Integration Test Plan Document

Simone Mosciatti & Sara Zanzottera

January 15, 2017

Contents

1 Introduction

1.1 Purpose

[New purpose]

1.2 Scope

PowerEnJoy is a digital management system for car sharing that exclusively employs electric cars to provide its service. The system provides all the functionalities normally provided by a car sharing service: registering to the service, find the location of nearby available cars, reserve cars up to a short amount of time, unlock the chosen car once found, ride it and then park it in a safe area, when it will be automatically locked and the fee paid.

In addition, the system gives bonuses and penalities in term of discounts or over-prices depending on the behavior of the user, in order to promote virtuous behaviors.

PowerEnJoy is therefore a inherently distributed system, based on a central server interactions with many distributed nodes. All these components will be examined in more detail in the subsequent sections of the document.

1.3 Definitions, Acronyms, Abbreviations

RASD Requirements and Specification Document.

DD Design Document.

User A customer of PowerEnJoy using the service.

Staff Operator (Operator) An employee of PowerEnJoy which takes care of the cars.

Ride The action of getting onboard of a PowerEnJoy car, start its engine, drive to destination and park.

Issue Any problem a car may incur in, or a user may face while using the service.

Nearby Cars Available cars located within a maximum distance to a specific position.

Available Cars Cars whose Availability Status is set to "Available".

Nearby Issues Issues that are affecting cars close to a specific position.

Booking (Reservation) The act to reserve a car for a limited amount of time for future use by a user.

Driver Whoever is driving a regularly booked PowerEnJoy car.

Driving License The state's issued driving license of the user.

Notification A form of comunication where the user is actively notified of some event.

Issue Report An incoming notification that states a car incurred in an issue.

Fine A fine issued by the local law enforcing officers to a user while driving a PowerEnJoy car.

Pending Bills Bills that an user still need to pay to PowerEnJoy .

Safe Area An parking area, predefined by the company, where is possible to safely park the cars of the PowerEnJoy fleet.

Battery Charge The amount of charge that is kept inside the car's battery.

Charging Station (Power Station) Dedicated areas where is possible to plug the PowerEnJoy cars to charge their batteries.

Car's Onboard System The controll system of the car that is able to exchange data with the central system and to relevate operation parameters.

Customer's App An implementation of the system frontend tailored to the need of the customers.

Staff's App An implementation of the system frontend tailored to the need of the staff.

Central System (Main Server) The central system for PowerEnJoy . All the command and all the data are streamed, analyzed and used here.

GPS : Global Positioning System is a global navigation satellite system (GNSS) that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

Location Pair of integer values as provided by GPS sensors.

Payment Method Set of data relative to a credit card.

Email address (Email) Unique string identifiying an email box to which email messages are delivered.

Identity ID Personal code provided by local authorities to uniquely identify citizens.

Driving License ID The unique code reported on every legal driving license.

Session Key A string representing a key for a secured channel. Used to secure communications between the server and the nodes.

Scanned License An high quality image of the driving license acquired by the car's onboard system.

1.4 Reference Documents

- *Assignments AA 2016-2017.pdf* (Assignments document given by the teacher)
- *Reqiurements And Specification Document* (referring to this project)
- *Design Document* (referring to this project)

2 Integration Strategy

We believe that the integration of different components should not be a completely different phase from the development of each component.

Integration should happen as soon as possible even with mock implementation of the components. Such early integration helps in identifying possible inconsistencies in the design documents: the sooner those inconsistencies are discovered, the simpler it is to fix them.

2.1 Entry criteria

As long as the integration tests are automatic and fast, we do not advocate for any strong or specific precondition.

On the other side we advocate to run the integration test after each commit. This procedure helps in catching regressions as early as possible.

Unit tests should be used in order to guarantee the interface exposed and documented are respected, and that the component runs in isolation, fulfilling its assigned functionalities.

2.2 Elements to be integrated

All the components that interact with the user must be integrated with the User FrontEnd. Similarly all the components that need to interact with the Staff need to be integrated with the Staff FrontEnd.

Also, the GEOLOCATION component needs to be integrated with the Car GUI.

Concerning external services, only a few components need to interact with them. In particular we need to integrate the BILLING_SYSTEM, POSITION, GEOLOCATION, USER_MANAGER with external services.

Finally, only a few services must be integrated between each other in the system, thanks to the high degree of decoupling reached during the design phase.

The services that must be integrated inside the system are the following:

IT01

Component1 : GEOLOCATION

Component2 : POSITION

Functionality : GEOLOCATION/AvailableCar

Description : The position of each car, retrieved using POSITION/Car, is used in order to describe which cars are close to a specific position.

IT02

Component1 : GEOLOCATION

Component2 : POSITION

Component3 : CAR_MANAGER

Functionality : GEOLOCATION/Issues

Description : The position of each car, retrieved using POSITION/Car, that has some issues (detectable using CAR/Telemetry) is used to communicate to the staff where are cars that need their intervent.

IT03

Component1 : RIDE_MANAGER

Component2 : CAR_MANAGER

Functionality : RIDE/Start, RIDE/End

Description : When a ride starts (ends), detected using CAR/Telemetry, the RIDE/Start (RIDE/End) method is invoked.

IT04

Component1 : BILLING_MANAGER

Component2 : CAR_MANAGER

Functionality : BILL/Pay

Description : When the car is locked, the payment procedure of the user starts.

IT05

Component1 : BILLING_MANAGER

Component2 : BOOKING_MANAGER

Functionality : BILL/CalculateExpireBookFee

Description : When a booking expires, detected using BOOKING/Expire, the user should be asked to pay the fee calculated using BILL/CalculateExpireBookFee

IT06

Component1 : BILLING_MANAGER

Component2 : RIDE_MANAGER

Component3 : POSITION

Component4: GEOLOCATION

Functionality : BILL/CalculateUnsafeParkingFine

Description : When the user ends a ride, detected using RIDE/End, and the car is unsafely parked, detected using POSITION/Car and GEOLOCATION/IsSafeArea, the user is prompted to pay for the ride plus the fee for having parked the car in an unsafe position.

IT07

Component1 : BILLING_MANAGER

Component2 : CAR_MANAGER

Functionality : BILL/CalculateRideFee

Description : After a ride is concluded, the amount of the fee the user is prompted to pay is determinated on the base of several bonuses that may be applied. To determinate which bonuses apply, CAR/Telemetry is used.

2.3 Integration testing strategy

We chosed, basically, to implement a bottom-up approach to our integration testing strategy.

Actually, the only components that requires a sub-components integration are the three services that are physically distributed. Once integrate, we can proceed to a higher-level integration reasoning about them in terms of a simpler logical component.

1. CAR_MANAGER: this component needs its own integration. It is distributed between the car system and the main server; it is also the only components that touches the hardware side of the project. In our opinion it should be the first component to be integrated.

2. ISSUE_MANAGER: another component that needs its own integration. However its functionality is more bounded that the one provided by CAR_MANAGER.

3. POSITION: this component needs it own integration too. However, its functionality is pretty standard and well known.

In parallel we can proced to integrate the external services with our own high-level components, namely BILLING_SYSTEM , POSITION , GEOLOCATION, and USER_MANAGER.

2.4 Sequence of components integration

Our design tried to keep the components very decoupled. This gives us a lot of flexibility in the integration process.

There are not hard constraints in what component should be integrated first, as long as sub-components get integrated into higher-level components before proceeding with the integration of high-level ones.

Of course all the components needs to be integrated with the user interface. We are not showing all those integrations, that are extremely simple but necessary. Since the NOTIFIER components is part of the user interface, we are not showing the integrations that concern this component.

Before going deeply inside the description of the high-level integrations, we describe the sub-component integrations required.

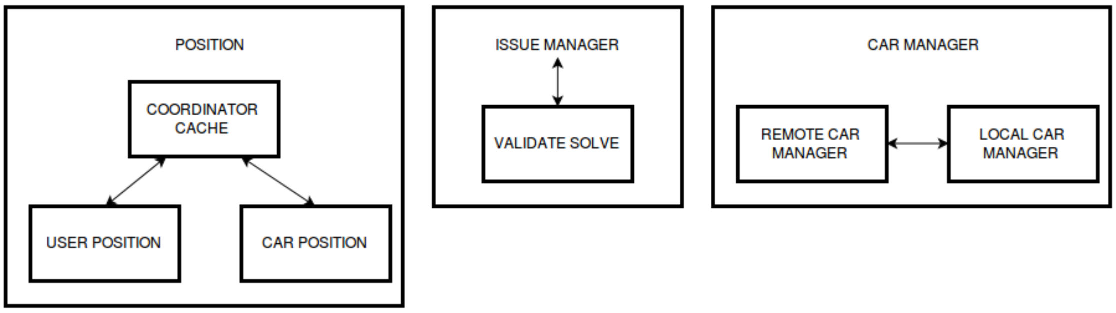

Figure 1: Sub-components integration regarding physically splitted components into a single, high-level logical component.

In the following diagram we show what components integrates with others. There are not strict dependencies: it is more a net that a pyramid.

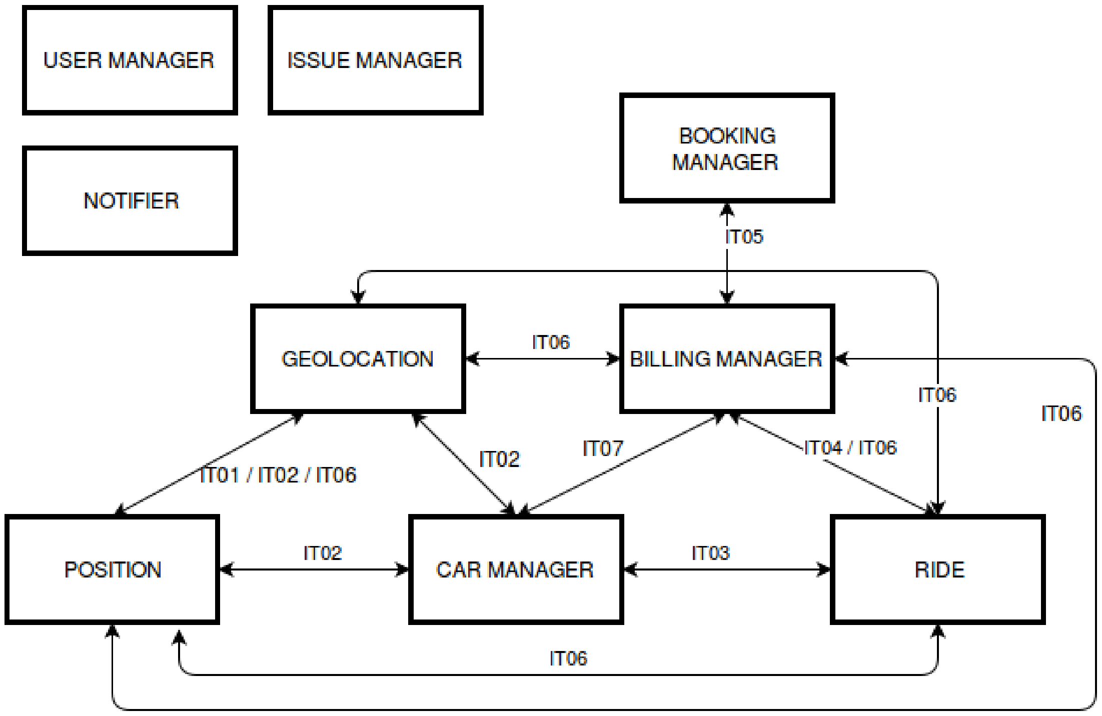

Figure 2: Integration net

3 Individual Steps and Tests Description

In this section we describe what test are necessary for each integration.
For each integration we list and describe the most important tests.

Integration IT 01

1. **Test Case ID: IT01C1**

   Test Items: GEOLOCATION → POSITION

   Input Specification: A circle including the point A is submitted to GEOLOCATION/AvailableCar

   Environment State: An available car is located at coordinates A

   Output Specification: A list of cars including the car located in A

   Purpose: Verify the output given by GEOLOCATION/AvailableCar in case of at least one car available in the selected area

   Dependencies: Unit tests for GEOLOCATION and POSITION

2. **Test Case ID: IT01C2**

   Test Items: GEOLOCATION → POSITION

   Input Specification: A circle centered in C with radius equal or less than R is submitted to GEOLOCATION/AvailableCar

   Environment State: No cars are available into the circle centered in C with radius R

   Output Specification: An empty list

   Purpose: Verify the output given by GEOLOCATION/AvailableCar in case of no cars available in the selected area

   Dependencies: Unit tests for GEOLOCATION and POSITION

Integration IT 02

1. **Test Case ID: IT02C1**

   Test Items: GEOLOCATION → POSITION ← CAR_MANAGER

   Input Specification: A circle including the point A is submitted to GEOLOCATION/Issue

   Environment State: A car with an issue is located at coordinates A

   Output Specification: A list of cars including the car located in A

   Purpose: Verify the output given by GEOLOCATION/Issue in case of at least one car issued in the selected area

   Dependencies: Unit tests for GEOLOCATION, POSITION and CAR_MANAGER

2. **Test Case ID: IT02C2**

   Test Items: GEOLOCATION → POSITION ← CAR_MANAGER

   Input Specification: A circle centered in C with radius equal or less than R is submitted to GEOLOCATION/Issue

   Environment State: No cars with issues are located into the circle centered in C with radius R

   Output Specification: An empty list

   Purpose: Verify the output given by GEOLOCATION/Issue in case of no cars with issues present in the selected area

   Dependencies: Unit tests for GEOLOCATION, POSITION and CAR_MANAGER

Integration IT 03

1. **Test Case ID: IT03C1**

   Test Items: CAR_MANAGER → RIDE_MANAGER

   Input Specification: The signal meaning that the engine ignited is sent to CAR_MANAGER

   Environment State: All the precondition for a ride to start are fulfilled (a non-expider booking exist for that car, the driver scanned successfully the correct driving licence, etc)

   Output Specification: A Ride is started successfully.

   Purpose: Verify the behavior of RIDE/Start

   Dependencies: Unit tests for CAR_MANAGER and RIDE_MANAGER

2. **Test Case ID: IT03C2**

   Test Items: CAR_MANAGER → RIDE_MANAGER

   Input Specification: The signal meaning that the driver exited is sent to CAR_MANAGER

   Environment State: A ride for that car is open.

   Output Specification: The relative ride is closed successfully

   Purpose: Verify the behavior of RIDE/End

   Dependencies: IT03C1 and unit tests for CAR_MANAGER and RIDE_MANAGER

Integration IT 04

1. **Test Case ID: IT04C1**

   Test Items: CAR_MANAGER → BILLING_MANAGER

   Input Specification: The locking signal is received by CAR_MANAGER

Environment State: A ride for that car exist and is closed.

Output Specification: The payment procedure starts.

Purpose: Verify that after the locking signal has been sent, the payment procedure starts immediately.

Dependencies: IT03C2 and unit tests for CAR_MANAGER and BILLING_MANAGER

Integration IT 05

1. **Test Case ID: IT05C1**

   Test Items: BOOKING_MANAGER → BILLING_MANAGER

   Input Specification: A reservation expires.

   Environment State: A reservation exist, but no rides are related to it.

   Output Specification: The fine for the expired booking is calculated and the payment procedure starts.

   Purpose: Verify that when a reservation expires, the correct fine is calculated and the payment procedure starts.

   Dependencies: Unit tests for BOOKING_MANAGER and BILLING_MANAGER

Integration IT 06

1. **Test Case ID: IT06C1**

   Test Items: {GEOLOCATION → POSITION } ← RIDE_MANAGER → BILLING_MANAGER

   Input Specification: A ride ends.

   Environment State: The car whose ride is ending is located outside from a safe area.

   Output Specification: The fine for the unsafe parking is calculated and the payment procedure starts.

   Purpose: Verify that when a car is parked outside from a safe area, the correct fine is calculated and the payment procedure starts.

   Dependencies: IT03C2 and unit tests for GEOLOCATION, POSITION, RIDE_MANAGER and BILLING_MANAGER

Integration IT 07

1. **Test Case ID: IT07C1**

   Test Items: CAR_MANAGER → BILLING_MANAGER

   Input Specification: A car is locked.

   Environment State: A ride and a booking exsist relative to that car and the user who locked it.

Output Specification: The fee is calculated and the payment procedure starts.

Purpose: Verify that when a car is locked, the correct fee is calculated and the payment procedure starts.

Dependencies: IT03C2 (which is an included procedure) and unit tests for CAR_MANAGER and BILLING_MANAGER

These tests are to be considered as guidelines. We described only the most straightforward test that must be implemented; however, other tests can be deployed too.

Usually it is a good idea to test not only that a method is invoked and used, but also that a method is not being invoked. We do not specifically describe those tests because they are extremely implementation dependent.

Developers should take into consideration if such kind of tests are useful during the testing phase, and if necessary, implement them.

4 Tools and Test Equipment Required

Any tools or framework works just as well in this project: our suggestion is to use a well know libraries and framework.

A very reasonable choice is **JUnit4**, which is the default testing framework for Java application.

4.1 Continuos Integration and Unit Testing

A server of continuos integration is necessary in order to execute the integration tests and the unit tests after each commit to the main code base.

Each developer should still be able to run every test in its developing machine.

The unit test suite is expected to be extremelly fast so that it can be executed at will by every developer without slowing down their workflow.

For the integration test, we don't require them to be fast, although is still a very desiderable property. But they should be, at very least, completely automatic.

4.2 Testing Strategy

We do not advocate for rigid methodology. Test should be produced in order to avoid regression.

After fixing a bug, a test relative to that same bug should be put in place and should be continuosly run.

4.3 Deploying

The deploy of the whole architecture should require the least amount of steps possible, ideally just one.

Considering the application is serving live traffic, a deploy of a new version should be done following the incremental GREEN / BLUE methodology: start running the new software only a small fraction of the overall traffic to the new version and, if everything on the control metrics still looks fine, increase the amount of traffic redirected to the new version until reaching the totality. Immediately roll back to the previous version at the first sign of problems.

5 Program Stubs and Test Data Required

5.1 Stubs

At this time, no stubs are strictly required to run our integration test suite.

Concerning the integration of our system with external services, we noticed that most interactions can be covered with the unit tests of single components: even in these cases, stubs are not required as all our external services expose some specific testing APIs that can be used for this purpose.

5.2 Test Data

The necessary test data will be generated using ad-hoc software.

There should be two different databases: one for tests and another for production. If it is necessary to simulate the load of the production database, we will not directly use it, but we populate the test database.

5.3 Hardware Emulation

Running integration tests using a real car may be difficult. That's why an emulation environment for the car's onboard system should be enough to run all our integration tests: this solution makes running tests faster and easier.

Also, the developing time of the hardware is usualy longer than the developing time of software, and relying on the actual hardware for running integration test may lead to delays or to skip the run of fundamental integration tests: a situation we would prefer to avoid.

6 Conclusions

6.1 Tools used

During the development of this document we used the following tools:

- **Github** to version control the project
- **LaTeX** on TeXworks to redact this document

6.2 Hours of work

- SZ: 1h on 21/12
- SM: 3h on 23/12
- SM: 5h on 27/12
- SM: 4h on 3/01
- SM: 6h on 7/01
- SZ: 2h on 11/01
- SZ: 3h on 14/01
- SM: 1h on 15/01