**Overview**

MyTaxiService involves different users communicating over the internet with a single system. Such users may use different platform (mobile and web) and can send requests of different types. The system must not only accept those requests and elaborate an answer in a short time, but it is required that it notifies multiple users of the occurring of some events. Usually a single event provide notifications for two types of users: taxi drivers and customers. Events notification and users requests might also necessitate to access stored data, like taxi identifier or users information.

This brief analysis clearly highlight the need of implementing MTS as a client-server-like architecture, eventually subdivided into multiple physical tiers and logical layers: this will allow to model properly the request-answer requirement. The notification and updates part, instead, requires in our opinion a particular event-based paradigm: publish-subscribe. This allows user (the subscribers) to be notified by an entity (the publisher) on specific topics (a ride, for example).

These styles will be explained in detail in the following chapter.

**Selected architectural styles and patterns**

*Three-Tier Architecture*

The image shows the tier architecture of the MTS system, composed by three physical tiers. We will now analyze every tier and explain its logical functions.

* *Top tier (Client)*

The users’ machines, that in our domain are mobile phones and computers, will have the only purpose to load the Graphical User Interface (GUI), which shows the services that can be requested from the MTS’s system. No application logic is involved at this level: Clients will only be able to send requests to the web server and application server.

Notice that users identified as clients are limited to the followings: Taxi Drivers, Customers, Guest.

* *Middle tier*

This tier encapsulate:

* The Web Server, which is the component of the system that manages the HTTP requests sent by clients using the web application.

This component can handle such requests in two ways:

* if the request can be resolved with a static content page, the web server will generate and send the response itself
* if the request comport a dynamic content, the web server will delegate the dynamic response generation to the application server
* The Application Server, which provides access to the business logic, to be used by the client application programs. This component is the central part of MTS’s system, and will contain all the logic that provides MTS’s services. To accomplish this, it will be able to execute complex algorithms and access the Database tier.

The Application Server will also provide lightweight APIs to be used directly by mobile application clients. It will answer mobile’s requests by sending only the strictly necessary information, reducing the amount of data transiting over the mobile network and thus increasing the performance of the application.

Web application clients, instead, will be able to access this component only indirectly, through the Web Server.

* The Admin’s GUI, the specific interface for Administrators, is actually included and provided by the Application Server. It allows Admins to access to their exclusive functions dialoguing directly with the business logic of the system. It’s completely disjointed from the other users’ GUI and functions.
* *Bottom tier (Database)*

This tier, which will be separated from the previous one with a (possibly local) network, contains all the data that MTS needs to store, ranging from the users’ information to the city map.

*Even-based system*

As anticipated, the core of MTS’s application logic is based on the publish-subscribe pattern.

Customers and taxi drivers have the role of subscriber: the system will automatically register them to specific topics, and they will receive update messages related to such topics. A topic is created every time customers request or reserve a ride. When a taxi driver is associated to that ride, he will be subscribed to the same topic too, and receive the relative notifications. Additionally, every taxi driver has an exclusive topic strictly bound to its status (available, busy…), which allows them to receive notification when the status is switched.

In this pattern we intend to use a broker, an intermediary component which performs the queue management and the filtering of the messages. The broker will allow to filter messages based on their content, so that taxi drivers and customers related to the same topic won’t receive necessary the same notification or messages.

The role of the publisher, instead, is associated to the logic components of the system that manages the rides, the research of available taxis, etc. Basically, there are more components of the system that may generate an update for a certain topic, even users (for example, when a taxi driver change its status generates an update for its topic).

Despite the event-based pattern may not be strictly required to model the actual MTS system, it provides much space for future extendibility. For example, it allows to handle notification for multiple customer connected to the same ride (e.g. in a taxi sharing service), or permits to easily add new types of notifications.