CS490 Senior Design

Design Document

MyMapper

Team Members:

Chen Gong

Yunkai Sun

Yang Xu

Zhihao Hu

Ben Pastene

Mingsheng Xu

Contents

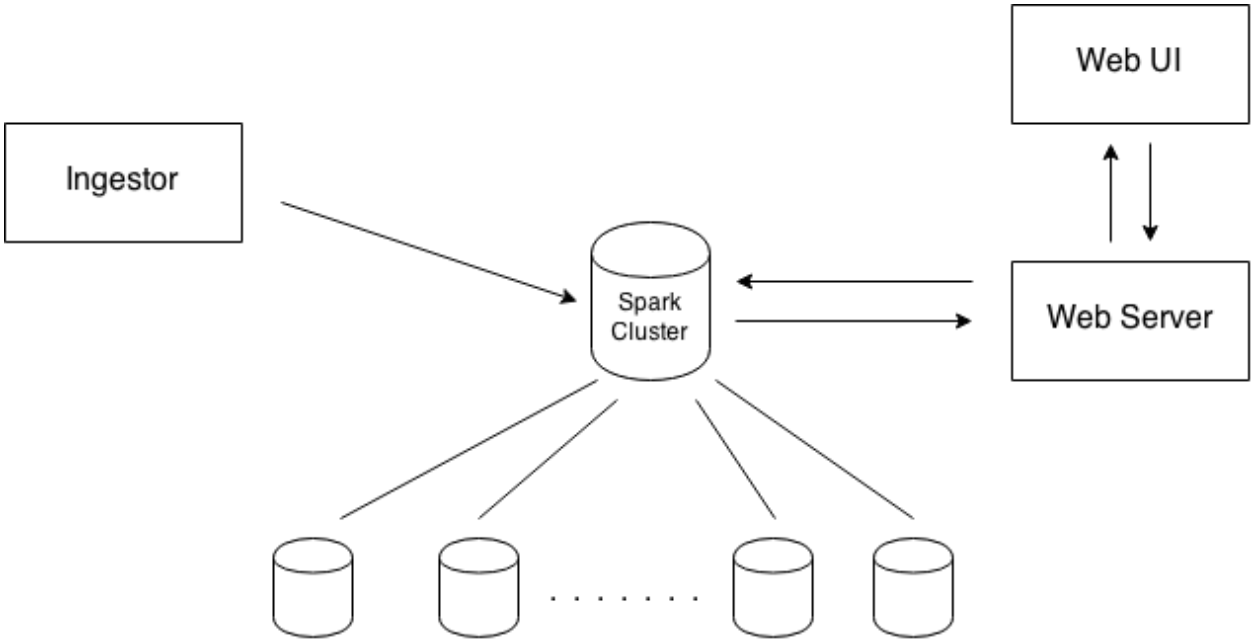
1. Purpose

MyMapper is an online mapping application. It will be used to help the user locate and navigate to any significant point of interest they so desire. Additionally, it will support moving points of interest, allowing the user to view up-to-date information on various moving actors in their vicinity (busses, taxis, police cars, etc.) Since the actual location for these moving objects in the real world are not readily available, we will simply mock the data. This will serve as a proof-of-concept for any future endeavors should the data ever become easily accessed.

2. Design Outline

The MyMapper system will have multiple components. First, in the front end, our system will consist of a client-server architecture to handle user requests. For the backing data store, we will use Spark, a distributed in-memory database cluster. To update the location of all dynamic objects within the database, we will use a scheduled

script that will insert data into the spark cluster. This script will be called the ingestor. The high level structure of the system is depicted below:



2.1 System Components

There are four main components of our system.

1) Web UI: This is the client-facing application. It will be a sleek and intuitive website that will take user input and display the requested information to the user. All client interactions with our product will go through here.
2) Web Server: This is a centralized server that will process all user requests. These requests are sent to the server from the client-facing web UI via http. The server will then process these requests, communicating with the database if necessary, and send the results back up the the user.
3) Spark Cluster Database: This is the backing data store for our system. It will store all location information for static and dynamic objects. It is an in-memory distributed database, so all transactions will have very low latency.

4) Ingestor: This is a script that will be scheduled to run very frequently. It will update all location information in the database for moving objects. Since the actual location of moving objects in the world are not readily available, this script will randomly generate these data points.

2.2 Component Interactions

1) Interaction between Ingestor and Database: The ingestor script will connect to the database to insert updated object locations.
2) Interaction between Web UI and Server: All user requests will be processed by the webserver, consequently the web UI will hand off to the web server all requests entered by the user. Additionally, once the request has been processed and the relevant data collected, the server will send the results back up to the web UI to be displayed to the user.
3) Interaction between Server and Database: Many user requests will query for location information. Consequently, the server will need to ask the database for this data to successfully process the user's request.

3. Design Issues

3.1 Functional Issues:

Design Issue #1: Platforms (Client Application)

Options:

1. Website
2. Smart phone application

Decision: Website

Justification: If it's built as a smartphone application then it will only be able to run on that device. But if it's built as an website, you can not only run it on your computer but also on any of your mobile devices.

Design Issue #2: Software Architecture

Options:

1. Client Server Design
2. Server Oriented Design
3. Application Only (Only Client Application)

Decision: Client Server Design (Web Application)

Justification: MyMapper needs to deal with large amount of data stream, so obviously we can not only build a Client. Because a Server is necessary, and our server will support a big database (the database will be on multi machine). Server will do the bulk of the calculations and data searching, and Client side will only send request with basic informations. Most important role of Client is the user interface, which will be a user friendly web client.

Design Issue #3 Database system selection

Options:

1. Spark
2. Hadoop
3. Google Cloud SQL

Decision: Spark

Justification: MyMapper needs to process a large amount of data, so the processing efficiency is very important in our project. Thus, we decide to choose Spark as our database framework because it is much faster than Hadoop. In addition, we want to save the cost of this project, so we drop the option of using Google Cloud SQL since it needs to be paid for using.

Design Issue #4 Backend Programming language

Options:

1. Java
2. C++

3. Python

Decision: Java

Justification: Spark contains support for only Java, Scala, and Python, so C++ is out of the question. We are all much more familiar with Java, so we went with Java to ease the development process.

Design Issue #5 Map information

Options:

1. OpenStreetMap
2. IMapper

Decision: OpenStreetMap

Justification: It provides detailed information of all the places in a specific area. And we can get all those information by export an OMS file from the openstreetmap website. The OpenStreetMap is free and open source.

Design Issue #6 Client Design

Options:

1. Thin Client
2. Fat Client
3. Balanced Client

Decision: Thin Client

Justification: We choose thin client because we want to centralize our map and point information on our spark structured database. On the one hand, thin client could enhance our web application performance, providing faster loading and faster searching. On the other hand, spark structured database could perfectly handle fat server issues because it is a distributed database system that data are stored in different servers in order to increase the system's stability and security.

Design Issue #7 Query Language

Options:

1. SQL
2. QBE
3. Datalog

Decision: SQL

Justification: We choose SQL firstly because SQL is the most common use query language and we all have some basic experience of it. Even though QBE did a good job formatting tables and Datalog did a good job depicting rules, SQL is the only language supported by SPARK, otherwise we have to build our own parser.

3.2 Non-functional Issues:

Design Issue #1: Version Control

Options:

1. GitHub
2. TortoiseSVN
3. BitBucket

Decision: GitHub

Justification: We made this decision based on the fact that all teammates are all familiar with GitHub and it's a distributed version control system. Though Bitbucket is similar to Github, we choose GitHub because we all have GitHub accounts. Centralized version control systems such as TortoiseSVN are easy to use but hard to manage a server and backups. GitHub can help us merge codes, synchronize our codes and trace the changes in an efficient manner, therefore our decision is GitHub.

Design Issue #2: Listing point of interest

Options:

1. Let the user to search for the POI.
2. Only give the user a list of description and let user choose

3. Give user a list of POI interest picture

Decision: Let user select from a list of POI description

Justification: Because if we let user can search for POI, It will be another searching mechanism to add. Because we have a type for each POI to add, we can easily classify all the POI to let user to choose. However if we use all the picture instead of description, the interface of the POI searching will be not neat. So the list of description is the best choice.

Design Issue #3: User type

Options:

1. Program works same to all users.
2. Program offers some privileges to some special users.

Decision: Program offers some privileges to some special users.

Justification: Because some users like hotel manager, restaurant manager or small business owners they may want to update their businesses information very frequently. So we want to make those kinds of users have the right to update the information.

Design Issue #4: User login design

Options:

1. MyMapper login (sign up from MyMapper and login)
2. Associate login (login from Facebook, Purdue ID...)
3. No login required

Decision: No login required at this moment

Justification: We want to provide no-difference service to every user because we are based on open street map. Also we believe when you are using our application while driving, it is dangerous and inconvenience to ask for a login before start searching points of interests. After all, user want it to be simple and fast. However, we may consider some methods to identify users if time allows, because we do want to

remember each user's preference in order to provide better service.

4. Design Details

4.1 Class Level Design

Class name: Server

● Basic class for Server, which stores the information of IP address and connection of database and the Client connection.

●Server will management all the activity of Server such as responding to the user activity and call the Info Collector to gather information.

●Server class will interact directly with validation class when there is any requests passed to server from client. Then server will interact with information collector and updater to get data information from database or modify data in database. Server will directly return the data to client class after processing each request.

Class name: Static Point

●The class that deal with static point, which is also called Point of Interest.

●This class will be called when User ask Client to use a query to search for a Static Point of Interest (both search within a distance and search the nearest location), it will react and print the points on the map by interacting with UI class.

Class name: User Classification

●This class will record with the user activity for further usage.

●When user do a search, this class will record the activity and give the searching option in the future searches from the user. This class will interact directly with client class.

Class name: Moving Point

●This is the class called by Client to deal with activity of moving point

●When the UI send a request to client need information from database about a moving object type this Class will be called and put all the point on the map by interacting with UI class.

Class name: Client

●This is the basic class that react to all the UI request and send request to Server, and process the data returned by server.

●This class will interact with Static point class, moving point class and UI classes to perform all UI layer changes. This class will also interact with User Classification to handle user preferences. When there are request from the user, Client class will pass all requests to validation class to validate the requests before processing by server.

Class name: Validation

●Will be called when any user request passed from Client to Server, so this class interact directly with client class and server class.

●This class will validated all user request passing to server, in order to make sure all request being processed are logically correct and secure.

Class name: UI class

●General UI class, deal with all the UI activity such as show map, give user searching selection refresh page

●This class will call Client whenever the user need any information from the server. This class will interact with Query selection view when user is selecting queries from the web interface. This class will also call point UI layer and map UI layer when points need to move or map need to change, these interaction will simply changes marking new point locations and showing new

map segments.

Class name: Query Selection View

●This is a UI class for query selection. This class will provide user several drop-down selection bars. Users will selection what kinds of POIs/moving points and what is the range they are going to search. Also, they can select what is the range shape, either a circle around them or a shell range in front of them.

●This class will call UI class when user is selecting queries.

Class name: Point UI layer

●This is a layer in UI class for marking points according to their latlang values. Moving points will refresh their location according to a refreshing rate.

●This class interact with UI class to mark points on the web interface.

Class name: Map UI layer

● This is another layer in UI class for displaying map basic information. All map data will be included from open street map, so this class will show a similar layout as open street map's web UI.

●This class interact with UI class to show maps on the web interface.

Class name: Info Collector

●Will be called when the server need to gather data from the database. So this class interact with server class and will directly query from database.

●This class main job is to connect server and database, after getting basic information from server, this class will request data from database and return the results.

Class name: Updater

●Will be called when the server need to update information in the

database. So this class interact with server class and will directly modify data in database.

●This class main job is to monitor the process when server is updating information to the database in order to make sure the updating process won't corrupt the database by checking insert integrity, modify integrity and delete integrity.

Class name: Database

● Basic class for database, in charge of processing all query and updating requests passed from server and return with information.

●Database will provide valid and efficient algorithm to connect with distributed data centers, this is a specific design inherited from SPARK.

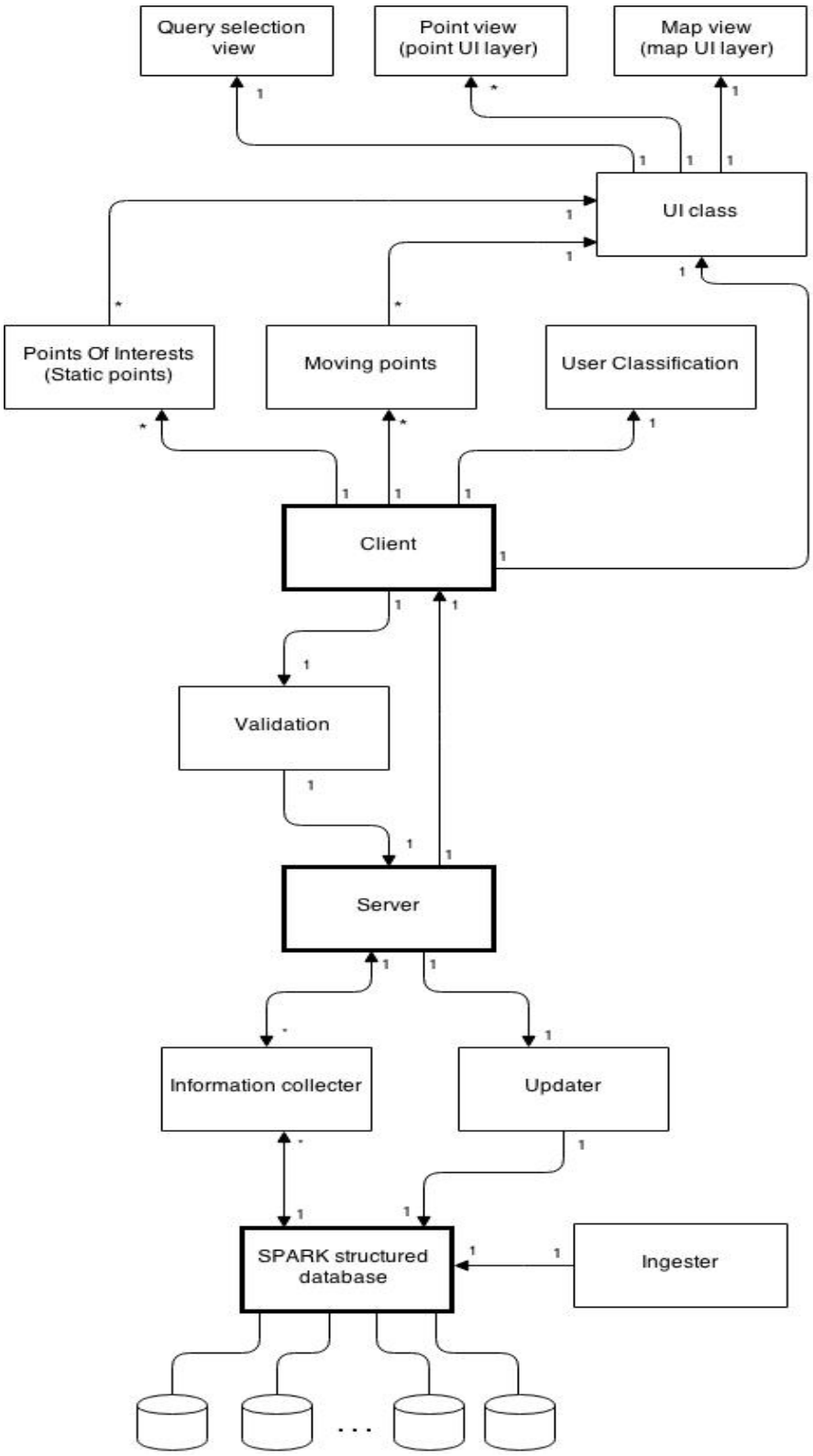
●This class will be called by Information collector and updater when there are requests coming from server. Also this class will be directly called from ingester continuously in order to update data in real time, such as make the moving points moved.

Class name: Ingestor

●Ingester class is closely linked with database class. The scripts in this class is in charge of modifying database, such as updating moving points' locations. It will recursively running in a time interval of Δt, which is a pre-set refreshing rate.

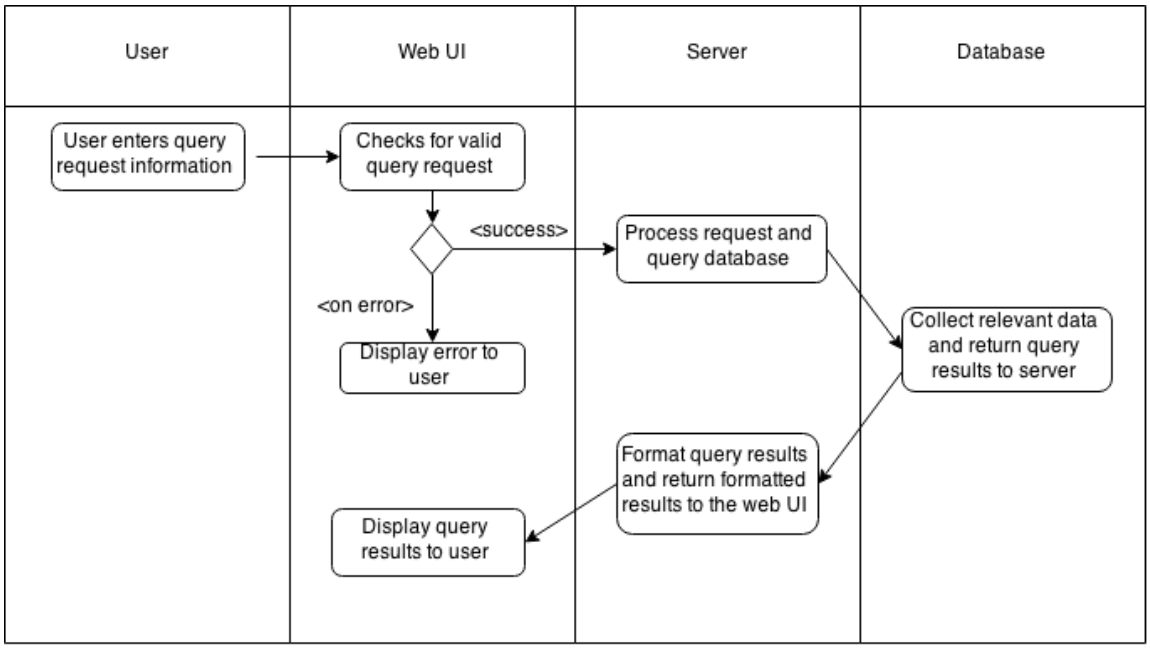
●Ingestor will automatically check data from OpenStreetMap and update the new insert data into the database. Also the Ingestor will update the location of the user, if we have the permission to access the user's location.

4.2 Class Diagram

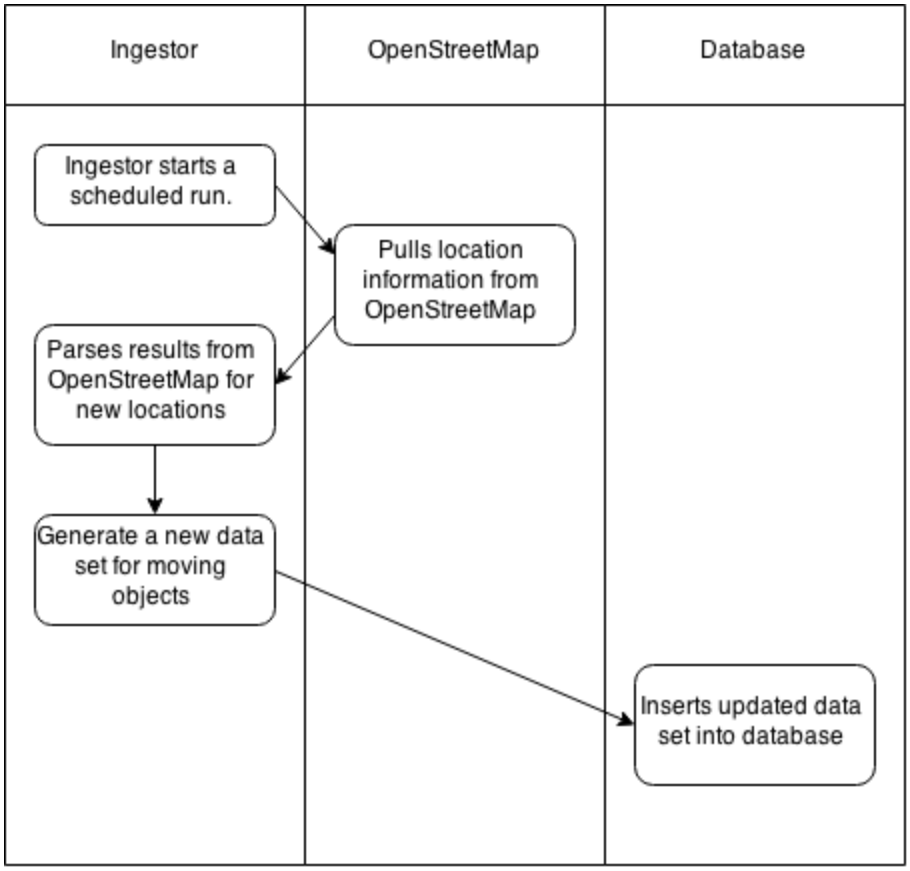


4.3 Sequence Diagram/Activity Logic Flow

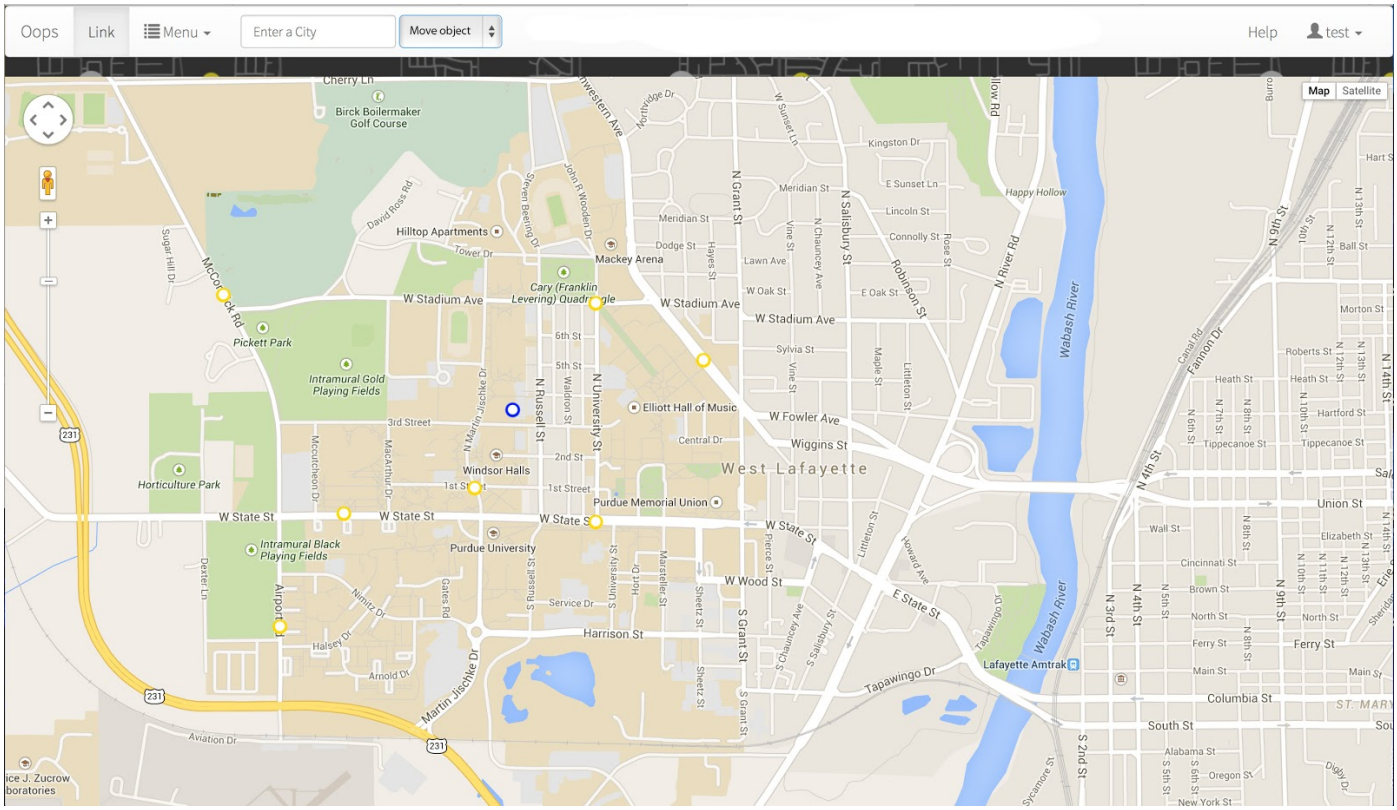
User Query Activity Diagram:

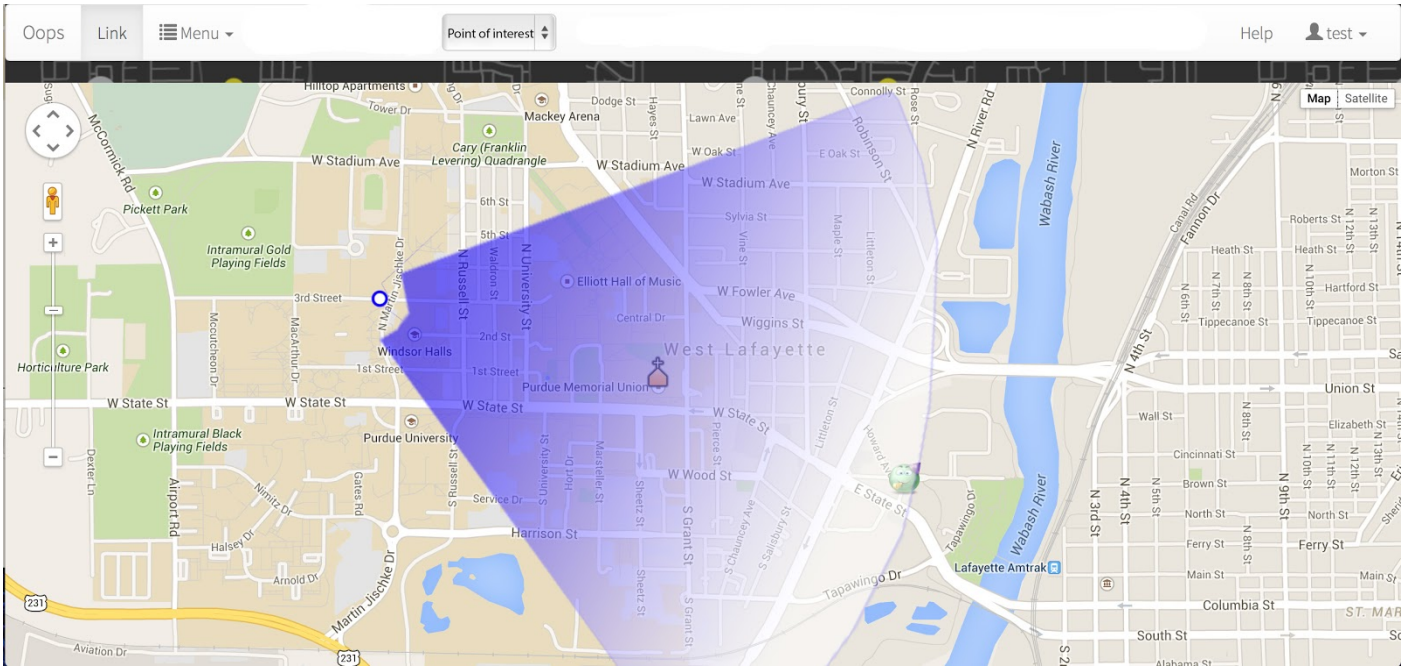


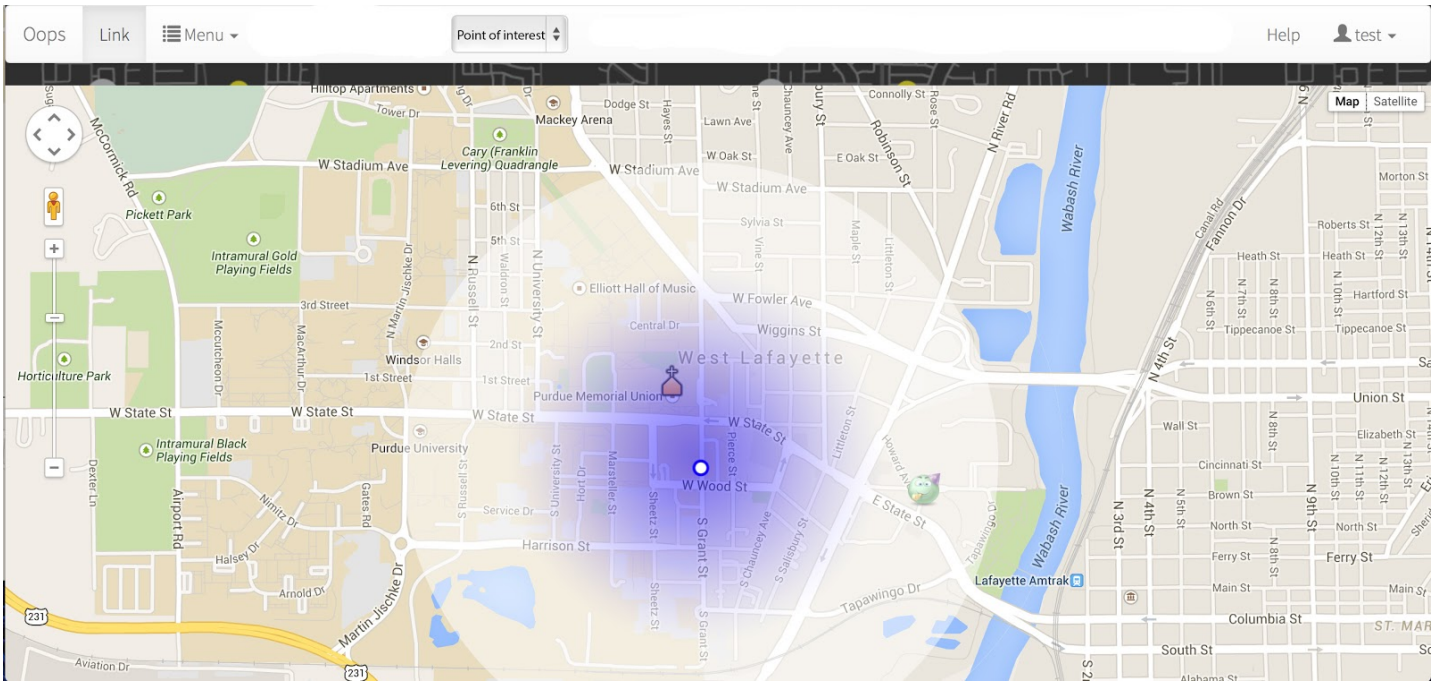
Ingestor Insertion Activity Diagram:



4.4 State Diagram/UI mockups




Oops
Link
Menu
Point of interest
Help
test
Map
Satellite
Hilltop Apartments
Tower Dr
Dodge St
Mackey Arena
Lawn Ave
Cary (Franklin Levering) Quadrangle
W Stadium Ave
W Oak St
E Oak St
Robinson St
N River Rd
Connolly St
Rose St
Wabash River
Canal Rd
Fannon Dr
Morton St
Roberts St
Heath St
Hartford St
Tippecanoe St
McCormick Rd
Pickett Park
Intramural Gold Playing Fields
6th St
5th St
Sylvia St
Maple St
Littleton St
Vine St
Waldron St
N Martin Jischke Dr
N Russell St
N University St
Elliott Hall of Music
W Fowler Ave
Central Dr
Wiggins St
3rd Street
2nd St
1st Street
Windsor Halls
West Lafayette
Purdue Memorial Union
Horticulture Park
Mccutcheon Dr
MacArthur Dr
W State St
Intramural Black Playing Fields
Purdue University
Dexter Ln
Airport Rd
Nimitz Dr
Gates Rd
S Russell St
S University St
Hort Dr
Marstellar St
Service Dr
Sheetz St
S Grant St
S Chauncey Ave
S Salisbury St
W Wood St
Harrison St
Halsey Dr
Arnold Dr
Martin Jischke Dr
231
Aviation Dr
E State St
Howard Ave
Tapawingo Dr
Lafayette Amtrak
N 4th St
N 3rd St
N 9th St
Union St
Wall St
Elizabeth St
Cincinnati St
Brown St
North St
Ferry St
Main St
Columbia St
South St
Alabama St


Oops
Link
Menu
Point of interest
Help
test
Map
Satellite
Hilltop Apartments
Tower Dr
Dodge St
Mackey Arena
Lawn Ave
Cary (Franklin Levering) Quadrangle
W Stadium Ave
W Oak St
E Oak St
Robinson St
N River Rd
Connolly St
Rose St
Wabash River
Canal Rd
Fannon Dr
Morton St
Roberts St
Heath St
Hartford St
Tippecanoe St
McCormick Rd
Pickett Park
Intramural Gold Playing Fields
6th St
5th St
Sylvia St
Maple St
Littleton St
Vine St
Waldron St
N Martin Jischke Dr
N Russell St
N University St
Elliott Hall of Music
W Fowler Ave
Central Dr
Wiggins St
3rd Street
2nd St
1st Street
Windsor Halls
West Lafayette
Purdue Memorial Union
Horticulture Park
Mccutcheon Dr
MacArthur Dr
W State St
Intramural Black Playing Fields
Purdue University
Dexter Ln
Airport Rd
Nimitz Dr
Gates Rd
S Russell St
S University St
Hort Dr
Marstellar St
Service Dr
Sheetz St
S Grant St
S Chauncey Ave
S Salisbury St
W Wood St
Harrison St
Halsey Dr
Arnold Dr
Martin Jischke Dr
231
Aviation Dr
E State St
Howard Ave
Tapawingo Dr
Lafayette Amtrak
N 4th St
N 3rd St
N 9th St
Union St
Wall St
Elizabeth St
Cincinnati St
Brown St
North St
Ferry St
Main St
Columbia St
South St
Alabama St

4.5 Database Schema:

The database will store two types of objects. The first type are static objects (buildings, restaurants, hospitals, etc.) and the second type are dynamic objects (busses, taxis, police cars, etc.) Spark SQL supports importing JavaBeans to define the database's schema. The definition for these JavaBean classes are as follows:

StaticObject Class:

```
public static class StaticObject implements serializable {
	private int id;
	private int type;
	private double latitude;
	private double longitude;
	private String description;
}
```

DynamicObjects Class:

```
public static class DynamicObject implements serializable {
	private int id;
	private int type;
	private double latitude;
	private double longitude;
	private long timestamp;
	private String description;
}
```

4.6 Supported Queries:

With the database schema as it is, the system is able to support a number of different queries on the data set as a whole. These queries, listed below, will be used to build the functionality of the system, providing the user with a variety of different options when using the application. These queries

- GetNearestPOI(Type, Location): This query returns the point-of-interest of the specified type closest to the specified location. The Type field will be used to classify different categories of objects (restaurants vs libraries, McDonalds vs Burger Kings, etc.)

- GetAllPOIs(Type, SearchArea): This query returns all points-of-interest of the specified type within the specified search area.
- GetHistory(ID, TimeRange): This query returns all the past locations within the specified TimeRange of the moving object uniquely identified by the specified ID.