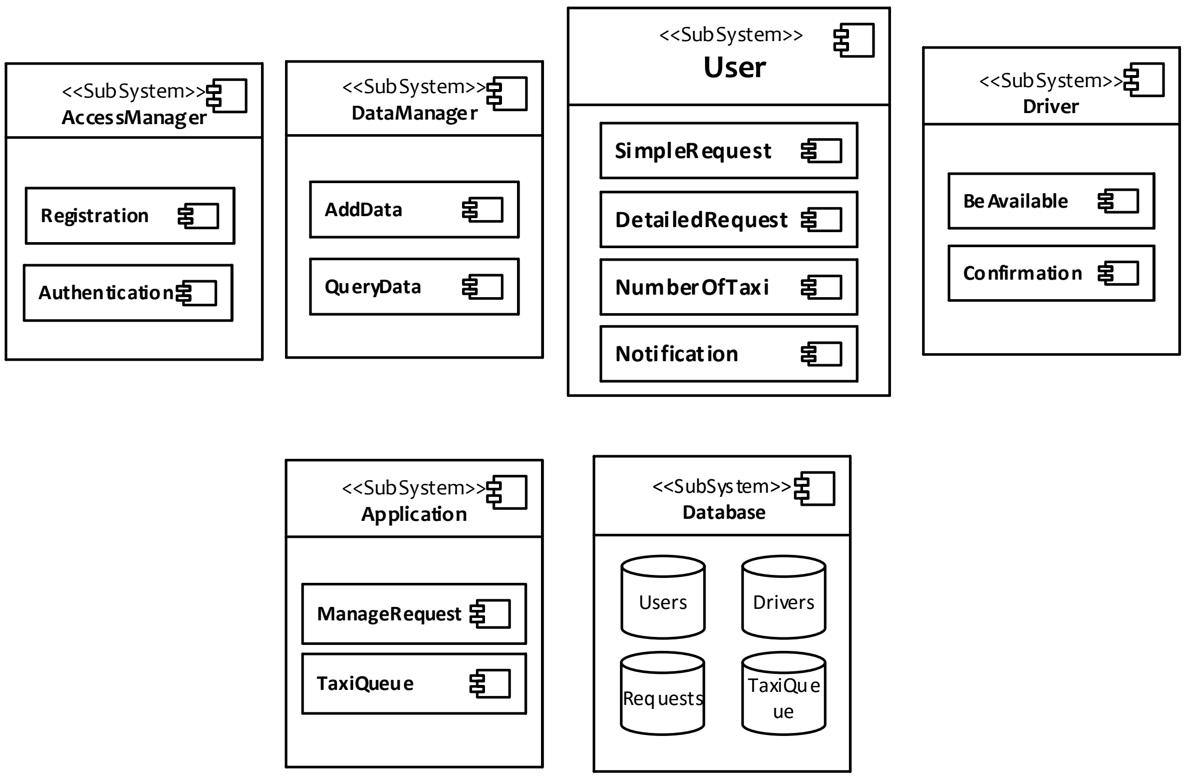
Integration Strategy

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.

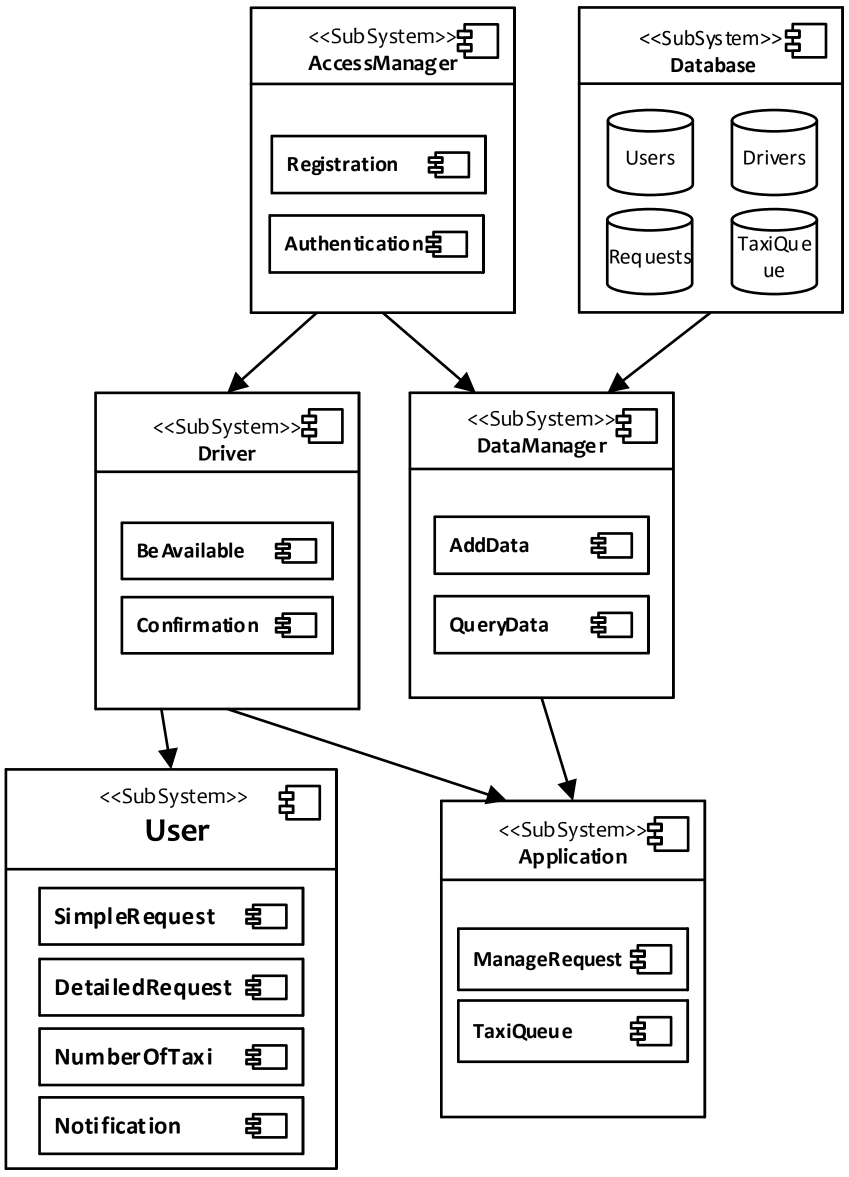
2.2 Elements to be integrated.

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 bottom-up approach. In this approach testing is conducted from sub-modules to main module, if the main module is not developed, a temporary program called DRIVERS is used to simulate the main module. The reason of our decision is principally about the major flaws that occur toward the bottom of the program and also because with this approach development and testing can be done together so that the product or application will be efficient.
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 our server. It is very important to have tested this aspect about the performance in order to be safe about an application's crash due to bigger traffic on the server.