CS673F14 Software Engineering
Group Project - UPark
Software Design Document

Your project Logo here if any

Team Member	Role(s)	Signature	Date
Fanghui Zhang	Team Leader		09/12/14
Xin Shan	Configuration Leader		09/12/14
Lu Zhang	Environment and Integration Leader		09/12/14
Rui Li	Requirement Leader		09/12/14
Xi Tang	Design Leader		09/12/14
Yilun Xie	Implementation Leader		09/12/14
Feiyu Shi	QA Leader		09/12/14
Mingli Li	Architect	*Mingli Li*	09/12/14

Revision history

Version	**Author**	**Date**	**Change**
1.0	**Fanghui Zhang**	**11/13/2014**	
1.1	**Fanghui Zhang Mingli Li**	**12/07/2014**	

1. Introduction

This software design documentation describe the detailed structure of the components of the software and the implementation to satisfy the requirement as specified in the SPPP. It assumed that the reader has read the SPPP.

The design description defined in this document serves purposes below:

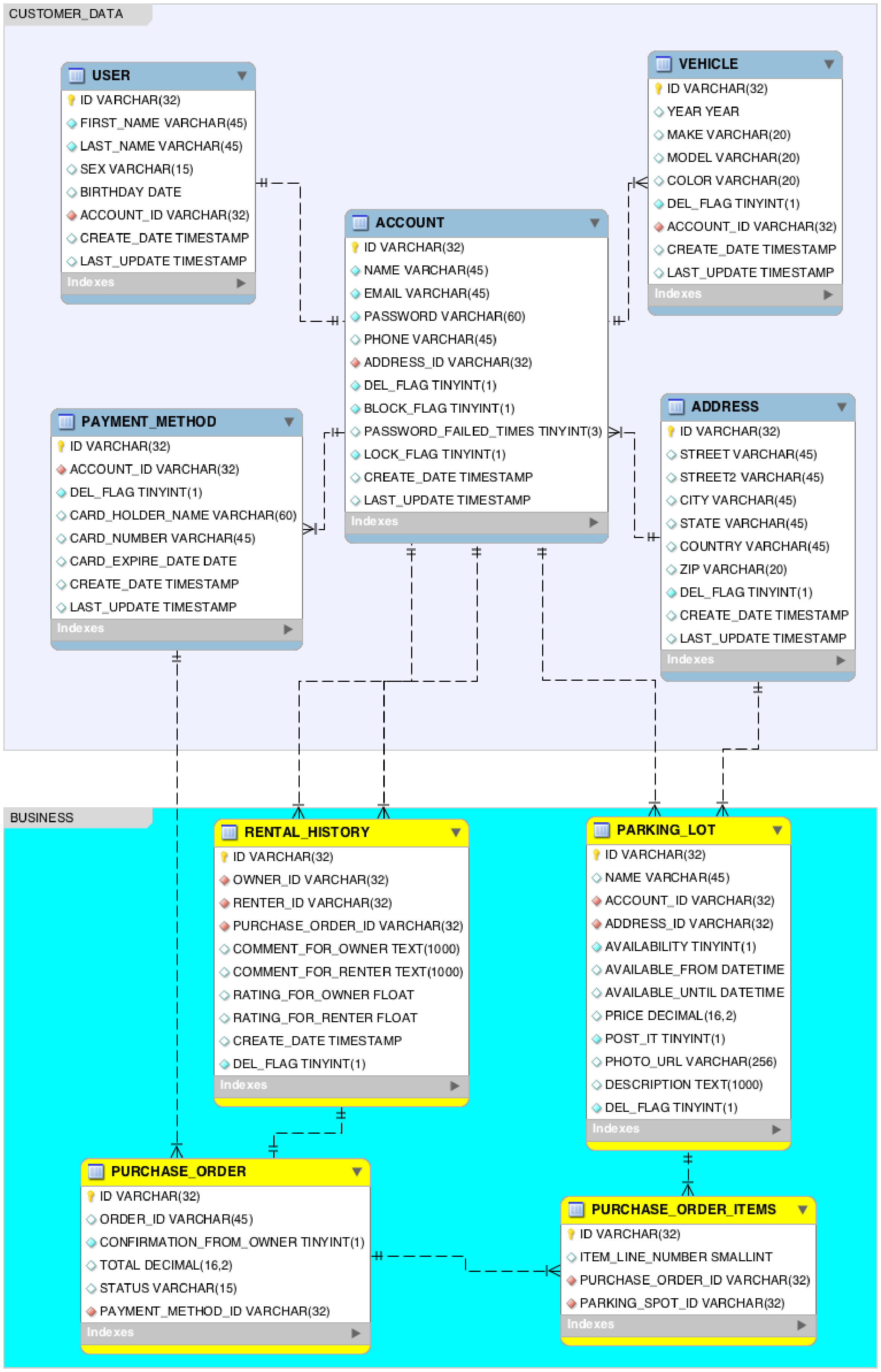
- To describe the functional structure, data and algorithms to be implemented
- To help to produce test cases
- To be used to verify compliance with requirement
- To help for further developing (If needed)

As is stressed in the SPPP, the goal of our software system is to build a online parking marketplace. We connect renters with drivers through our platform and provide them services they need to finish the rent activity.

2. Software Architecture

In this section, you will describe the decomposition of your software system, which include each component (which may be in terms of package or folder) and the relationship between components. You shall have a diagram to show the whole architecture, and class diagram for each component. The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too. Database design should also be described if used.

2.1 Database Design
2.1.1 Database Schema
MySQL is the chosen database for this project, for it is open source and capable of dealing with large amount of transaction data of the website. Below is the database schema for the project. It not only describes the entity structures but also shows the relationships between them. This schema only depicts the client-side of the system; administration-side system may be designed later as the system extends. The content of the schema might also be changed due to possible requirement adjustment.


CUSTOMER_DATA
USER
ID VARCHAR(32)
FIRST_NAME VARCHAR(45)
LAST_NAME VARCHAR(45)
SEX VARCHAR(15)
BIRTHDAY DATE
ACCOUNT_ID VARCHAR(32)
CREATE_DATE TIMESTAMP
LAST_UPDATE TIMESTAMP
Indexes
VEHICLE
ID VARCHAR(32)
YEAR YEAR
MAKE VARCHAR(20)
MODEL VARCHAR(20)
COLOR VARCHAR(20)
DEL_FLAG TINYINT(1)
ACCOUNT_ID VARCHAR(32)
CREATE_DATE TIMESTAMP
LAST_UPDATE TIMESTAMP
Indexes
ACCOUNT
ID VARCHAR(32)
NAME VARCHAR(45)
EMAIL VARCHAR(45)
PASSWORD VARCHAR(60)
PHONE VARCHAR(45)
ADDRESS_ID VARCHAR(32)
DEL_FLAG TINYINT(1)
BLOCK_FLAG TINYINT(1)
PASSWORD_FAILED_TIMES TINYINT(3)
LOCK_FLAG TINYINT(1)
CREATE_DATE TIMESTAMP
LAST_UPDATE TIMESTAMP
Indexes
PAYMENT_METHOD
ID VARCHAR(32)
ACCOUNT_ID VARCHAR(32)
DEL_FLAG TINYINT(1)
CARD_HOLDER_NAME VARCHAR(60)
CARD_NUMBER VARCHAR(45)
CARD_EXPIRE_DATE DATE
CREATE_DATE TIMESTAMP
LAST_UPDATE TIMESTAMP
Indexes
ADDRESS
ID VARCHAR(32)
STREET VARCHAR(45)
STREET2 VARCHAR(45)
CITY VARCHAR(45)
STATE VARCHAR(45)
COUNTRY VARCHAR(45)
ZIP VARCHAR(20)
DEL_FLAG TINYINT(1)
CREATE_DATE TIMESTAMP
LAST_UPDATE TIMESTAMP
Indexes
BUSINESS
RENTAL_HISTORY
ID VARCHAR(32)
OWNER_ID VARCHAR(32)
RENTER_ID VARCHAR(32)
PURCHASE_ORDER_ID VARCHAR(32)
COMMENT_FOR_OWNER TEXT(1000)
COMMENT_FOR_RENTER TEXT(1000)
RATING_FOR_OWNER FLOAT
RATING_FOR_RENTER FLOAT
CREATE_DATE TIMESTAMP
DEL_FLAG TINYINT(1)
Indexes
PARKING_LOT
ID VARCHAR(32)
NAME VARCHAR(45)
ACCOUNT_ID VARCHAR(32)
ADDRESS_ID VARCHAR(32)
AVAILABILITY TINYINT(1)
AVAILABLE_FROM DATETIME
AVAILABLE_UNTIL DATETIME
PRICE DECIMAL(16,2)
POST_IT TINYINT(1)
PHOTO_URL VARCHAR(256)
DESCRIPTION TEXT(1000)
DEL_FLAG TINYINT(1)
Indexes
PURCHASE_ORDER
ID VARCHAR(32)
ORDER_ID VARCHAR(45)
CONFIRMATION_FROM_OWNER TINYINT(1)
TOTAL DECIMAL(16,2)
STATUS VARCHAR(15)
PAYMENT_METHOD_ID VARCHAR(32)
Indexes
PURCHASE_ORDER_ITEMS
ID VARCHAR(32)
ITEM_LINE_NUMBER SMALLINT
PURCHASE_ORDER_ID VARCHAR(32)
PARKING_SPOT_ID VARCHAR(32)
Indexes

2.1.2 Details of Tables

Tables are roughly divided into two groups: customer data and business data. Customer data includes user and account information. Payment, address, and vehicle are also stored. Business data contains parking lot information, rental history, orders and order items. These are essential in transactions.

All tables have an 'id' as its primary key. It's a 32 bit varchar, storing a 32 bit UUID (universally unique identifier). This ensures the uniqueness of every entry in the database.

Some of tables have an 'del_flag', which is a boolean type data. It is true when the entry is "deleted" by the user or administrator. The entry is not actually removed from the table; this prevents problems when the data is referenced by another table but is deleted in the current table.

We will outline each table below:

User table:

Primary information of the user, including names, sex, birthday, etc. It also contains the id of the accounts belonging to the user. Statistics like creation and last update date are also recorded.

Account table:

An active image of the user on the website. It includes primary security information, for example, email, password, phone, and some flags for reset account. User will use these information to login the website and involve in transactions with other users.

Vehicle table:

Basic vehicle information associated with the account. Information are like year, make, model, color, etc.

Payment method table:

Credit card information associated with the account. This type of payment method may not be used as it might be up-front cash transaction.

Address table:

Address information associated with the account or parking lot.

Parking lot table:

Information for available parking lots for rent. Information like available period, price, description are included. 'post it' flag is for search display use.

Rental history table:

It records the history of every transactions between users. It also includes ratings and comments for the owner of parking lots and renters. Each history entry contains one order id linked to an order.

Purchase order table:

This contains some information for the order, for example, subtotal, status, payment method, etc. It contains several order items (id).

Purchase order item table:

This entry includes a parking lot id, and its associated order id. It is 'one item' in an order.

2.1.3 Controllers

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.
Additionally, the `@Controller` annotation indicates that a particular class serves the role of a *controller*.

LoginController
---LoginService()
RegisterController
---RegisterService()
LogoutController
---LogoutService()
SearchController
---SearchService()
LogoutController
---LogoutService()
InfoReviewController
--InfoReviewService()
InfoDeleteController
---InfoDeleteService()

2.1.4 Restful API

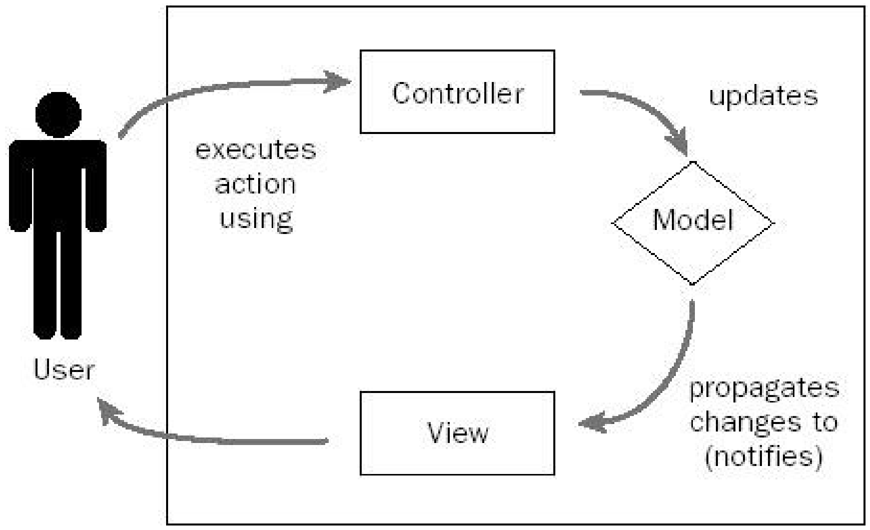
1 /login GET
Description: login into Upark.
2 /logout GET
Description: logout Upark.
3 /register POST
Description: register into Upark.
4 /search POST
Description: search loaction information.
5 /review POST
Description: review user's posted location infomation.
6 /delete DELETE
Description: delete posted location infomation

3. Design Patterns

- MVC
  We use model-view-controller(MVC) to isolate business logic from the user interface. Using MVC, the model represents the information(data) of the application and the business rules used to manipulate the data, the view corresponds to elements the user interface as we will show on the demo, and

the controller manages details involving the communication between the model and view. The controller handles user actions and pipes them into the model or view as required.

Additionally, we use spring web MVC framework, the request processing workflow of the Spring Web MVC is illustrated in the following diagram.



4. Key Algorithms

In this section, you shall describe any key algorithms used in your software system, either in terms of pseudocode or flowchart.

In this iteration, we are going to implement the basic search algorithm, we call this “stupid algorithm”, we will test the performance to see if we can improve.
Figure 1 shows how search function goes in every step. Figure 2 shows the diagram of the “stupid algorithm”, basically, the parameter will be decided by mathematical calculation.That is, the conversion for kilometers and latitude is about 111km/1°, and for kilometers and longitude is about 111cosα km/1°.

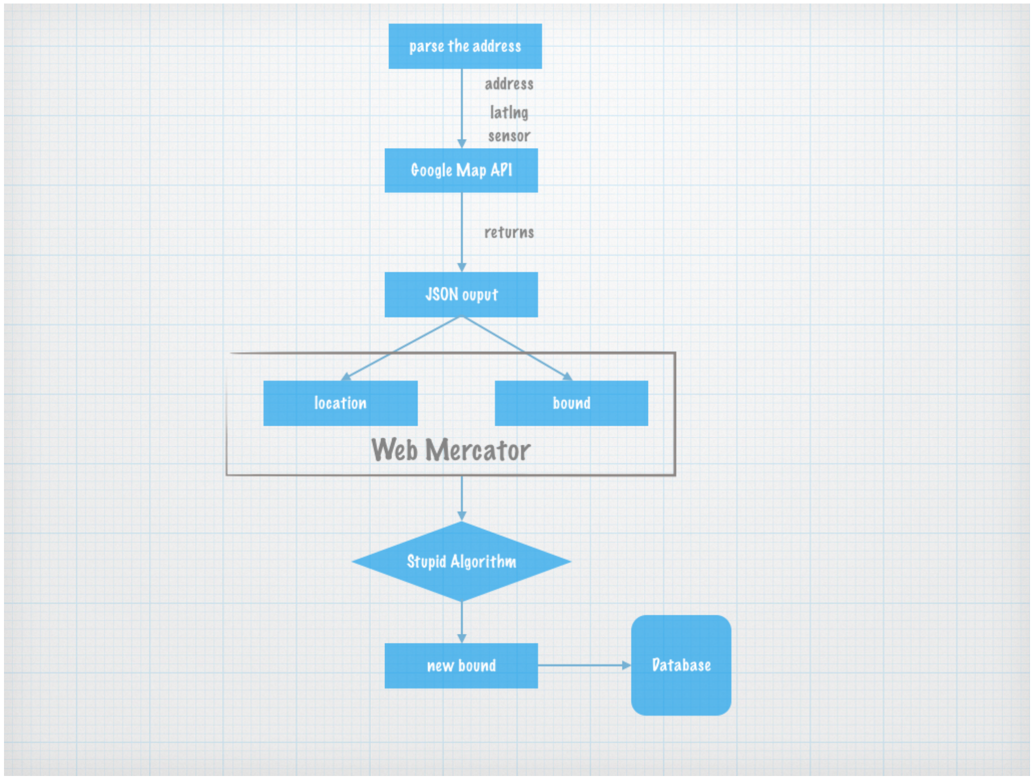


Figure 1

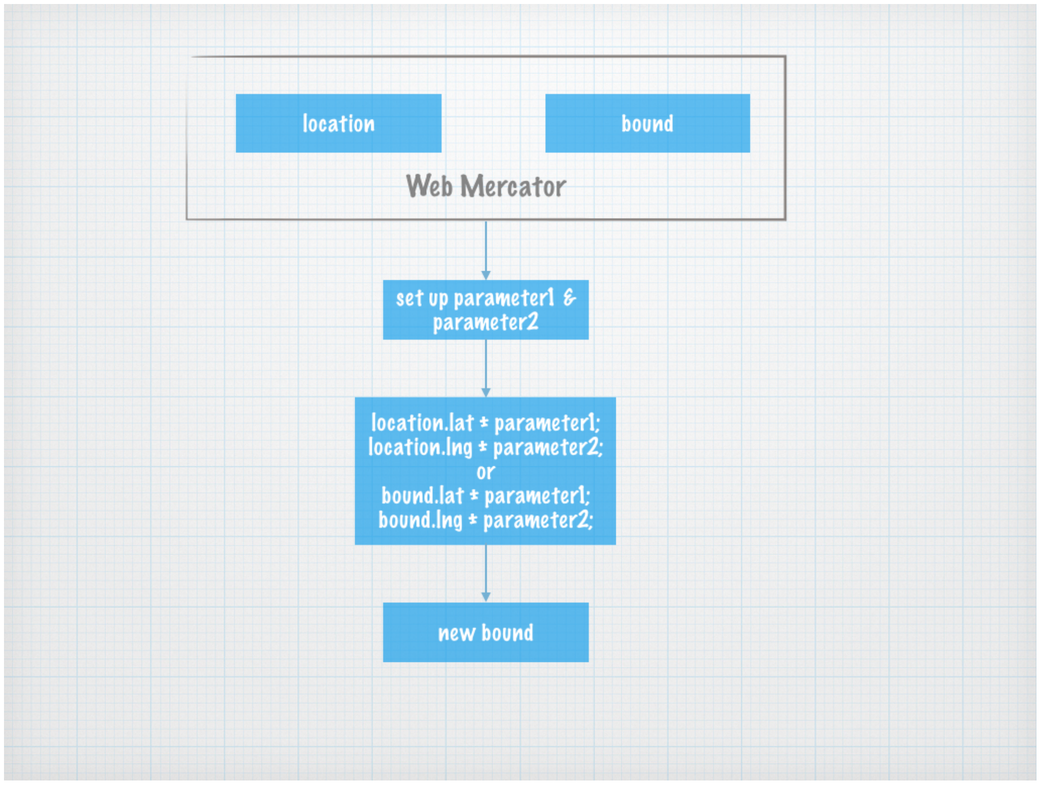
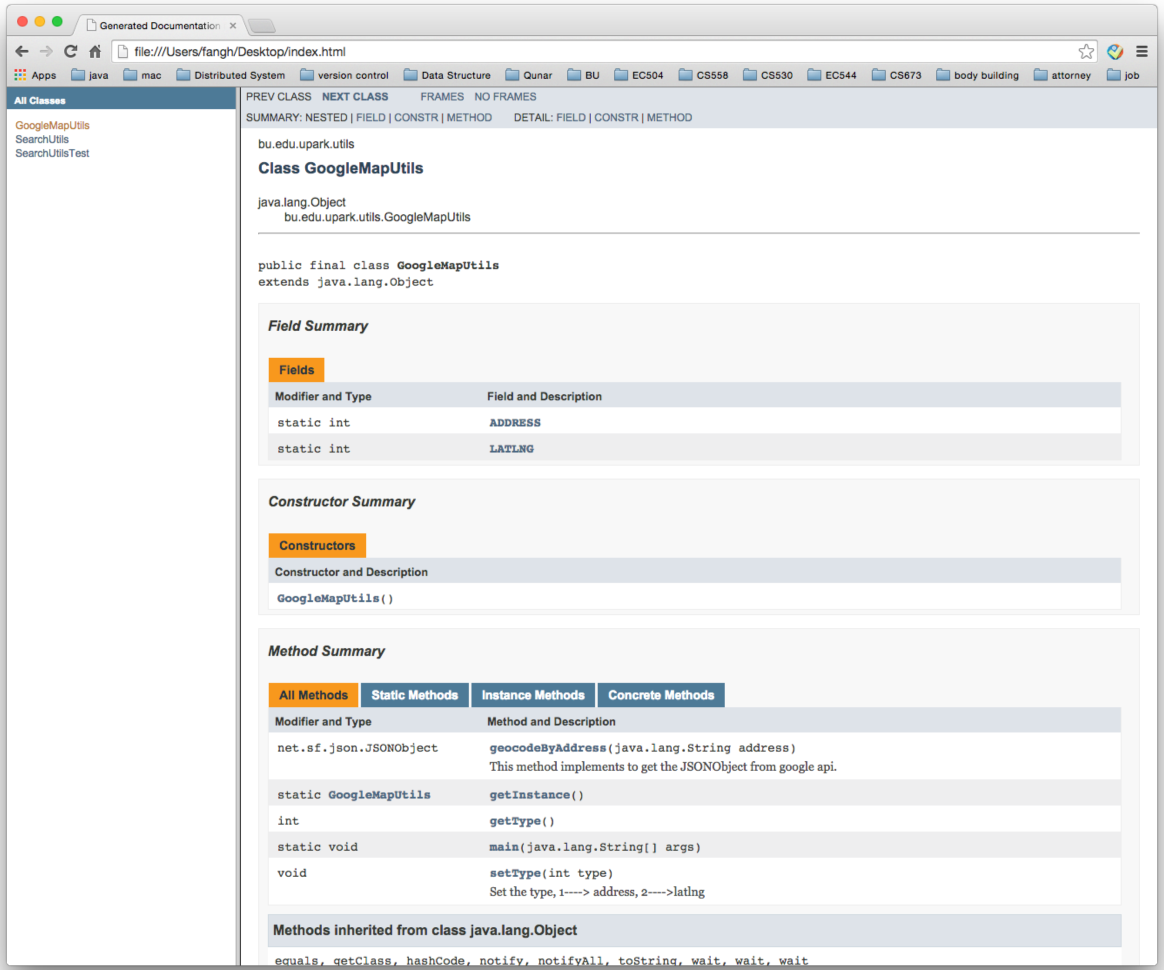

location
bound
Web Mercator
set up parameter1 & parameter2
location.lat ± parameter1;
location.lng ± parameter2;
or
bound.lat ± parameter1;
bound.lng ± parameter2;
new bound

Figure 2 “Stupid Algorithm” for searching

5. Classes and Methods

This part can be a reference to automatic generated document for all classes and methods.

Utility Methods for parsing data from google api and return formatted address:


Generated Documentation
file:///Users/fangh/Desktop/index.html
Apps
java
mac
Distributed System
version control
Data Structure
Qunar
BU
EC504
CS558
CS530
EC544
CS673
body building
attorney
job
All Classes
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

GoogleMapUtils
SearchUtils
SearchUtilsTest

bu.edu.upark.utils

Class GoogleMapUtils

java.lang.Object
    bu.edu.upark.utils.GoogleMapUtils

```
public final class GoogleMapUtils
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	**ADDRESS**
`static int`	**LATLNG**

Constructor Summary

Constructors

Constructor and Description
`GoogleMapUtils()`

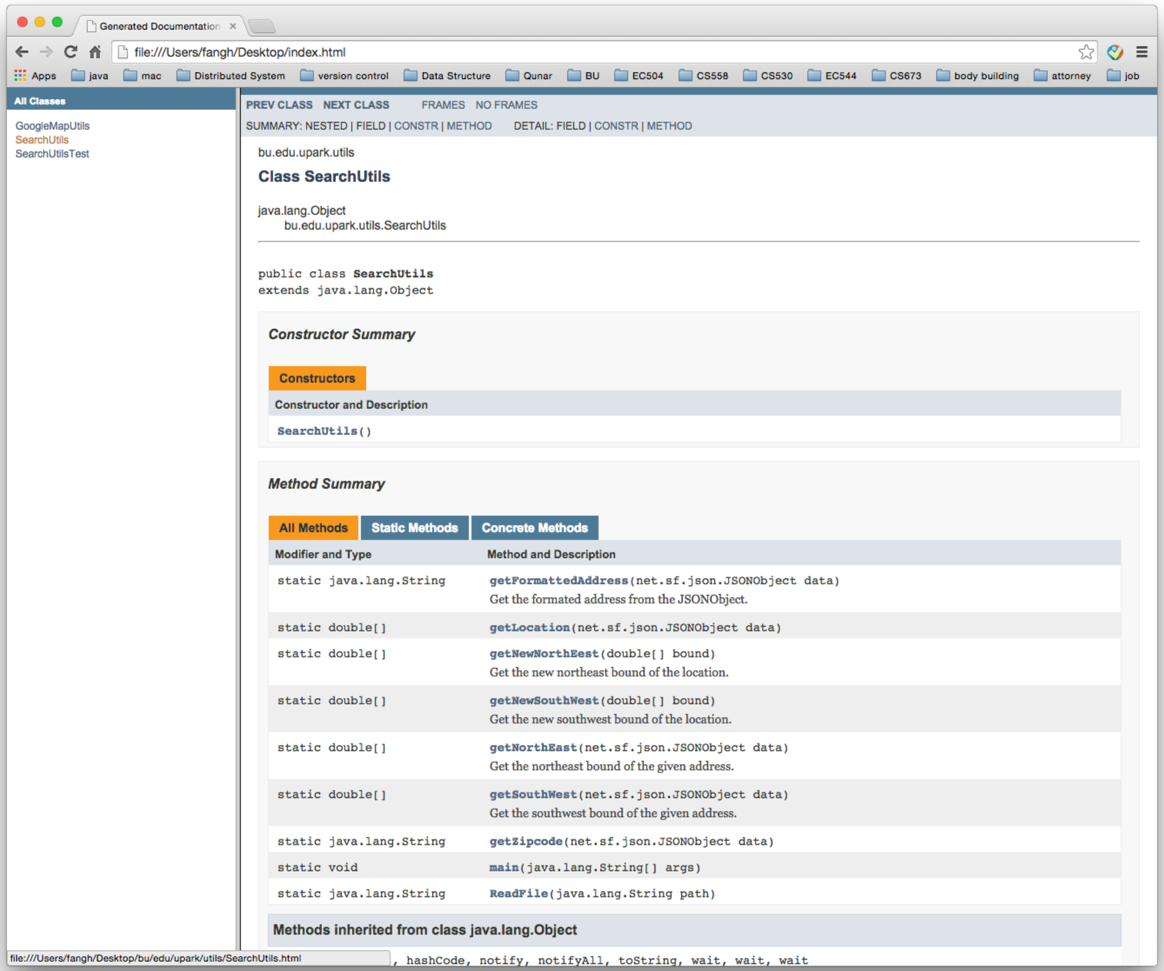
Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`net.sf.json.JSONObject`	`geocodeByAddress(java.lang.String address)` This method implements to get the JSONObject from google api.
`static GoogleMapUtils`	`getInstance()`
`int`	`getType()`
`static void`	`main(java.lang.String[] args)`
`void`	`setType(int type)` Set the type, 1----> address, 2---->latlng

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`


Generated Documentation
file:///Users/fangh/Desktop/index.html
Apps
java
mac
Distributed System
version control
Data Structure
Qunar
BU
EC504
CS558
CS530
EC544
CS673
body building
attorney
job
All Classes
GoogleMapUtils
SearchUtils
SearchUtilsTest
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
file:///Users/fangh/Desktop/bu/edu/upark/utils/SearchUtils.html

bu.edu.upark.utils

Class SearchUtils

java.lang.Object
    bu.edu.upark.utils.SearchUtils

```
public class SearchUtils
extends java.lang.Object
```

Constructor Summary

Constructors

Constructor and Description
`SearchUtils()`

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
`static java.lang.String`	`getFormattedAddress(net.sf.json.JSONObject data)` Get the formated address from the JSONObject.
`static double[]`	`getLocation(net.sf.json.JSONObject data)`
`static double[]`	`getNewNorthEest(double[] bound)` Get the new northeast bound of the location.
`static double[]`	`getNewSouthWest(double[] bound)` Get the new southwest bound of the location.
`static double[]`	`getNorthEast(net.sf.json.JSONObject data)` Get the northeast bound of the given address.
`static double[]`	`getSouthWest(net.sf.json.JSONObject data)` Get the southwest bound of the given address.
`static java.lang.String`	`getZipcode(net.sf.json.JSONObject data)`
`static void`	`main(java.lang.String[] args)`
`static java.lang.String`	`ReadFile(java.lang.String path)`

Methods inherited from class java.lang.Object

`, hashCode, notify, notifyAll, toString, wait, wait, wait`

6. RESTful API

GET /user

Description:
This method will return information about all users in the database.
Input:
None

Returns:

```
<data>
  <batch_program_data>
    <batch_program batch_name="HealthCheck"
                   platform_type="VNX"
                   program_name="Rockies">
      <identities>
        <identity name="batch_program_id"
                   value="1" />
        <identity name="batch_info_id"
                   value="1" />
      </identities>
    </batch_program>
    <batch_program batch_name="Upgrade"
                   platform_type="VNX"
                   program_name="Rockies">
      <identities>
        <identity name="batch_program_id"
                   value="1" />
        <identity name="batch_info_id"
                   value="2" />
      </identities>
    </batch_program>
    <batch_program batch_name="HealthCheck"
                   platform_type="VNXe"
                   program_name="KittyHawk">
      <identities>
        <identity name="batch_program_id"
                   value="3" />
        <identity name="batch_info_id"
                   value="3" />
      </identities>
    </batch_program>
    <batch_program batch_name="Upgrade"
                   platform_type="VNXe"
                   program_name="KittyHawk">
      <identities>
        <identity name="batch_program_id"
                   value="3" />
        <identity name="batch_info_id"
                   value="4" />
      </identities>
    </batch_program>
  </batch_program_data>
</data>
```

7. References

[1] Web MVC framework
http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[2] Design Patterns
http://sourcemaking.com/design_patterns

[3] Example of Software Design Document
http://portal.unimap.edu.my:7778/portal/page/portal30/Lecturer%20Notes/KEJURUTERAAN_KOMPUTER/Semester%202%20Sidang%20Akademik%2020112012/EKT420%20Software%20Engineering/Example%20of%20Software%20Design%20Document(SDD)/EDDISS.pdf

8. Glossary