Politecnico di Milano

Industrial and Information Engineering
Computer Science and Engineering

Software Engineering II Assignment

PowerEnJoy - car sharing

Academic Year: 2016/2017

Prof. Elisabetta Di Nitto

Alice Segato matr. 875045
Mark Edward Ferrer matr. 876650
Davide Bonacina matr. 876199

PowerEnJoy

Car Sharing App

TABLE OF CONTENT

1 INTRODUCTION

1. Revision History

Version	Date	Authors	Summary
1.0		Mark Edward Ferrer, Alice Segato, Davide Bonacina	Initial release

2. Purpose and Scope

This document describes the Integration Testing Plan for PowerEnjoy system.

This testing is necessary in order to avoid unexpected behavior of the system and guarantee its ability to fulfill all the requirements.

The ITPD outlines the testing activities organization for the subsystems that make up the system.

This document is composed by five parts:

- Integration Strategy: explains the selection of subsystems and their subcomponents for the testing and outlines, for each one, the project status that has to be met in order to start the testing.
- Individual Steps and Test Description: describes the integration testing approach, the sequence in which components and subsystems will be integrated and the planned testing activities for each integration step, including their input data and the expected output.
- Performance Analysis: performance measures on the components to check in order to verify the requirements fulfillment.
- Tools and Test Equipment Required: list of tools that will be employed during the testing activities and description of the environment for the test execution.
- Required Program Stubs and Test Data: list of program stubs and drivers to perform the necessary method invocations on the components to be tested.

1.3 List of Definitions and Abbreviations

- DD: Design Document.
- RASD: Requirements analysis and Specification Document.
- DB: the database layer, handled by a DBMS.
- UI: User Interface.
- GUI: graphical user interface is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators;
- Application server: the layer, which provides the application logic and interacts with the DB and with the front-ends.
- Back-end: term used to identify the Application server.
- Front-end: the components, which use the application server services, namely the web front-end and the mobile applications.

- Web server: the component that implements the web-based front-end. It interacts with the application server and with the users’ browsers.
- Acknowledge: is a signal passed between communicating processes or computers to signify acknowledgement, or receipt of response, as part of a communications protocols.

1.4 List of Reference Documents

- RASD;
- Assignment AA 2016-2017.pdf;
- …
- …

2 INTEGRATION STRATEGY

2.1 Entry Criteria

In order to start testing the integration of all the components previously described in the Design Document, there are some conditions that have to be fulfilled before the integration:

- The components must have been already designed in the Design Document in order to figure out their role in the system;
- The components must have been unit tested and they must be correct.

2.2 Elements to be Integrated

The components that we are going to test are all the internal components of the Main System, the Car Computer and the Client App (as described in the section 2.3.1 of the Design Document). The components of the Third Party System and the relative clients (both web and mobile applications) have to be already tested and fully working, according to our assumptions described in the section 1.6 of the Requirements Analysis and Specification Document.

[imagine dei component che devono essere testati]

Realistically speaking, handling a full system testing is very difficult, for this reason we clamp together the components that have strong dependencies into subsystems to be tested, and then we integrate the subsystem gradually until we obtain the complete system.

Main System

- The Application Controller depends on User Manager, Reservation Manager, Utility Manager subsystems;
- The API Controller depends on the Algorithm Controller;
- The API Controller depends also on the Application Controller.

Car Computer

- The Mobile application controller relies on the Sensor Manager.

User App

- The Mobile application controller depends on the GPS Manager;

These three components generate five subsystems and the components themselves must be integrated in this order:

1. Main System with Car Computer;
2. User App with Main System;
3. All these three components with the Third Party System and the Database.

2.3 Integration Testing Strategy

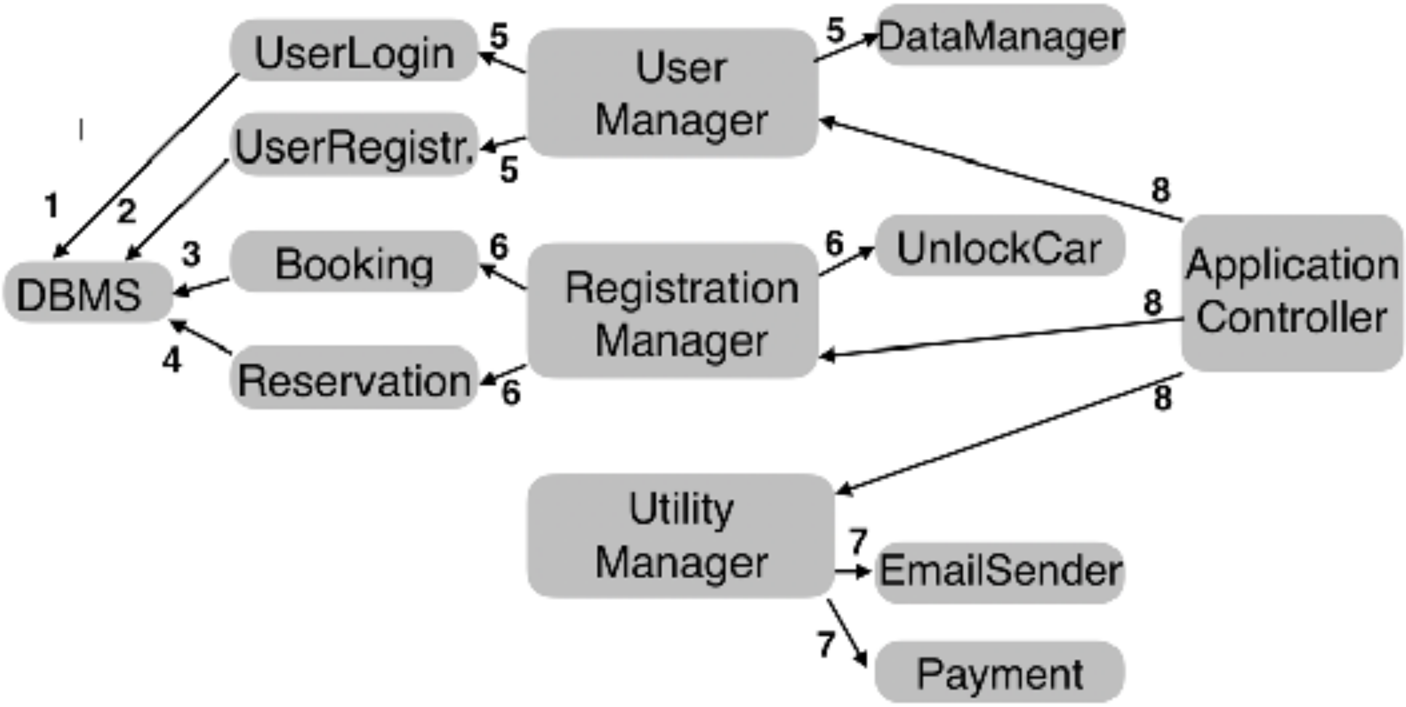
To approach the integration test phase we decided to adopt the bottom-up strategy to test first the lower level components until we obtain greater subsystems. At this point we follow the critical-module-first approach to integrate together the subsystems found in the previous step. We will need only one stub of the Main System component to be used during User App subsystem testing since some of the functionalities of the User App involve strongly some subcomponents of the Main System.

2.4 Sequence of Component/Function Integration

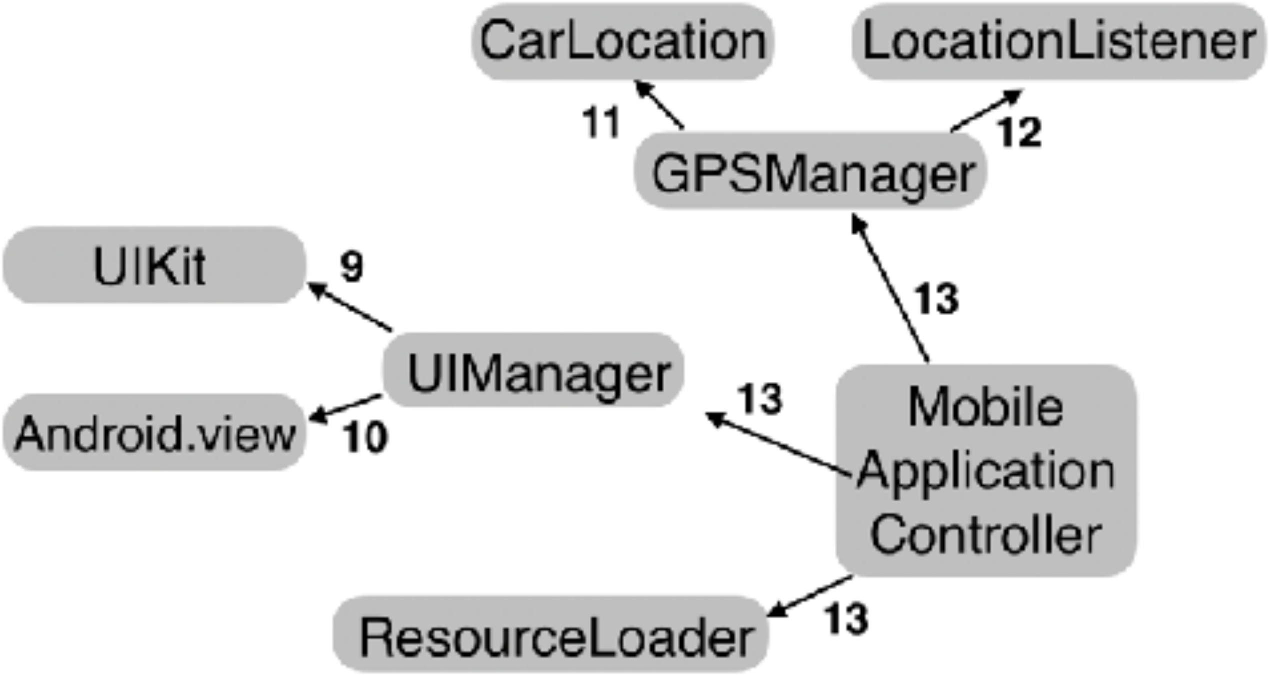
2.4.1 Software Integration Sequence

The components are tested starting from the most independent to the less one. This gives the opportunity to avoid the implementation of useless stubs, because when less independent components are tested, the components which they rely on have already been integrated. The components are integrated within their classes in order to create an integrated subsystem which is ready for subsystem integration.

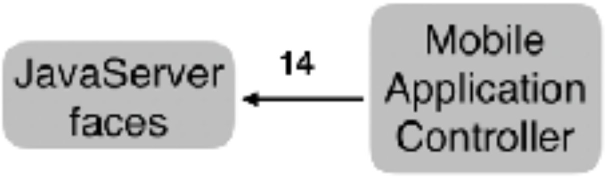
Application Server component integration:



Mobile Application and car computer component integration:



Web Application component integration



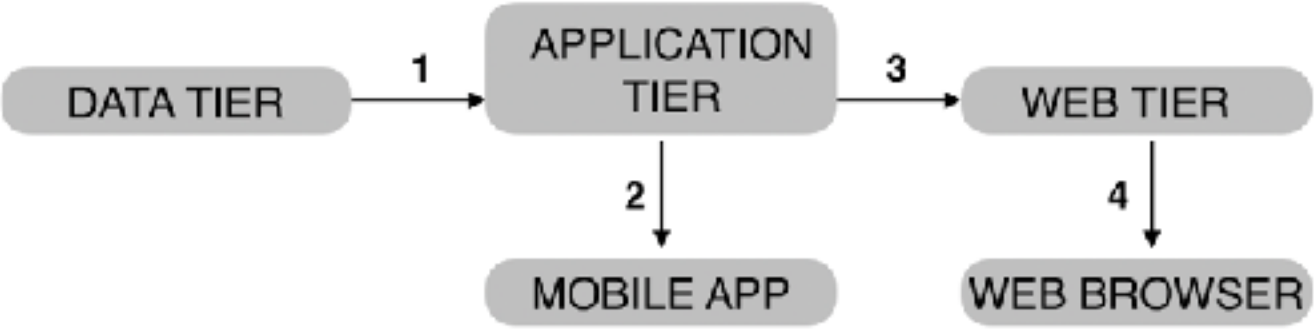
Integration of the system component:

N	SUBSYSTEM	COMPONENT	INTEGRETES WITH
I1	Data, Application tier	UserLogin	DBMS
I2	Data, Application tier	UserRegistration	DBMS
I3	Data, Application tier	Booking	DBMS
I4	Data, Application tier	Reservation	DBMS
I5	Application tier	UserManager	UserLogin UserRegistration DataManager
I6	Application tier	ReservationManager	Booking Reservation UnlockCarManager
I7	Application tier	UtilityManager	EmailSender PaymentManager
I8	Application tier	ApplicationController	UserManager ReservationManager UtilityManager
I9	Mobile	UIManager	UIKit
I10	Mobile	UIManager	Android.view
I11	Mobile	GPSManager	CarLocator
I12	Mobile	GPSManager	LocationListener
I13	Mobile	MobileAppController	UIManager GPSManager ResourceManager
I14	Web	WebAppController	JavaServerFaces

2.4.2 Subsystem Integration Sequence

We made a choice to proceed with the integration process from the server side towards the client applications, integrating the mobile app before the web tier. The reason to do so is that in order to have a functioning client you need to have a working Application tier. The Application tier, instead, can be tested without any client, by making API calls also in an automated fashion. By integrating the mobile application before the web tier, we aim to obtain a fully operational client-server system as soon as possible, since car sharing service can not work without the mobile app. The web tier is less essential and can be integrated after the app.

Directed Acyclic Graph representing the order of integration of the subsystems:



Integration order of the subsystems:

N	SUBSYSTEM	INTEGRETES WITH
SI1	APPLICATION TIER	DATA TIER
SI2	MOBILE APPLICATION	APPLICATION TIER
SI3	WEB TIER	APPLICATION TIER
SI4	WEB BROWSER	WEB TIER

3 INDIVIDUAL STEPS AND TEST DESCRIPTION

This chapter describes the individual test cases to be executed. Each test case is identified with a code and is directly mapped with the two Table for the integration between components and the integration between subsystems. Test cases whose code starts with SI are integration tests between subsystems; test cases whose code starts with I are integration tests between components.

3.1 Integration test case SI1

Test Case Identifier	SI1T1
Test Item(s)	Application tier —> Data Tier
Input Specification	Typical calls to the methods of the JPA Entities, mapped with tables in the Data tier.
Output Specification	The Data tier shall respond by doing the correct queries on the test database. It must also react in the right way both if the requests are made correctly and if they come from unauthorized sources that are trying to access the data.
Environmental Needs	Complete implementation of the Java Entity Beans, Java Persistence API, Test Database, driver that calls the Java Entity Beans.
Test Description	The response will be compared with the expected output of the queries.
Testing Method	Automated with JUnit.

3.2 Integration test case SI2

Test Case Identifier	SI2T1
Test Item(s)	Mobile Application —> Application Tier
Input Specification	Typical API calls to the Application tier (REST API).
Output Specification	The Application tier shall respond accordingly to the API specification. Also, it must react correctly if the requests are malformed or maliciously crafted.
Environmental Needs	Complete implementation of the Application tier; REST API client (driver) that mocks the actual mobile client.
Test Description	The clients should make typical API calls to the application tier; the responses are then evaluated and checked against the expected output. The driver of this test is a standard REST API client that runs on Java.
Testing Method	Automated with JUnit.

Test Case Identifier	SI2T2
Test Item(s)	Mobile Application —> Application Tier
Input Specification	Multiple concurrent requests to the REST API of the application tier.
Output Specification	The business tier must answer the requests in a reasonable time with the applied load.
Environmental Needs	Apache Server, fully developed application tier, Apache JMeter.
Test Description	This test case assesses whether the business tier fulfills the performance.
Testing Method	Automated with Apache JMeter.

3.3 Integration test case SI3

Test Case Identifier	SI3T1
Test Item(s)	Web Tier —> Application Tier
Input Specification	Requests for services offered by the application tier, also invalid ones.
Output Specification	The web tier must call the proper REST APIs or report an error.
Environmental Needs	Apache Server, Web tier
Test Description	This test has to ensure the right translation from HTTPS requests into REST APIs calls, reporting errors when needed.
Testing Method	Automated with JUnit.

Test Case Identifier	SI3T2
Test Item(s)	Web Tier —> Application Tier
Input Specification	Multiple concurrent API calls to the Application tier.
Output Specification	Web requests should be served without problems when a reasonable load is applied on the Application tier.
Environmental Needs	Apache Server, Web tier, Apache JMeter.
Test Description	This test case assesses whether the business tier fulfills the performance
Testing Method	Automated with Apache JMeter.

3.4 Integration test case SI4

Test Case Identifier	SI4T1
Test Item(s)	web browser —> Web tier
Input Specification	Typical and well-formed HTTPS requests from client browser; incomplete, malformed and maliciously crafted requests.
Output Specification	The web tier shall display the requested pages if the requests are valid; if the requests are invalid it shall display a generic error message.
Environmental Needs	Apache Server, fully developed web tier, HTTP client (driver).
Test Description	This test should emulate HTTP requests from typical users of the service and also incorrect requests.
Testing Method	Automated with JUnit.

Test Case Identifier	SI4T2
Test Item(s)	web browser —> Web tier
Input Specification	Multiple concurrent requests to the web server.
Output Specification	Web pages should be served without problems when a reasonable load is applied on the web server.
Environmental Needs	Apache Server, fully developed web tier, Apache JMeter.
Test Description	This test case assesses whether the web tier fulfills the performance
Testing Method	Automated with Apache JMeter.

3.5 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	UserLogin —> DBMS
Input Specification	Typical queries on database tables
Output Specification	The queries return the correct results.
Environmental Needs	Apache server, Test Database, driver for the Java Entity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.6 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	UserRegistration—> DBMS
Input Specification	Typical queries on database tables
Output Specification	The queries return the correct results.
Environmental Needs	Apache server, Test Database, driver for the Java Entity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

4 TOOLS AND TEST EQUIPMENT REQUIRED

5 PROGRAM STUBS AND TEST DATA REQUIRED

6 EFFORT SPENT