Politecnico di Milano

A.A. 2016-2017

Software Engineering 2:

PowerEnJoy

Integration **T**est **P**lan **D**ocument

Lorenzo Frigerio (mat. 879203)

Martina Perfetto (mat. 808869)

Ivan Vigorito (mat. 879204)

15th January 2017

Version 1.0

Contents

1. Introduction

1.1. Purpose and Scope

This document explains a strategy to test all the components of the system and the sequence of integration of them. Its overall purpose is to efficiently test everything and verify functional requirements and the reliability. 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

User / registered user: he/she is the client of the service; he/she is able to rent a car in order to travel around the city. He is associated with:

-Name

-Surname

-Other personal information

-Method of payment

-Number of driving licence and expiration date

-Password

Employee/operator: is the one that help users in case of emergency and has the responsibility of managing cars in case of malfunction. Users can call them by using the telephone exchange of the application.

Method of payment: is inserted by the user during the registration phase but can be updated over the time. Only one method is active at once and payment are concluded using services offered by the different companies holding the credit card. An invoice containing all the charges collected is generated monthly.

Car: sometimes referred as vehicle is the means of transport rented by users. It contains a set of sensors that analyse the number of passengers presents on the car, control the charge of the battery and detect when a door is closed. Moreover, it includes a module that transmit this information to the system using the Internet.

Available car: car that is not in use at the moment by any user, has at least 20% of charge and is not reserved by anyone.

Reservation: made by a user that wants to use a car. Has a duration of one hour maximum and is associated with a unique car. Once the user asks to unlock the car the car becomes associated to the user until he decides to end the ride.

Charge: amount of money that users have to pay due to the use of the service. It is immediately calculated by the system after a ride but money is transferred only at the end of the month.

Penalty: fee derived from a bad behaving of the user such as a damage on the car or a fine for exceeding speed limits. The fee will be notified to the user and included in the monthly invoice.

GPS navigation device: system that equip each car and that is able to calculate the exact position of the car and display to the user the route to follow. Its display is also used to show the current fee of the ride and the status of the battery.

Special Safe Area: special parking areas that contain plugs that allow cars to be recharged. They are provided with sensors that detect the number of spots that are currently used and communicate the number to the system. They are also called power grid stations.

Safe Area: Space included in boundaries that determine where users can park a car. It covers entire metropolitan cities in order to facilitate users to find a park and they may also contain power grid stations. Users cannot terminate a ride while outside from a safe area.

In this document we use ‘Safe Area’ to identify both Safe Area and Special Safe Area.

Ride/Rental: it last from when the user picks up the car until when the system stops charging the user. It includes a possible set of temporary stops and the total path travelled by the user.

Park: is when a user leaves the car and wants to end the rental. At this point the system stops charging the user.

Stop: is when a user leaves the car but wants to resume the ride in the future. The car will be locked by the system that, however, will continue to charge the user for the ride.

1.2.1. Acronyms

SPA: Special Safe Areas

SA: Safe Areas

1.3. References

- Requirements and Specification Document, RASD
- Design document, DD
- IEEE Standards for Information Technology Systems, Design Document

2. Integration Strategy

2.1. Entry Criteria

Before starting the testing process there are several achievements related to the project that need to be performed.

Being the testing phase complex and long, it is important that each low-level component and function has been revised and approved from a code point of view with the code inspection, so as to avoid any kind of possible failure during the tests.

Moreover, the Design Document (DD) and Requirement Analysis and Specification Document (RASD) need to be fully completed and approved. The last one, allows the team to know all the aspects and environments in which our infrastructure will work in, but also the relations and boundaries of the system.

The integration phase should start only when the estimated percentages of completion of modules are:

- Total completion 100% for the Persistence Module
- At least 40% for client tier 1 (user application)
- At least 80% for client tier 2 (car)
- At least 70% for the car management
- At least 60% for the user management

All these percentages are obtained considering several aspects. In order to release a better product, it is important that the core functionalities are fully tested and available. This is the reason, we decided to assign an high grade of completion to some fundamental components. Furthermore, considering the time to fully perform the tests some percentages are also assigned taking into account the order of integration

2.2. Elements to be integrated

The elements that we are going to integrate are the three component which we have already discussed in the Design Document. Each component implements a specific set of functionalities. They are:

- Business logic tier, that is the system main part and has the application logic inside;
- Persistence module, that stores and manages the data;
- Client tier, which is the part of the system that the client interacts with (web and mobile application and the car)
- External part, that is necessary for the system, is composed by sms and mail server for the client notifications and by the payment system.

2.3. Integration Testing Strategy

The integration strategy is based on a bottom-up approach.
This decision comes from the fact that the small components of the infrastructure are tested and gradually integrated. Starting from easiest components increases the robustness and reliability of the system.

The initial components which are chosen to be tested, are selected owing to they don’t depend on other components to function. Different stubs are used during these phases replacing the other subsystems, which are not tested yet. After the testing of all the components that compose a subsystem (in our case, the business logic is built up firstly), the subsystems are tested together.
This strategy of starting from the business logic and after go on with a bottom-up approach allows us to spend more time and efforts upon the core aspects and functionalities of PowerEnJoy infrastructure.

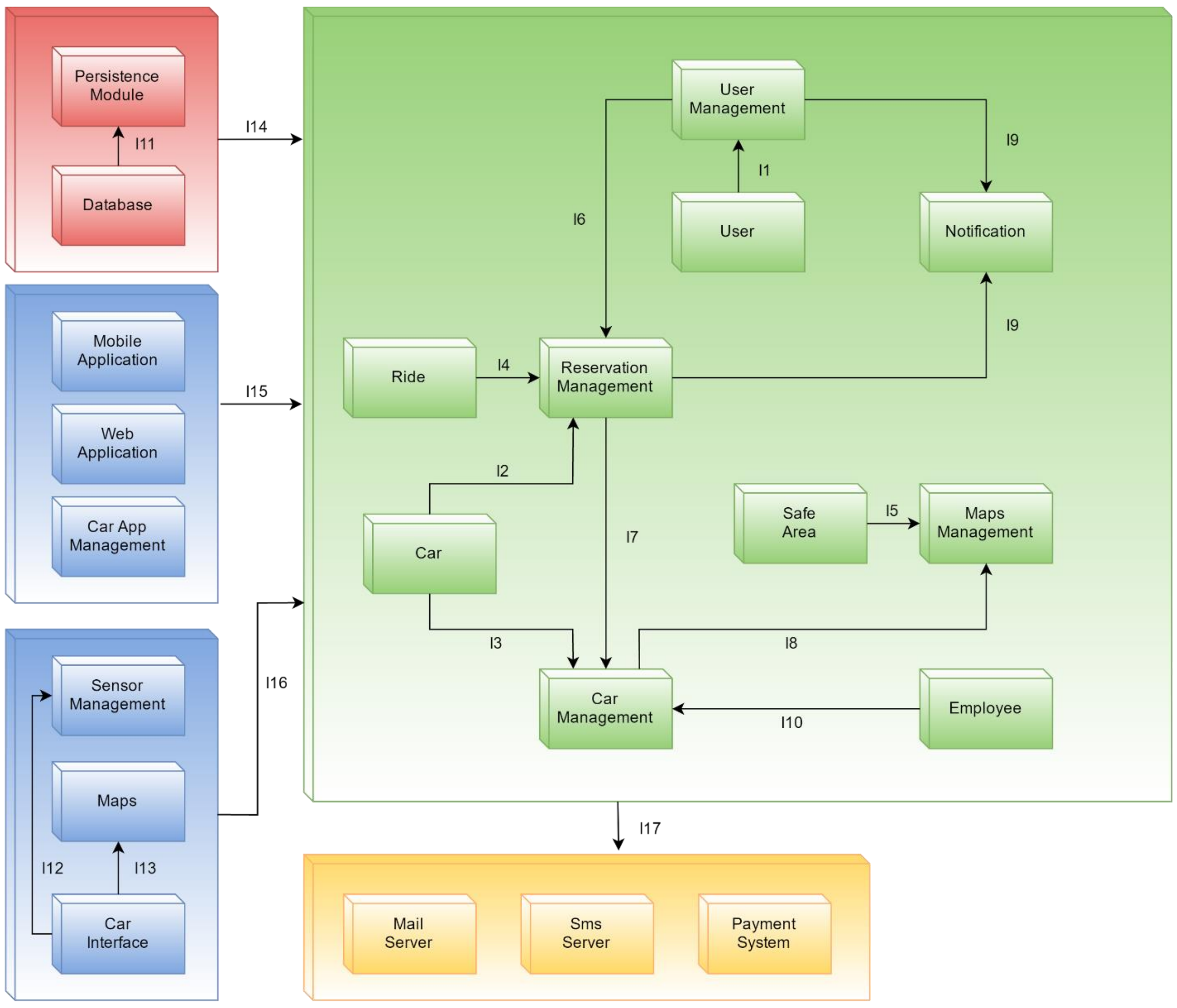
2.4. Subcomponent and Subsystem Integration Sequence

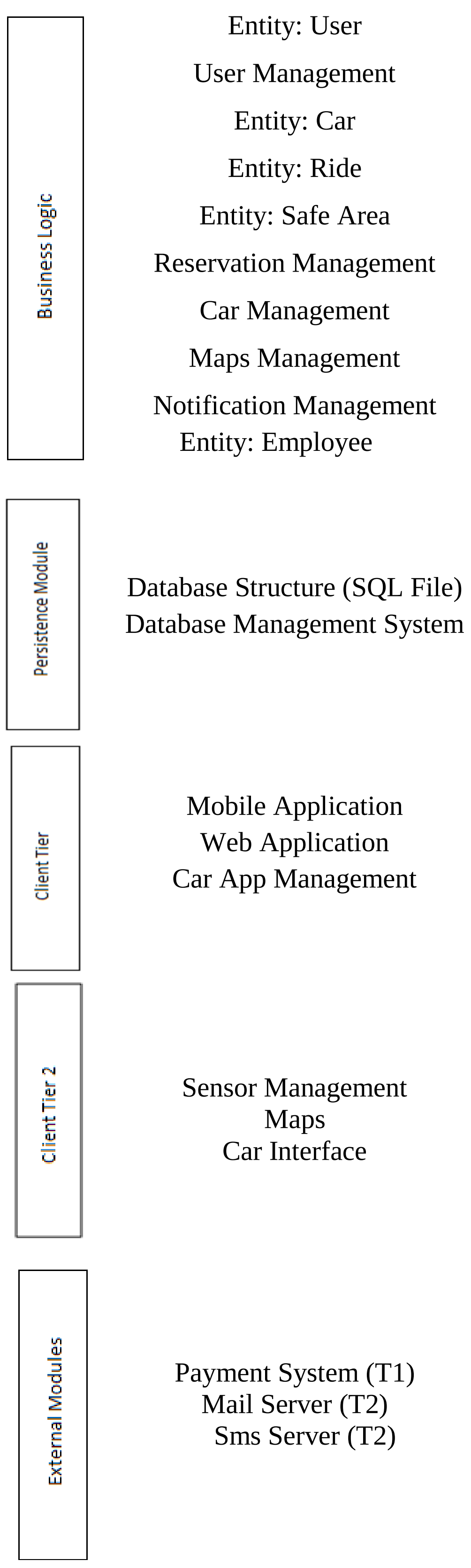
This diagram depicts the overview of all components but also the sequence in which they are integrated one after the other. The integration is based on a bottom-up approach, and the system is integrated starting from the business logic. After the tests inside the business logic, the persistence module is integrated.

Next, the client tiers are connected to the systems, they will create the environment for the system.

After that, external modules (such as mail server, sms server and payment system) are integrated in order to test the functionalities concerning the payment activities and the different notification methods.

It is clear that during the tests inside the business logic, a stub replaces all the services and data provided by the database (more information are provided inside the fifth paragraph).




Business Logic
Entity: User
User Management
Entity: Car
Entity: Ride
Entity: Safe Area
Reservation Management
Car Management
Maps Management
Notification Management
Entity: Employee
Persistence Module
Database Structure (SQL File)
Database Management System
Client Tier
Mobile Application
Web Application
Car App Management
Client Tier 2
Sensor Management
Maps
Car Interface
External Modules
Payment System (T1)
Mail Server (T2)
Sms Server (T2)

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 subcomponents of a subsystem, the next step is to integrate two subsystems among themselves.

3. Individual steps and test description

3.1. Business Logic (B)

Test Case Identifier	B I1 T1
Test Items(s)	User → User Management
Input Specification	Generate typical User traffic
Output Specification	Check User Management
Environment Needs	N/A

Test Case Identifier	B I2 T1
Test Items(s)	Car → Reservation Management
Input Specification	Test different rental activities
Output Specification	Check Reservation Management
Environment Needs	N/A

Test Case Identifier	B I3 T1
Test Items(s)	Car → Car Management
Input Specification exchange	Generate car information
Output Specification	Check Car Management
Environment Needs	N/A

Test Case Identifier	B I4 T1
Test Items(s)	Ride →Reservation Management
Input Specification	Test different type of rides
Output Specification	Verify the different responses
Environment Needs	N/A

Check that each reservation generates a ride.

Test Case Identifier	B I5 T1
Test Items(s)	Safe Area → Maps Management
Input Specification	Test with different safe areas
Output Specification	Check responses
Environment Needs	N/A

The test will be executed with different request on isInsideSafeArea() method. In addition it will be tested that CalculateRoute() generate destinations inside a SafeArea.

Test Case Identifier	B I6 T1
Test Items(s)	User Management→ Reservation Management
Input Specification	Test reservation with valid and not valid users
Output Specification	Check the validity of the results
Environment Needs	I1, I2, I4 test succeeded

Test will be executed over different kinds of users. An exception will be generated in case the user does not have a valid driving licence or in case he has not paid the last monthly invoice.

Test Case Identifier	B I7 T1
Test Items(s)	Reservation Management→Car Management
Input Specification	Different requests performed with all possible combination of status
Output Specification updated	verify that all the status are correctly
Environment Needs	I2, I3, I4 test succeeded

Test will be executed by updating a car status with the different possible status. Exception will be raised in case a car status could not be updated such as when an InRide car is asked to pass to underChargeAvailable directly without passing from Available. In addition other exceptions are raised when the car is asked to open unlockCar(Car car) or to close when the car is not under reservation

Test Case Identifier	B I8 T1
Test Items(s)	Car Management→Maps Management
Input Specification	Invoke functions with valid and invalid parameter
Output Specification	Verify that maps are returned correctly
Environment Needs	I5, I7 test succeeded

The Car Management traffic is composed of requests of calculating a route and isInsideSafeArea(). Exception are raised in case the departure or the destination of a route are not correct or when one of the parameters is missing.

Test Case Identifier	B I9 T1
Test Items(s) →Notification	User Management, Reservation Management Management
Input Specification	SMS and EMAIL requests are checked
Output Specification	Check notification behaviour
Environment Needs	I6, I7 test succeeded

This integration, test the Notification Management to verify the correct behaving of the component under the different requests coming from the User Management and the Reservation Management.

Test Case Identifier	B I10 T1
Test Items(s)	Employee→Car Management
Input Specification	Manage employee' status
Output Specification	Check behaviour
Environment Needs	N/A

The test checks the correct behaving of the Car Management.

3.2. Persistence Module Tests (P)

Test Case Identifier	P I11 T1
Test Items(s)	Database → Persistence Module
Input Specification	Create queries
Output Specification	Check database structure and concurrency
Environment Needs	N/A

Exceptions are raised if the query is invalid or returns no parameter

3.3. Car Test (C)

Test Case Identifier	C I12 T1
Test Items(s)	Car Interface→Car Maps
Input Specification	Different requests to show and update maps
Output Specification	Verify that maps are displayed correctly
Environment Needs	N/A

The test verifies the behaving of car when showing maps and perform requests to calculate the correct route.

Test Case Identifier	C I13 T1
Test Items(s)	Car Interface→Car Sensor Management
Input Specification	Different requests to test all the sensors in different circumstances
Output Specification	Verify that results are returned correctly
Environment Needs	N/A

The test is based on verify the behaving of all the sensors present on the car, such as the one that calculate the number of passengers on the car and the one on the battery level. Exceptions are raised in case one the sensors do not behave correctly

3.4. Subsystems Test (S)

Test Case Identifier	S I14 T1
Test Items(s)	Persistence Module → Business Logic
Input Specification	Different entities to be stored or updated
Output Specification in	verify the presence of the entities form of tuples
Environment Needs	I10 test succeeded

All the different functions provided by the persistence module are tested. Exceptions are raised when the Persistence Module is not able to connect to the DataBase or when the DataBase notify a consistency problem, such the request to register of a user already registered in the application.

Test Case Identifier	S I15 T1
Test Items(s)	Client Tier 1 → Business Logic
Input Specification	Generate typical client traffic
Output Specification	Check the result of the operations
Environment Needs	B block test succeeded

The user traffic is generated in different possible ways: login access, sign-up or making reservations.
The interaction between business logic and the applications could raise exceptions. It happens when the applications make invalid requests, for

example when the user try to make a reservation but he’s not logged. This integration is also critical to verify the quickness of the answers of the business logic.

Test Case Identifier	S I16 T1
Test Items(s)	Client Tier 2 → Business Logic
Input Specification	Generate typical car requests (e.g. get the route)
Output Specification	Check the validity of the result
Environment Needs	B block test succeeded

The test checks the answer provided by the system to the possibly different requests coming from the car. This phase is critically important because of the robustness required for the communication between these two subsystems.

Test Case Identifier	S I17 T1
Test Items(s)	Business Logic →External Application
Input Specification Method	Simulate the use of Payment as external Application
Output Specification payments	Check the correctness of all functionalities related to and invoices
Environment Needs	block B and P test succeeded

The business logic needs to check the compatibility and the functionalities of external applications. In this test the business logic uses some services provided by the payment method. The possible exceptions which may be raised concern a wrong payment method introduced by the user during the sign-up phase (checked with the method checkPayment() described in DD). Another possible exception is raised when the user tries to pay with an outdated / expired payment method or when not enough money are available.

Test Case Identifier	S I17 T2
Test Items(s)	Business Logic → External Application
Input Specification	Send notification through Sms or Mail
Output Specification	Notification sent
Environment Needs	block B and P test succeeded

The business logic has to interact with its registered users. For example, the users, as soon as they send the registration form, receive a password in their mail box. They also receive notification through sms by the business logic. In the case in which a user inserts a wrong email, during the signup form the business logic will not be able to send the registration password and a particular exception will be raised in order to remove this registered user with a wrong mail.

4. Tools and test equipment required

In order to perform the integration some tools are used to test functional and non-functional requirements:

NON-FUNCTIONAL REQUIREMENTS:

- Apache JMeter: It is used to simulate heavy load on a server or object to analyse the performances. In this case the tool is used to test the responses of the application server to an heavy load of requests especially during specific hours of the day.

FUNCTIONAL REQUIREMENTS:

- Mockito: It is the software used to create the three main stubs required and to do unit tests.
- JUnit: It is used for the integrations of subsystems.
- Manual testing: finally, a user simulates the behaving of the systems under the different requests to verify the overall functioning of the system.

The integration testing phase described in this document requires a specific testing environment. Therefore, some equipment is necessary to make the integration phase. As regard to the Business Logic and the Persistence Module a cloud infrastructure is necessary to simulate the real environment where per system is going to be executed. On the other hand, a set of three devices using the three different OS. chosen is necessary to perform the final tests of the performances of the mobile applications. However, an emulator of the three devices is sufficient during the initial testing phase. Finally, a personal computer is used for testing the web application and the Car Management Application of the employees.

5. Program stubs and test data required

Having followed a bottom up approach we reduced as much as possible the necessity of stubs. However, in some circumstances they revealed to be still useful to replace part of the code not yet integrated. During the integration process, in fact, stubs are necessary when:

- Persistence Module: In this case a stub replaces the whole database until I11 that is when the module is integrated with the business logic. Until that moment a stub implemented with the same interface of the Persistence Module replace that component and return dummy results to the requests of the Business Logic. This approach is facilitated by the use of a repository pattern that separates in an easy to manage way the business logic from the retrieval of data.

- External Module:

    - Payment System: A stub is used to replace the payment system until I17 to simulate its functioning during the testing of the business logic. The stub will be pre-written to answer to the requests performed.
    - SMS and Mail Server: A stub replaces this component during the testing of the notification component by simply simulate the sending of mails and SMSs.

5.1. Data required

For each interface, we illustrate all the methods and the data required, followed by the input that can generate errors.

- DB Service

+insertUser(User usr)
+insertReservation(Reservation reserve)
+insertEmployee(Employee empl)
+insertRide(Ride rid)
+insertCar(Car car)
+modifyUser(User usr)
+updateStatus(Status stat, Car car)

– Null object
– Invalid value, for example an employee's matriculation number is a wrong value or the card of the user has expired
– An object User, Reservation, Employee, Ride or Car already exists in the database

- Reservation

+reserveCar(String username, Car car, Date reservation_date, Time reservation_time)
+searchAvailableCar(GPSData local)
+searchAvailableCar(String departure_address)
+unlockCar(Car car, String username, GPSData local_user)

– Null object
– Invalid value, for example a gpsdata correspond with a wrong position or the date and time are in the future, or try to unlock the car when its status is free

- Ride

+beginRide(Car car, Reservation reserv)
+endRental(Car car)

– Null object
– Some incompatible objects, for example the car does not correspond with the reservation's car

- Map Management

  +calculateRoute(String departure, String destination)
  +isInsideSafeArea(Car car, GPSData local_car)

  – Null object
  – Invalid value, like an erroneous gpsdata

- Status

  +lockCar(Car car)
  +unlockCar(Car car)
  +getCarPosition(Car car)
  +updateStatus(Car car, Status stat)

  – Null object
  – Invalid value, for example a wrong status for the car

- User Manager notification

  +sendEmail(String email, String event_type, String user_destination, String password)
  +sendNotification(String username, String event_type, String event_id)

  – Null object
  – Invalid value, like a wrong email
  – Some incompatible objects, the email does not correspond with the user’s email

- Payment Method

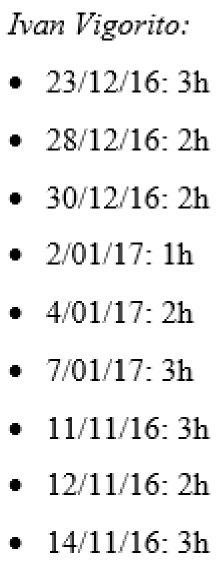
  +checkPayment(Data payment)
  +payInvoice(Data invoice)

  – Null object
  – Invalid value, for example a data in the future

6. Hours of work

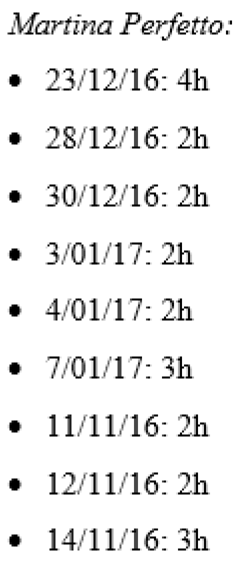
Lorenzo Frigerio:

- 23/12/16: 4h
- 28/12/16: 2h
- 29/12/16: 2h
- 2/01/17: 2h
- 4/01/17: 2h
- 7/01/17: 3h
- 11/11/16: 3h
- 14/11/16: 3h



Ivan Vigorito:

- 23/12/16: 3h
- 28/12/16: 2h
- 30/12/16: 2h
- 2/01/17: 1h
- 4/01/17: 2h
- 7/01/17: 3h
- 11/11/16: 3h
- 12/11/16: 2h
- 14/11/16: 3h



Martina Perfetto:

- 23/12/16: 4h
- 28/12/16: 2h
- 30/12/16: 2h
- 3/01/17: 2h
- 4/01/17: 2h
- 7/01/17: 3h
- 11/11/16: 2h
- 12/11/16: 2h
- 14/11/16: 3h