**TECHNICAL DOCUMENT DESIGN SPECIFICATION**

**for**

COMPUTERIZED RIDESHARE PROGRAM

**GROUP B**

Steven Rago

Rajeev Reddy Vishaka

Mohanakrishnakumar Karunakaran

Saulius Milevicius

Subramanian Arumugam

**ABSTRACT**

This document is used to specify the design specification for the Computerized Rideshare Program. The overall architecture of the system to be developed is included in this document supported with the Unified Modeling Diagrams. A description of the diagrams has been added and this document will be used until the demonstration of the final product.

The Computerized Rideshare System will be used by the employees’ of the Southern North Dakota University(SNDU). The main aim of the system is to save employees’ money and also alleviate overcrowding problems in the university parking lots. The system is to be developed, by taking into consideration in its design, several factors such as the scheduling algorithm, information pertaining to each employee, the interface between each module and the database design. The design will also present a set of sample user interface screens which would provide an overview of the system and its functions. These screens are not the final user interface for the system. All the interaction diagrams mentioned in this design document are used to represent the flow of messages between the various tiers. The names of the classes and messages in the interaction diagram might differ from the class diagram, but the overall flow and functionality depicted would be the same.

**TABLE OF CONTENTS**

Page No.

1. Introduction 4

1.1 Purpose of the document 4

1.2 Document Scope 4

1.3 Definitions, Acronyms and Abbreviations 4

1.4 Specific Design Considerations 4

1.5 Operating Environment 4

2. Proposed Software Architecture 5

2.1 Software Development Life Cycle 5

2.2 Architecture Overview 6

2.3 Architecture Component Description 7

2.3.1 Introduction 7

2.3.2 Client Tier 7

2.3.3 Back End Server 7

2.3.4 Presentation Tier - User Interface 7

2.3.5 Controller 7

2.3.6 Persistence Tier 7

3. Unified Modeling Diagrams 8

3.1 Sequence Interaction Diagram 8

3.2 Class Diagram 16

3.3 Object Diagram 19

4. Pseudo code 20

5. Graphical User Interface 21

6. Database Design 24

5.1 Table Schemas 24

5.2 Table SQL Queries 26

5.3 Transaction Implementation 27

7. References 27

**1. Introduction**

**1.1 Purpose of the document**

The purpose of this document is to detail the architecture, interface, and data designs of the Computerized Rideshare Program.

**1.2 Document Scope**

This design document deals with the design of the Computerized Rideshare Program and anyone interested in understanding its implementation design should read this document. This document does not deal with the functionality of the software and servers which are to be used to develop the application.

**1.3 Definitions, Acronyms and Abbreviations**

DHTML - Dynamic Hypertext Markup Language - representing a collection of

technologies like HTML, CSS and JavaScript.

MVC - Model View Controller design pattern

**1.4 Specific Design Considerations**

Wrapper pattern

MVC Pattern

Iterator pattern

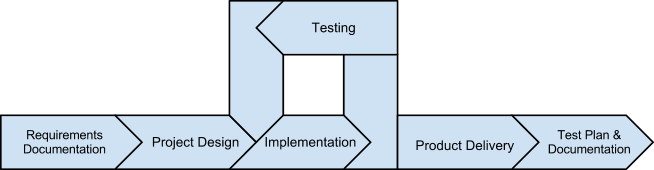
Composition

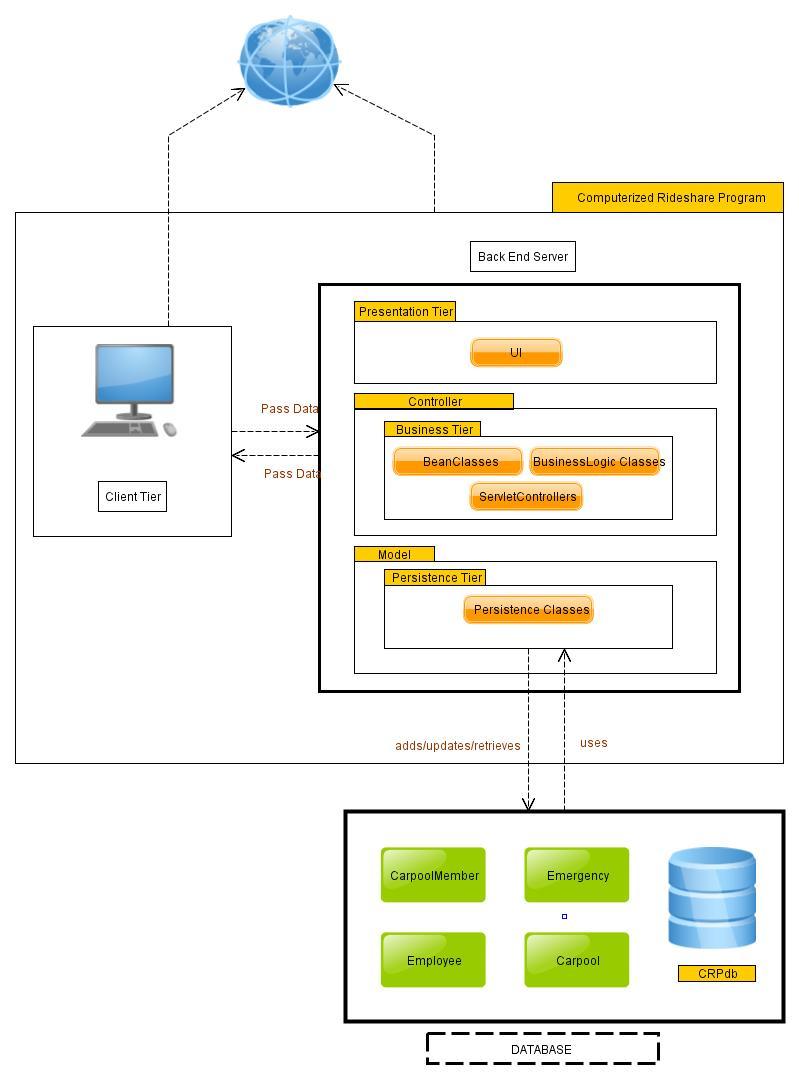
**1.5 Operating Environment**

The system will be deployed into Apache Tomcat Web Server 7.X and will use an instance of the MySQL server in Windows 8. The server environment should be running with at-least 2GB of RAM and have at-least 1GB of hard disk space. The supported browsers for the clients to run this system are Google Chrome, Mozilla Firefox, Safari and Internet Explorer and the javascript enabled in them.

**2. Proposed Software Architecture**

**2.1 Software Development Life Cycle**



**2.2 Architecture Overview**

**2.3 Architecture Component Description**

**2.3.1 Introduction**

The CRP system will be a web application where information will be passed and received as http requests and responses.

**2.3.2 Client Tier**

The client tier consists of all the web pages using DHTML which will be loaded by the client’s browser. There will not be much changes in this tier, unless there is a requirement to change the user interface.

**2.3.3 Back-End Server**

The back-end tier consists of all the files that reside on the web application server. It consists of three parts where each tier performs a specific task and there is low coupling between each tiers.

**2.3.4 Presentation Tier - User Interface**

The presentation tier consists of the dynamic web pages using JSP, Servlets, Struts framework which keeps all the user interface related files in a separate package. All the screens related to the CRP will be present in this tier as a package.

**2.3.5 Controller**

The controller forms the business tierthe business logic for the CRP will be present. Any change in the presentation tier will not require a change in the business tier. The controller contains the bean classes and the business logic classes. The bean classes contain as instance variables, all the attributes present in the database tables and will the source of data for the business logic classes. Bean classes are public classes which can considered as First-Class objects which can be passed as arguments and returned as value. The business logic for each changing behavior will be encapsulated in its own class which can be instantiated when required and its methods can be used.

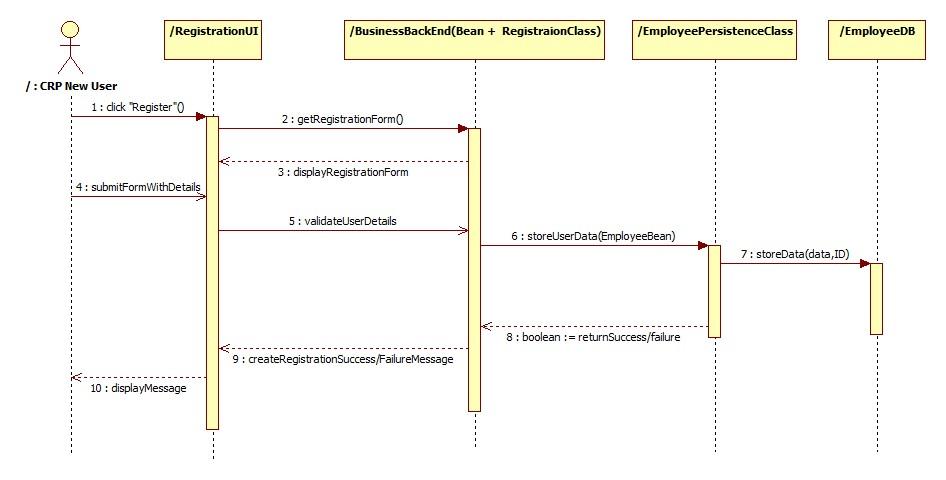
**2.3.6 Persistence Tier**

The persistence tier contains the classes which are responsible for changing the state of the CRP system. The classes present in this tier are responsible for maintaining the consistent state of the CRP application. Any operation involving the database, whether retrieval or storage or updating of data, will be performed using the objects of the persistence classes. There will be a persistence class responsible for each table present in the database schema (CRPdb).

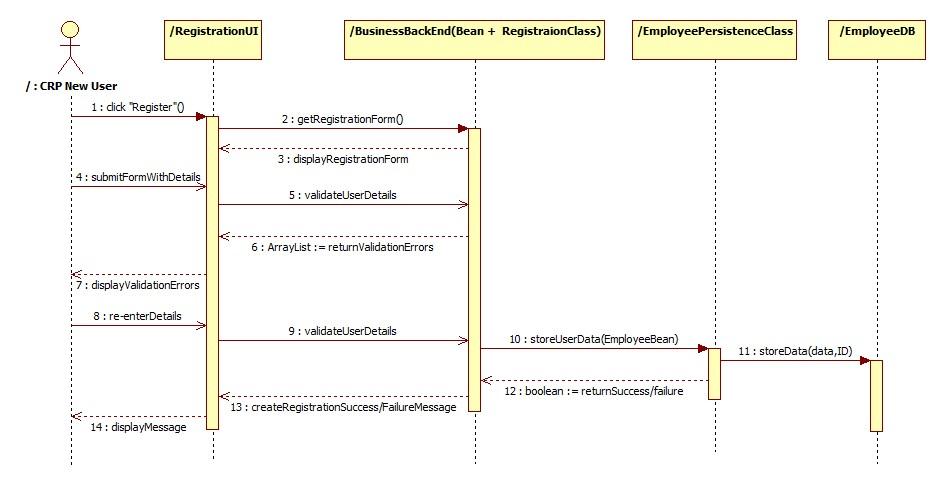
**3. Unified Modeling Diagrams**

**3.1 Sequence Interaction Diagram**

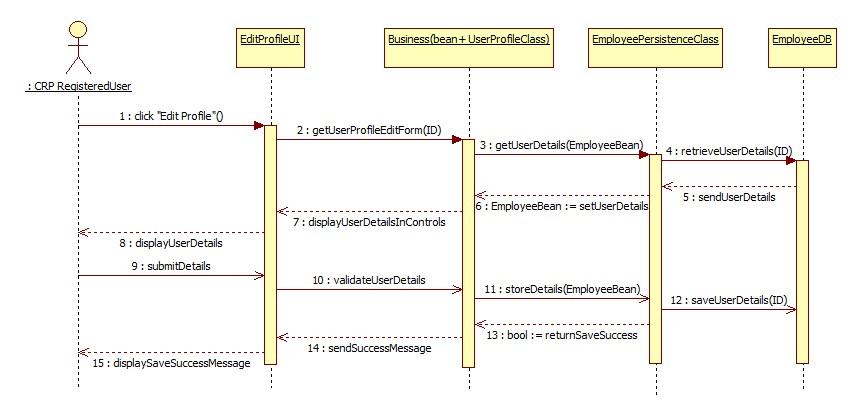
**3.1.1 Successful User Registration**



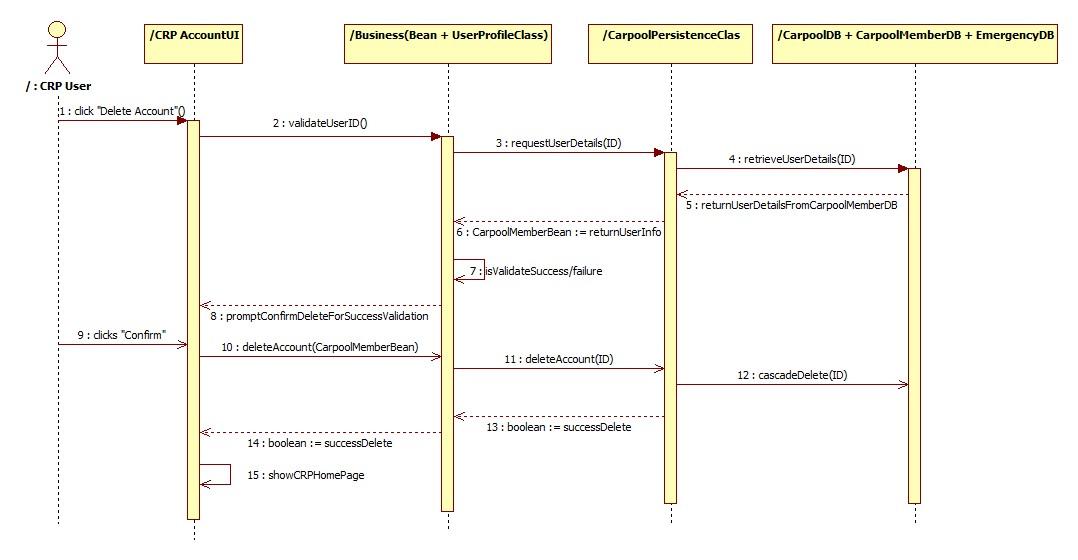
**3.1.2 User Registration Exception**

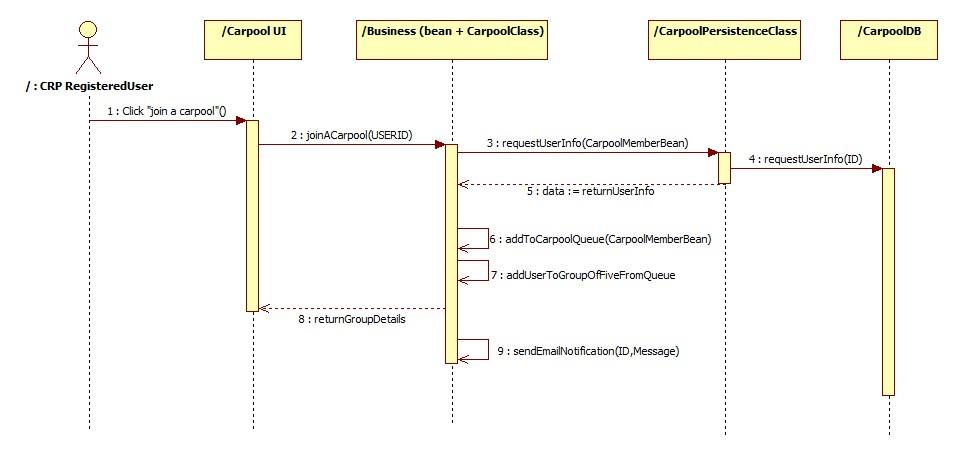


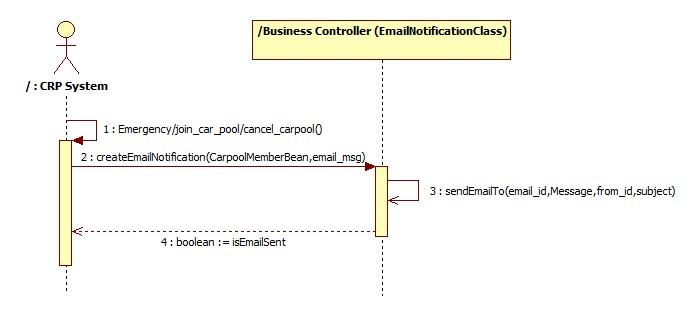
**3.1.3 Edit User Profile**



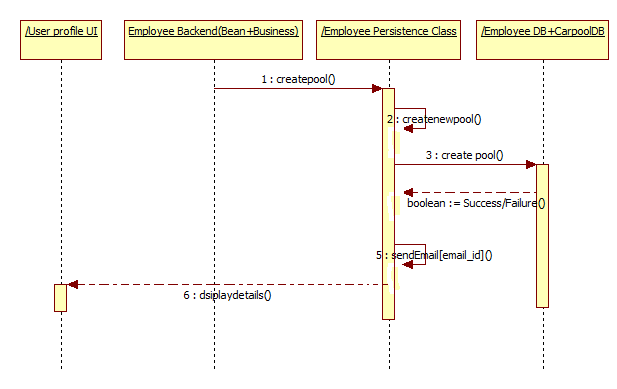
**3.1.4 Remove Member from Carpool**



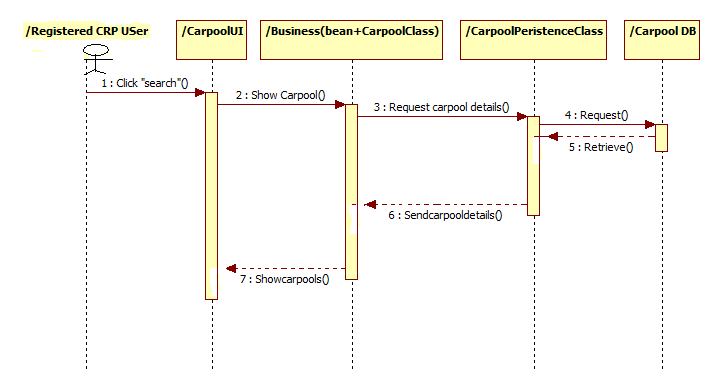
**3.1.5 Apply to join a carpool**

**3.1.6 Email Notifications (Cancel Carpool/Join Carpool/ Emergency)**

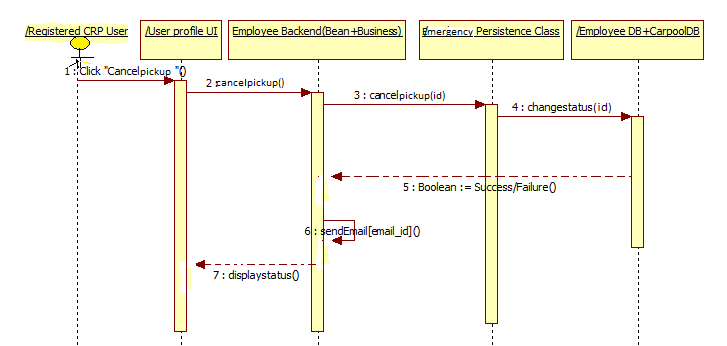
**3.1.7 Allocating Carpool**



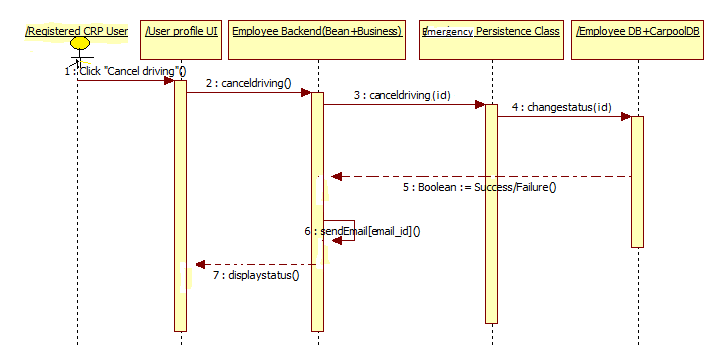
**3.1.8 Search for a Carpool**

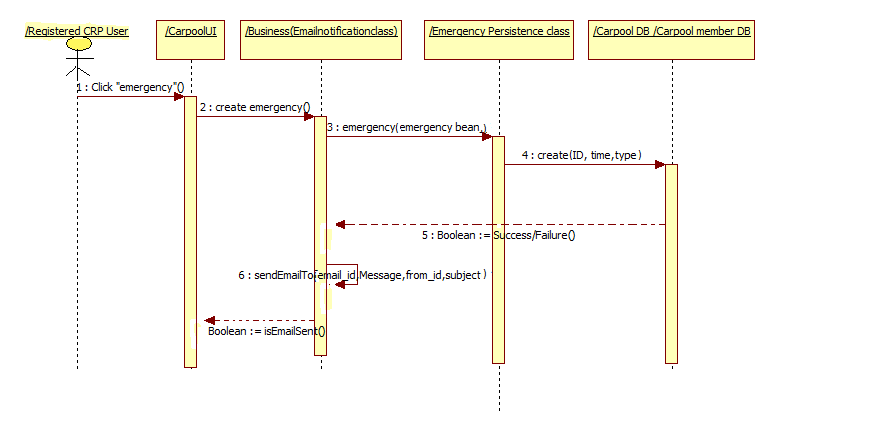


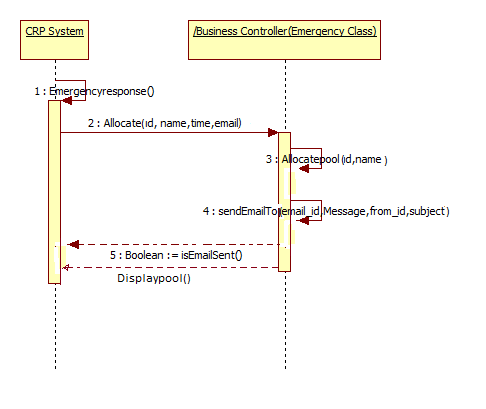
**3.1.9 Cancel pickup**



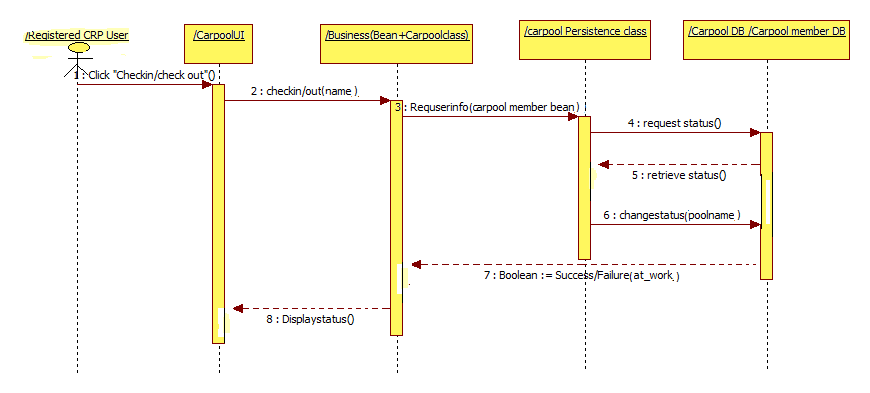
**3.1.10 Cancel driving**

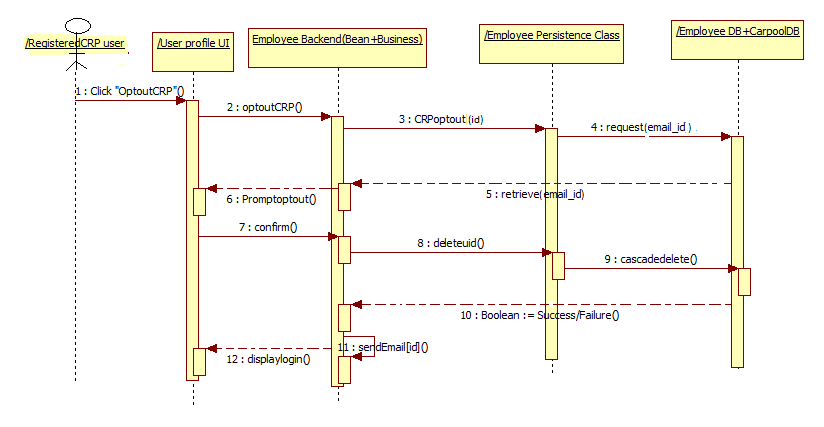


**3.1.11. Issuing emergency****3.1.12.Responding to Emergency**

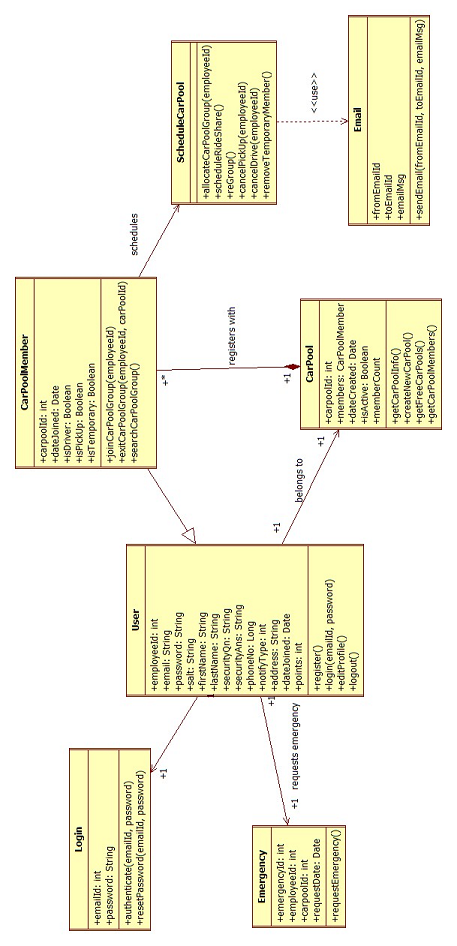


**3.1.13 Carpool Checkin/Checkout**



**3.1.14 Opt out of CRP**

**3.2 Class Diagram**



**Class Diagram Description**

**Email**

|  |  |
| --- | --- |
| boolean sendEmail(member\_id, message) | Used by Schedule Class to send Email to the car pool members. Returns true if the email is sent successfully. |

**Schedule**

|  |  |
| --- | --- |
| boolean allocateCarPoolGroup(employeeId) | This method allocates a user to a CarPoolGroup initially when he registers with the system |
| void scheduleRideShare() | Schedules the rideshare program daily. Invokes sendEmail method. Decides on the driver and the passengers of a group on a day. |
| void reGroup() | Regroups the car pool group when someone opts out of a group. |
| boolean cancelPickUp(employeeId) | cancels the pickup for the particular day and notifies the members of the group. Returns true on successful completion |
| boolean cancelDrive(employeeId) | When the driver invokes cancel drive, it cancels the drive option for that particular employee and assigns driver and notifies other members. Returns true on successful completion |
| boolean removeTemporaryMembers() | Ran at beginning (or end) of each day - Remove temporary members from the group (if any assigned) for that particular day. Returns true on successful completion |

**Login**

|  |  |
| --- | --- |
| User authenticate(email, password) | authenticates with the data in the User table and returns the corresponding user object |
| boolean logout() | Logs out the user from the session |

**Emergency**

|  |  |
| --- | --- |
| CarPool requestEmergency(employeeId) | Invoked when the user requests emergency pick up request. Invokes getFreeCarPools() of CarPool class and displays the groups where vacancies are available to the user. |

**Carpool**

|  |  |
| --- | --- |
| int carpoolId;  CarPoolMember[] members;  long dateCreated;  boolean isActive;  int memberCount; |  |
| int createNewCarpool() | Creates a new Car Pool group after a car pool group is filled. |
| int[] getFreeCarpools() | Fetches carpool groups for displaying all carpools with less than 4 members |
| CarPoolMember getCarPoolInfo(carpoolId) | Fetches car pool information details such as date created and number of members details |
| User[] getCarpoolMembers(carpoolId) | Fetches car pool information details such as date created and number of members details |

**CarpoolMember**

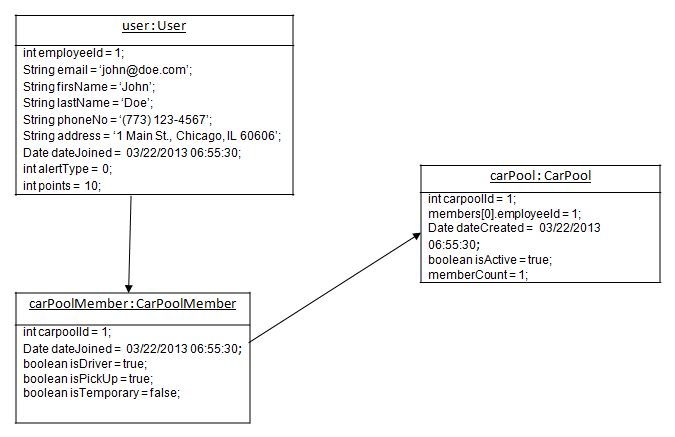
|  |  |
| --- | --- |
| int carpoolId;  int employeeId;  long dateJoined;  boolean isDriver;  boolean isPickUp;  boolean isTemporaryMember; |  |
| boolean joinCarPoolGroup(employeeId) | This method is invoked when a member wants to join a carpool group and is not part of any group. i.e after opting out from a group |
| boolean exitCarPoolGroup(employeeId, carPoolId) | Invoked when a member wants to exit from a carpool group. |

**User**

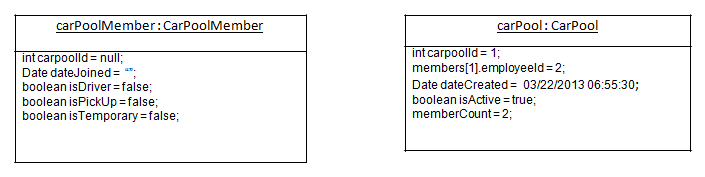
|  |  |
| --- | --- |
| int employeeId;  String emailId;  String firstName;  String lastName;  String phoneNo;  String address;  long dateJoined;  int alertType;  int points; |  |
| login(emailId,password) | invokes authenticate of Login class and obtains the corresponding user object on successful authentication. |
| void register(first\_name, last\_name, phone, email, password, sec\_question, sec\_answer, address) | Registers his id with the system. |
| boolean editProfile(User,employeeId) | This method edits the user information and returns true on successful completion. |

**3.3 Object Diagram**

(Assuming first user to join the system) User Registers with the system. The system creates a new group with id 1 and allocates the user to the car pool group 1.



Consider the car pool group 1 with 3 members and John Doe opts out of the group. The state of the CarPoolMember object and carpool objects will be,



**4. Pseudo Code:**

1. User registers with the system if he is not registered yet.

2. He is automatically placed in a group by the system. The system creates a new group once the previous group created is allocated with 4 members. So a new group is created only after a group is filled.

3. User is presented with the details of the group he belongs to and the members in that group.

4. The system allocates a member as a driver in a group for a particular day and notifies via email.

5. A member cancels his drive, his points will be reduced and the member to be driven next day will be made the driver and notified to all members via email. The system will allocate the cancelled drive member to drive the following day. So each day a member will drive and fair share is maintained.

6. Points of a member and the date joined fields are used to schedule drives for a member.

6. A member cancels his pickup, all the members will be notified and points will be deducted.

7. When a member creates an emergency pick up, the system will look for carpool groups with less than 4 members and allocates to the group and emails everyone in the group and the member. He is a “temporary” member for that group.

8. When a member opts out of group A and if there is a group E with one member(that is formed latest) the member in group E will be moved to group A and notified. Points will be reduced.

9. When a member is not part of any group, he will be given the choice to search for groups and join any group where there is vacancy.

**Points Deduction Scenario**

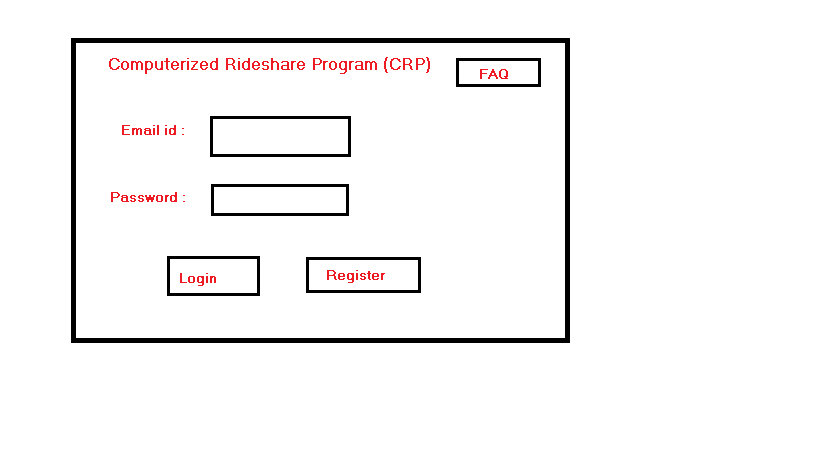
1. Opt out of carpool - 5 points

2. Cancel Pickup - 2 points

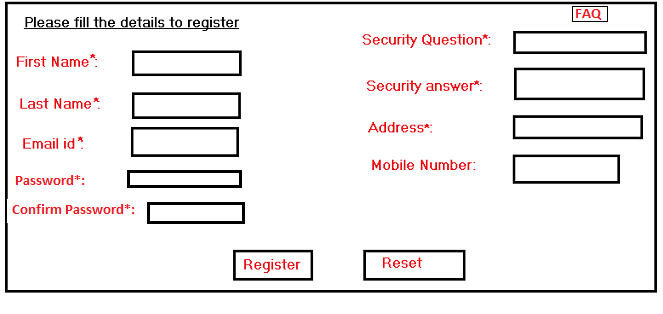
3. Cancel Driving - 3 points

4. emergency usage( leave in between-temporary car member) - 4 points

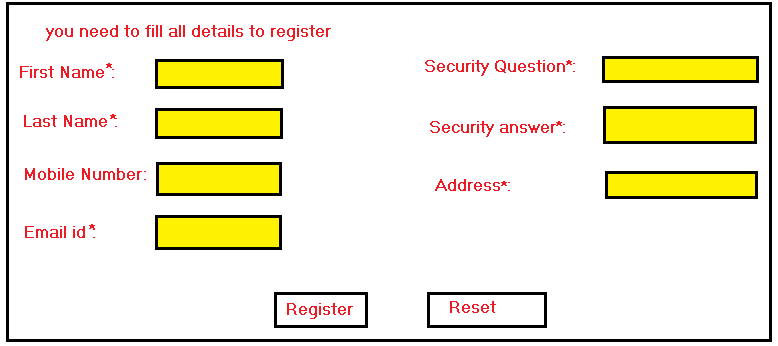
**5. Graphical User Interface**

**5.1 Login Screen.**

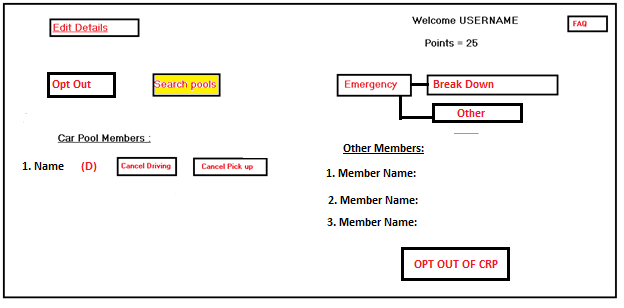
**5.2.Register Screen :**



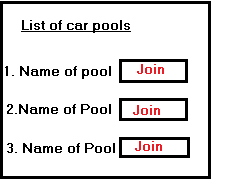
**5.3. Registration screen, with no input or wrong input**



**5.4. After login, screen:**



**5.5. Clicking Search Pools:**



**6. Database Design**

**6.1 Table Schemas**

**Notation**

|  |  |
| --- | --- |
| **P** | Primary Key |
| **F** | Foreign Key |
| U | Unique/Candidate Key |

**Employee**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Type** | **Example of Values** |
| employee\_id  **(P)** | Employee’s ID | INT | 1 to 4294967295 |
| email | Employee’s email/username | TEXT | ‘john@gmail.com’ |
| password | Employee’s password, md5 encrypted | VARCHAR(32) | ‘2bd0a5b9d773f57d7654791e952933c9’ |
| salt | Employee’s password salt | VARCHAR(12) | ‘0345\_#%)#@Kd’ |
| first\_name | Employee’s first name | TEXT | ‘John’ |
| last\_name | Employee’s last name | TEXT | ‘Doe’ |
| sec\_question | Employee’s selected security question | TEXT | ‘Mother’s maiden name?’ |
| sec\_answer | Employee’s security question answer | TEXT | ‘Smith’ |
| phone | Employee’s phone number | VARCHAR(14) | ‘(773) 123-1234’ |
| notify\_type | The type of alert to use to contact the employee | INT(1) | 0 for email  1 for text |
| address | Employee’s address | TEXT | ‘1 Main St., Chicago, IL 60606’ |
| date\_joined | Employee’s system join date | TIMESTAMP | ‘2013-03-17 16:24:58’ |
| points | Employee’s current points | INT | 1 to 4294967295 |

**Carpool**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Type** | **Example of Values** |
| carpool\_id  **(P)** | Carpool’s ID | INT | 1 to 4294967295 |
| date\_created | Date when the carpool was created | TIMESTAMP | ‘2013-03-17 16:27:00’ |
| at\_work | Is the carpool currently at work | INT(1) | 0 for not at work  1 for at work |

**CarpoolMember**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Type** | **Example of Values** |
| employee\_id  (U**F**) | Employee’s ID | INT | 1 to 4294967295 |
| carpool\_id  (U**F**) | Employee’s carpool ID | INT | 1 to 4294967295 |
| date\_joined | Employee’s carpool join date | TIMESTAMP | ‘2013-03-17 16:27:00’ |
| is\_driver | Defines whether the employee is currently a driver or passenger | INT(1) | 0 for passenger  1 for driver |
| pick\_up | Defines whether the employee is to be picked up | INT(1) | 0 for no  1 for yes |
| is\_temporary | Defines whether the member is a temporary addition to the group | INT(1) | 0 for fulltime member  1 for temporary |

**6.2 Table SQL Queries**

**Employee**

New employee

|  |
| --- |
| INSERT INTO Employee (email,password,salt,first\_name,last\_name,sec\_question,sec\_answer,phone,address) VALUES (‘x’,‘x’,’x’,’x’,’x’,’x’,’x’,’x’,’x’) |

Get employee

|  |
| --- |
| SELECT \* FROM Employee WHERE employee\_id = 0 |

Update information

|  |
| --- |
| UPDATE Employee SET phone = ‘y’, address = ‘y’, password = ‘y’ WHERE employee\_id = 0 |

Cancel pickup

|  |
| --- |
| UPDATE CarpoolMember SET pick\_up = 0 WHERE employee\_id = 0 |

Update points

|  |
| --- |
| UPDATE Employee SET points = ((SELECT points FROM Employee WHERE employee\_id=0)+1) WHERE employee\_id = 0 |

**Carpool**

New carpool

|  |
| --- |
| INSERT INTO Carpool (name,owner\_id) VALUES (‘x’,0) |

Change owner

|  |
| --- |
| UPDATE Carpool SET owner\_id = 1 WHERE carpool\_id = 0 |

Delete carpool

|  |
| --- |
| DELETE FROM Carpool WHERE carpool\_id = 0 |

Update name

|  |
| --- |
| UPDATE Carpool SET name = ‘y’ WHERE carpool\_id = 0 |

Checkin

|  |
| --- |
| UPDATE Carpool SET at\_work = 1 WHERE carpool\_id = 0 |

Get non-departed carpools with less than 4 members

|  |
| --- |
| SELECT \* FROM Carpool AS c INNER JOIN CarpoolMember AS cm ON c.carpool\_id=cm.carpool\_id WHERE c.at\_work = 1 GROUP BY cm.carpool\_id HAVING COUNT(cm.employee\_id) < 4 |

**CarpoolMember**

Add new member to carpool

|  |
| --- |
| INSERT INTO CarpoolMember (employee\_id,carpool\_id,is\_driver,is\_temporary) VALUES (0,0,1,0) |

Get carpool members

|  |
| --- |
| SELECT \* FROM Employee AS e INNER JOIN CarpoolMember AS cm ON e.employee\_id=cm.employee\_id WHERE cm.carpool\_id = 0 |

Change driver

|  |
| --- |
| UPDATE CarpoolMember SET is\_driver = 0 WHERE employee\_id = 0  UPDATE CarpoolMember SET is\_driver = 1 WHERE employee\_id = 1 |

Remove member from carpool

|  |
| --- |
| DELETE FROM CarpoolMember WHERE employee\_id = 0 |

Remove temporary members

|  |
| --- |
| DELETE FROM CarpoolMember WHERE is\_temporary = 1 |

**6.3 Transaction Implementation**

We will be using MySQL as our database server which handles transaction for us and preserves the ACID properties of the database.

**7. References**

1. [www.csis.pace.edu/~scharff/cs3892005/designdoc.doc](http://www.csis.pace.edu/~scharff/cs3892005/designdoc.doc)

2. <http://www.jguru.com/faq/view.jsp?EID=262429>