PowerEnJoy
Integration Test Plan Document
Software Engineering 2 project

POLITECNICO
MILANO 1863

Authors:
Arcari Leonardo
Bertoglio Riccardo
Galimberti Andrea

January 15, 2017

Document version: 1.0

Contents

1 Introduction

1.1 Revision History

- 15/01/2017 - Version 1.0

1.2 Purpose and Scope

Purpose The purpose of this document is to describe the process to be followed in order to perform integration testing of the PowerEnJoy system. It states the elements of the system to be integrated in accordance with the Design Document, which is referred for each component name, semantic and inter-components relations. It defines the approach and strategy to follow in developing software tests by means of integration testing, with the intention of discovering system faults due to the interaction among components. A sequence of components integration is provided, to establish the order of tests development, which come along with a set of test cases. Finally testing supporting tools and test data are described.

Scope The required product is a digital management system for PowerEnJoy, a car sharing service employing exclusively electric vehicles.

The system has to provide functionalities to support both the users of the service and the employees of the company who interact with customers and cars.

Typical functions of a car-sharing service have to be supported, such as reserving a car and finding a parking area where to plug it in at the end of the rent.

A critical issue that has to be considered is the management of the electric vehicles, which have to be continuously recharged as they get used by customers and have to be distributed in a quite uniform way around the town; hence, the user has to be incentivized through discounts to recharge the car in power plugs-equipped parking areas, don't leave the car with a low battery level and park the car where there are few other available vehicles.

The product also has to provide a way to attract more potential customers to the service, therefore sharing the car with other passengers will be incentivized through appropriate discounts. To achieve such functionalities, the system shall provide applications to its users, both customers and employees.

1.3 List of Definitions and Abbreviations

Assistance Operator functionalities Set of functionalities provided by the system involving the Assistance Operator only.

Car availability functionalities Set of functionalities provided by the system to tackle issues related to Car availability, like battery recharging, parking it in a SafeArea or restoring it from a fault.

Car functionalities Set of functionalities provided by the system involving the Car only.

Customer functionalities Set of functionalities provided by the system involving the Customer only.

Customer service functionalities Set of functionalities provided by the system to tackle issues related to Customer service, like handling the inability of the Customer to unlock the Car through the Customer App.

ECU Electronic Control Unit.

Elementary Component An atomic component. That means that from the DD point of view, the component has no sub-components within it.

ITPD Integration Test Plan Document.

Road Operator functionalities Set of functionalities provided by the system involving the Road Operator only.

1.4 List of Reference Documents

- Project description
- Requirements Analysis and Specification Document
- Design Document
- spinGRID ITPD (provided as example)

2 Integration Strategy

2.1 Entry Criteria

In order to begin with the components integration, the following requirements must be met.

- Each elementary component has been unit tested.
- Each third-party component is considered stable, therefore if preconditions hold, the output is the one expected.
- Not all the functionalities have to be definitely implemented, as long as the most critical ones are. See section 2.3.
- The system integration test environment has been configured, is available and is ready for test.
- The appropriate test tools have been configured with system integration test cases for execution.

2.2 Elements to be integrated

As described in the DD, the architecture of the software system is highly hierarchical, thus the integration testing process must follow this structure. Of course, the elements to be integrated are all the components that are described in DD but in this section a logical division of test cases is provided for better understanding of the following sections.

The syntax kept is: *{<component>|<function>} <subcomponent>[, <subcomponent>].* The semantic is: *<component>(or <function>) has <subcomponent>[, <subcomponent>] as sub-components.*

The first step is to test the integration of the *elementary components.* As atomic units of the system, their degree of interaction is high, as they participate in building the functionalities granted by higher-level components. Hence, these are the elements to be integrated at the lowest level of the hierarchy:

Car Application ECUDataFetcher, ECUWriter, CarStatus, Ride, GUI, UIController, CommunicationManager

Car Logic CarRemoteManager, CarStatusManager

Road Operator Logic CarControlManager, AssistanceRequestManager, StatusManager

Assistance Operator Logic AssistanceRequestManager, CarControlManager

Customer Logic CustomerManager, AccessManager, RentalManager

It must be noted that in each of the above list entries *Entities* are missing. Because of the fact they interact with many the components within the *Application Server*, integration testing of them is due, and of course they are taken into account in the integration test sequence in section 2.4

The second step is to test the integration of the *Logic* components. *Logic* components are within the *Application Server* component and together build the business logic of the system.

Car availability functionalities Road Operator Logic, Car Logic

Customer service functionalities Assistance Operator Logic, Road Operator Logic, Car Logic

The third step is to test the integration of client and (application) server software, according to the logical grouping of functionalities:

Car functionalities Car Application, Car Logic

Customer functionalities Customer App, Customer Logic

Road Operator functionalities Road Operator App, Road Operator Logic

Assistance Operator functionalities Assistance Operator App, Assistance Operator Logic

Finally, highest-level components and third party components integration is tested, with the exception of the DBMS. In fact, because of the design choice of adopting JEE application server, database interaction is considered reliable.

Car GPS localization Car Application, Google Maps API

User apps localization Customer App, Road Operator App, Google Maps API

Server GPS localization Car Logic, Google Maps API

Customer payments Customer Logic, Payment Service

2.3 Integration Testing Strategy

In this section the integration testing strategy is explained to clarify why section 2.4 and section §3 are structured that way.

The testing strategy follows the *incremental integration test* approach in a *bottom-up* fashion. Because of the knowledge developed with meetings and talks to draw up the RASD we feel confident enough proceed in a bottom-up way, having defined different levels of abstraction in the DD to split responsibilities among the different components. Therefore it is clear how to structure the sequence of component integration. This approach is also preferred to the top-down one because of the lower amount of scaffolding code needed.

On the top of that, a *critical modules first* strategy is adopted. As a matter of fact, it is clear that the sooner critical modules faults are fixed the less the system will cost. Thus, priority is given to the integration of the *Car and Customer applications* with the *Application Server* as they represent the core business of our system.

2.4 Sequence of Component / Function Integration

Here is reported the sequence of integration tests to be performed to ensure the correct cooperation inside subcomponenets and between subcomponents.

The used notation is to mean that when an arrow goes from component A to component B, then A is required for the functioning of B, hence A is required to be already correctly working and unit tested for B to behave in the desired way.

2.4.1 Software Integration Sequence

Integration Tests of Car Logic

ID	Integration Test
I1	Rental→ CarStatusManager
I2	Car→ CarStatusManager
I3	SafeArea → CarStatusManager
I4	SpecialParkingArea→ CarStatusManager
I5	Car → CarRemoteManager

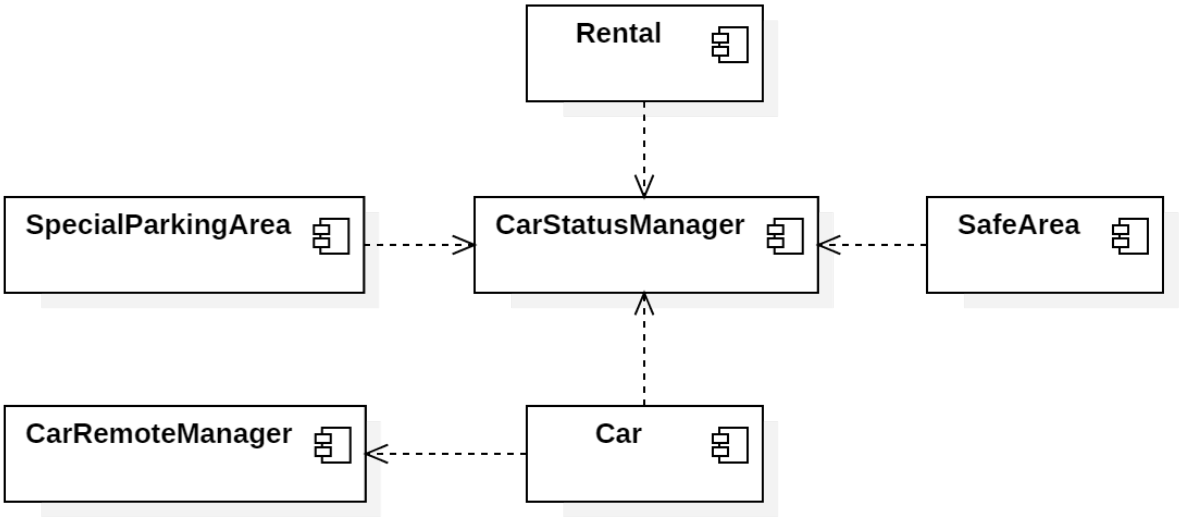

Figure 1: Components of Car Logic

Integration Tests of Car Application

ID	Integration Test
I6	Car ECU → ECUDataFetcher
I7	Car ECU → ECUWriter
I8	CarStatus → ECUDataFetcher
I9	CarStatus → CommunicationManager
I10	ECUWriter→ CommunicationManager
I11	Ride → CommunicationManager
I12	Ride → GUI
I13	CarStatus→ UIController
I14	UIController → GUI
I15	CarStatus→ GUI

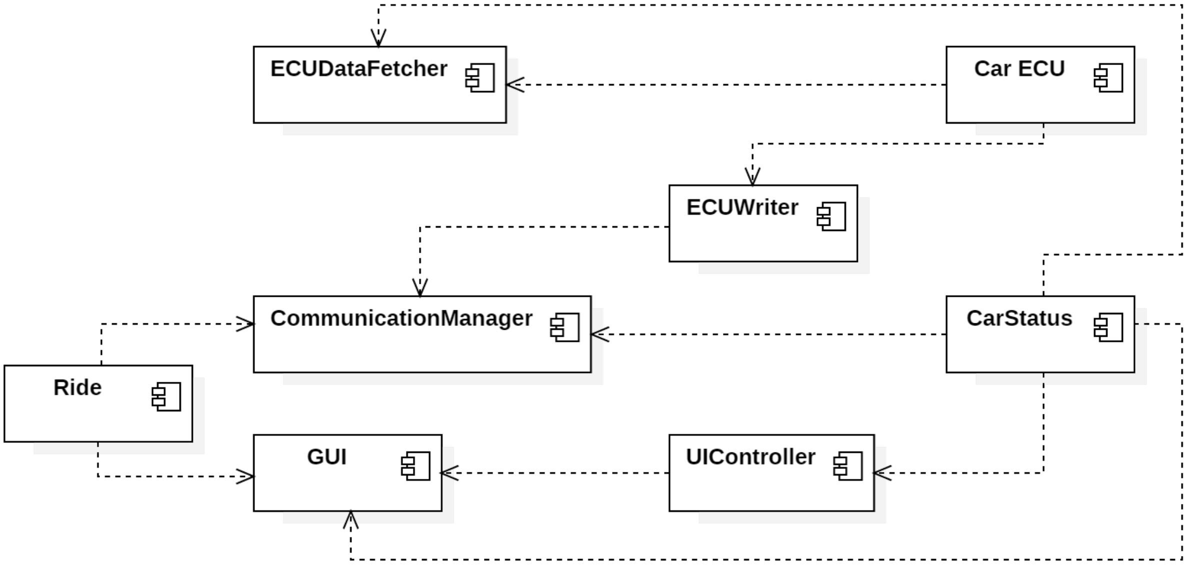

Figure 2: Components of Car Application

Integration Tests of Road Operator Logic

ID	Integration Test
I16	RoadOperator → StatusManager
I17	StatusManager→ AssistanceRequestManager
I18	AssistanceRequest→ AssistanceRequestManager
I19	AssistanceRequestManager → CarControlManager
I20	Car → CarControlManager

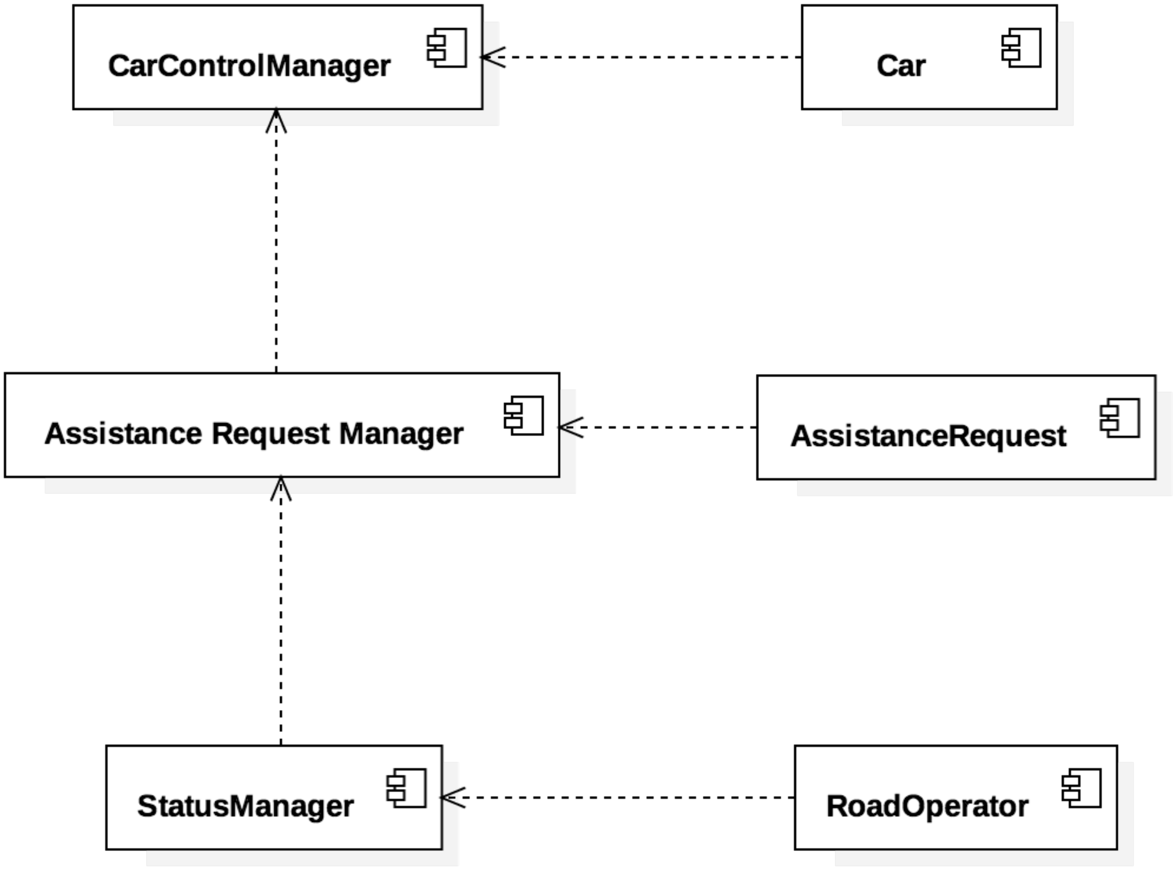

Figure 3: Components of Road Operator Logic

Integration Tests of Assistance Service Operator Logic

ID	Integration Test
I21	AssistanceServiceOperator→ AssistanceRequestManager
I22	RoadOperator→ AssistanceRequestManager
I23	AssistanceRequest→ AssistanceRequestManager
I24	Car→ CarControlManager

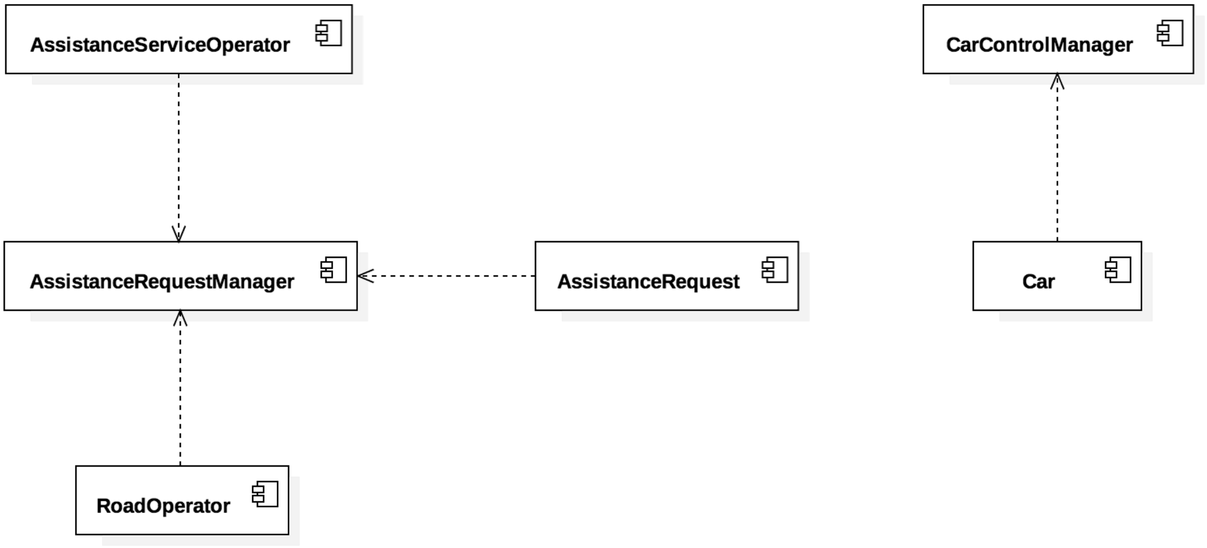

Figure 4: Components of Assistance Service Operator Logic

Integration Tests of Customer Logic

ID	Integration Test
I25	Reservation, Rental, SafeArea, Car → RentalManager
I26	Customer → AccessManager
I27	Customer, AccessManager, RentalManager→ CustomerManager

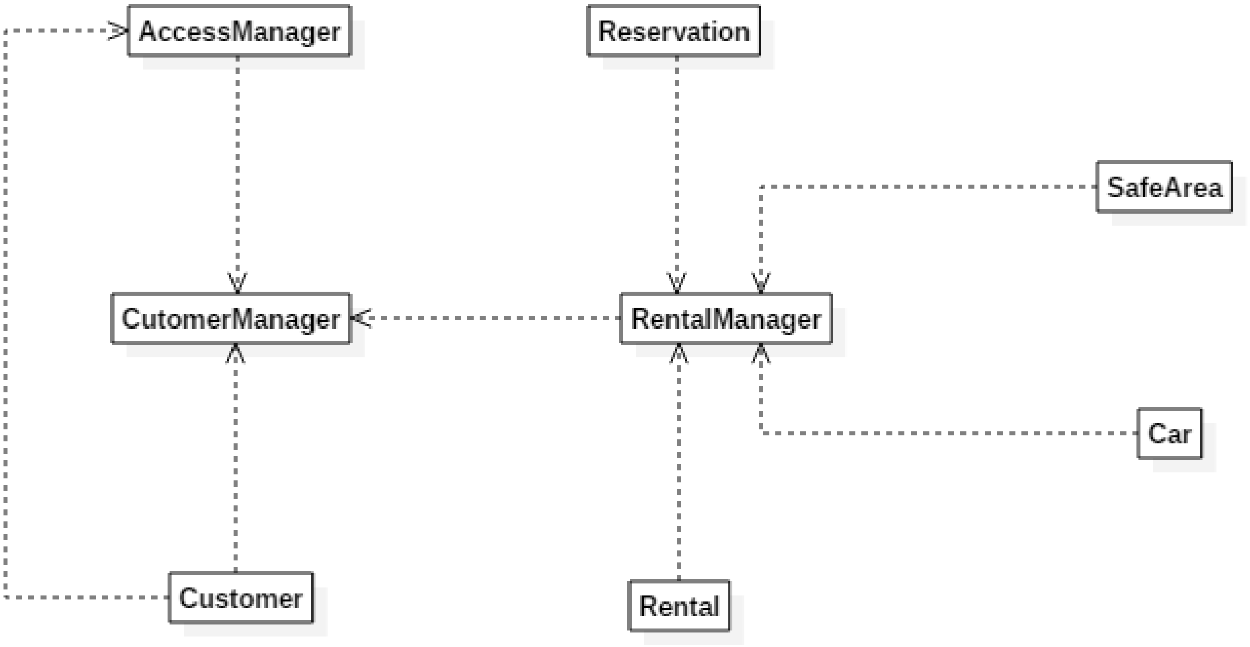

Figure 5: Components of Customer Logic

2.4.2 Subsystem Integration Sequence

Integration Tests of Functionalities

ID	Integration Test
I28	Car Application → Car Logic
I29	Customer App → Customer Logic
I30	Road Operator Application → Road Operator Logic
I31	Assistance Service Operator Application → Assistance Service Operator Logic
I32	Google Maps API→ Car Application
I33	Google Maps API→ Customer Application
I34	Google Maps API→ Road Operator Application
I35	Google Maps API→ Car Logic
I36	Payment Service→ Car Logic
I37	Payment Service→ Customer Logic

3 Individual Steps and Test Description

3.1 Integration test case I1

Test Case Identifier	I1T1
Test Item(s)	Rental→ CarStatusManager
Input Specification	New GPS location message
Output Specification	Rental price is updated coherently with the space driven
Environmental Needs	Car Application sends a CarStatusMessage

Test Case Identifier	I1T2
Test Item(s)	Rental→ CarStatusManager
Input Specification	New number of people on the car count
Output Specification	A ShareARide discount is applied
Environmental Needs	Car Application sends a CarStatusMessage

Test Case Identifier	I1T3
Test Item(s)	Rental→ CarStatusManager
Input Specification	Available ServiceStatus message from the Car
Output Specification	Rental is ended, discounts/overcharges are applied and the Customer is charged for Rental fees
Environmental Needs	Car Application sends a CarStatusMessage

3.2 Integration test case I2

Test Case Identifier	I2T1
Test Item(s)	Car→ CarStatusManager
Input Specification	New LockStatus message
Output Specification	Locking status on Car is updated coherently
Environmental Needs	Car Application sends a CarStatusMessage

Test Case Identifier	I2T2
Test Item(s)	Car→ CarStatusManager
Input Specification	New ServiceStatus message
Output Specification	Service status on Car is updated coherently
Environmental Needs	Car Application sends a CarStatusMessage

Test Case Identifier	I2T3
Test Item(s)	Car→ CarStatusManager
Input Specification	New GPS location message
Output Specification	GPS location on Car is updated coherently
Environmental Needs	Car Application sends a CarStatusMessage

Test Case Identifier	I2T4
Test Item(s)	Car→ CarStatusManager
Input Specification	New number of people on the car count
Output Specification	Number of people count on Car is updated coherently
Environmental Needs	Car Application sends a CarStatusMessage

3.3 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	SafeArea → CarStatusManager
Input Specification	Available ServiceStatus message
Output Specification	Add the Car to the parked cars list of SafeArea whose GPS location matches the Car one
Environmental Needs	Car Application sends a CarStatusMessage

3.4 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	SpecialParkingArea → CarStatusManager
Input Specification	PluggedIn message
Output Specification	Add the Car to the plugged-in cars list of SafeArea whose GPS location matches the Car one
Environmental Needs	Car Application sends a CarStatusMessage

3.5 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	Car → CarRemoteManager
Input Specification	Acknowledge message
Output Specification	Stop remote unlocking routine since unlock message has been received
Environmental Needs	Car Application sends a CarRemoteControl message

3.6 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	Car ECU → ECUDataFetcher
Input Specification	A typical OBD-II message is received from Car ECU bus
Output Specification	The ECUDataFetcher deserializes the message into the correct ADT
Environmental Needs	Car ECU

3.7 Integration test case I7

Test Case Identifier	I7T1
Test Item(s)	Car ECU → ECUWriter
Input Specification	A remote unlock request is made
Output Specification	The Car door unlocks
Environmental Needs	Car ECU

Test Case Identifier	I7T2
Test Item(s)	Car ECU → ECUWriter
Input Specification	A remote lock request is made
Output Specification	The Car door locks
Environmental Needs	Car ECU

3.8 Integration test case I8

Test Case Identifier	I8T1
Test Item(s)	CarStatus→ ECUDataFetcher
Input Specification	An ADT representing some Car status is created
Output Specification	CarStatus is updated coherently with the new status value
Environmental Needs	Car ECU → ECUDataFetcher succeeded

3.9 Integration test case I9

Test Case Identifier	I9T1
Test Item(s)	CarStatus → CommunicationManager
Input Specification	A CarStatus update event is fired
Output Specification	The corresponding CarStatusUpdate message is sent to the Application Server
Environmental Needs	CarStatus→ ECUDataFetcher || CarStatus→ UIController succeeded

3.10 Integration test case I10

Test Case Identifier	I10T1
Test Item(s)	ECUWriter→ CommunicationManager
Input Specification	A CarRemoteControl message is created
Output Specification	The ECUWriter applies the action requested
Environmental Needs	Car ECU → ECUWriter succedeed && Application Server

3.11 Integration test case I11

Test Case Identifier	I11T1
Test Item(s)	Ride→ CommunicationManager
Input Specification	A message with Ride information is created
Output Specification	Ride is updated coherently with received data
Environmental Needs	Application Server

3.12 Integration test case I12

Test Case Identifier	I12T1
Test Item(s)	Ride→ GUI
Input Specification	A Ride update event is fired
Output Specification	GUI is updated coherently with new Ride data
Environmental Needs	Ride→ CommunicationManager succeeded

3.13 Integration test case I13

Test Case Identifier	I13T1
Test Item(s)	CarStatus → UIController
Input Specification	A GUI button interacting with CarStatus is pressed
Output Specification	CarStatus is updated coherently with the action related to the button pressed
Environmental Needs	UIController → GUI succeeded

3.14 Integration test case I14

Test Case Identifier	I14T1
Test Item(s)	UIController → GUI
Input Specification	*Stop* button is pressed
Output Specification	UIController updates CarServiceStatus to *Stopped* in CarStatus
Environmental Needs	Button click simulation

Test Case Identifier	I14T2
Test Item(s)	UIController → GUI
Input Specification	*Resume* button is pressed
Output Specification	UIController updates CarServiceStatus to *Driving* in CarStatus
Environmental Needs	Button click simulation

Test Case Identifier	I14T3
Test Item(s)	UIController → GUI
Input Specification	*End Rental* button is pressed
Output Specification	UIController updates CarServiceStatus to *Available* in CarStatus
Environmental Needs	Button click simulation

3.15 Integration test case I15

Test Case Identifier	I15T1
Test Item(s)	CarStatus→ GUI
Input Specification	A CarStatus update event is fired
Output Specification	GUI is updated coherently with new CarStatus data
Environmental Needs	ECUDataFetcher → CarStatus || UIController→ CarStatus succeeded

3.16 Integration test case I16

Test Case Identifier	I16T1
Test Item(s)	RoadOperator → StatusManager
Input Specification	Request to set the RoadOperator Status to Available
Output Specification	Availability status on RoadOperator is updated coherently
Environmental Needs	StatusManager driver

Test Case Identifier	I16T2
Test Item(s)	RoadOperator → StatusManager
Input Specification	Request to set the RoadOperator Status to Unavailable
Output Specification	Availability status on RoadOperator is updated coherently
Environmental Needs	StatusManager driver

3.17 Integration test case I17

Test Case Identifier	I17T1
Test Item(s)	StatusManager→ AssistanceRequestManager
Input Specification	An AssistanceRequest has been accepted
Output Specification	Status of the RoadOperator set to Unavailable
Environmental Needs	I16 tests succeeded

Test Case Identifier	I17T2
Test Item(s)	StatusManager→ AssistanceRequestManager
Input Specification	An AssistanceRequest has been closed
Output Specification	Status of the RoadOperator set to Available
Environmental Needs	I16 tests succeeded

3.18 Integration test case I18

Test Case Identifier	I18T1
Test Item(s)	AssistanceRequest → AssistanceRequestManager
Input Specification	Acceptance message for an Assistance Request
Output Specification	The AssistanceRequest is definitively assigned to the RoadOperator who accepted
Environmental Needs	AssistanceRequestManager driver

Test Case Identifier	I18T2
Test Item(s)	AssistanceRequest → AssistanceRequestManager
Input Specification	Refusal message for an Assistance Request
Output Specification	The AssistanceRequest is tentatively assigned to another available RoadOperator
Environmental Needs	AssistanceRequestManager driver

Test Case Identifier	I18T3
Test Item(s)	AssistanceRequest → AssistanceRequestManager
Input Specification	Request to close the currently served AssistanceRequest
Output Specification	The corresponding AssistanceRequest is set to Closed
Environmental Needs	AssistanceRequestManager driver

3.19 Integration test case I19

Test Case Identifier	I19T1
Test Item(s)	AssistanceRequestManager→ CarControlManager
Input Specification	Request for the ID of the Car of which an Assistance Request exists
Output Specification	Car ID
Environmental Needs	I17 and I18 tests succeeded

3.20 Integration test case I20

Test Case Identifier	I20T1
Test Item(s)	Car → CarControlManager
Input Specification	Request to lock the car
Output Specification	Locking status on Car is updated coherently
Environmental Needs	CarControlManager driver

Test Case Identifier	I20T2
Test Item(s)	Car → CarControlManager
Input Specification	Request to unlock the car
Output Specification	Locking status on Car is updated coherently
Environmental Needs	CarControlManager driver

Test Case Identifier	I20T3
Test Item(s)	Car → CarControlManager
Input Specification	Request to get the Car Status data
Output Specification	Car Status data is provided back
Environmental Needs	CarControlManager driver

3.21 Integration test case I21

Test Case Identifier	I21T1
Test Item(s)	AssistanceServiceOperator→ AssistanceRequestManager
Input Specification	Request to get informations about the Assistance Service Operator
Output Specification	Informations are provided back
Environmental Needs	AssistanceRequestManager driver

3.22 Integration test case I22

Test Case Identifier	I22T1
Test Item(s)	RoadOperator→ AssistanceRequestManager
Input Specification	Request for a RoadOperator whose Status is Available and in a certain Zone
Output Specification	Identifier of a RoadOperator as requested
Environmental Needs	AssistanceRequestManager driver

3.23 Integration test case I23

Test Case Identifier	I23T1
Test Item(s)	AssistanceRequest → AssistanceRequestManager
Input Specification	Data about a new AssistanceRequest
Output Specification	A new AssistanceRequest is instantiated
Environmental Needs	AssistanceRequestManager driver

Test Case Identifier	I23T2
Test Item(s)	AssistanceRequest → AssistanceRequestManager
Input Specification	Request to close a pending AssistanceRequest
Output Specification	The corresponding AssistanceRequest is set to Closed
Environmental Needs	AssistanceRequestManager driver

Test Case Identifier	I23T3
Test Item(s)	AssistanceRequest → AssistanceRequestManager
Input Specification	A RoadOperator is assigned to an AssistanceRequest
Output Specification	The corresponding AssistanceRequest is coherently updated
Environmental Needs	AssistanceRequestManager driver

3.24 Integration test case I24

Test Case Identifier	I24T1
Test Item(s)	Car→ Car ControlManager
Input Specification	Request to lock the car
Output Specification	Locking status on Car is updated coherently
Environmental Needs	CarControlManager driver

Test Case Identifier	I24T2
Test Item(s)	Car→ Car ControlManager
Input Specification	Request to unlock the car
Output Specification	Locking status on Car is updated coherently
Environmental Needs	CarControlManager driver

Test Case Identifier	I24T3
Test Item(s)	Car→ Car ControlManager
Input Specification	Request to get the Car status data
Output Specification	Car status data is provided back
Environmental Needs	CarControlManager driver

3.25 Integration test case I25

Test Case Identifier	I25T1
Test Item(s)	Reservation → RentalManager
Input Specification	Create typical RentalManager input in order to request Reservation information
Output Specification	Check if the correct methods are called in Reservation component and data retrieved are consistent with those in the database
Environmental Needs	CustomerManager driver

Test Case Identifier	I25T2
Test Item(s)	Rental → RentalManager
Input Specification	Create typical RentalManager input in order to request Rental information
Output Specification	Check if the correct methods are called in Rental component and data retrieved are consistent with those in the database
Environmental Needs	CustomerManager driver

Test Case Identifier	I25T3
Test Item(s)	SafeArea→ RentalManager
Input Specification	Create typical RentalManager input in order to request SafeArea information
Output Specification	Check if the correct methods are called in SafeArea component and data retrieved are consistent with those in the database
Environmental Needs	CustomerManager driver

Test Case Identifier	I25T4
Test Item(s)	Car → RentalManager
Input Specification	Create typical RentalManager input in order to request Car information
Output Specification	Check if the correct methods are called in Car component and data retrieved are consistent with those in the database
Environmental Needs	CustomerManager driver

3.26 Integration test case I26

Test Case Identifier	I26T1
Test Item(s)	Customer → AccessManager
Input Specification	Create typical AccessManager input in order to request Customer information
Output Specification	Check if the correct methods are called in Customer component and data retrieved are consistent with those in the database
Environmental Needs	CustomerManager driver

3.27 Integration test case I27

Test Case Identifier	I27T1
Test Item(s)	Customer → CustomerManager
Input Specification	Create typical CustomerManager input in order to request Customer information
Output Specification	Check if the correct methods are called in Customer component and data retrieved are consistent with those in the database
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T2
Test Item(s)	AccessManager → CustomerManager
Input Specification	Create login data for a Customer
Output Specification	Check that access is permitted when valid data are provided and it is forbidden when the user is not yet registered
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T3
Test Item(s)	AccessManager → CustomerManager
Input Specification	Create new Customer personal information
Output Specification	Check if the user has been inserted in the database
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T4
Test Item(s)	RentalManager → CustomerManager
Input Specification	Create new Reservation data
Output Specification	Check if the reservation has been inserted in the database
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T2
Test Item(s)	RentalManager → CustomerManager
Input Specification	Create new Reservation data
Output Specification	Check if the reservation expires in the due time
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T5
Test Item(s)	RentalManager → CustomerManager
Input Specification	Create new Rental data
Output Specification	Check if the rental has been inserted in the database
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T6
Test Item(s)	RentalManager → CustomerManager
Input Specification	Create a request of unlocking the reserved car
Output Specification	Check if the car has been unlocked
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T7
Test Item(s)	RentalManager → CustomerManager
Input Specification	Create position coordinates
Output Specification	Check if only safe areas near to the position provided are returned
Environmental Needs	Customer App driver, I25 and I26 succeeded

Test Case Identifier	I27T8
Test Item(s)	RentalManager → CustomerManager
Input Specification	Create a safe area position
Output Specification	Check if only cars parked in the requested safe area are returned
Environmental Needs	Customer App driver, I25 and I26 succeeded

3.28 Integration test case I28

Test Case Identifier	I28T1
Test Item(s)	Car Application → Car Logic
Input Specification	A CarStatusUpdate message is created and sent as parameter to a CarStatus API method
Output Specification	CarStatusUpdate message related action is applied to the system
Environmental Needs	An HTTP connection is up and running between Car Application and Application Server

Test Case Identifier	I28T2
Test Item(s)	Car Application → Car Logic
Input Specification	A CarRemoteControl message is created and sent to the Car Application
Output Specification	CarRemoteControl related action is applied to the Car
Environmental Needs	A WebSocket connection is up and running between Car and Application Server

3.29 Integration test case I29

Test Case Identifier	I29T1
Test Item(s)	Customer App → Customer Logic
Input Specification	Retrieved data about safe area position
Output Specification	The data are properly showed to the client
Environmental Needs	An HTTP connection is up and running between the Customer Application and the Application Server

Test Case Identifier	I29T2
Test Item(s)	Customer App → Customer Logic
Input Specification	Retrieved data about a safe area parked cars
Output Specification	The data are properly showed to the client
Environmental Needs	An HTTP connection is up and running between the Customer Application and the Application Server

Test Case Identifier	I29T3
Test Item(s)	Customer App → Customer Logic
Input Specification	Retrieved data about a reservation
Output Specification	The data are properly showed to the client
Environmental Needs	An HTTP connection is up and running between the Customer Application and the Application Server

Test Case Identifier	I29T4
Test Item(s)	Customer App → Customer Logic
Input Specification	Retrieved data about a rental
Output Specification	The data are properly showed to the client
Environmental Needs	An HTTP connection is up and running between the Customer Application and the Application Server

3.30 Integration test case I30

Test Case Identifier	I30T1
Test Item(s)	Road Operator Application → Road Operator Logic
Input Specification	Accept an assistance request by clicking on the apposite button
Output Specification	The status of the assistance request is properly updated server-side
Environmental Needs	An HTTP connection is up and running between the Road Operator Application and the Application Server

Test Case Identifier	I30T2
Test Item(s)	Road Operator Application → Road Operator Logic
Input Specification	An assistance request is assigned to the road operator
Output Specification	The road operator gets notified on the Road Operator Application
Environmental Needs	A WebSocket connection is up and running between the Road Operator Application and the Application Server

3.31 Integration test case I31

Test Case Identifier	I31T1
Test Item(s)	Assistance Service Operator Application → Assistance Service Operator Logic
Input Specification	Inserted data about a new Assistance Request to be opened and clicked the apposite button
Output Specification	The new AssistanceRequest is properly added server-side
Environmental Needs	An HTTP connection is up and running between the Assistance Service Operator Application and the Application Server

3.32 Integration test case I32

Test Case Identifier	I32T1
Test Item(s)	Google Maps API→ Car Application
Input Specification	The Car GPS module detects the current position
Output Specification	The Car position showed on the onboard navigator matches the detected one
Environmental Needs	The Car is connected to the Internet to access the Google Maps service

3.33 Integration test case I33

Test Case Identifier	I33T1
Test Item(s)	Google Maps API→ Customer Application
Input Specification	The Customer device GPS module detects the current position
Output Specification	The Customer position showed on their device matches the detected one
Environmental Needs	The Customer device is connected to the Internet to access the Google Maps service

3.34 Integration test case I34

Test Case Identifier	I34T1
Test Item(s)	Google Maps API→ Road Operator Application
Input Specification	The application detects the current position
Output Specification	The position is correctly updated
Environmental Needs	The application is connected to the Internet to access the Google Maps service

3.35 Integration test case I35

Test Case Identifier	I35T1
Test Item(s)	Google Maps API→ Car Logic
Input Specification	A pair (Current GPS position, Destination GPS position)
Output Specification	A pair (Distance, Traveling time)
Environmental Needs	None

3.36 Integration test case I36

Test Case Identifier	I36T1
Test Item(s)	Payment Service→ Car Logic
Input Specification	A Rental marked as *Ended* having a fee amount > 0
Output Specification	A “Payment process went fine” message
Environmental Needs	A valid and working credit card

3.37 Integration test case I37

Test Case Identifier	I37T1
Test Item(s)	Payment Service→ Customer Logic
Input Specification	Customer payment data are provided
Output Specification	Validity of the given payment data
Environmental Needs	A valid and working credit card

4 Tools and Test Equipment Required

4.1 Tools

4.1.1 Unit Testing

Unit testing involves both state testing (evaluating the consistency of properties inside each single object) and interaction testing (evaluating how the objects interact with each other); for what concerns interaction testing, it will be crucial to have the possibility to define stubs and drivers, enabling us to test functionalities that require not yet implemented classes.

Unit tests are therefore performed by using a combination of JUnit and Mockito; all the unit tests are written and run inside the JUnit framework, while, for the interaction tests, Mockito allows us to create mocks, which can replace real collaborators of a class, thus making it possible to implement some test cases while cooperating classes haven't been coded yet.

4.1.2 Integration Testing

The designed PowerEnjoy service is a distributed service, and it will be therefore implemented by making use of the Java EE framework functionalities.

The server-side components of the system are deployed in containers, which manage the lifecycle (from deployment to activation or instantiation to termination) of the single components, the communication between components and interface them with the low level functionalities offered by the JEE platform.

Hence why testing the components outside of containers is not meaningful enough to validate and verify the whole system; instead, a proper testing of how the application as a distributed service satisfies functional and non-functional requirements is required.

The Arquillian framework will be therefore employed as it allows to execute unit tests against containers (always in combination with the JUnit framework); integration tests will be executed to validate how the macro components of the server-side business logic (modules taking care of the logic behind cars, customers and road and assistance operators) integrate and interact with each other, with the DBMS and with the client applications.

The integration testing obviously requires that the components involved at each step have already successfully passed the unit testing.

4.1.3 Performance Testing

Aside from the validation and verification of the behavior of the implemented against the functional requirements explicated by the customer, it is important to also evaluate how the service will perform when it is deployed, running and used by the real car-sharing users. The JMeter tool will be used to create test plans that enable us to measure the server-side performance of our system, by injecting different load types (heavy load, long-lasting load, when the capacity of the system has reached its saturation point, etc.) and analyzing how the system fares in terms of throughput, number of users served, memory usage, response time, and other metrics.

Performance will also be evaluated on the mobile clients that will be used by the car-sharing users and the operators through the usage of appropriate analytics frameworks.

4.1.4 Manual Testing

Another important factor to consider as we test our system is that it has to effectively and correctly operate in the real environment, therefore an accurate on-field testing reveals itself necessary, and also cars are therefore required in the later part of the testing process.

In an earlier part of the testing process, the car logic and application can be instead simulated via an appropriate software.

Also the applications, both for the customers and the road and assistance service operators, will have to be manually tested on mobile devices to properly evaluate their usability, responsiveness and more in general their performance.

4.2 Testing Equipment

A fundamental component of the required testing equipment for the integration testing is a server infrastructure similar to the one that will be effectively deployed at the start of the real-world PowerEnjoy service; hence, at least a Java Enterprise Runtime, a MySQL Database Management System and a Glassfish Application Server will be needed, to be deployed for testing on a dedicated server.

Other required equipment are mobile devices on which to deploy the customer, road operator and assistance service operator application; the mobile devices could also be simulated, but using real ones is deemed more effective for their immediate availability and their faster configuration process compared to the setup of a simulator.

In order to appropriately cover a reasonably wide pool of devices without incurring into excessive costs, the testing of the applications will be performed on the smartphones and tablets currently having the highest market share, more precisely:

- the 5 most widely used Android smartphones (of any size),
- the 3 most widely used Android tablets (of any size),
- all the iPhones since 2014 (for both the 4.7" and 5.5" sizes),
- all the iPads since 2014 (for the 7.8", 9.7" and 12.9" sizes).

In an earlier part of the testing process, a software simulator on which to deploy the car logic and the car application will also be needed. Meanwhile, in a later part of the testing process, at least one car for each model that will be employed in the PowerEnjoy service will be required, to test both the car logic component interacting with the car's hardware and the car application component which the customer interfaces with through a touch screen display.

5 Program Stubs and Test Data Required

5.1 Program stubs and drivers

Because of the bottom-up approach we only need drivers. Here follows a list of the required drivers:

- Car Logic
  - CarStatusManager driver, mocking a RPC call to the CarRemoteManager in order to text the interaction with Car, SafeArea and Rental entities;
  - CarRemoteManager driver, subscribing the CarRemoteManager module to the Car entity, updating Car entity and sending a message through the Car Remote API according to the update received;
- Road Operator Logic
  - StatusManager driver, mocking a module that makes calls to the StatusManager module in order to stimulate its interaction with the RoadOperator entity;
  - AssistanceRequestManager driver, a testing module calling methods from AssistanceRequestManager that require the intervention of the AssistanceRequest entity and the StatusManager module;
  - CarControlManager driver, having the goal to drive the CarControlManager so that it is possible to test how it integrates with the Car entity and the AssistanceRequestManager subcomponent;
- Assistance Service Operator Logic
  - AssistanceRequestManager driver, mocking a component that makes calls to the AssistanceRequest-Manager component so that it can properly stimulate the AssistanceServiceOperator, RoadOperator and AssistanceRequest entities to allow the evaluation of their correct integration;
  - CarControlManager driver, a module that makes calls to CarControlManager to see how it interacts with the Car entity, e.g. when a request to lock/unlock a car is performed;
- Customer Logic
  - CustomerManager driver: this testing module will call methods of RentalManager in order to verify the interaction between this component and Reservation, Rental, Car, SafeArea entities. Furthermore, it will call methods of AccessManager in order to verify the interaction between this component and the Customer entity;
  - Client driver: this testing module simulates calls from the Customer application sent to the CustomerManager component. The purpose of this driver is to stimulate the interaction between the CustomerManager component and the RentalManager, AccessManager components, Customer entity;
- Car Application
  - CommunicationManager driver: this testing module mocks the Application Server sending a message through Car Remote API and Car Status API, in order to test the integration of CommunicationManager module with CarStatus, Ride and ECUWriter modules;
  - UIController driver: to mock event fired by a User interacting with the GUI and test whether CarStatus is update coherently;
  - ECUDataFetcher driver: this module mocks the Car ECU producing OBD-II messages to test ECUDataFetcher module integration with the CarStatus, verifying whether messages are deserialized coherently;

5.2 Test data

Here follows a list of special test data required for each integration step:

- *I27T2 AccessManager → CustomerManager*
  - Valid data of an existing customer
  - Invalid login data of an existing customer
  - Invalid login data of a not yet registered customer
- *I27T7 RentalManager → CustomerManager*
  - Valid position
  - Invalid position coordinates, i.e., a location outside the service area
- *I27T8 RentalManager → CustomerManager*
  - Valid safe area position
  - Invalid position of a safe area that doesn't exist
- *I28T1 Car Application → Car Logic*
  - Empty message
  - Empty message fields
  - Very unlikely GPS status updates (very far from the previously received ones)
  - Fields values out of their domain
  - Unexpected messages (like a remote lock message while Car is driving)
- *I30T1 Road Operator Application → Road Operator Logic*
  - Null request
  - Fields values out of their domain
- *I30T2 Road Operator Application → Road Operator Logic*
  - Non-existing RoadOperator assigned to a request
  - Non-existing request
  - Null request
- *I31T1 Assistance Service Operator Application → Assistance Service Operator Logic*
  - Null request
  - Fields values out of their domain
- *I37T1 Payment Service→ Car Logic*
  - Negative fees
  - Unlikely fees (orders of magnitude higher than mean ones)
  - High frequency payment requests

6 Effort Spent

Arcari Leonardo

- 29/11/2016 - 2h
- 03/01/2017 - 5h
- 05/01/2017 - 4h
- 15/01/2017 - 2h

Bertoglio Riccardo

- 03/01/2017 - 2h
- 08/01/2017 - 2h
- 10/01/2017 - 1h
- 12/01/2017 - 3h
- 13/01/2017 - 30min
- 15/01/2017 - 1:30h

Galimberti Andrea

- 29/12/2016 - 2h
- 04/01/2017 - 2h
- 06/01/2017 - 1h
- 07/01/2017 - 3h
- 10/01/2017 - 2h
- 11/01/2017 - 2h
- 14/01/2017 - 3h
- 15/01/2017 - 1h