PowerEnJoy
Software Engineering II

Integration Test Plan Document

Giovanni Scotti, Marco Trabucchi

Document version: 1
January 2, 2017

Contents

Section 1

Introduction

1.1 Purpose and Scope

1.1.1 Purpose

The Integration Test Plan Document (ITPD) is intended to provide the guidelines to accomplish the integration test phase planning in sufficient detail. This also includes determining which tools are needed and will be used during the testing process itself, as well as the required stubs, drivers and data structures that will be useful during said process.

1.1.2 Scope

PowerEnJoy is a car sharing service that only employs electric vehicles; it is provided for a large city, and aims to support the sharing process and car management of the electric cars, as well as the booking and payments for the service itself.

1.2 Definitions, Acronyms, Abbreviations

RASD: Requirements and Specification Document.

DBMS: DataBase Management System.

DD: Design Document.

ITPD: Integration Test Plan Document.

JEB: Java Entity Bean.

SB: Session Bean.

1.3 Reference Documents

The indications provided in this document are based on the ones stated in the previous deliverables for the project, the RASD document [1] and the DD document [2].

Moreover it is strictly based on the test plan example [4] presented during lectures and on the specifications concerning the RASD assignment [3] for the Software Engineering II project, part of the course held by professors Luca Mottola and Elisabetta Di Nitto at the Politecnico di Milano, A.Y. 2016/17.

Section 2

Integration Strategy

2.1 Entry Criteria

This section expresses the prerequisites needed to be met before the integration phase takes place.

Documentation: The documentation for every method and class must be provided for each individual component, in order to make it easier to reuse classes and understand their functioning; this is in fact also a prerequisite for the unit tests to be performed before the integration test phase. When necessary, a formal language specification of the classes' behaviours can be used (such as JML - Java Modelling Language).

Unit tests: All the classes and methods must be tested thoroughly using JUnit, in order to assure a properly correct behaviour of the internal mechanics of the individual components. It is required that the test coverage of each class and package reaches 90% of the code lines; moreover, test cases must be written with continuity and executed at every consecutive build of the project: this is needed in order to ensure that newly added lines do not interfere with the stability of the rest of the code.

Code Inspection and Analysis: Both automated data-flow analysis and code inspection must be performed on the whole project classes. This will reduce the risk that, during the integration test phase, any code-related issues or bugs rise, leading to more complex problematic situations to be solved in latter phases of the project development, with much greater effort for the development team.

RASD and DD: Along with the indications provided in this very document (ITPD), the two previous documents for this project, RASD and DD, must be delivered before the integration test phase can begin.

2.2 Elements to be Integrated

The integration test phase for the *PowerEnJoy* system will be structured based on the architectural division in tiers that is described in the Design Document [2], as well as the indication of the elements of which said subsystems are composed of.

With respect to this, the subsystems to be integrated in this phase are the following four:

Database Tier This includes all the commercial database structures that will be used for the data storage and management of the system, namely the DBMS and the Database Engine; the two data layer components are already developed to work properly when coupled together, so the only component to be integrated is the DBMS.

Application Logic Tier This includes all the business logic for the application, the data access components and the interface components towards external systems and clients. All the interactions among internal logic components must be tested and all the subsystems that interact with this tier must be individually integrated.

Web Tier This includes all the components in charge of the web interface and the communication with the application logic tier and the browser client. The integration tests must be performed both ways for this tier, and the Web Controller must be thoroughly tested also for the interaction with the Java Server Pages component.

Client Tier This includes the various types of clients, which is to say the Mobile Application Client, the Web Browser Client and the On-Board Application Client, and their internal components. Single clients must behave properly with respect to their internal structure, and must be individually integrated with the tier they interface with.

The integration process will be performed in two steps:

- A *first phase* in which the individual components of the subsystems (i.e. Java classes, Java Beans and Containers), are integrated one by one.

- A *second phase* in which, after having ensured a proper internal behaviour, the above specified subsystems are integrated as well.

2.3 Integration Testing Strategy

As far as the integration testing process is concerned, a **bottom-up approach** will be followed.

The choice of the bottom-up testing strategy is natural since the integration testing can start from the smallest and lowest-level components, that are already tested at a unit level and do not depend on other components or not-already-developed components. In this way the total amount of needed stubs to accomplish the integration will be deeply reduced, but temporary programs for higher-level modules (drivers) will be necessary to simulate said modules and invoke the unit under test.

The bottom-up strategy will be mixed with a **critical-module-first approach**, in order to avoid issues related to the failures of core components and threats to the correct implementation of the entire *PowerEnJoy* system.

Moreover the higher-level subsystems described in section 2.2 are loosely coupled and fairly independent from one another because they correspond to different tiers. In this case, the critical-module-first approach is used to establish the integration order and get to the full system.

Notice that the DBMS is a commercial component already developed that can be used directly in the bottom-up approach and does not have any dependency.

At this level of integration testing, the communication functionalities with external systems must be covered as well, especially considering the relevance of said interaction in the context of the application. With respect to this, stubs and drivers will be used appropriately, based on the type of interface and interaction with the individual external systems.

2.4 Sequence of Components/Function Integration

The following sections aim to describe the integration testing sequence of the different components and subsystems of *PowerEnJoy*. From now on the following notation will be used: C1 $\rightarrow$ C2 indicates that C2 is necessary for C1 to work properly.

2.4.1 Software Integration Sequence

The components of each subsystem are tested starting from the most to the least independent one.

Data Access

The first components to be integrated are those relative to the data access, starting from the database core: the DBMS. This will be integrated with all the Java Entity Beans (JEB) defined in the Design Document [2].

In order to do so, the DBMS will need a driver for each Entity Bean to carry out queries and verify their correctness on a dummy database, containing a greatly reduced number of test information. Said test database will be structured based on the E-R schema that will be adopted for the final implementation of the data layer.

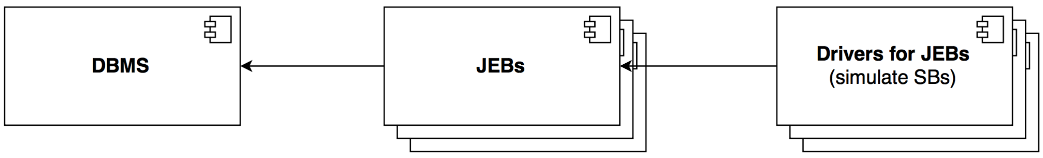
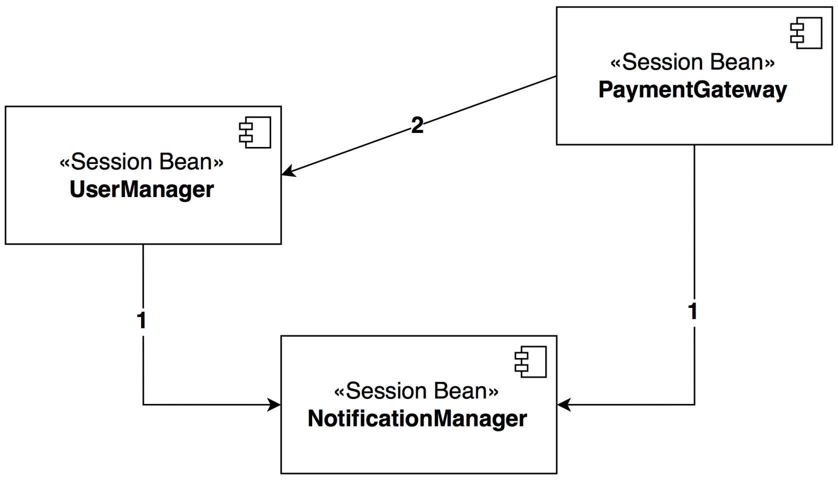


Figure 2.1: The integration of the system components at a Data Access level. The detailed integration steps are described at the end of the section in an overall comprehensive diagram.

The next steps involve the integration of the Session Beans which take advantage of said Entity Beans and are in charge of accessing them in the final application.

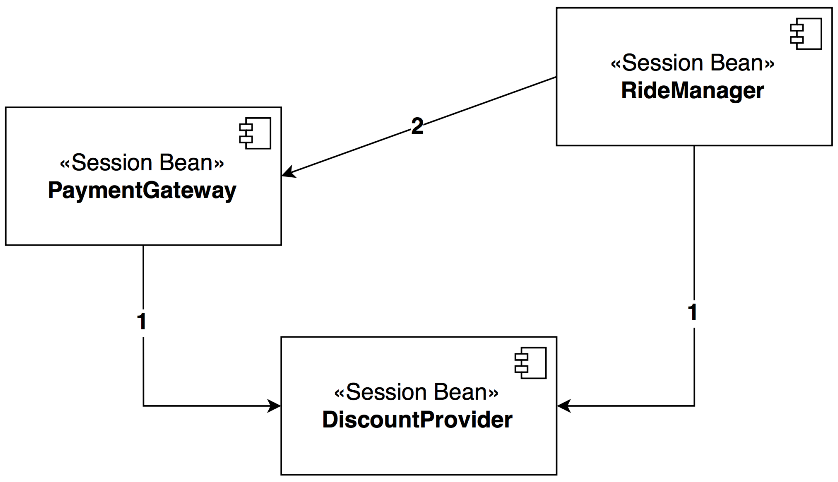
User and Utilities Management

The integration can begin by covering the user management and the business logic utilities, that are considered relevant to support the rest of the application functionalities. To begin with, the most independent bean is, in this case, the NotificationManager, that requires two drivers to invoke and test methods later used by UserManager and PaymentGateway (1). The UserManager component can then in turn be integrated (2), using a driver to be replaced with the PaymentGateway together with the previously mentioned one, and to call the needed methods appropriately for the case.



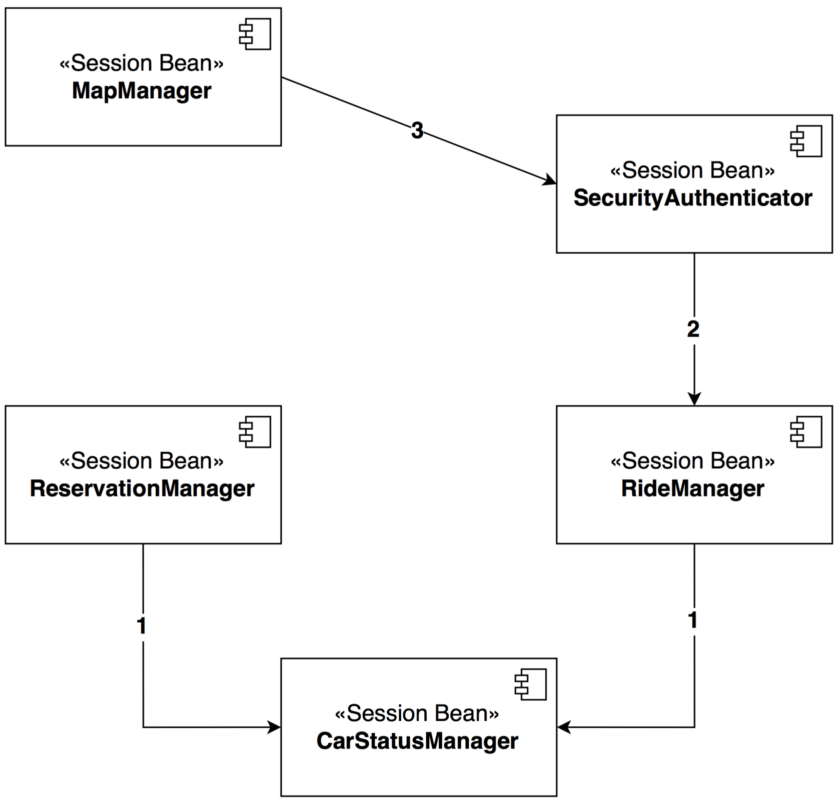
Payment Management

The payment management context can be covered next in the integration process, starting once more from the most independent component, represented by the DiscountProvider. This will need two different drivers, one for the methods called by the RideManager and another for the PaymentGateway (1). The PaymentGateway itself can in turn be integrated by using another driver, that will be replaced with the RideManager (2). RideManager is then going to replace the two drivers that simulated its behaviour altogether.



Ride and Reservation Management

The most critical features of the application revolve around the management of rides and reservations. Within this context, the most independent functionality is provided by the CarStatusManager bean, since no other bean depends on it apart from the already integrated Entity Beans. In order to integrate the CarStatusManager, there will be the need of two drivers: invoking methods in place of the ReservationManager and of the RideManager (1). In its turn, the RideManager itself needs a driver in order to be integrated (2), which will represent the SecurityAuthenticator session bean and call the methods exposed by RideManager in its place. The last component to be integrated in this context is the SecurityAuthenticator, which will need a driver to simulate method calls by the MapManager bean on it (3).



Application Logic Overall Integration

To conclude the integration process for the application logic tier, drivers for the EJB Containers must be provided, in order to have a means to emulate multiple requests for session bean instances; this will help in testing the underlying system effectiveness in managing heavy loads and concurrency during ordinary activity. Hence, in order to simulate the correctness of requests to the individual containers, a driver for each individual container must be used to bypass the runtime behaviour and reproduce said requests in a deterministic way. This approach will avoid the necessity of implementing the whole system before having the possibility to test the correctness of the requests to the containers.

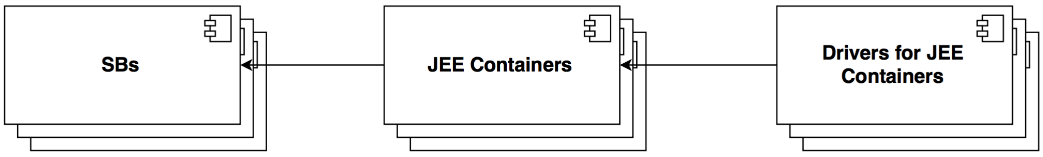
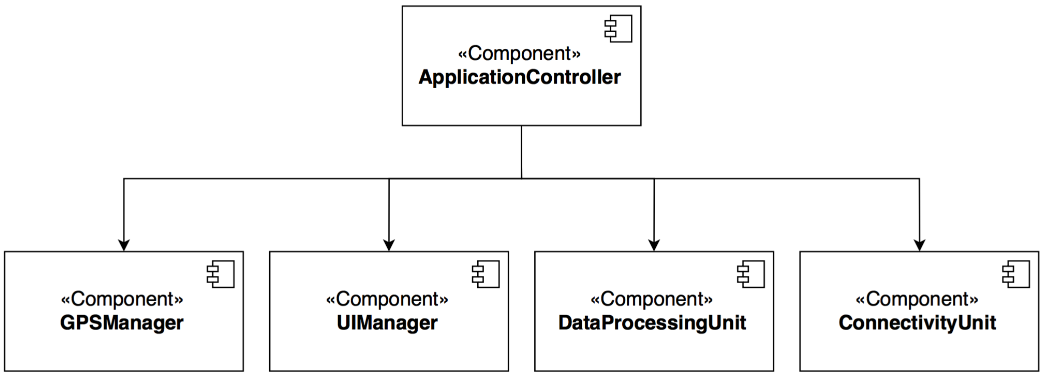


Figure 2.2: The integration of the system components at a container level. The detailed integration steps are described at the end of the section in an overall comprehensive diagram.

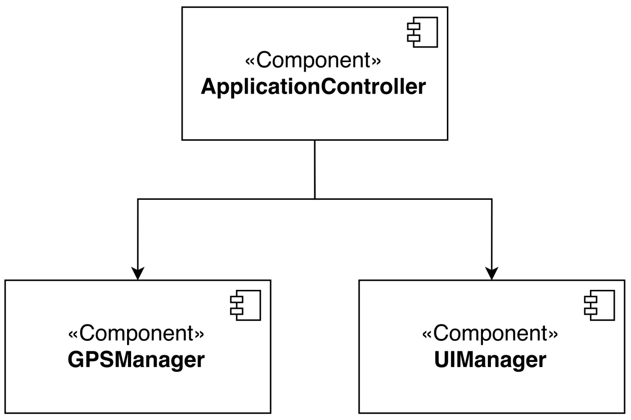
On-Board Application

With respect to the On-Board Application, the integration process must proceed for each of the base components individually with the application controller.


«Component» ApplicationController
«Component» GPSManager
«Component» UIManager
«Component» DataProcessingUnit
«Component» ConnectivityUnit

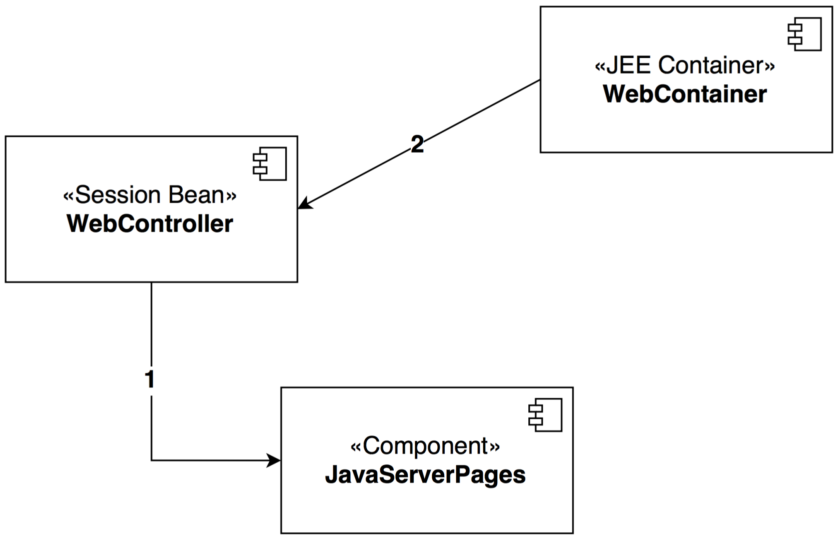
Mobile Application

Similarly to what is stated for the on-board application, the mobile application will follow the order imposed by the centrality of the application controller; the other components will be integrated individually with the controller itself.



Web Application

As far as the Web Application is concerned, the Java Server Pages will be integrated with the web controller, which will be in turn integrated with the container in which it is defined.



An overall view of the integration sequence is provided below:

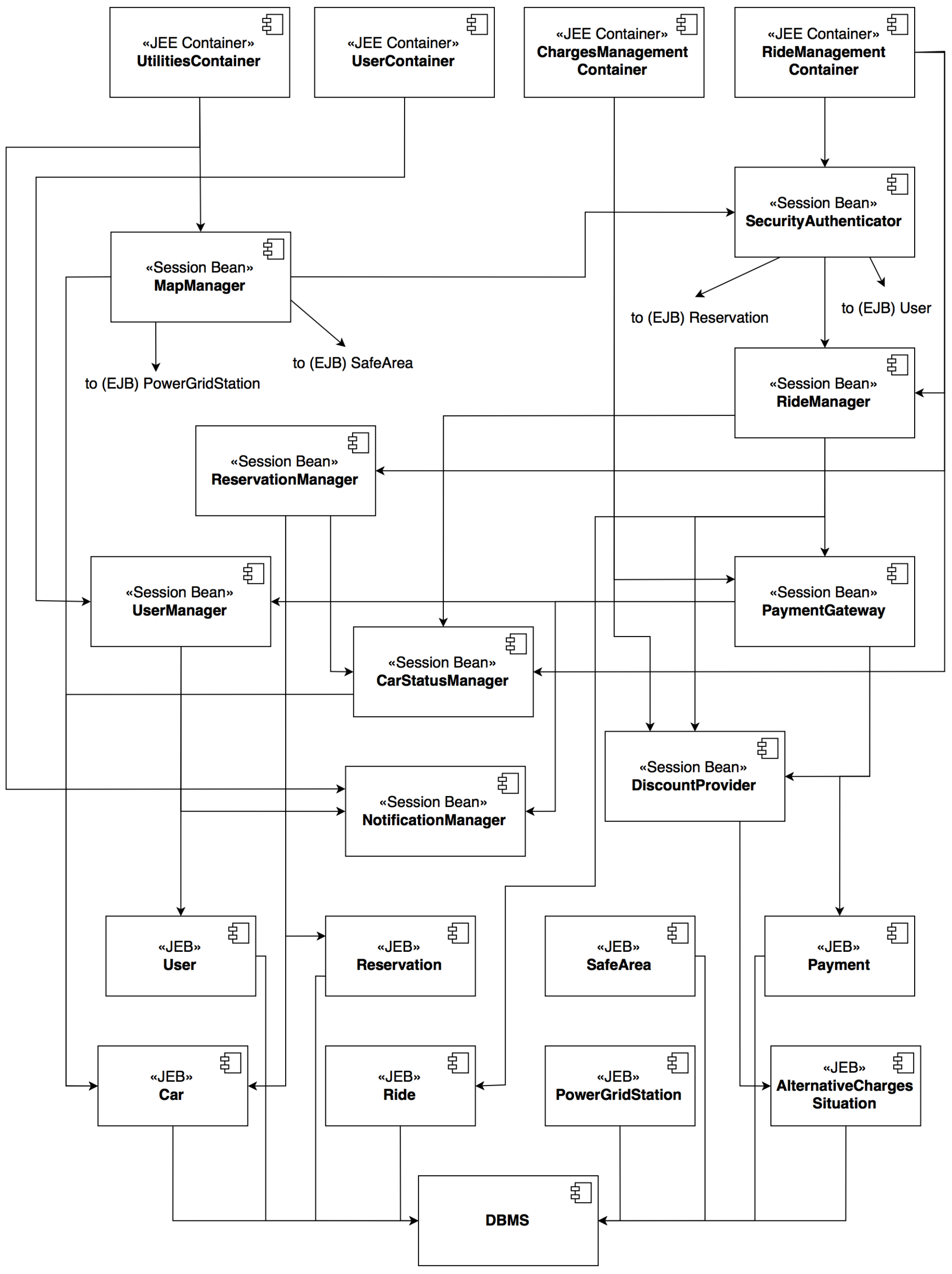


Figure 2.3: •

N.	Subsystems	Component	Integrates with

I01	Database, App. Logic	(JEB) User	DBMS
I02	Database, App. Logic	(JEB) Car	DBMS
I03	Database, App. Logic	(JEB) Reservation	DBMS
I04	Database, App. Logic	(JEB) Ride	DBMS
I05	Database, App. Logic	(JEB) SafeArea	DBMS
I06	Database, App. Logic	(JEB) PowerGridStation	DBMS
I07	Database, App. Logic	(JEB) AlternativeChargesSitutation	DBMS
I08	Database, App. Logic	(JEB) Payment	DBMS
I09	App. Logic	(SB) DiscountProvider	(JEB) AlternativeChargesSituation
I10	App. Logic	(SB) CarStatusManager	(JEB) Car
I11	App. Logic	(SB) ReservationManager	(SB) CarStatusManager
I12	App. Logic	(SB) UserManager	(JEB) User (SB) NotificationManager
I13	App. Logic	(SB) PaymentGateway	(JEB) Payment, (SB) UserManager, (SB) DiscountProvider
I14	App. Logic	(SB) RideManager	(JEB) Ride, (SB) PaymentGateway, (SB) DiscountProvider, (SB) CarStatusManager
I15	App. Logic	(SB) ReservationManager	(JEB) Car, (JEB) Reservation, (SB) CarStatusManager
I16	App. Logic	(SB) SecurityAuthenticator	(JEB) User, (JEB) Reservation, (SB) RideManager
I17	App. Logic	(SB) MapManager	(JEB) Car, (JEB) SafeArea, (JEB) PowerGridStation, (SB) SecurityAuthenticator
I18	App. Logic	(EJB Container) UserManagementContainer	(SB) UserManager

I19	App. Logic	(EJB Container) UtilitiesContainer	(SB) MapManager, (SB) NotificationManager
I20	App. Logic	(EJB Container) ChargesManagementContainer	(SB) PaymentGateway, (SB) DiscountProvider
I21	App. Logic	(EJB Container) RideManagementContainer	(SB) RideManager, (SB) ReservationManager, (SB) SecurityAuthenticator, (SB) CarStatusManager
I22	On-Board Client	ApplicationController	UIManager, GPSManager, ConnectivityUnit, DataProcessingUnit
I23	Mobile Client	ApplicationController	UIManager, GPSManager
I24	Web	WebController	JavaServerPages
I25	Web	WebContainer	WebController

Table 2.1: Integration order of the system components.

2.4.2 Subsystem Integration Sequence

The integration sequence of the high-level subsystems is described in Figure 2.4 and Table 2.2.

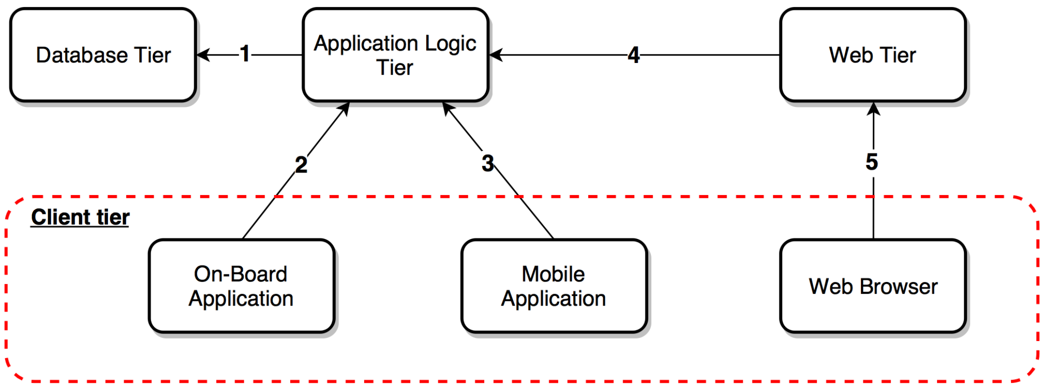


Figure 2.4: Diagram representing the order of the subsystems integration.

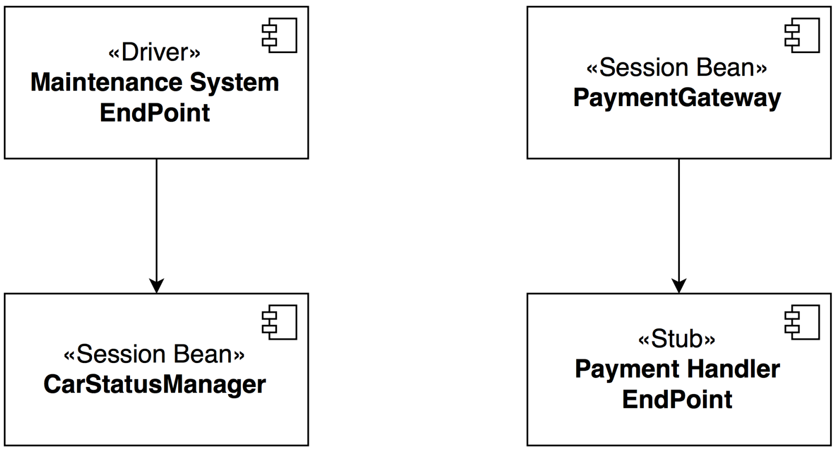
N.	Subsystem	Integrates with
SI1	Application Logic Tier	Database Tier
SI2	Application Logic Tier	(EXT) MaintenanceSystem
SI3	Application Logic Tier	(EXT) PaymentHandler
SI4	On-Board Application	Application Logic Tier
SI5	Mobile Application	Application Logic Tier
SI6	Web Tier	Application Logic Tier
SI7	Web Browser	Web Tier

Table 2.2: Integration order of the subsystems described in Section 2.2. External system interfaces to be integrated with *PowerEnJoy*'s subsystems are marked with (EXT).

Note that the base for the subsystem integration is the data tier, which is considered the most critical component; for the same reason, the application logic tier comes before all kinds of clients, since a working business logic is mandatory to have properly functioning clients. The choice of integrating the on-board application before other clients is due to the critical-module-first approach that has been chosen for this step of the integration process, since the on-board functionalities are meant to be core for the application itself. Lastly, the mobile application will be integrated first, since the integration of the web tier and browser client is heavier and more complex; moreover, this choice will allow the development team to have a working part of the system implementing a client-server structure even before having fully developed the web application.

External Systems

As stated in Section 2.3 of this document, the relevance of the interactions with external systems makes it necessary to integrate some of said functionalities at an application logic level. To be precise, the components to be integrated are the endpoints of the Payment Handler and of the Maintenance System. Since in the final implementation of the application the Payment Handler will provide the APIs to interface with it, the integration will need a *stub* of the Payment Handler endpoint, which will simulate the behaviour of the external payment system upon being invoked for any function. The Maintenance System will instead use the APIs provided by the *PowerEnJoy* system itself, hence the integration process will need a *driver* to simulate its calls to the *PowerEnJoy* system.



Section 3

Individual Steps and Test Description

The following are the test cases to be carried out. They are directly mapped with Table 2.1 for the integration among components and Table 2.2 for the subsystems integration. Test cases starting with **I** are integration test among components while test cases starting with **SI** concern the integration among subsystems. The adopted structure of the tables refers to the provided inegration test example [4].

3.1 Integration Test Cases I01, I02, I03, I04, I05, I06, I07

The following test cases concern the integration among the Java Entity Beans and the Database Tier running the DBMS. Since the test cases are very similar to each others, they are grouped together as follows:

Test Case Identifier	I01T1
Test Item(s)	(JEB) User $\rightarrow$ DBMS
Input Specification	Typical queries on table User.
Test Case Identifier	I02T1
Test Item(s)	(JEB) Car $\rightarrow$ DBMS
Input Specification	Typical queries on table Car.
Test Case Identifier	I03T1
Test Item(s)	(EJB) Reservation $\rightarrow$ DBMS
Input Specification	Typical queries on table Reservation.
Test Case Identifier	I04T1

Test Item(s)	(EJB) Ride → DBMS
Input Specification	Typical queries on table Ride.
Test Case Identifier	I05T1
Test Item(s)	(EJB) SafeArea → DBMS
Input Specification	Typical queries on table SafeArea.
Test Case Identifier	I06T1
Test Item(s)	(EJB) PowerGridStation → DBMS
Input Specification	Typical queries on table PowerGridStation.
Test Case Identifier	I07T1
Test Item(s)	(EJB) AlternativeChargesSituation → DBMS
Input Specification	Typical queries on table AlternativeChargesSituation.
Output Specification	The queries shall return the correct results.
Environmental Needs	GlassFish server, Test Database.
Test Description	The purpose of these tests is to check the correctness of the queries to the DBMS that the JEBs shall perform and implement. Drivers for the entity beans are used to properly initialize and call their methods before the related Session Beans are fully developed.
Testing Method	Automated with JUnit.

3.2 Integration Test Case I08

Test Case Identifier	I08T1
Test Item(s)	(SB) DiscountProvider → (JEB) AlternativeCharges-Situation
Input Specification	Calls from DiscountProvider to methods of the AlternativeChargesSituations entity to manage and update the information about discounts and additional charges accumulated during a rental.
Output Specification	The additional charges or discount information updates must be correctly memorized and made persistent.
Environmental Needs	GlassFish server, Test Database.

Test Description	The purpose of this test is to verify that the additional charges or discount information are correctly managed and updated by the AlternativeChargesSituation entity upon being prompted by the DiscountProvider.
Testing Method	Automated with JUnit.

3.3 Integration Test Case I09

Test Case Identifier	I09T1
Test Item(s)	(SB) CarStatusManager → (JEB) Car
Input Specification	Calls from CarStatusManager to methods of the Car entity to update the car status information according to specific conditions.
Output Specification	The car status information updates must be correctly memorized and made persistent.
Environmental Needs	GlassFish server, Test Database.
Test Description	The purpose of this test is to check that every update to the status attribute of the Car entities performed upon requests from the CarStatusManager is applied correctly.
Testing Method	Automated with JUnit.

3.4 Integration Test Case I10

Test Case Identifier	I10T1
Test Item(s)	(SB) CarStatusManager → (JEB) Car
Input Specification	Calls from CarStatusManager to methods of the Car entity to update the car status information according to specific conditions.
Output Specification	The car status information updates must be correctly memorized and made persistent.
Environmental Needs	GlassFish server, Test Database.
Test Description	The purpose of this test is to check that every update to the status attribute of the Car entities performed upon requests from the CarStatusManager is applied correctly.

Testing Method	Automated with JUnit.

3.5 Integration Test Case SI2

Test Case Identifier	I10T1
Test Item(s)	(EXT) MaintenanceSystemEndpoint → (SB) CarStatusManager
Input Specification	Typical queries on table AlternativeChargesSituation.
Output Specification	The queries shall return the correct results.
Environmental Needs	GlassFish server, Test Database, driver for the Java Entity Beans.
Test Description	A driver emulating the Maintenance System endpoint is used to simulate the external processing of intervention requests triggered by the CarStatusManager. The purpose of this test is to check that upon every request, a confirmation of the ... of the maintenance intervention is received from the external system.
Testing Method	Automated with JUnit.

3.6 Integration Test Case SI1

Test Case Identifier	SI1T1
Test Item(s)	Application Logic Tier → Database Tier
Input Specification	Calls to the methods offered by the JPA Entities that are mapped on tables in the Database Tier.
Output Specification	The Database Tier must reply correctly by executing queries on the Test Database. In the case requests are coming from unauthorized sources that are maliciously trying to access the data, they must be blocked
Environmental Needs	The test needs a complete implementation of the JEBs, the Java Persistence API, the Test Database and a bunch of drivers that calls the JEBs' methods.
Test Description	The replies to the queries coming from the Database Tier will be compared with the expected output results.
Testing Method	Automated with JUnit.

Section 4

Tools and Test Equipment Required

Section 5

Program Stubs and Test Data Required

Appendix A

Appendix

A.1 Software and tools used

- LaTeX, used as typesetting system to build this document.
- draw.io - `https://www.draw.io` - used to draw diagrams and mock-ups.
- GitHub - `https://github.com` - used to manage the different versions of the document and to make the distributed work much easier.
- GitHub Desktop, the GitHub official application that offers a seamless way to contribute to projects.

A.2 Hours of work

The absolute major part of the document was produced in group work. The approximate number of hours of work for each member of the group is the following:

- Giovanni Scotti:
- Marco Trabucchi:

NOTE: indicated hours include the time spent in group work.

Bibliography

[1] AA 2016/2017 Software Engineering 2 - *Requirements Analysis and Specification Document* - Giovanni Scotti, Marco Trabucchi

[2] AA 2016/2017 Software Engineering 2 - *Design Document* - Giovanni Scotti, Marco Trabucchi

[3] AA 2016/2017 Software Engineering 2 - *Project goal, schedule and rules*

[4] SpinGrid Project - *Integration Test Plan*