUrls:['app/yelpComponent/yelp.component.css'],
    providers: [YelpService]
})
export class YelpComponent implements OnInit {
```

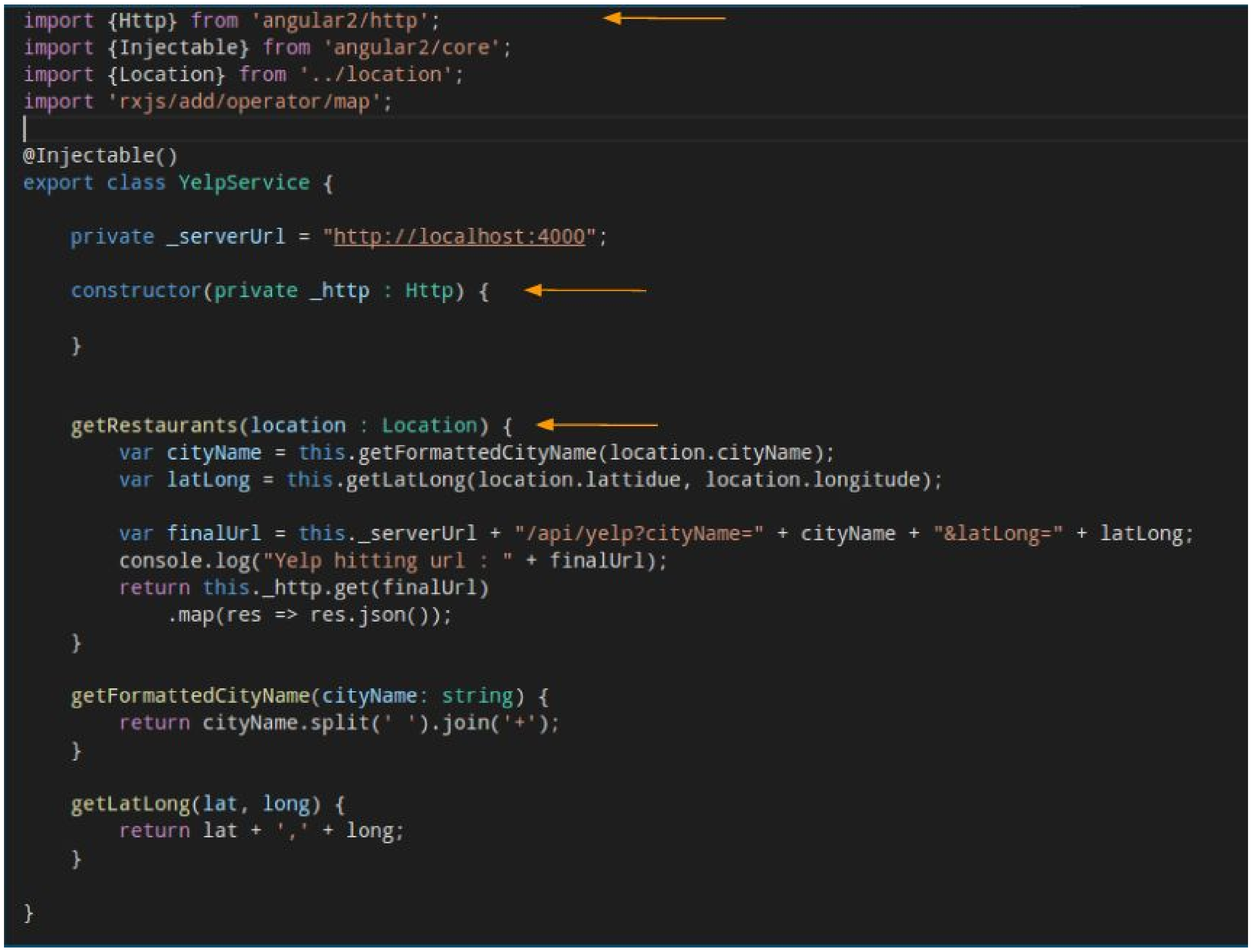
We can see from the above diagram that how we have included css file and html file in our component decorator. This is the way in which we tell angular that for this component, we are using html file as a template and css style for styling. We also include yelp service as a provider.



```
export class YelpComponent implements OnInit {
    selectedCity;
    searchByCity;
    selectedLocation: Location;
    restaurants = [];
    mapUrl = "https://www.google.com/maps/place/";

    isExpanded = true;

    constructor(private _yelpService : YelpService,
        private _appService: AppService) {

    }

    ngOnInit() {
        if(this.searchByCity) {
            // Search by city name
            this._appService.validateCity(this.selectedCity)
            .subscribe(res => {
                if(res.length == 0) {
                    return;
                }
                this.selectedLocation = new Location(
                    res[0].zipCode,
                    res[0].stateCode,
                    res[0].cityName,
                    res[0].stateName,
                    res[0].lattitude,
                    res[0].longitude,
                    res[0].cityKey
                );

                this._yelpService.getRestaurants(this.selectedLocation)
                    .subscribe(res => {
                        console.log(res.toString());
                        this.restaurants = res.businesses;
                    });
            });
```

In ngOnInit() method, we first validate the user entered location. If the location is invalid we return straight away displaying error to the user. If the user entered location is valid, we call getRestaurants() method in Yelp service class to get list of restaurants.

yelp.service.ts



```
import {Http} from 'angular2/http';
import {Injectable} from 'angular2/core';
import {Location} from '../location';
import 'rxjs/add/operator/map';

@Injectable()
export class YelpService {

    private _serverUrl = "http://localhost:4000";

    constructor(private _http : Http) {

    }


    getRestaurants(location : Location) {
        var cityName = this.getFormattedCityName(location.cityName);
        var latLong = this.getLatLong(location.lattidue, location.longitude);

        var finalUrl = this._serverUrl + "/api/yelp?cityName=" + cityName + "&latLong=" + latLong;
        console.log("Yelp hitting url : " + finalUrl);
        return this._http.get(finalUrl)
            .map(res => res.json());
    }

    getFormattedCityName(cityName: string) {
        return cityName.split(' ').join('+');
    }

    getLatLong(lat, long) {
        return lat + ',' + long;
    }

}
```

In this service class, we trigger events in our back end server engine to get restaurants near the user entered location. We are using city name, latitude and longitude attribute of the location object.

User interface

Good UI/UX design practices have been extensively followed in designing user interface for Locus. Here are some good practices and how they reflect in our different components.

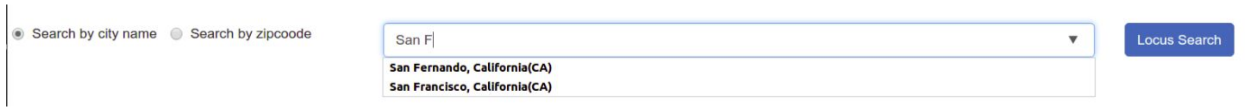
UI/UX Choice depends largely on the types of the users application is targeted for. An application targeted for commercial users would have considerable differences than the one targeted for the non-commercial users which can be attributed to the difference in depth of functionality between the two.

Locus is for noncommercial usage and the complexity hence is less.

There is no user-registration and hence no access policies needs to be managed, all the content is available to all the users across the system.

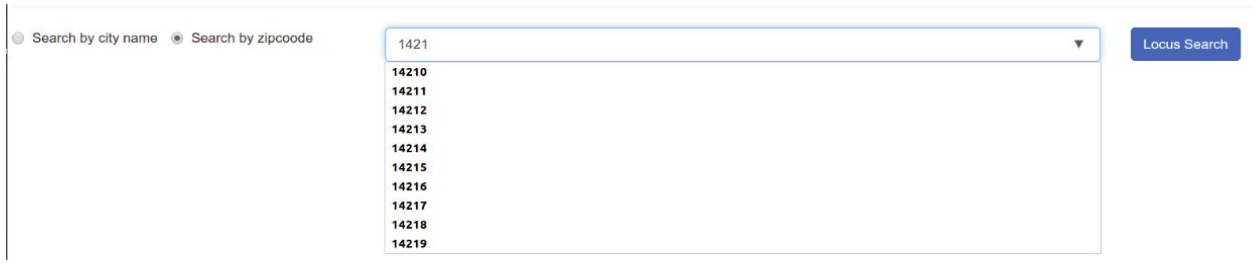
Basic UI is just one form and two radio buttons to make it easy to navigate and use. User can select whether he wants to enter the zipcode or the city name. Auto complete feature is provided to the user to aid with input.

Search by City Name



San Fernando, California(CA)
San Francisco, California(CA)

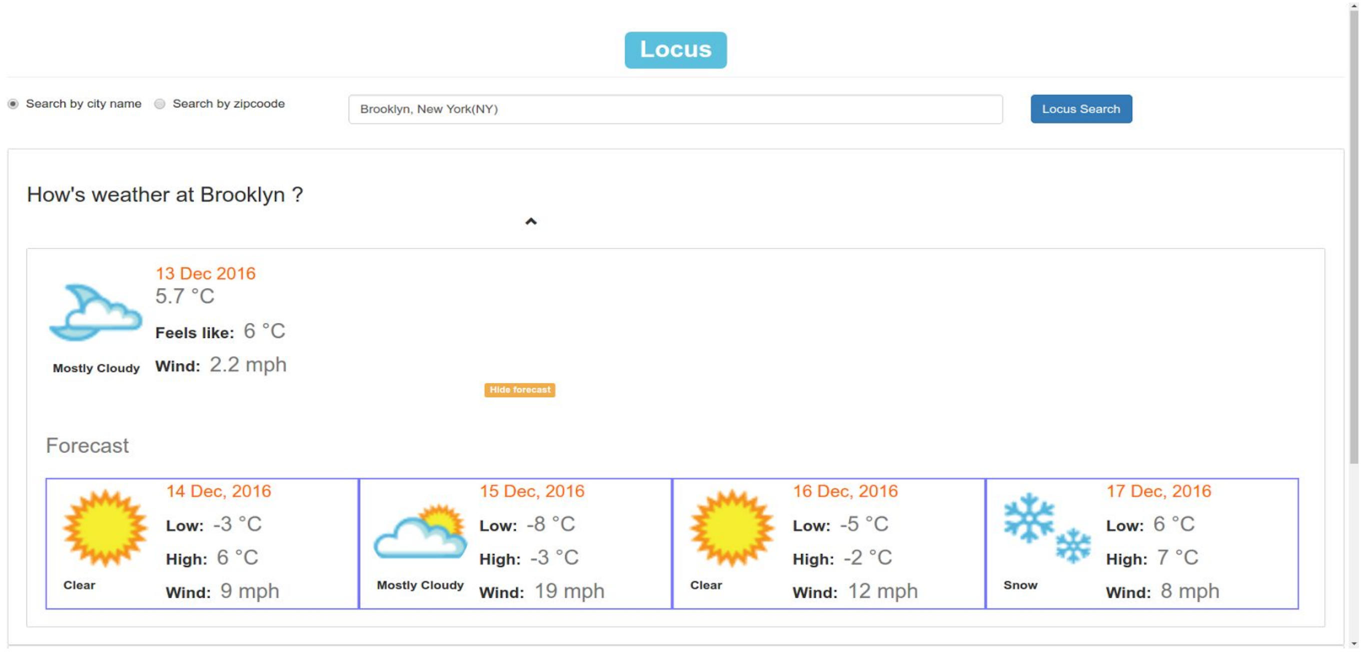
Search by zipcode



14210
14211
14212
14213
14214
14215
14216
14217
14218
14219

Each component is neatly arranged in it's own division. There is a coherent representation of information across the application.

Weather component which is a major decision influencer has been kept on top in the search results representation.



How's weather at Brooklyn ?

13 Dec 2016
5.7 °C
Feels like: 6 °C
Mostly Cloudy
Wind: 2.2 mph

Forecast

14 Dec, 2016
Low: -3 °C
High: 6 °C
Clear
Wind: 9 mph

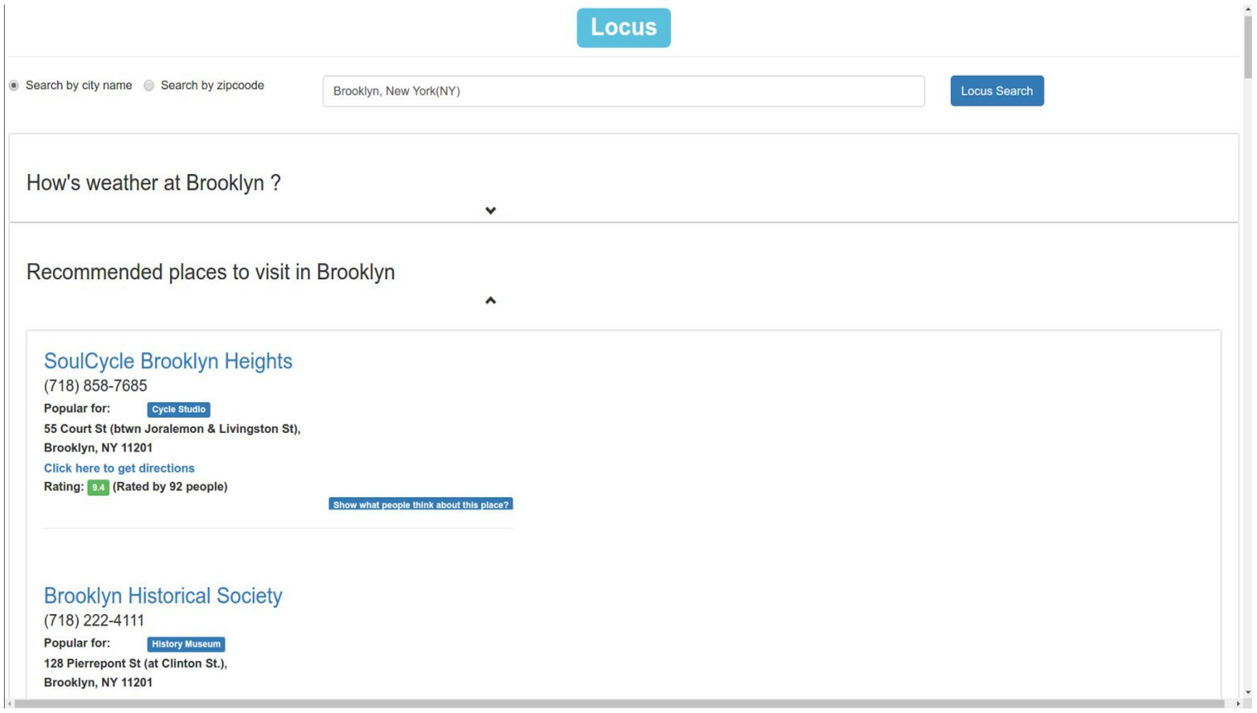
15 Dec, 2016
Low: -8 °C
High: -3 °C
Mostly Cloudy
Wind: 19 mph

16 Dec, 2016
Low: -5 °C
High: -2 °C
Clear
Wind: 12 mph

17 Dec, 2016
Low: 6 °C
High: 7 °C
Snow
Wind: 8 mph

Each component is placed compactly inside the zippy component.
Zippy is a toggle view with a little arrow next to it, which can be clicked to show or hide the specific component.

Places to visit have been placed just below the weather component because it is again a major decision influencer, since a city is as interesting as the popular places it has to offer.



How's weather at Brooklyn ?

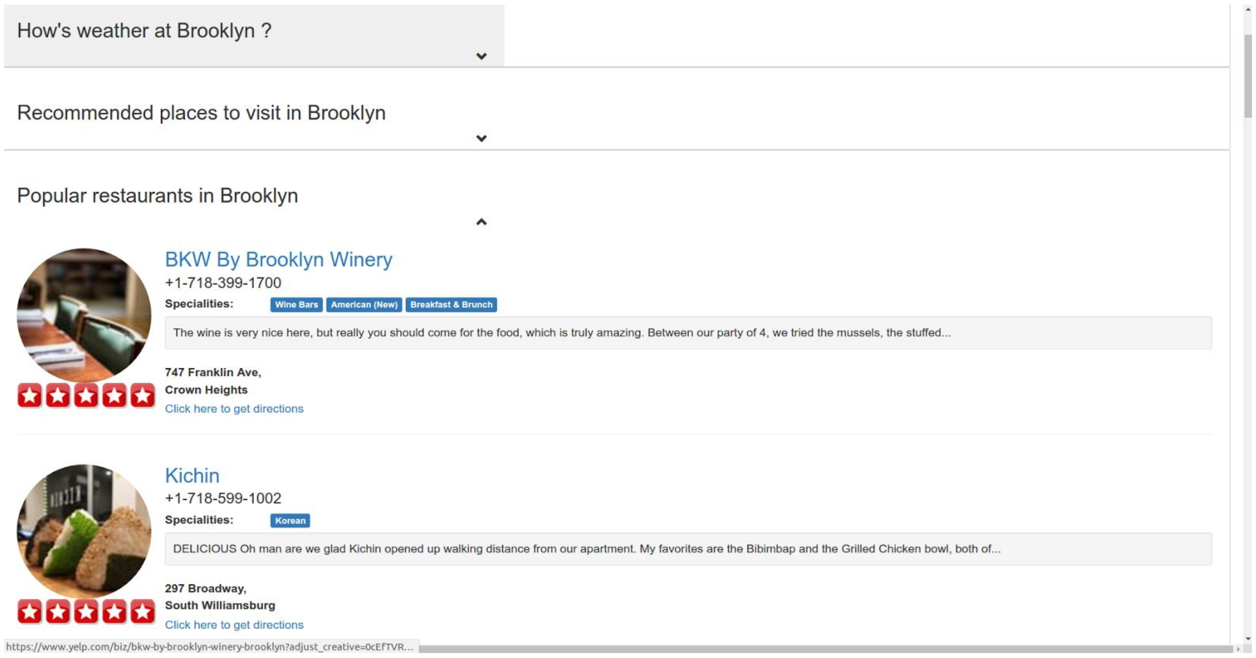
Recommended places to visit in Brooklyn

SoulCycle Brooklyn Heights
(718) 858-7685
Popular for: Cycle Studio
55 Court St (btwn Joralemon & Livingston St),
Brooklyn, NY 11201
Click here to get directions
Rating: 9.4 (Rated by 92 people)
Show what people think about this place?

Brooklyn Historical Society
(718) 222-4111
Popular for: History Museum
128 Pierrepont St (at Clinton St.),
Brooklyn, NY 11201

Places to eat comes just after that, since places to eat may not be top priority for majority of users as a measure when it comes to making the decisions to visit a place. But it is important nevertheless to give an idea of cuisine options available.

Each popular restaurant has been displayed in it's own division, which displays it's name, contact number, & the specialties. A link is provided just below it to locate the restaurant using map.



How's weather at Brooklyn ?

Recommended places to visit in Brooklyn

Popular restaurants in Brooklyn

BKW By Brooklyn Winery
+1-718-399-1700
Specialities: Wine Bars American (New) Breakfast & Brunch

The wine is very nice here, but really you should come for the food, which is truly amazing. Between our party of 4, we tried the mussels, the stuffed...

747 Franklin Ave,
Crown Heights
Click here to get directions

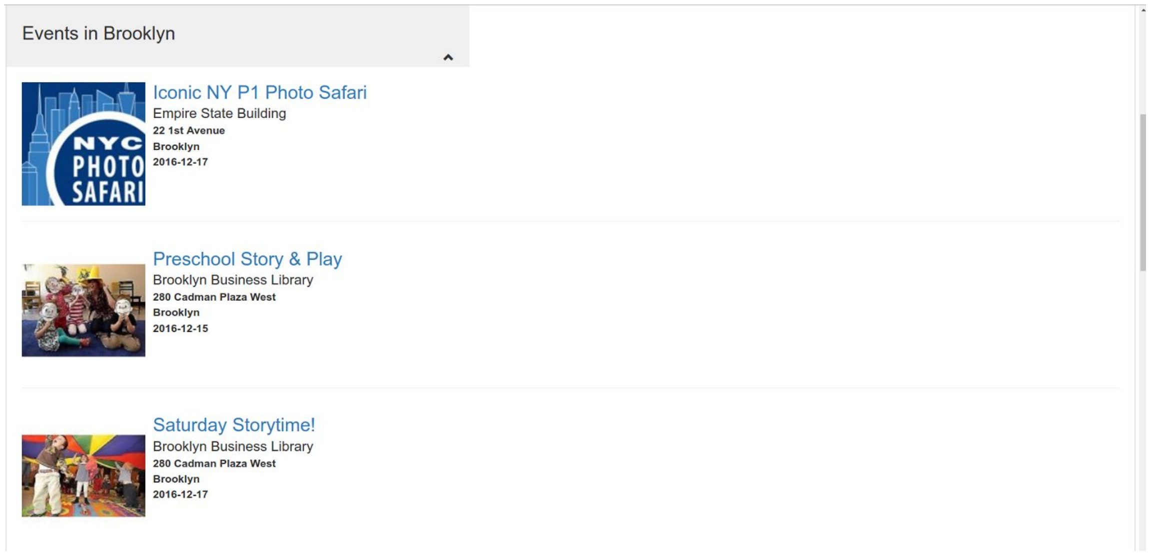
Kichin
+1-718-599-1002
Specialities: Korean

DELICIOUS Oh man are we glad Kichin opened up walking distance from our apartment. My favorites are the Bibimbap and the Grilled Chicken bowl, both of...

297 Broadway,
South Williamsburg
Click here to get directions

https://www.yelp.com/biz/bkw-by-brooklyn-winery-brooklyn?adjust_creative=0cEfTVR...

Then there is an event component. User might be looking for specific events but it is less likely a reason to be visiting a place, and hence it has been placed at this level of hierarchy. Event component shows the Name and place of the event with the address.



Events in Brooklyn

Iconic NY P1 Photo Safari
Empire State Building
22 1st Avenue
Brooklyn
2016-12-17

Preschool Story & Play
Brooklyn Business Library
280 Cadman Plaza West
Brooklyn
2016-12-15

Saturday Storytime!
Brooklyn Business Library
280 Cadman Plaza West
Brooklyn
2016-12-17

Justification of the design choices

We have implemented MEAN stack [7] which comprises of four components viz. Mongo DB, Express Framework, Angular 2 [6] and Node JS. Each component serves some purpose in our system design.

As discussed in our system architecture, our system can be divided into three components viz.

1. Front end - Client engine
2. Back end - Server engine
3. Database

1. Front end - Client engine

a. **HTML 5/CSS:** Layout of a search and results page.

b. **Angular2:**

   i. Provides MVC framework which helps modularizing code

   ii. It is easier to implement different components and services.

c. **Bootstrap:** Styling and user interface components

2. Back end - Server engine

a. Server Engine

   i. **Node JS:** Application server which is written in javascript so it is extremely compatible with all the front end components.

   ii. **Express JS:** Framework for running server.

3. Database

a. **Mongo DB:** Useful for storing all locations as a JSON object.

Hardware and software platforms

Development environment specifics

Hardware platform

- Ubuntu 14.04 LTS 64 Bit
- Intel® Core™ i7-5500U CPU @ 2.40GHz × 4
- 8 GB RAM

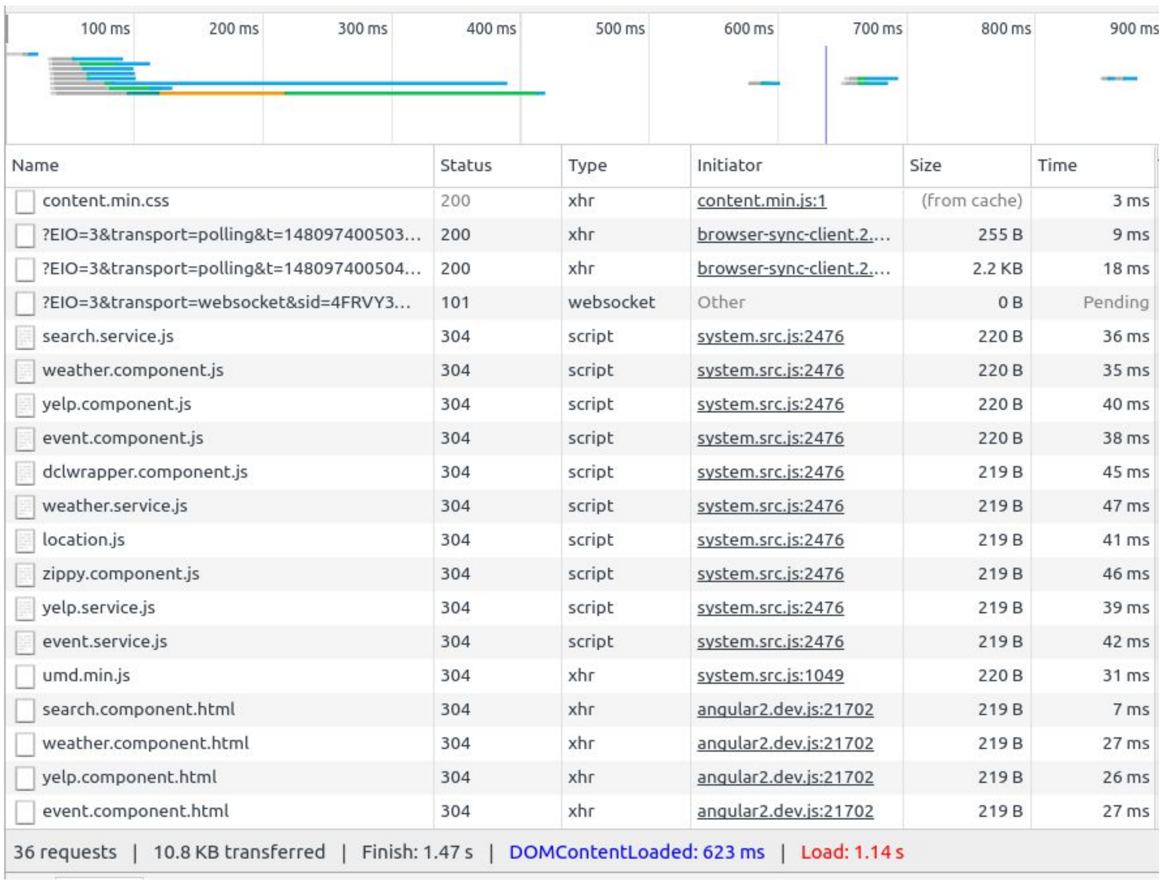
Software platform

- IDE: Visual Studio
- Browser: Chrome and Firefox
- Server: Node, Express Framework.
- Database: Mongo DB

Experimental evaluation

Experimental evaluation has been done using **chrome tools and loadtest** (Node package).

Page load test



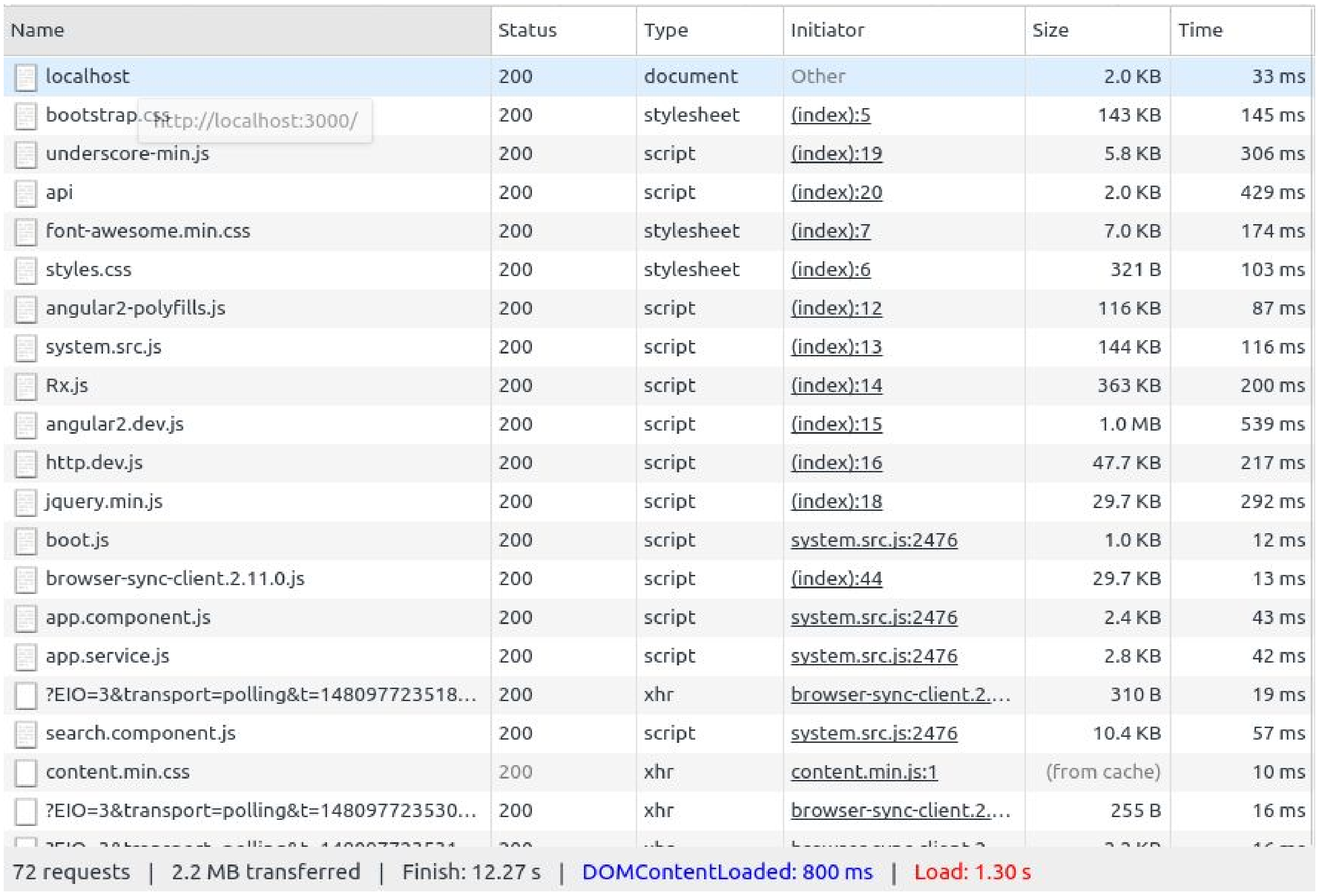
Name	Status	Type	Initiator	Size	Time
content.min.css	200	xhr	content.min.js:1	(from cache)	3 ms
?EIO=3&transport=polling&t=148097400503...	200	xhr	browser-sync-client.2....	255 B	9 ms
?EIO=3&transport=polling&t=148097400504...	200	xhr	browser-sync-client.2....	2.2 KB	18 ms
?EIO=3&transport=websocket&sid=4FRVY3...	101	websocket	Other	0 B	Pending
search.service.js	304	script	system.src.js:2476	220 B	36 ms
weather.component.js	304	script	system.src.js:2476	220 B	35 ms
yelp.component.js	304	script	system.src.js:2476	220 B	40 ms
event.component.js	304	script	system.src.js:2476	220 B	38 ms
dclwrapper.component.js	304	script	system.src.js:2476	219 B	45 ms
weather.service.js	304	script	system.src.js:2476	219 B	47 ms
location.js	304	script	system.src.js:2476	219 B	41 ms
zippy.component.js	304	script	system.src.js:2476	219 B	46 ms
yelp.service.js	304	script	system.src.js:2476	219 B	39 ms
event.service.js	304	script	system.src.js:2476	219 B	42 ms
umd.min.js	304	xhr	system.src.js:1049	220 B	31 ms
search.component.html	304	xhr	angular2.dev.js:21702	219 B	7 ms
weather.component.html	304	xhr	angular2.dev.js:21702	219 B	27 ms
yelp.component.html	304	xhr	angular2.dev.js:21702	219 B	26 ms
event.component.html	304	xhr	angular2.dev.js:21702	219 B	27 ms

36 requests | 10.8 KB transferred | Finish: 1.47 s | DOMContentLoaded: 623 ms | Load: 1.14 s

We can see that the total load time here is **1.14s** & the finish time is **1.47s.**
'Finished' time in Chrome devtools includes the asynchronously loading(non blocking) objects/elements on the page which may continue downloading even after the onload event for page has fired.

The response time for a website means 'Load' time. At this point user can see browser has finished working and page is ready on their screen.

Query response test

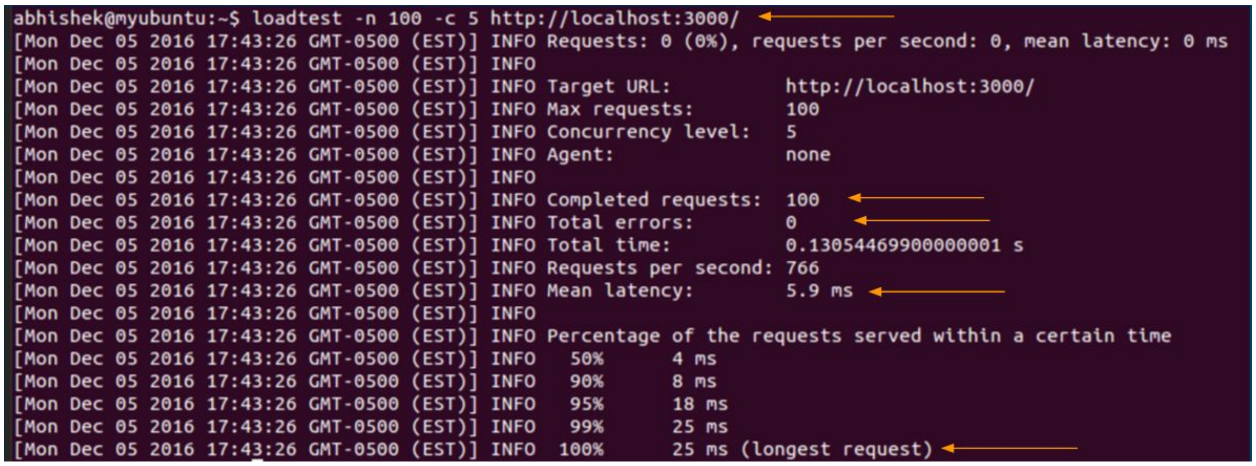


Name	Status	Type	Initiator	Size	Time
localhost	200	document	Other	2.0 KB	33 ms
bootstrap.css http://localhost:3000/	200	stylesheet	(index):5	143 KB	145 ms
underscore-min.js	200	script	(index):19	5.8 KB	306 ms
api	200	script	(index):20	2.0 KB	429 ms
font-awesome.min.css	200	stylesheet	(index):7	7.0 KB	174 ms
styles.css	200	stylesheet	(index):6	321 B	103 ms
angular2-polyfills.js	200	script	(index):12	116 KB	87 ms
system.src.js	200	script	(index):13	144 KB	116 ms
Rx.js	200	script	(index):14	363 KB	200 ms
angular2.dev.js	200	script	(index):15	1.0 MB	539 ms
http.dev.js	200	script	(index):16	47.7 KB	217 ms
jquery.min.js	200	script	(index):18	29.7 KB	292 ms
boot.js	200	script	system.src.js:2476	1.0 KB	12 ms
browser-sync-client.2.11.0.js	200	script	(index):44	29.7 KB	13 ms
app.component.js	200	script	system.src.js:2476	2.4 KB	43 ms
app.service.js	200	script	system.src.js:2476	2.8 KB	42 ms
?EIO=3&transport=polling&t=148097723518...	200	xhr	browser-sync-client.2....	310 B	19 ms
search.component.js	200	script	system.src.js:2476	10.4 KB	57 ms
content.min.css	200	xhr	content.min.js:1	(from cache)	10 ms
?EIO=3&transport=polling&t=148097723530...	200	xhr	browser-sync-client.2....	255 B	16 ms

72 requests | 2.2 MB transferred | Finish: 12.27 s | DOMContentLoaded: 800 ms | Load: 1.30 s

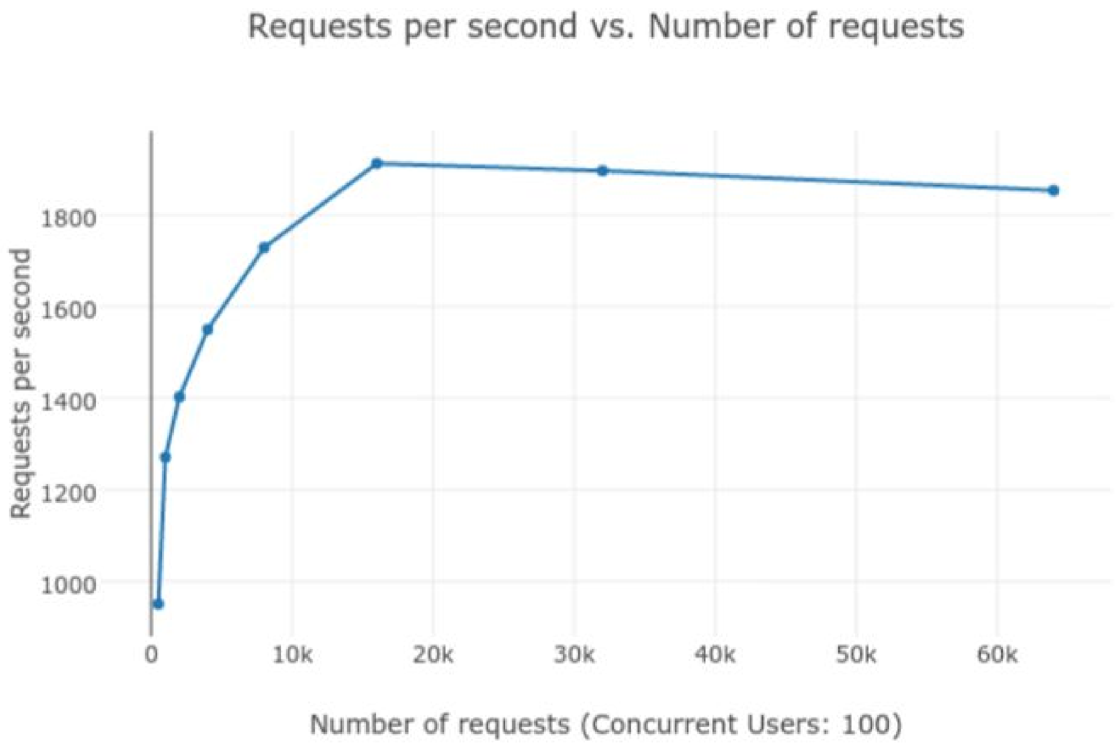
1.30s is a good response time, since it has been proven by various experiments done by Website usage evaluation firms that 1 second keeps the user's flow of thought seamless. Users feel in control of the overall experience and that they're moving freely rather than waiting on the computer. This degree of responsiveness is needed for good navigation.

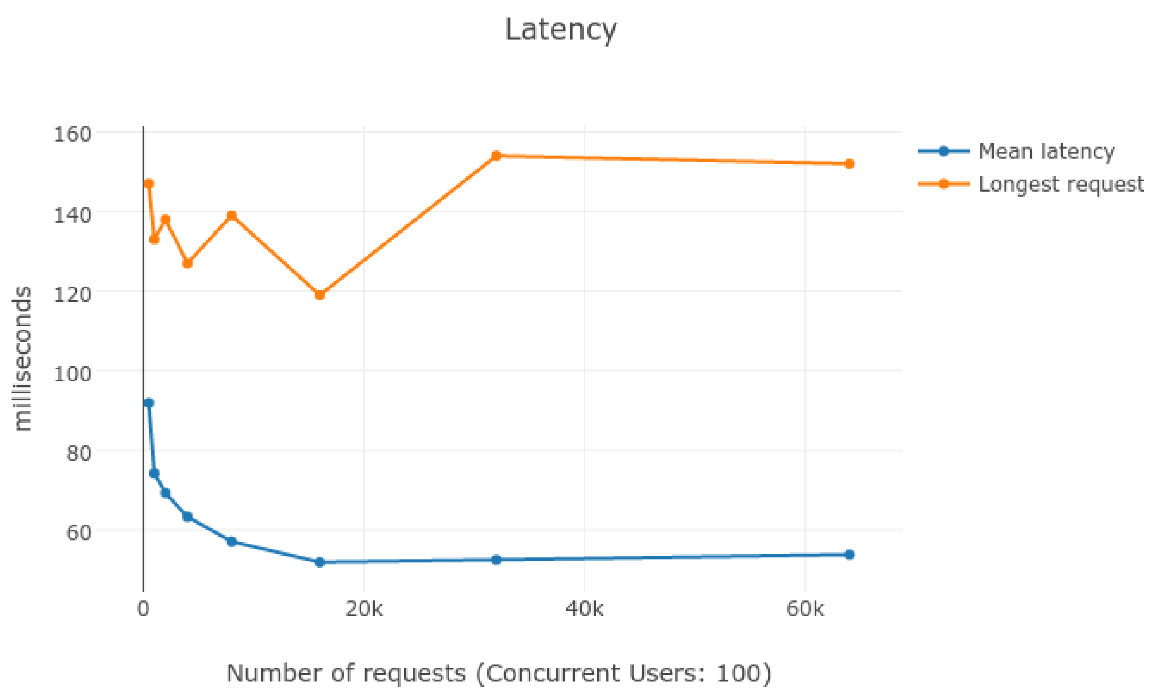
Load test



```
abhishek@myubuntu:~$ loadtest -n 100 -c 5 http://localhost:3000/
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Requests: 0 (0%), requests per second: 0, mean latency: 0 ms
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Target URL:          http://localhost:3000/
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Max requests:        100
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Concurrency level:   5
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Agent:               none
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Completed requests:  100
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Total errors:        0
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Total time:          0.13054469900000001 s
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Requests per second: 766
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Mean latency:        5.9 ms
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO Percentage of the requests served within a certain time
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO   50%      4 ms
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO   90%      8 ms
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO   95%      18 ms
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO   99%      25 ms
[Mon Dec 05 2016 17:43:26 GMT-0500 (EST)] INFO  100%      25 ms (longest request)
```

The screenshot above shows the results after running loadtest. They can be better interpreted using the graphs below





Number of concurrent requests have been kept to **100**. We can see that mean latency increases as the Number of requests increases. This can be attributed to the fact that with few requests power of concurrency is not being fully utilized, but as the number of requests increases, concurrent requests are handled more efficiently and the mean latencies follows a linear curve. Longest request also shows linear trend when the concurrent request handling kicks in.

Conclusions

In this project, we have solved data integration problem. We have found that this problem can be solved using schema mappings between target schemas and source schemas. For most of our components, we have used schema correspondence between target schema and source schema which is a different notion of schema mappings. As we are allowing user to search the locations by city name and zipcode, user has the flexibility to search with different granularities of the location. Auto suggest feature is a good assist to the user when user does not know the complete location.

From the above graphs, we can see that our application works in a multithreaded environment where different users are accessing our application concurrently. Also the mean latency of our application is 53.4 ms which is very acceptable for the web application at this scale.

Summary

To summarize the above discussion, we have built a single web application which is a mashup of a relevant data about user entered location. This feature is useful when user is unfamiliar about new location. We have built a lucid interface with features like auto-suggest, zippy component to show or hide forecast based on user action and clean user interface to show restaurants and events.

In this project, we have solved the data integration query problem using schema mappings where we have integrated data from four different sources like weather underground for weather forecast, yelp for best places to eat, eventful for popular events happening around and foursquare for interesting places to explore around user entered location.

Limitations of the proposed approach

Planning Trip in advance

An Improvement that can be made is to provide user with the ability to plan trips in advance. Locus currently only retrieves the results for the current week.

Enriching Places-to-eat component(Yelp) using multiple APIs

Components like places to eat components can still be improved. Placed to eat are currently being fetched from single API.

Place rating reliability can be further improved using multiple APIs for places-to-eat component. For example the ratings of the particular restaurant can be fetched from multiple APIs and then the ratings can be normalized using algorithms like z-score to provide more reliability.

Enriching Places-of-interest(Foursquare) using multiple APIs

Similar thing can be done for this component. Expedia.com has stopped new user registrations which has led to this limitation. Multiple API like expedia can be used together with existing Foursquare to provide more reliable and enriching places-of-interest data.

Future work

Considering current user location

In our current implementation, users has to explicitly enter the location in the search box. However we can extend the same functionality by fetching GPS coordinates from current location of the users with their consent.

Filtering events by Date

There is still a margin of improvement in the event information that are displayed. It might be a possibility that the user is interested in planning on attending events for on certain dates, more flexibility can be provided to the user to filter the events based upon specific dates.

Filtering events by Places

There is a scope of improving flexibility by providing the user with an option to group the events based on particular place.

Locus search in multiple countries

Currently the locus search is limited to the locations in United States. We can extend the same set of features for multiple countries around the world so that more users can get benefited from this service.

Bibliography

1. http://simplemaps.com/data/us-zips
2. https://www.wunderground.com/weather/api/
3. https://www.yelp.com/developers/documentation/v2/search_api
4. http://api.eventful.com/docs/events/search
5. https://developer.foursquare.com/docs/explore#req=venues/explore%3Fll%3D40.7,-74
6. https://angular.io/
7. http://mean.io/
8. http://stackoverflow.com/questions/4727226/on-a-high-level-how-does-oauth-2-work
9. http://stackoverflow.com/questions/30266960/website-response-time-difference-between-load-and-finish