DOCUMENT DESIGN

**3A) Purpose**

This software design specification is made whit the purpose of outlining the software architecture and design of myTaxi application system in detail. The documents will provide developers an insight in meeting client’s needs efficiently and effectively. Moreover the document facilitates communication and understanding of the system by providing several views of the system design.

**3B) Scope**

The software design document would demonstrate how the design will accomplish the functional and non-functional requirements captured in the Software Requirement specification (RASD). The document will provide a framework to the programmers trough describing the high level components and architecture, sub system, interfaces and algorithm design. This is achieved trough the use of architectural patterns, sequence diagrams and user interfaces.

**4A) Document overview**

The next chapter of the document has described architectural design of myTaxi system. The high level components and their interactions, suitable architectural patterns, physical arrangement of components and design decisions applied to the whole system.

**4B) High level components and their interaction**

**Components:**

* AccessManager:

This sub system consists of all the components responsible of registration and login process. It authenticates users and drivers allowing them to manage their relative activities. It’s also responsible for guest’s registration.

* + Registration
  + Authentication
* User:

This sub system consists of all the components that implements the functions related to the user operations such as asking a taxi for a simple ride, make a reservation for taxi and view the number of available taxies.

* + SimpleRequest
  + DetailedRequest
  + NumberOfTaxi
  + Notification
* Driver:

This sub system consist of all the components that implements the functions related to the driver operations such as inform about own availability and accept an user’s request.

* + BeAvailable
  + Confirmation
* DataManager:

This is the major sub system that is responsible for the security of sensible data. It communicates directly with the database improving several operation on it such as adding new users, adding new drivers, check saved data with data given by the user during the login process.

* + AddData
  + QueryData
* Database:

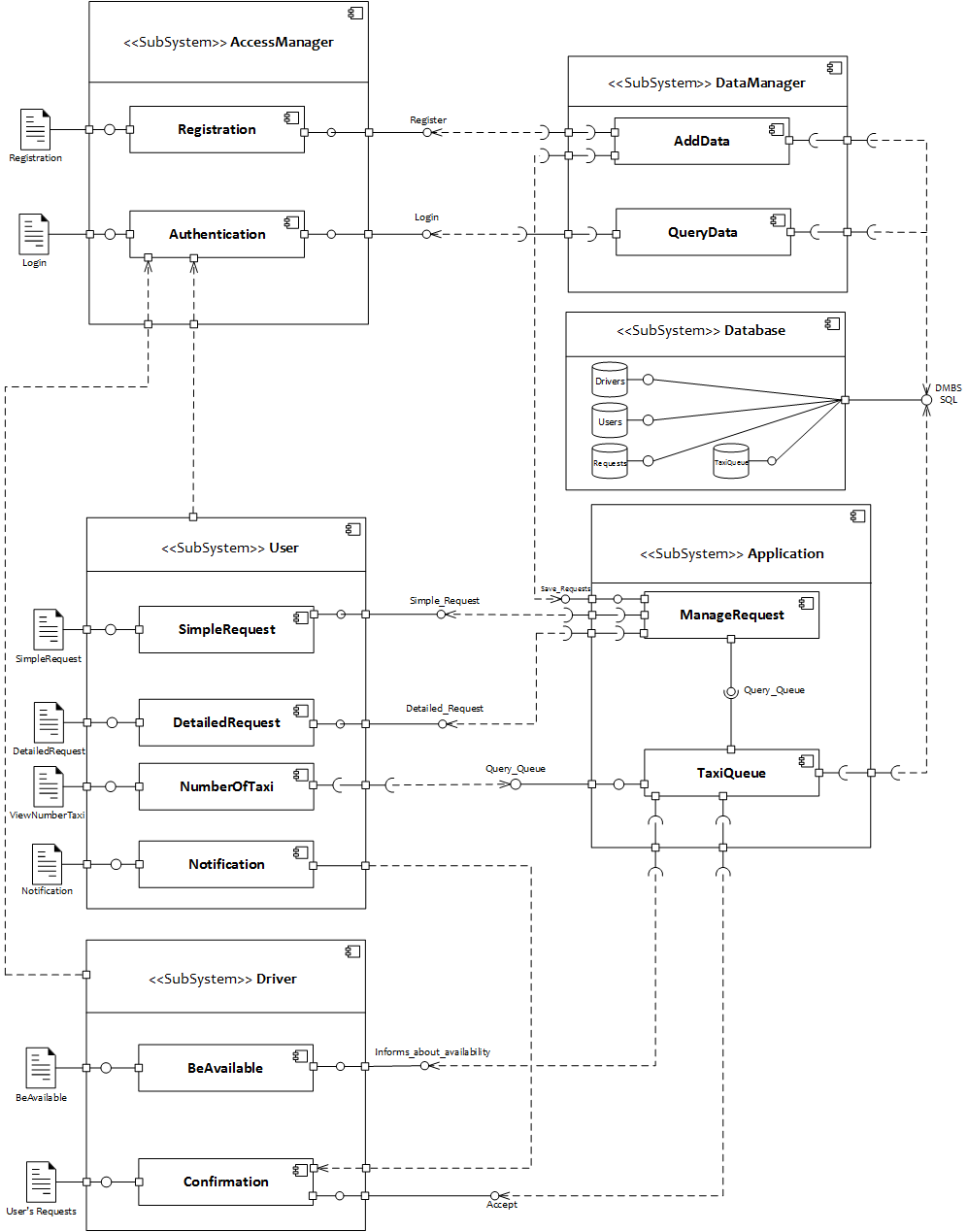
Essentially, it is formed by the databases and give all the functionalities by whose operate directly on the database.

* Drivers
* Users
* Requests
* TaxiQueue
* Application:

This is the major sub system that is responsible of the logic of myTaxi app. In particular it manages all the request done by the users forwarding them to the drivers according to the policy of forwarding.

* ManageRequest
* TaxiQueue

**4C) Component diagram**



**4F) Component interface**

**AccessManager**

**Registration:Register** This interface allows a guest to register himself into the system giving the mandatory information

**Authentication:Login** This is the interface that allows the users/drivers to login to the system. This will guide the user/driver to his relative home page.

**DataManager**

**AddData:** This component uses interfaces provided by other components in order to grant a major grade of security to the data tier interposing between the presentation tier and data one.

**QueryData:** This component, like AddData, communicates with the data tier querying it in order to check user’s information given by the user interface.

**Database**

The relevant interface given by this tier are the most common known methods by whose it can be possible to operate, manipulate and querying a database.

**User**

**SimpleRequest:Simple\_Request** This is the interface that allow the users to ask for a taxi for a simple taxi

**DetailedRequest:Detailed\_Request** This is the interface that allow the users to reserve a taxi.

**NumberOfTaxi** This component uses a required interface provided by the TaxiQueue in order to query it and know the number of available taxi in relation to the current position given by the gps

**Notification** This is a component that only have a user’s interface that depends on the confirmation of a own user request.

**Driver**

**BeAvailable:Informs\_About\_Availability** This is the interface that allow a driver to inform the system about his current availability. it can be changed by both system and driver according of what written in the RASD.

**Confirmation:Accept** This is the interface by which a driver can accept an incoming user’s request forwarded by the system.

**Application**

**ManageRequest:Save\_Request** This interface is given to the manager data component in order to retrieve information about requests saving them

**ManageRequest:Taxi\_Queue** This interface is the major central point of the application because it communicates with the other component whose tasks are to handle requests incoming from the users forwarding them correctly to the drivers.

**TaxiQueue:Query\_Queue** This interface allows to query the number of available taxi and no other information.

**TaxiQueue:** This component, with the required interfaces it needs, is the central point of our application because, as written for ManageRequest one, it is responsible of the correct management of requests, users and drivers on whose the service our app will provide is based.

**4G) Selected architectural styles and patterns**

myTaxi System will be developed under two main architectural styles/patterns. Development of the project will be done in MVC architectural style and also 3 tier Client/Server Architecture. Client can browse the internet and access myTaxi application.

* MVC Architecture Style (Model – View – Controller)

MVC Style separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other.

1. **The Model component** that manages the system data and associated operations on that data.
2. **The View component** that defines and manages how the data is presented to the user.
3. **The Controller component** that manages user interaction and passes these interactions to the View and the Model.

We will use this MVC Style for myTaxi System because there are multiple ways to view and interact with data. We have decided to use MVC architectural style to separate the application logic with the interface. The main advantage of this is style allows the data to change independently of its representation and vice versa. Support presentation of the same data in different ways with changes made in one representation shown all of them.

* Three-Tier Client Server Architecture

In a client server architecture, the functionality of the system is organized into services, with each service delivered from separate server. Clients are users of these services and access servers to make use of them.

We will use this 3- Tier Client Server Architecture because, when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is a variable.

In particular, there are three tier:

1. **Data Tier**

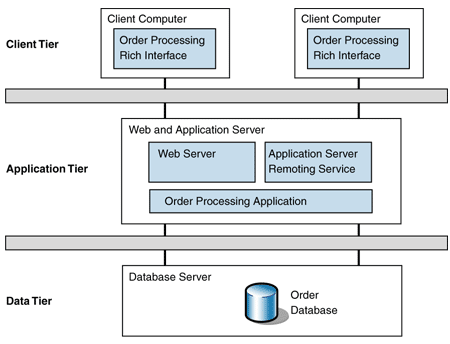
The data tire maintains the applications data such as Users’ data , Drivers’ data , Request’ data , TaxiQueue’ data and the SQL queries . It stores these data in a relational database management system (RDBMS). All the connections with the RDBMS are managed in this tier.

1. **Application Tier**

The application tier (web / application server) implements the business logic , controller logic and presentation logic to control the interaction between the applications’ clients and data. Business rules enforced by the business logic dictate how clients and cannot access application data and how applications process data.

1. **Client Tier**

The client tire is the applications user interface connecting data entry forms and client side applications. It displays data to the user. User interact directly with the application through user inter-face. The client tier interacts with the web/ application server to make requests and to retrieve data from the database. It then displays to the user the data retrieved from the server.

In the picture below it’s shown how does it work the 3-tier architecture style:

In the picture below it’s shown how we have combined the two style/pattern. On the presentation tier we have the view and the controller that invokes methods on the model in which there are the two other tier. In particular, in the model the chance become from controller on the application tier which in turn imparts to the data.

