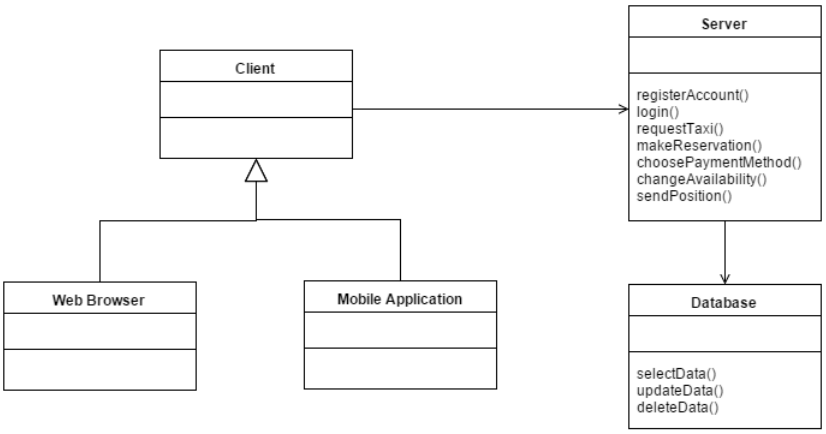
1 Architectural Design

1.1 Overview

This section deals with the specific description of the architectural choices made for the myTaxiService system. These decisions are supported by different diagrams, whose aim is to highlight and simplify the view of the components of the system, and explanations about the architectural styles adopted.
In the first subsections there is a representation of the components of the system with diagrams. Here are presented the high level components and their interaction, with a brief definition of the functionalities of the three components individuated, then these are unfolded and there is a more specific description of the components, the interfaces and their relationships and features.
The last subsection explains and justifies the architectural styles selected for the myTaxiService system, which consists in a client/server, event-based system, with a service-oriented, three tier architecture (client, server and database).

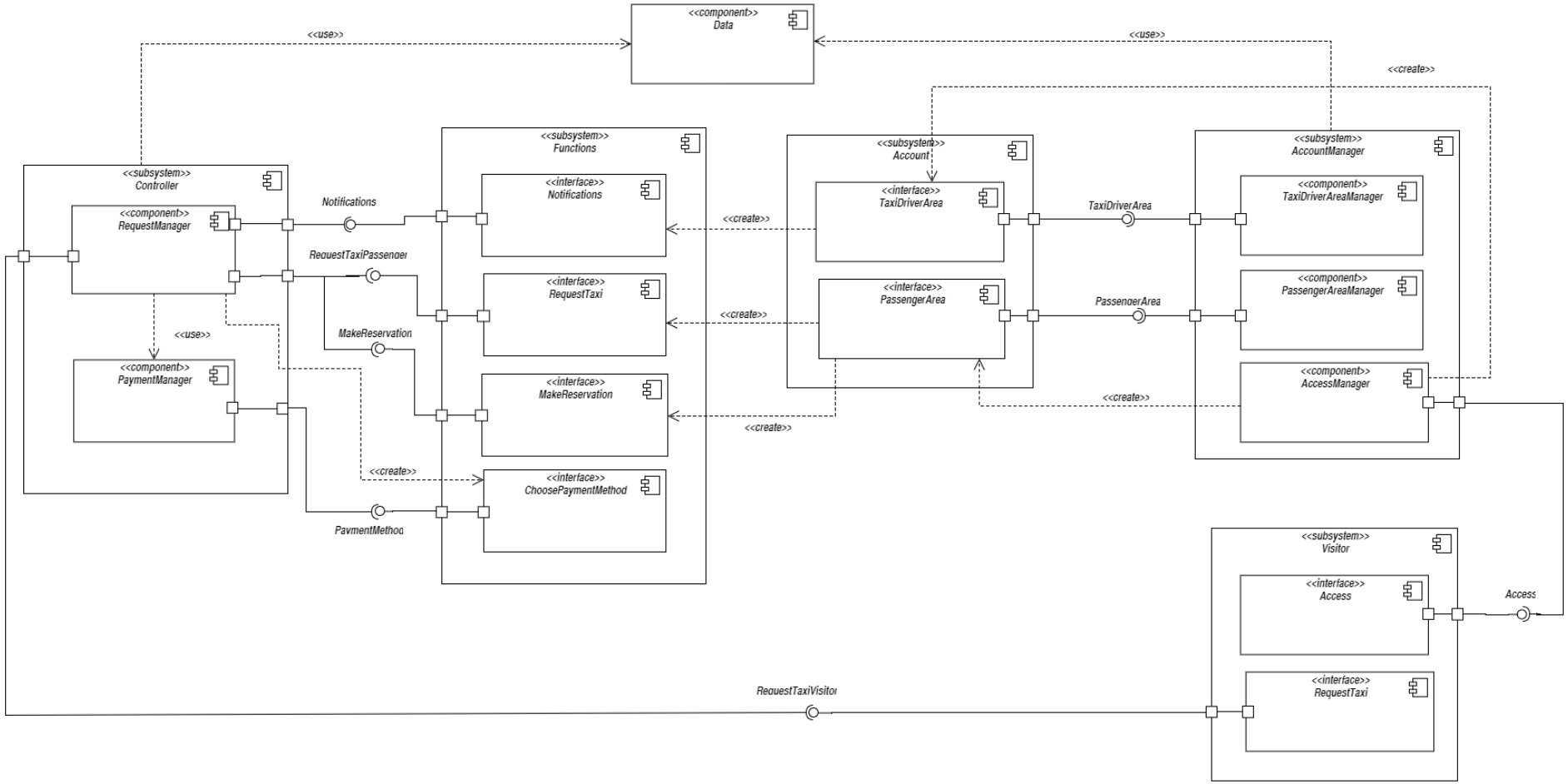
1.2 High level components and their interaction



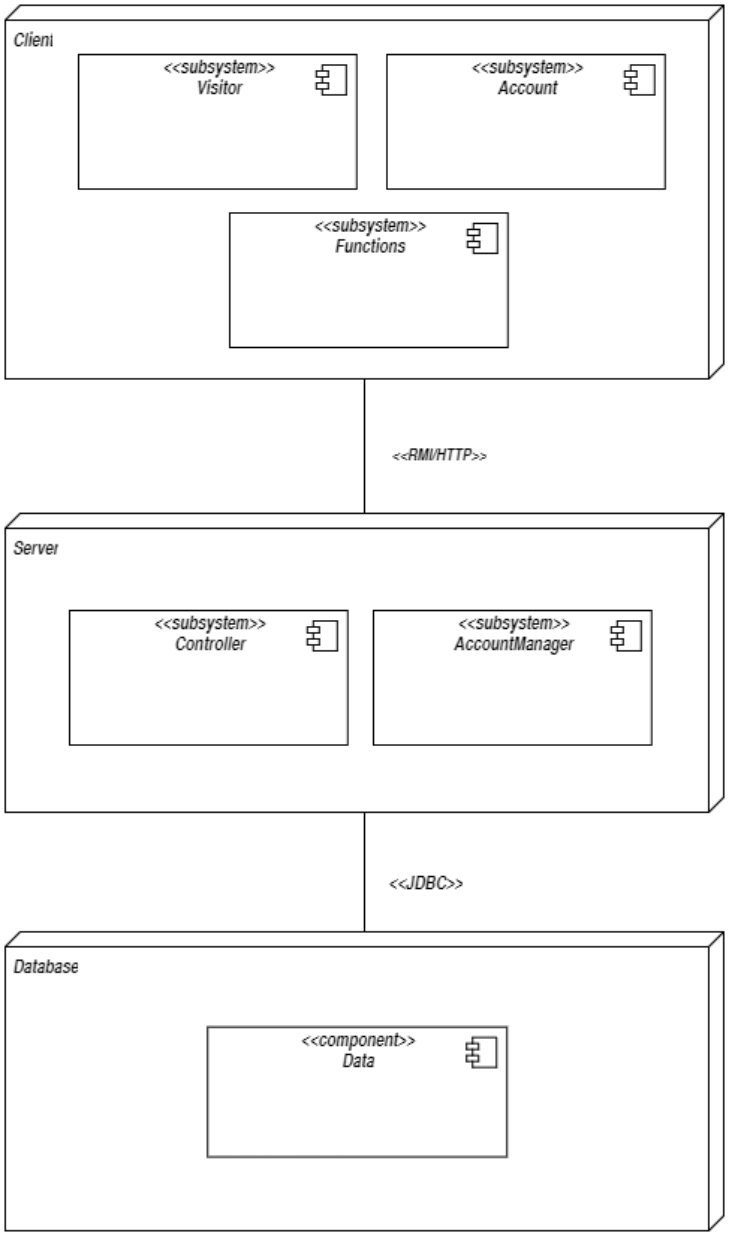
- **Client**: this part runs on the client devices via a Web browser or the mobile application. It allows the users to insert and submit the data in the input forms, that are sent to the application server. On the other hand, the taxi drivers can send information about their availability to the server and the application client monitors their GPS position in order to move the taxi drivers to another queue if they change their area.
- **Application Server**: this part runs on the JEE server. It is composed of a web server part, which always listens to all the clients requests and is responsible for the creation of the faces and pages of the client interfaces. The application server, instead, contains the logical part of the application, collecting and managing the information from the clients and the database. In fact, it analyses the data coming from the clients and, according to the requests, it modifies or asks for the required information stored in the database, then it is able to answer the client request, sending it the result.
- **Database**: this part contains the database where all the application data are stored. It is not only accessed by the application server, but also by the administrators, who can, for example, directly add a taxi driver account to the database.

1.3 Component view

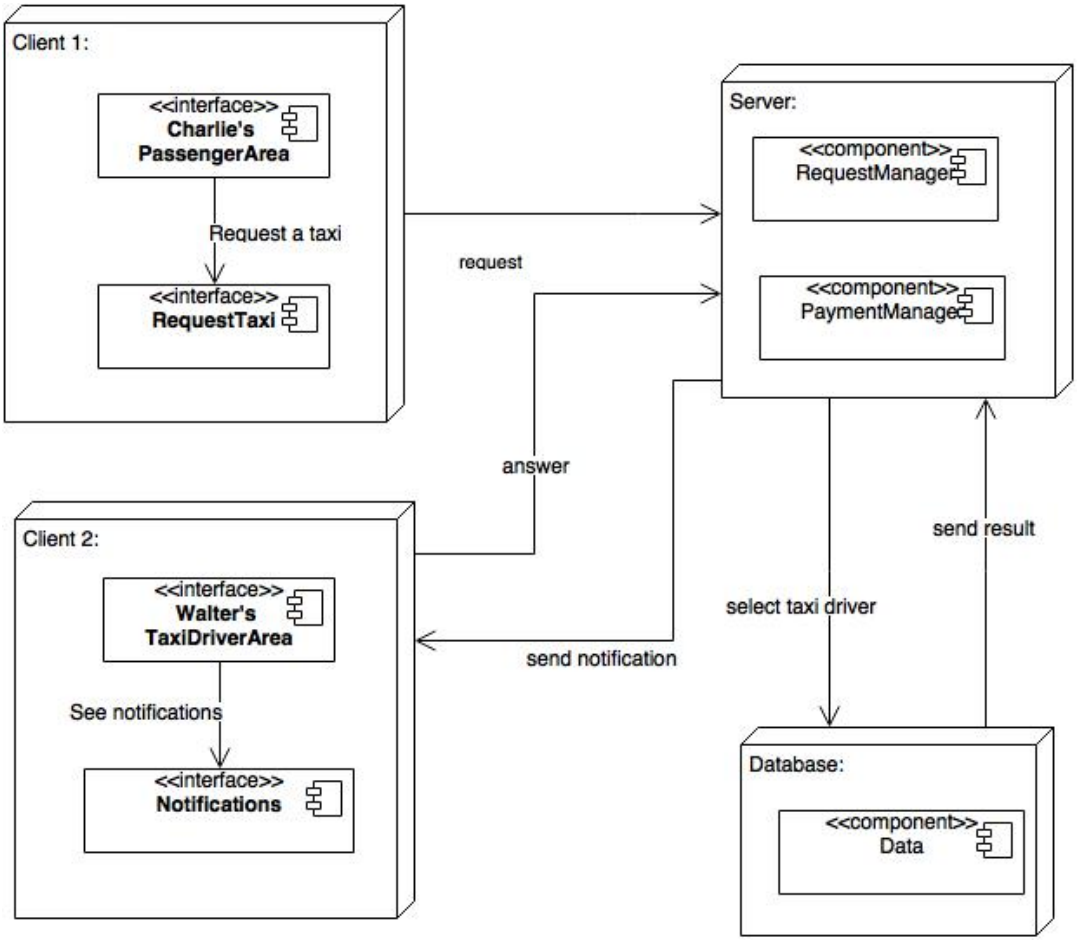
These are the components that define the myTaxiService system architecture. This diagram is composed by many subsystems and an external component which belongs to the database.
Starting from the left, in the first subsystem, the **Controller**, there are two components that manage the requests of the users and the payments.
The *RequestManager* component manages all the logic of the requests, getting the information from the users and forwarding them to the designated taxi driver. As it deals with several tasks, he needs more than one interfaces to be implemented: *RequestTaxiVisitor*, *Notifications*, *RequestTaxiPassenger* and *MakeReservation* are the required interfaces for this component. It also relates with other components: it uses the *PaymentManager* and the *Data*, in order to complete its operations, and creates the *ChoosePaymentMethod* interface, which is required by the *PaymentManager*.
The second subsystem is called **Functions**, as it provides the possible actions that the user can do while using the system. It is composed by four interfaces provided to the **Controller**: *PaymentManager*, *RequestTaxiPassenger*, *MakeReservation* and *Notifications*.
This last interface can be accessed from the taxi driver mobile application to see the notifications of the requests, in fact it is produced by the *TaxiDriverArea* interface. The *RequestTaxiPassenger* and *MakeReservation* interfaces, instead, are created by the *PassengerArea* interface, which is part of the subsystem **Account** with *TaxiDriverArea*. They are used by the passenger when he needs to request a taxi or make a reservation from his personal area.
Another subsystem is the **AccountManager** whose main goal is to manage the access of the users, forwarding them to the proper designated area. The *TaxiDriverAreaManager* requires the interface *TaxiDriverArea*, which is provided by the *TaxiDriverArea* interface in **Account**. The same relation is defined between the *PassengerAreaManager* component and the *PassengerArea*. The third component of this subsystem, the *AccessManager*, which is responsible for the log in and the registration of the users, creates an instance of *PassengerArea*, while requires the interface *Access*.
The last subsystem is the **Visitor**, which provides the interface *Access* to the *AccessManager* and another interface, *RequestTaxi* to the *RequestManager*. In fact, this subsystem represent the home page where the visitor can register or log in and request a taxi.


<<component>>
Data
<<use>>
<<use>>
<<create>>
<<subsystem>>
Controller
<<component>>
RequestManager
<<component>>
PaymentManager
<<use>>
Notifications
RequestTaxiPassenger
MakeReservation
PaymentMethod
<<create>>
<<subsystem>>
Functions
<<interface>>
Notifications
<<interface>>
RequestTaxi
<<interface>>
MakeReservation
<<interface>>
ChoosePaymentMethod
<<create>>
<<create>>
<<create>>
<<subsystem>>
Account
<<interface>>
TaxiDriverArea
<<interface>>
PassengerArea
TaxiDriverArea
PassengerArea
<<create>>
<<subsystem>>
AccountManager
<<component>>
TaxiDriverAreaManager
<<component>>
PassengerAreaManager
<<component>>
AccessManager
<<subsystem>>
Visitor
<<interface>>
Access
<<interface>>
RequestTaxi
Access
RequestTaxiVisitor

1.4 Deployment view



1.5 Runtime view



1.6 Component interfaces

Here is presented a list of the interfaces defined in the component diagram and their functionalities.

- **Access.**
- **TaxiDriverArea.**
- **PassengerArea.**
- **RequestTaxiVisitor.**
- **RequestTaxiPassenger.**
- **MakeReservation.**
- **PaymentMethod.**

1.7 Selected architectural styles and patterns

- **Client/Server.** The client/server architecture is the optimal solution for the myTaxiService system, as it is necessary to have a central system that listens, manages and forwards the requests of the different clients. There is a central server that contains the logic of the application and the clients are the users of the system, such as visitors, passengers and taxi drivers.

- **Three-tier.** It has been adopted a three-tier architectural model, composed by thin client, application server and database.
  This architecture is the best choice for our system, even if it has some cons, such as the complexity of the structure and the difficulty of set up and maintenance, it still has several pros. For example, it guarantees increased performance and great flexibility, useful if there will be any future change concerning the architecture. Moreover it is granted a great security level, thanks to the decoupling of logic, data and presentation, which is essential as the system deals with several personal data.

- **Event-Based System.** The myTaxiService application is based on the event firing.
  It is necessary, in fact, that the system is reactive and that does different quick operations according to the action of the clients. For example the visitors and passengers make requests and reservations, which the system has to manage and forward to the first taxi of the area, and the taxi drivers can change their availability state and the system has to start or end the monitoring of their position, and they can accept or decline a request and the system has to manage the queue.
  The users and the taxi drivers are registered to different events and expect to receive notifications about what they need, whether it is the arrival of a taxi or the requests of the users.
  The events are asynchronous and based on a "send and forget" paradigm, where the system only cares for sending the notifications to the designated clients or doing the actions needed in response of the event fired.

- **Service-Oriented Architecture.** A service-oriented architecture is necessary if the system wants to be more flexible and expandable. In fact, the myTaxiService application needs to provide programmatic interfaces in order to be open to future implementations of additional services. This can be guaranteed with this architectural choice, which is based on the loose coupling of the services, allowing to easily add more functionalities, without starting from scratch when a change is needed, and to simplify the maintenance of the system.