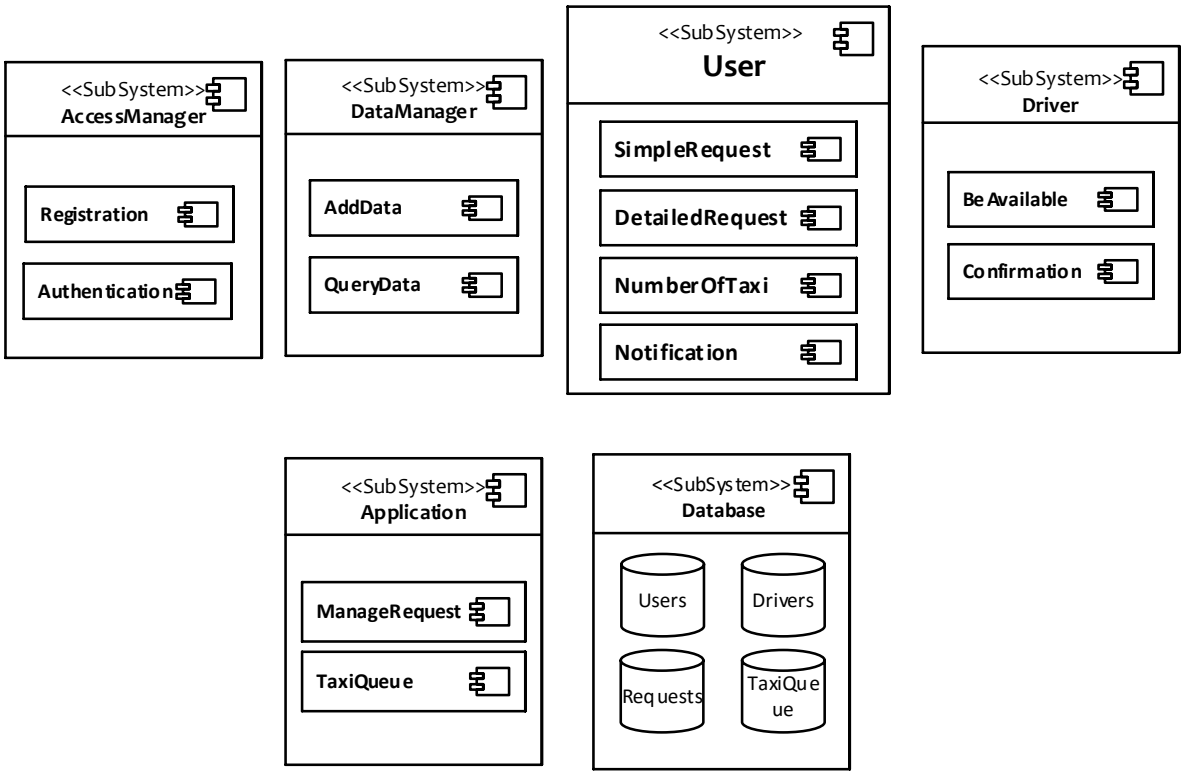
Integration Strategy

2.1 Entry criteria.

Before starting the integration testing of the subsystems, we recommend to perform a unit test on the “application” sub-system in order to check the algorithm correctness, the functional specs and the structural coverage. This is the major class that contains the logic of myTaxiService. In this class will be implemented, according with the document design already provided, the algorithm of management of the queue of taxis and the algorithm of forwarding the requests arrived from the user to the taxi's drivers and that’s why it’s important to perform a unit test on this class before proceed to the integration test.
The documentation about what is needed to perform this kind of test is in the fifth paragraph of this document.

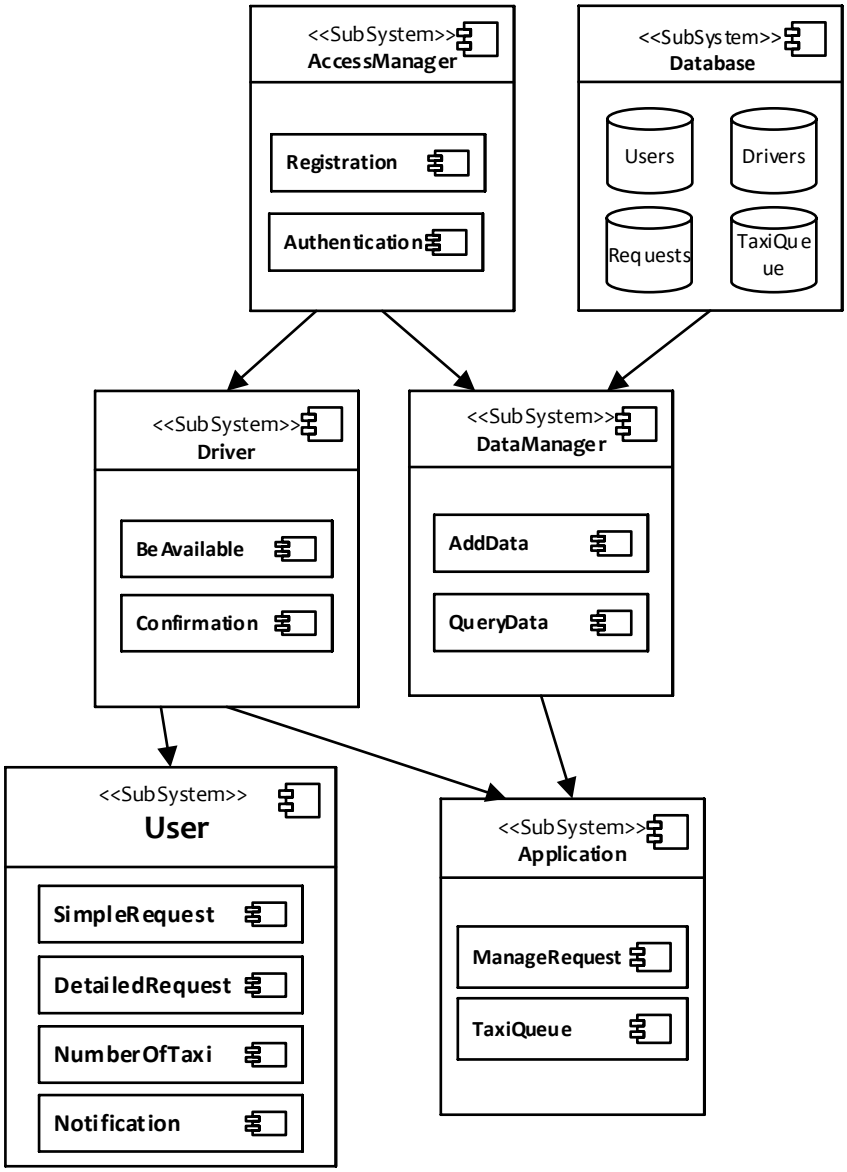
2.2 Elements to be integrated.

The items to be tested consists of the integration of all the subsystem developed that form myTaxyService.



2.3 Integration Testing Strategy.

For testing, we choose the top-down approach. In this approach testing is conducted from the high level interfaces to the core of our application. It's advantageous because the major flaws occur toward the top of the program. It's also a way for simplify the readability of the results of the test case because once the I/O functions are added, representation of test cases is easier.
Due to these considerations, in the pictures below are shown trough the arrows, the order of integrations about the subsystems.



Regarding the User subsystem, the first software components to be integrated are "Simple Request" and "Detailed Request".
Regarding the Application subsystem, the first software component to be integrated is TaxiQueue.
Regarding all the others subsystems, each component has to be integrated with the entire subsystem.

4 Tools and Test Equipment Required

In this paragraph, we suggest some tools by which perform different kind of tests. It is only a suggestion that can be followed or not.

A manual testing that helps the activity of verification and validation is the v&V activities and software artefacts (the V model). This is an easy understandable way to fix what the program developed really does compared to what it should do.

In myTaxiService application, the objects are in continuously interaction and it is useful to know what are the methods called “n” times or never called. It is useful too knowing if a controller is calling the right service. We suggest Mockito software, available at this link http://mockito.github.io , in order to perform this kind of test.

Another important tool to verify other aspects is JMeter, available at http://jmeter.apache.org/download_jmeter.cgi . myTaxi Service could have a very high contemporary audience especially during peak hours. With JMeter we can simulate logging into the web site and application, clicking buttons, sending requests, performs generally some action that are going to make heavy the server. It is very important to have tested this aspect about the performance and network’s stress in order to be safe about an application’s crash due to bigger traffic on the server.

5. Program Stubs And Test Data Required

In this paragraph we are going to highlight what it’s necessary about program stubs, drivers or anything other else, in order to perform the integration test following the guideline we have proposed. Since we have chosen a top-down approach, stubs are required at every single steps during the integration test. A ‘Stub’ is a piece of software that works similar to a unit which is referenced by the Unit being tested, but it is much simpler that the actual unit. A Stub works as a ‘Stand-in’ for the subordinate unit and provides the minimum required behaviour for that unit. A Stub is a dummy procedure, module or unit that stands in for an unfinished portion of a system.

1. At the first step of our integration, a stub that simulates the database response correlated at a logging or registering action is required.
   In example:

```
public void login(String username, String passwrod) {

    if username.equalsTo(“MarMas”) and password.equals(“Admin”) then
            return true

    if username.equalsTo(“”) or password.equalsTo(“”) then
            return false
    }
```

   - This is a very law level of example just only to show and make understandable what kinds of stubs we aspect the tester will perform. It’ needed only to simulat e a function not already implemented.

2. A stub that simulate operations on a database is also needed.

3. At the second step of the integration test are required different stubs:

    - A stub for drivers which by clicking on be_available gain the right to enter in a queue list.
    - A stub that simulate a pending request
    - A stub that simulate the timing out of the response's time with the relative consequences.
4. Finally, by integrating the remaining components there is no stubs to do, that's why now all the entire application is integrated and no more functionalities have to be implemented yet

More detail on what to expect for a single test case can be found in the third paragraph of this document.

The documentations that must be provided before starting with the integration test are:

- The Design Document (DD)
- All this document, in particular the specific paragraphs concerning the strategy.

The documentations and materials that must be provided before starting a specific integration test are:

- The paragraph concerning the order of the component's integration.
- All the stubs needed for testing the components.
- The unit test report concerning the "application" component

At the end of the integration test we advise for a system test checking not only the design, but also the behaviour and even the believed expectations of the customer.