CS 441 Project Technical Documentation

iGo

Group Members
Chetana Vedantam
Harry Cordewener
Kanishka Garg
Lokesh Gunda
Shouvik Sayef
Sudhanshu Malhotra

Section 1: Introduction

iGo is a software application focused on improving travel experiences by making relevant travel data easy to access. Outside of this primary functionality, iGo will also serve as a tool for performance comparison of SQL vs noSQL databases over distributed systems.

Section 2: Architecture Overview

iGo would be implemented using 3-tier architecture consisting of Presentation Tier, Logic Tier, and Data-Access Tier.

Presentation Tier: This is the top-most tier in our application. End-user interacts with this tier in order to give inputs and see the outputs. It communicates with the logic tier using RMI technology. This tier calls methods on remote objects hosted by the logic tier.

Logic Tier: This is the middle layer in our application. It controls the application functionality by doing major processing activities like handling request loads from all the clients, distributing the requests, performing computations, interacting with the data access tier, and preparing the results to be sent to the presentation tier. It implements the remote interface on which presentation layer invoke method calls.

Data Access Tier: This is the innermost layer in our application. This tier provides mechanisms to access and manipulate data by constructing queries. It interacts with relational databases using JDBC technology. In order to communicate with NoSQL databases, it sends REST service call, and obtains data in the JSON format.

Reasons for using 3-tier architecture:

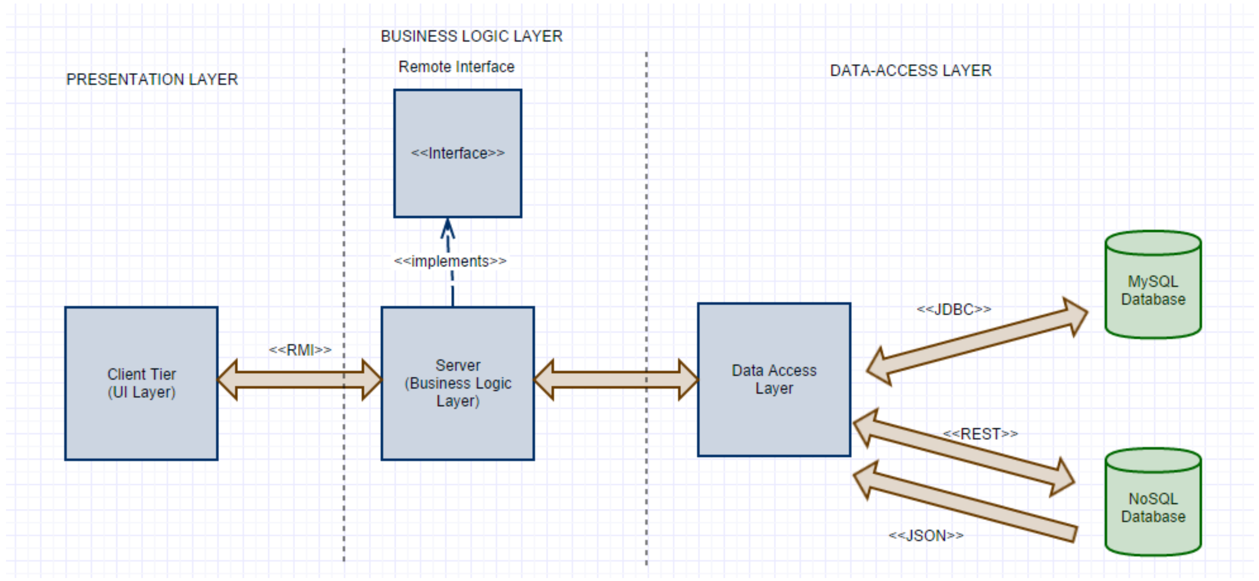
1. Flexibility: By separating the business logic of an application from its presentation logic, a 3-Tier architecture makes the application much more flexible to changes.

2. Maintainability - Changes to the components in one layer should have no effect on any others layers. Also, if different layers require different skills (such as HTML/CSS is the presentation layer, PHP/Java in the business layer, SQL in the data access layer) then these can be managed by independent teams with skills in those specific areas.

3. Reusability - Separating the application into multiple layers makes it easier to implement reusable components. A single component in the business layer, for example, may be accessed by multiple components in the presentation layer, or even by several different presentation layers (such as desktop and the web) at the same time.

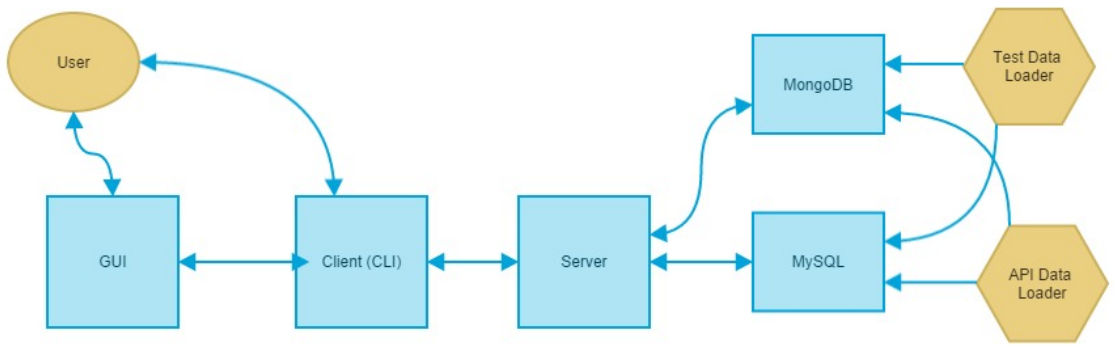
4. Scalability - A 3-Tier architecture allows distribution of application components across multiple servers thus making the system much more scalable.

5. Reliability - A 3-Tier architecture, if deployed on multiple servers, makes it easier to increase reliability of a system by implementing multiple levels of redundancy.



There are 6 major component types for our software:
-MySQL Database
-MongoDB Database
-Server Backend
-Client
-Client UI
-API Data Loader

There is an additional component we will use to provide data for testing purposes:
-Test Data Loader



Section 3: Component Overview

Database Handlers

We will implement a few scripts to automatically load our schema into existing databases. The initial data provided to our databases will allow functionality of all of our potential use cases as well as providing sample data to be manipulated when testing SQL vs noSQL performance.

MySQL

DESCRIPTION:
The MySQL loader will consist of an SQL script which contains our database schema as well as some hard coded sample data (Atleast 5-10 points per table). It should be able to detect if the schema already exist in an existing database and create any tables if necessary.

INPUT:
-Database Location (IP Address/hostname)
-Database Port
-Database Credentials

FUNCTIONS:
-Create and insert hardcoded schema and data into existing database

OUTPUT:
-Success/Failure Indication

MongoDB

DESCRIPTION:

The MongoDB Script will consist of either and Expect or JavaScript script which will interact with Mongo shell. The script will create the noSQL equivalent of the data and tables in the MySQL database.

INPUT:
-Database Location (IP Address/hostname)
-Database Port
-Database Credentials

FUNCTIONS:
-Create and insert hardcoded schema and data into existing database

API Loading Application

DESCRIPTION:
Our final database component will modify the existing databases created by the scripts above and add in data retrieved from available resources from existing APIs. The difference from the server is that the loading application will be a standalone implementation of the API

INPUT:
-Database Location (IP Address/hostname)
-Database Port
-Database Credentials
-API data specifics

FUNCTIONS:
-Implementation of API's to retrieve data
-Conversion of data for insertion
-Communication with both MySQL and MongoDB databases to insert gathered data

OUTPUT:
-Success/Failure Indication

2: Server Application

The server application will contain all remote functions for use with connected client applications. The server will primarily handle database transactions on behalf of the client applications, and will provide remote functions to the client so that users have an interface to interact with our data.

INPUT:
-IP Address/Port - to listen on
-Database credentials- IP addresses, ports, security credentials, etc.

LOCAL FUNCTIONS/METHODS:
-Connect to MySQL
-Connect to MongoDB
-Handle user authentication
-Load Balancing
-Query logging and execution

REMOTE FUNCTIONS/METHODS: (Remote functions should cover all functionality needed to implement our use cases).

OUTPUT:
-Function status indicators for console

3: Client Application

The client application will act as an API to gather and format the requested data for thefont

INPUT:
-Accept user input equivalents from frontend
-User credentials
-User input for function calls
-Remote function calls

FUNCTIONALITY:
-Will translate frontend input to method calls
-Testing application will run on command line without a front end allowing manual calls to functions and manual input of data

OUTPUT:
-Formatted data for display on frontend (JSON).
-Command line menu/data display

4: Frontend

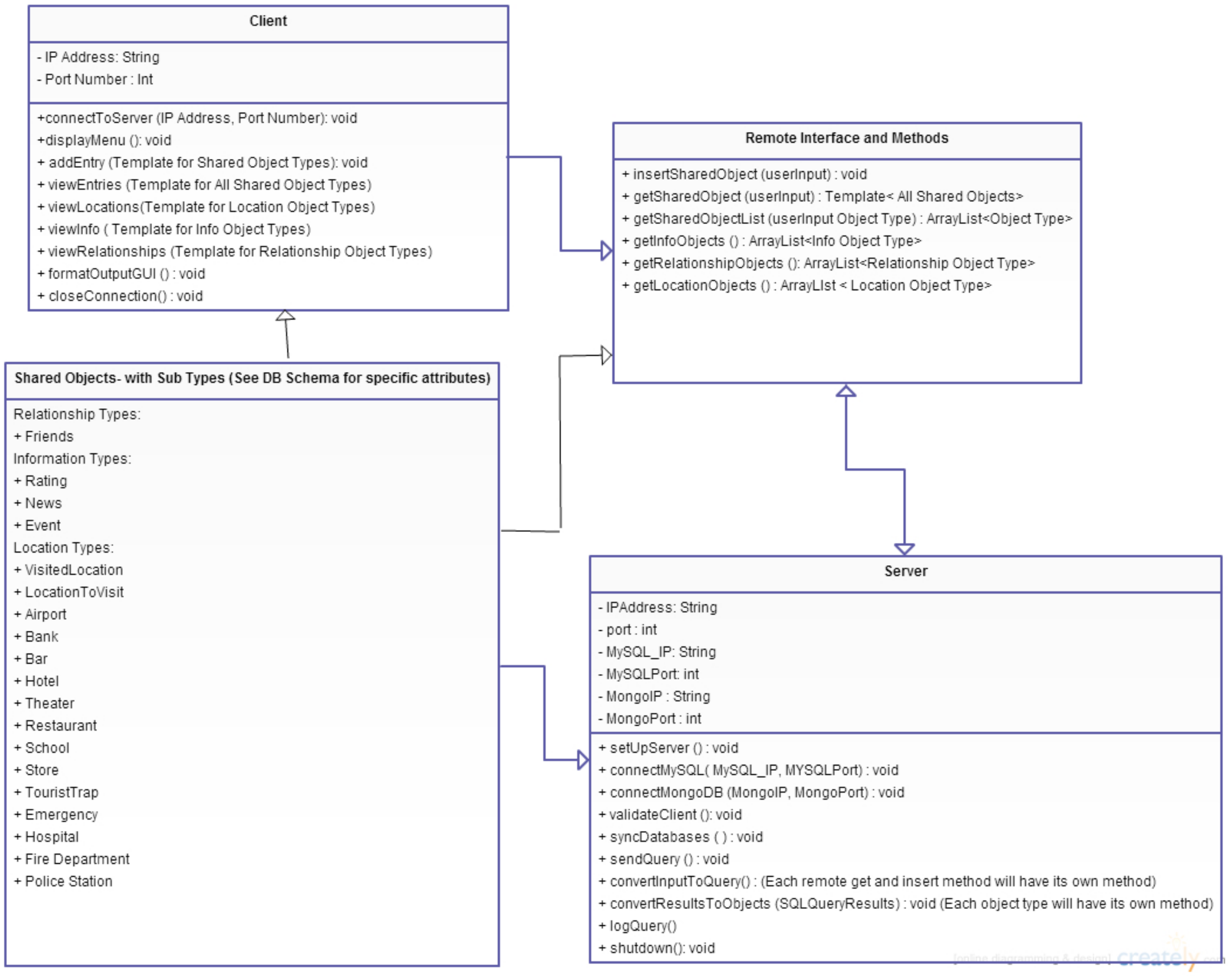
The frontend will act as a graphical equivalent to the command line version of the client program and allow forms for inserting data. The frontend will also display requested data in a neat and easier to read format. Our primary focus for the project skeleton will only include functionality up to the client CLI, if we have additional after the rest of the project is functional we will work on this graphical display.

Section 4: Table/Schema Specifications

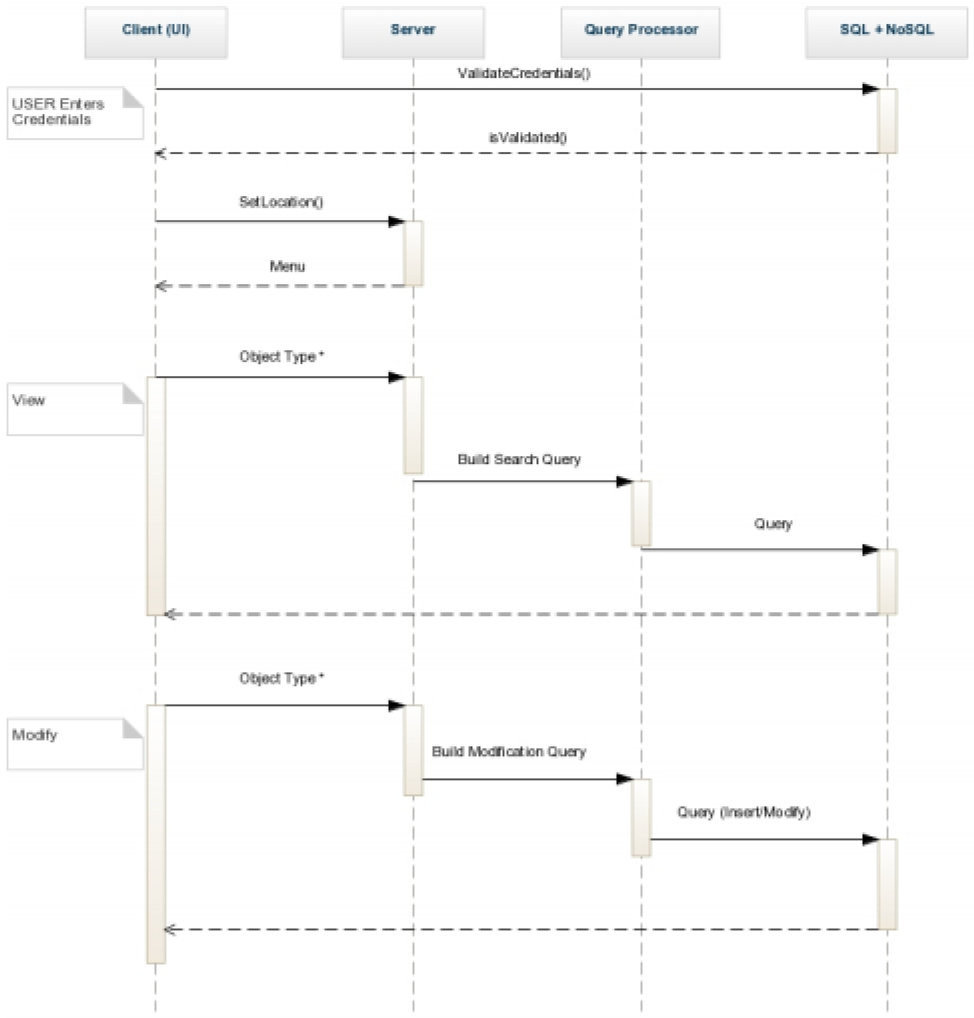
Table Name <category>	**Primary Key**	**Other Attributes**				
<Admin/ Relationships>						
User	**UserID**	**Name**	**eMailId**	**Mobile No**		
LocationVisited	**FriendID**	**LocationID**	**Occurences**			
Friendship	**FriendID**	**FriendName**	**UserID**			
LoginTable	**UserID**	**Password**	**SuperUser**	**FirstTileLogin**		
<Info Types>						
Reviews	**BusinessID**	**Rating**	**ReviewerID**	**Comments**	**Type**	
LocationNews	**LocationID**	**News**	**RSSFeeds**	**Date**		
Events	**EventID**	**EventName**	**EventDate**	**StartTime**	**EndTime**	**LocationID**
<Location Types>						
Location	**LocationID**	**LocationName**	**State**	**GPS Coordinate**		
LocationToVisit	**UserID**	**LocationID**	**StartDate**	**EndDate**		

Airport	**AirportID**	**AirportName**	**GPS Coordinate**	**Active**	**LocationID**	
Banks	**BankID**	**BankName**	**BankType**	**GPS Coordinate**	**Active**	**LocationID**
Bar	**BarID**	**BarName**	**BarType**	**GPS Coordinate**	**Active**	**LocationID**
Hotel	**HotelID**	**HotelName**	**HotelType**	**GPS Coordinate**	**Active**	**LocationID**
MovieTheater	**TheaterID**	**TheaterName**	**GPS Coordinate**	**ContactNo**	**Active**	**LocationID**
Restaurant	**RestaurantID**	**RestaurantName**	**RestaurantType**	**GPS Coordinate**	**Active**	**LocationID**
School	**SchoolID**	**SchoolName**	**ShoolType**	**GPS Coordinate**	**LocationID**	
Store	**StoreID**	**StoreName**	**StoreAddress**	**GPS Coordinate**	**Active**	**LocationID**
TouristPlaces	**TouristPlaceID**	**TouristPlaceName**	**Type**	**GPS Coordinate**	**LocationID**	
Emergency	**EmergencyID**	**LocationID**	**Type**			
Hospital	**HospitalID**	**HospitalName**	**HospitalType**	**ContactNumber**	**GPS Coordinate**	**LocationID**
FireDepartment	**FireDepartmentID**	**ContactNumber**	**GPS Coordinate**	**LocationID**	**Active**	
PoliceStation	**PoliceStationID**	**ContactNumber**	**GPS Coordinate**	**LocationID**	**Active**	

<u>Section 5: Class/Object Specifications</u>


Client
- IP Address: String
- Port Number : Int
+connectToServer (IP Address, Port Number): void
+displayMenu (): void
+ addEntry (Template for Shared Object Types): void
+ viewEntries (Template for All Shared Object Types)
+ viewLocations(Template for Location Object Types)
+ viewInfo (Template for Info Object Types)
+ viewRelationships (Template for Relationship Object Types)
+ formatOutputGUI () : void
+ closeConnection() : void
Remote Interface and Methods
+ insertSharedObject (userInput) : void
+ getSharedObject (userInput) : Template< All Shared Objects>
+ getSharedObjectList (userInput Object Type) : ArrayList<Object Type>
+ getInfoObjects () : ArrayList<Info Object Type>
+ getRelationshipObjects (): ArrayList<Relationship Object Type>
+ getLocationObjects () : ArrayLIst < Location Object Type>
Shared Objects- with Sub Types (See DB Schema for specific attributes)
Relationship Types:
+ Friends
Information Types:
+ Rating
+ News
+ Event
Location Types:
+ VisitedLocation
+ LocationToVisit
+ Airport
+ Bank
+ Bar
+ Hotel
+ Theater
+ Restaurant
+ School
+ Store
+ TouristTrap
+ Emergency
+ Hospital
+ Fire Department
+ Police Station
Server
- IPAddress: String
- port : int
- MySQL_IP: String
- MySQLPort: int
- MongoIP : String
- MongoPort : int
+ setUpServer () : void
+ connectMySQL(MySQL_IP, MYSQLPort) : void
+ connectMongoDB (MongoIP, MongoPort) : void
+ validateClient (): void
+ syncDatabases ( ) : void
+ sendQuery () : void
+ convertInputToQuery() : (Each remote get and insert method will have its own method)
+ convertResultsToObjects (SQLQueryResults) : void (Each object type will have its own method)
+ logQuery()
+ shutdown(): void

Section 6: Sequence Diagram



Section 7: Use Cases

Use Case 1: Login

1.1 Preconditions:
1.1.1) The user is not yet logged in.
1.1.2) The user is on the main login page.

1.2 Use Case:
As a user, I want to use the Login page, so that I can enter data.

1.3 Flow:
1. Enter Username
2. Enter Password
3. Confirm Login
4. On no-username, display error.
5. On wrong-username, display error.
6. On no-password, display error.
7. On wrong-password, display error.
8. Login User (create cookie)

Use Case 2: Data View

1.1 Preconditions:
1.1.1) The user is on the main View page.
1.1.2) The user is logged in.

1.2 Use Case:
As a user, I want to use the Data View page, so that I can see the visualization of the full system's data.

1.3 Flow:
1. The page displays a visualization of system data, showing any Locations that the user is planning on visiting.
2. The page displays all registered locations, within view.

3. The page displays all locations the user wants to visit.
4. The page will show, per location, how many times their friends have visited them.

Use Case 3: Data Entry (Location Visited)

1.1 Preconditions:
1.1.1) The user is on the Data Entry Page for Locations Visited
1.1.2) The user is logged in.

1.2 Use case:
As a user, I want to use the Data Entry page so that I can show that I visited a location. If I already visited it before, it should raise my visitation count.

1.3 Flow:
1. The user enters the available data. Location Type will be a Dropdown, and Location Sub Type is based on Location Type.
2. Upon submission, the entry is added to the global database.

Use Case 4: Data Entry (Event)

1.1 Preconditions:
1.1.1) The user is on the Data Entry Page for Events
1.1.2) The user is logged in.

1.2 Use case:
As a user, I want to use the Data Entry page so that I can add an event to a location.

1.3 Flow:
1. The user enters the available data. Locations will be referenced by ID for the sake of keeping this exercise simple.
2. Upon submission, the entry is added to the global database.

Use Case 5: Data Entry (Location)

1.1 Preconditions:

1.1.1) The user is on the Data Entry Page for Locations
1.1.2) The user is logged in.

1.2 Use case:
As a user, I want to use the Data Entry page so that I can show that I visited a location. If I already visited it before, it should raise my visitation count.

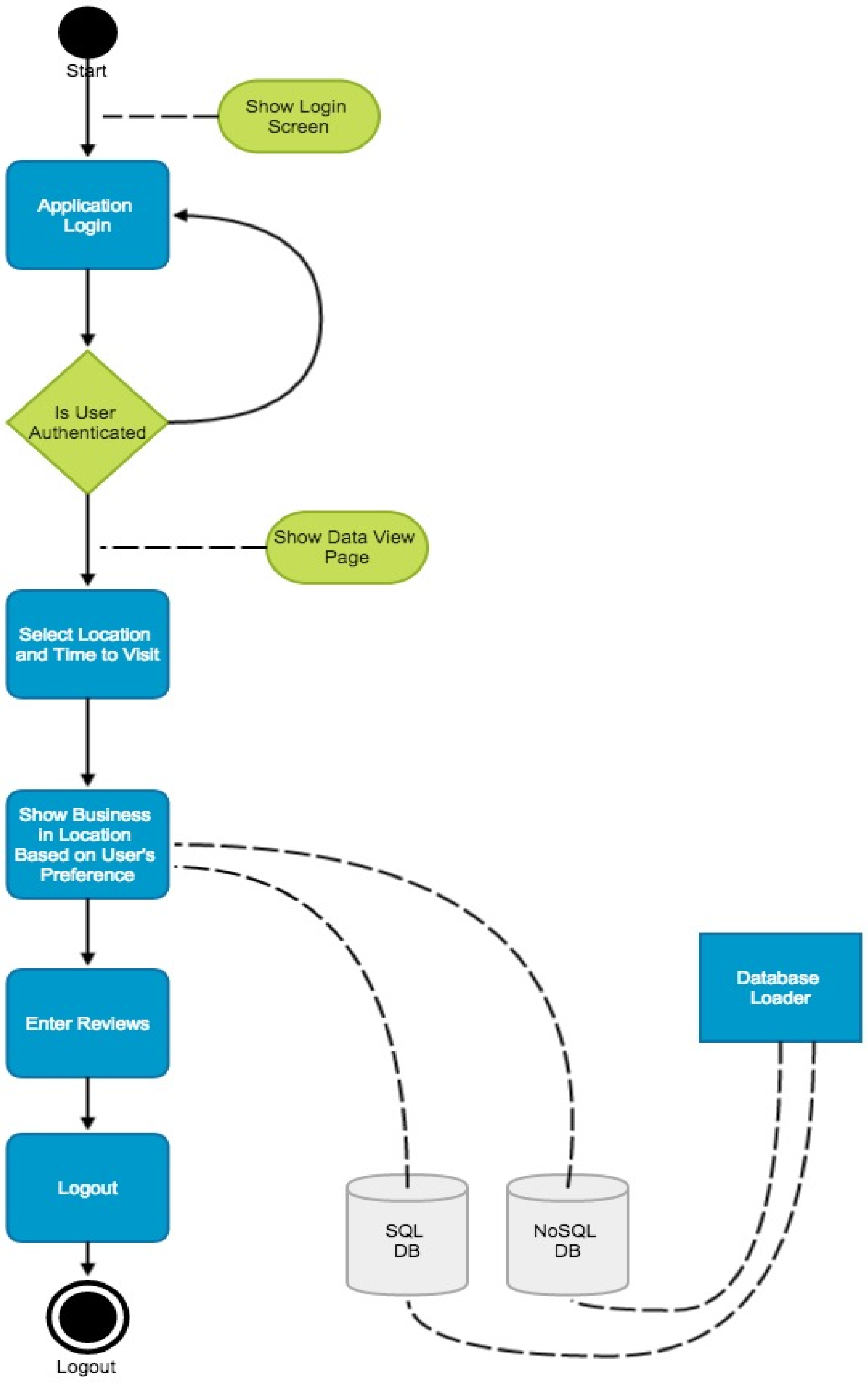
1.3 Flow:
1. The user enters the available data. Location Type will be a Dropdown, and Location Sub Type is based on Location Type.
2. Upon submission, the entry is added to the global database.

Use Case 6: Data Entry (Desired Visitation)
1.1 Preconditions:
1.1.1) The user is on the Data Entry Page for Desired Visitation
1.1.2) The user is logged in.

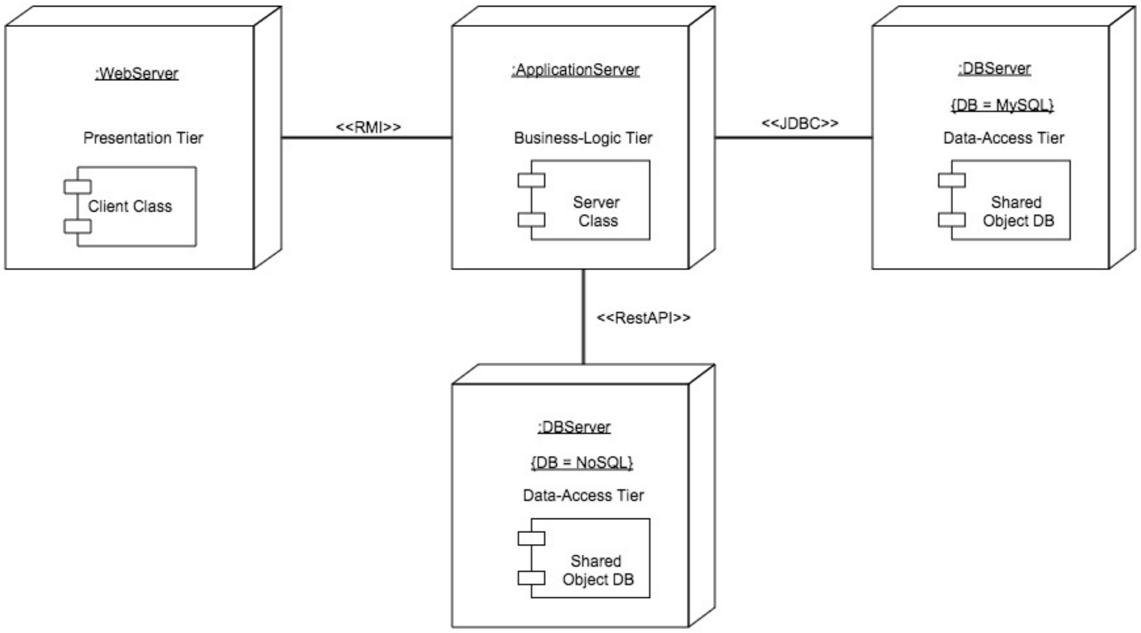
1.2 Use case:
As a user, I want to use the Data Entry page so that I can show that I intend to visit a location within a certain timespan.

1.3 Flow:
1. The user enters the available data. Locations will be referenced by ID for the sake of keeping this exercise simple.
2. Dates will be selected using a data-picker.
3. Upon submission, the entry is added to the global database.

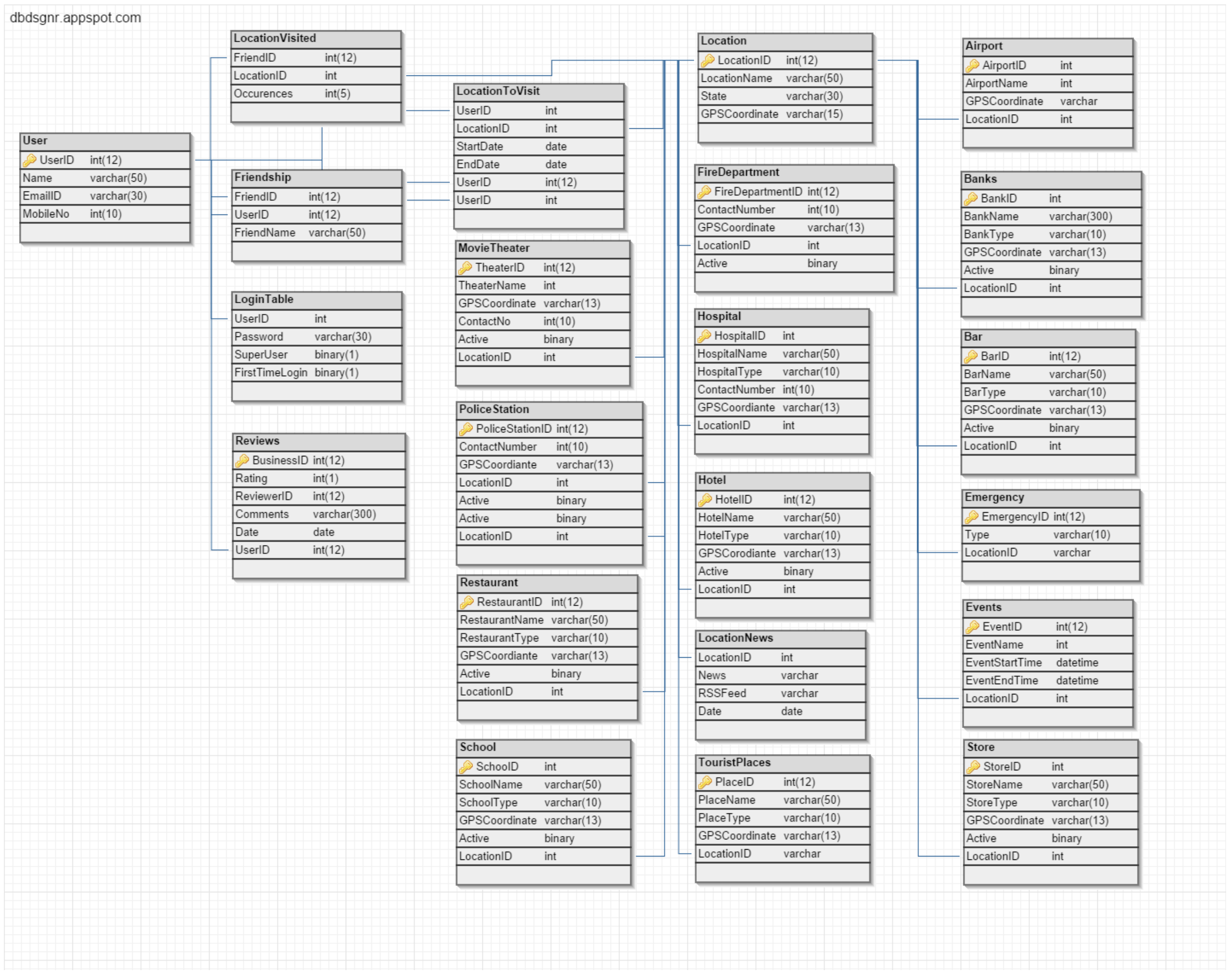
Section 8. Activity Diagram



Section 9. Deployment Diagram



Section 10 UML Diagram:


dbdsgnr.appspot.com
LocationVisited
FriendID int(12)
LocationID int
Occurences int(5)
User
UserID int(12)
Name varchar(50)
EmailID varchar(30)
MobileNo int(10)
Friendship
FriendID int(12)
UserID int(12)
FriendName varchar(50)
LoginTable
UserID int
Password varchar(30)
SuperUser binary(1)
FirstTimeLogin binary(1)
Reviews
BusinessID int(12)
Rating int(1)
ReviewerID int(12)
Comments varchar(300)
Date date
UserID int(12)
LocationToVisit
UserID int
LocationID int
StartDate date
EndDate date
UserID int(12)
UserID int
MovieTheater
TheaterID int(12)
TheaterName int
GPSCoordinate varchar(13)
ContactNo int(10)
Active binary
LocationID int
PoliceStation
PoliceStationID int(12)
ContactNumber int(10)
GPSCoordiante varchar(13)
LocationID int
Active binary
Active binary
LocationID int
Restaurant
RestaurantID int(12)
RestaurantName varchar(50)
RestaurantType varchar(10)
GPSCoordinate varchar(13)
Active binary
LocationID int
School
SchoolID int
SchoolName varchar(50)
SchoolType varchar(10)
GPSCoordinate varchar(13)
Active binary
LocationID int
Location
LocationID int(12)
LocationName varchar(50)
State varchar(30)
GPSCoordinate varchar(15)
FireDepartment
FireDepartmentID int(12)
ContactNumber int(10)
GPSCoordinate varchar(13)
LocationID int
Active binary
Hospital
HospitalID int
HospitalName varchar(50)
HospitalType varchar(10)
ContactNumber int(10)
GPSCoordiante varchar(13)
LocationID int
Hotel
HotelID int(12)
HotelName varchar(50)
HotelType varchar(10)
GPSCorodiante varchar(13)
Active binary
LocationID int
LocationNews
LocationID int
News varchar
RSSFeed varchar
Date date
TouristPlaces
PlaceID int(12)
PlaceName varchar(50)
PlaceType varchar(10)
GPSCoordinate varchar(13)
LocationID varchar
Airport
AirportID int
AirportName int
GPSCoordinate varchar
LocationID int
Banks
BankID int
BankName varchar(300)
BankType varchar(10)
GPSCoordinate varchar(13)
Active binary
LocationID int
Bar
BarID int(12)
BarName varchar(50)
BarType varchar(10)
GPSCoordinate varchar(13)
Active binary
LocationID int
Emergency
EmergencyID int(12)
Type varchar(10)
LocationID varchar
Events
EventID int(12)
EventName int
EventStartTime datetime
EventEndTime datetime
LocationID int
Store
StoreID int
StoreName varchar(50)
StoreType varchar(10)
GPSCoordinate varchar(13)
Active binary
LocationID int