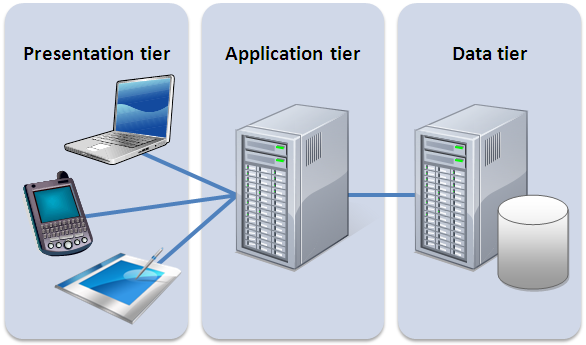
As previously introduced, a Three-Tier architecture is used in this project:



1. Presentation Tier (Client Tier)
   1. Composed mainly by:
      1. The app used by the user to find and rent a car
      2. The screen in the rented car
      3. The tools used by the Supervisor.
   2. It’s the layer that interacts with the serverto display useful information.
2. Application Server
   1. Where all the computations (reservations, locking/unlocking cars, payment) occur.
3. Data Tier:
   1. MySQL DBMS

Elaborating more, a three tier architecture like the one described here, represents a situation where the storage logic (database) is separate from the business logic (application) and from the presentation.

This kind of separation of duties allows for:

1. An easier implementation of the system
2. An efficient scalability (future proof architecture). We could expand the DB, should we need a bigger one, change the UI, etc. with a low impact to the untouched tiers.
3. It is also a matter of security, allowing for a better isolation of different processes.

The system can then be described as having an object-oriented architecture:

* All components are object
* Connectors are messages and method invocations
* The internal representation is hidden from other objects

That gives us all the benefits of the object-oriented paradigm, with focus on the scalability (as explained before), reusability and testability.



The system is meant to be developed with a test driven development (TDD): all the goals discussed in the RASD document (and in the following pages), will be converted into test cases and then programmers will code aiming to pass those tests.

The software will then evolve adding more goals/tests.

As always, integration tests need to pass with the addition of every new module.

The tools used in the testing process are not discussed in this document.

In addition, the approach used in this system is a top-down one: we started from the agreed specifications, we discussed the master features needed (the high-level structure of the system) and then we started to work down to detailed decisions about low-level constructs and individual algorithms.

Finally, a system like this one must comply with a number of QoS metrics like:

* Reliability: what if the user rents a car, parks it in a shopping mall car park and then, after buying a lot of frozen foods, it finds out that the service isn’t working and the car won’t open?
* Availability: the service must be continuously available to the user; we can’t allow downtimes in a competitive business like this one.
* Usability: a not so user friendly app (or car’s display) will make us lose the customer or we’ll need to invest a lot of resources in Public Relations and Customer Assitance.