MyTaxi

Integration Test Plan

Authors:

Bucci Giovanni

De Togni Riccardo

Summary

1. Introduction

1.1. Purpose and Scope

The purpose of this document is to verify functional, performance, and reliability requirements of the My Taxi service developed so far.

Since the project is vast and complex, it is useful to make this document in order to have a solid point of reference about ways and techniques of testing, and a schedule about the order of components to be tested.

1.2. Glossary

Below is reported the glossary already inserted in RASD and in the DD:

- CUSTOMER: a generic person that use any part of the system service, it could be either a user or a guest.
- DBMS: Database Management System, the set of machines and specific operation that allow the right Database working.
- ADMIN/ADMINISTRATOR: it is a particular type of user that has administrative functions.
- GUEST: a person who has not signed up yet. Guests have no power until they sign up with one exception. If a Guest just want to call a taxi, it could simply insert its identification data.
- USER: a person that has already signed up as a customer. It could call a taxi, as guest does, but it also could reserve it in advance, compiling a specific form.
- TAXI DRIVER: a person who has signed up as a taxi driver. In order to complete its registration it has to provide its identification data and its driving license too.
- SYSTEM: the environment formed by the application itself and its features.
- CITY ZONE: each city is divided in zones. Every zone has approximatively the same territorial extension, so a city zone is one of the portions of the metropolitan area.
- QUEUE: an ordered list of taxi drivers that have previously provided their availability.
- CALL A TAXI: the action which can be performed both by guests and user, that consists in asking for a single taxi ride without any advance.
- RESERVE A TAXI: the action that could be performed only by Users. A user can forward the request for a taxi from a specified place to another in advance.
- SERVICE: the service that is provided by the application.

- DENY/DENIAL: when a request is not satisfied. It produce the shifting of the considered taxi driver to the bottom of the queue.
- ACCEPT: when a request, both coming from a reservation or a taxi call, is taken by a taxi driver who assumes the charge to bring passengers to the destination.
- UI: the user interface i.e. the set of web pages that constitute the meeting point for users and system.
- RASD: Requirement Analysis and Specifications Document
- DD: Design Document

1.3. References

- RASD
- Design Document
- Assignment 4 Integration Test Plan

2. Integration Strategy

2.1. Entry Criteria

Before starting the testing process, every lowest-level component (like single Java classes) has to be checked alone. This means that every class code has been revised using code inspection, and every function has been tested alone.
It is also necessary to write and collect the documentation related to single classes, and previous RASD and DD are requested in order to know every detail and plan the testing in the best way.

2.2. Elements to be integrated

The elements that have to be integrated will correspond to the components already presented in MyTaxi Design Document, since they represent each one a stand-alone feature.

It is possible to identify three macro components in the Design Document, according to the three-tier structure [DD 2.1] which will represent our subsystems to be integrated:

- Client-tier subsystem: client side part of the system, which consists in a web browser for the web application and a mobile app for the mobile application.
- Business Logic-tier: the core of the system, it contains all the application logic.
- Persistence Module: it manages the data storage.

These macro components are themselves composed by some subcomponents that have to be integrated first following the rules explained in the next paragraph.

Another subsystem can be identified grouping all the external component used by the system. Further explanation will be in paragraph 2.4.

In particular, Client-tier has a subcomponent for each different type of client [DD 2.7].

Several Web Services compose the Business Logic:

- Request Web Service
- Reservation Web Service
- Notification Web Service
- Queue Web Service, that logically contains one subcomponent for each queue zone
- Entity Manager Service, that contains a subcomponent for each entity defined by ER-Diagram [DD 5.1]

The persistence module contains the Database component and its related interface to the Business logic that now on we call DBMS.

2.3. Integration Testing Strategy

The integration strategy is based on a bottom-up approach. This choice depends on the fact that every smallest unit has already been tested, and so it is easier in this situation to put low-level components together and test every subsystem created. This method is therefore efficient, and saves time and resources since it is not necessary to create stubs and to simulate components behaviours.
Finally, high-level components are separated from lower tiers by using defined interfaces and are easy to design and implement, so it is useful to produce them in the last phase.

2.4. Subcomponent and Subsystem Integration Sequence

Here is presented the table with the specific components to be integrated during the development of the application. The components listed are in integration order: from the persistence module to the client tier every component is integrated systematically following the table (and the order described in the diagram).

It is clear that also the subsystem are integrated in the order of their subcomponents. The Persistence module is created first, then the Business Logic level and the Client tier.

Finally external services, until now replaced by stubs, are integrated to the application.

Persistence Module	Database Structure (SQL file)
	Database Management System
Business Logic	Entity: User
	Entity: Guest
	Entity: Admin
	User Manager
	Entity: Taxi
	Taxi Manager
	Entity Manager
	Request Manager
	Entity: Queue
	Queue Manager
	Reservation Manager
	Notification Centre

Client Tier

Web Application

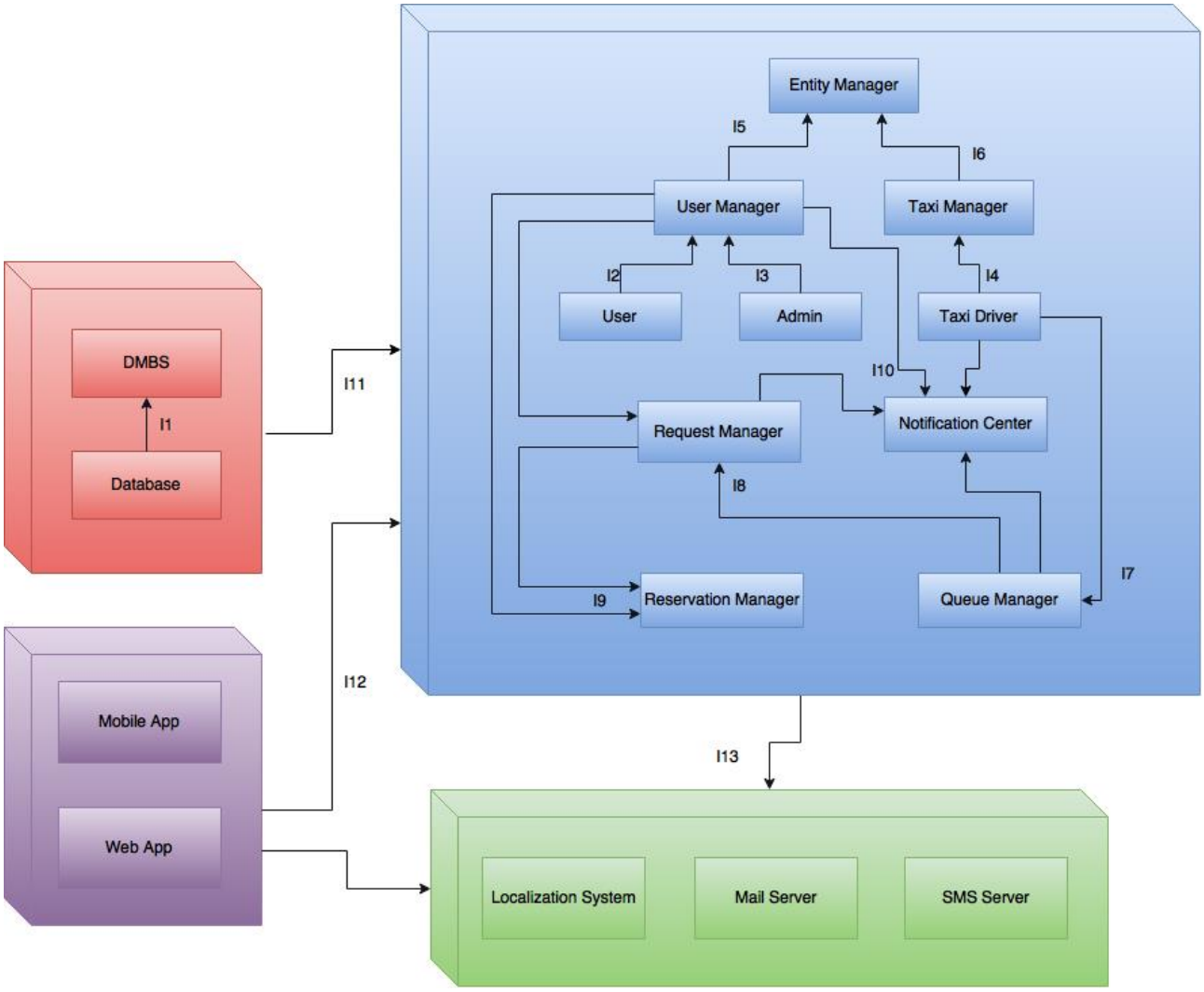
Mobile Application

External Modules

Localization System

Mail Server

SMS Server



The diagram above deepens the description of integration sequence. The internal component of each subsystem is integrated following the specific numeric sequence. After the integration of all the subcomponent of a subsystem, the next step is to integrate two subsystems among themselves.

3. Individual steps and test description

3.1. Persistence Module Tests (P)

Test Case Identifier	P I1 T1
Test Item(s)	Database → DBMS
Input Specification	Create random queries
Output Specification	Check database structure and concurrency
Environmental Needs	N/A

3.2. Business Logic (B)

Test Case Identifier	B I2 T1
Test Item(s)	User → User Manager
Input Specification	Generate typical User traffic
Output Specification	Check user management
Environmental Needs	N/A

Test Case Identifier	B I3 T1
Test Item(s)	Admin entity → User Manager
Input Specification	Generate typical Admin traffic
Output Specification	Check admin management
Environmental Needs	N/A

Test Case Identifier	B I4 T1
Test Item(s)	Taxi Manager → Taxi Manager
Input Specification	Generate typical Taxi Manager traffic
Output Specification	Check taxi management
Environmental Needs	N/A

Test Case Identifier	B I5 T1
Test Item(s)	User Manager → Entity Manager
Input Specification	Generate typical User Manager traffic
Output Specification	Check generic entity manager
Environmental Needs	I2 and I3 tests succeeded

Test Case Identifier	B I6 T1
Test Item(s)	Taxi Manager → Entity Manager
Input Specification	Generate typical Taxi Manager traffic
Output Specification	Check generic entity manager
Environmental Needs	I4 test succeeded

Test Case Identifier	B I7 T1
Test Item(s)	Taxi Driver →Queue Manager
Input Specification	Generate taxi driver availability
Output Specification	Check queue management
Environmental Needs	N/A

Test Case Identifier	B I8 T1
Test Item(s)	User Manager, Queue Manager → Request Manager
Input Specification	Generate typical request traffic
Output Specification	Check request management
Environmental Needs	I2, I3 and I7 tests succeeded

Test Case Identifier	B I9 T1
Test Item(s)	User Manager, Request Manager → Reservation Manager
Input Specification	Generate typical reservation traffic
Output Specification	Check reservation creation and management
Environmental Needs	I2, I3 and I8 tests succeeded

Test Case Identifier	B I10 T1
Test Item(s)	User Manager, Taxi Driver, Request Manager → Notification Center
Input Specification	Simulate typical notifications
Output Specification	Check notification behaviour
Environmental Needs	I2, I3, I7 and I8 tests succeeded

3.3. Subsystems Test

Test Case Identifier	S I11 T1
Test Item(s)	Persistence Module → Business Logic
Input Specification	Simulate interactions between the two subsystems
Output Specification	Check
Environmental Needs	P block and B block tests succeeded

Test Case Identifier	S I12 T1
Test Item(s)	Client Tier → Business Logic
Input Specification	Generate typical client traffic
Output Specification	Check
Environmental Needs	B block and C block tests succeeded

Test Case Identifier	S I13 T1
Test Item(s)	Client Tier → External Applications
Input Specification	Simulate remote call to the external application
Output Specification	Check external applications compatibility and functionalities
Environmental Needs	C block tests succeeded

Test Case Identifier	S I13 T2
Test Item(s)	Business Logic → External Applications
Input Specification	Generate remote calls to specific external applications
Output Specification	Check external applications compatibility and functionalities
Environmental Needs	B block tests succeeded

4. Tools and test equipment required

- Apache JMeter: It is used to simulate a heavy load on a server or object to test its strength or to analyse overall performance under different load types;
- Junit: unit-testing framework for the Java programming language;
- Mockito: software used to mock missing components;
- Manual testing.

5. Program stubs and test data required

The chosen testing strategy minimize the stubs needed to test the components. However, in order to fully test the application, it is necessary to create some stubs to overcome the external components not yet available, like the localisation system, and the message and e-mail service.
It is also necessary to generate sample data in order to test both database operation, like insertion and deletion, and the business tier that needs data to manipulate.