Politecnico di Milano

Industrial and Information Engineering
Computer Science and Engineering

Software Engineering II Assignment
PowerEnJoy - car sharing

Academic Year: 2016/2017

Prof. Elisabetta Di Nitto

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

PowerEnJoy

Car Sharing App

TABLE OF CONTENT

LIST OF FIGURES

1 INTRODUCTION

1.1 Revision History

Version	Date	Authors	Summary
1.0	07/01/2017	Mark Edward Ferrer, Alice Segato, Davide Bonacina	Initial release
1.1	07/01/2017	Mark Edward Ferrer, Alice Segato, Davide Bonacina	Better specification of subsystems

1.2 Purpose and Scope

This document describes the Integration Testing Plan for PowerEnJoy system. This testing is necessary in order to avoid unexpected behaviors of the system and guarantee its ability to fulfill all the requirements. The ITPD outlines the testing activities organization for the subsystems that make up the system. This document is composed by four parts:

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

1.3 List of Definitions and Abbreviations

- DD: Design Document.
- RASD: Requirements analysis and Specification Document.
- DB: the database layer, handled by a DBMS.
- UI: User Interface.
- GUI: graphical user interface is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators;
- Application server: the layer, which provides the application logic and interacts with the DB and with the front-ends.
- Back-end: term used to identify the Application server.
- Front-end: the components, which use the application server services, namely the web front-end and the mobile applications.
- Web server: the component that implements the web-based front-end. It interacts with the application server and with the users' browsers.
- Acknowledge: is a signal passed between communicating processes or computers to signify acknowledgement, or receipt of response, as part of a communications protocols.

1.4 List of Reference Documents

- RASD;
- Assignment AA 2016-2017.pdf;
- Integration testing example document.pdf;
- Sample Integration Test Plan Document.pdf.

2 INTEGRATION STRATEGY

2.1 Entry Criteria

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

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

2.2 Elements to be Integrated

The components that we are going to test are all the internal components of the Main System, the Car Computer and the Client App (as described in the section 2.3.1 of the Design Document). The components of the Third Party System and the relative clients (both web and mobile applications) have to be already tested and fully working, according to our assumptions described in the section 1.6 of the Requirements Analysis and Specification Document. Even if the third party system is not meant to be developed by our team, as specified in the previous documents, the testing must be done in order to guarantee the correctness of the integration between the two systems and of their outputs.

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

As explained better in the few next sections, we will test the components from their inner classes up to the complete subsystem, which are:

- Data Management subsystem that handles all the actions concerning login, registration, database storage and data manipulation for any kind of activity;
- Logic subsystem that handles all the business logic linked to the Main System, such as money saving option algorithm, reservation and booking management, signal dispatch to both user and admin clients and communication with the central database;
- Administration subsystem that coincides more or less with the third party system, handling all of its business logic, such as car management, user support and service maintenance but also communication with the central database;
- User client subsystem that handles the user app and its communication with the Main system;
- Admin client subsystem that includes both mobile and web app for admins and employees handling their communication with the third party system.

2.3 Integration Testing Strategy

To approach the integration test phase we decided to adopt the bottom-up strategy to test first the lower level components until we obtain greater subsystems. At this point we follow the critical-module-first approach to integrate together the subsystems found in the previous step. We will need only one stub of the User App component to be used during Main System subsystem testing. This stub will be called by the Main System to mock the behavior of the user app since this will be implemented later than the core system. The reason for this choice will be explained better in the Section 5 of this document.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

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

Application Server component integration:

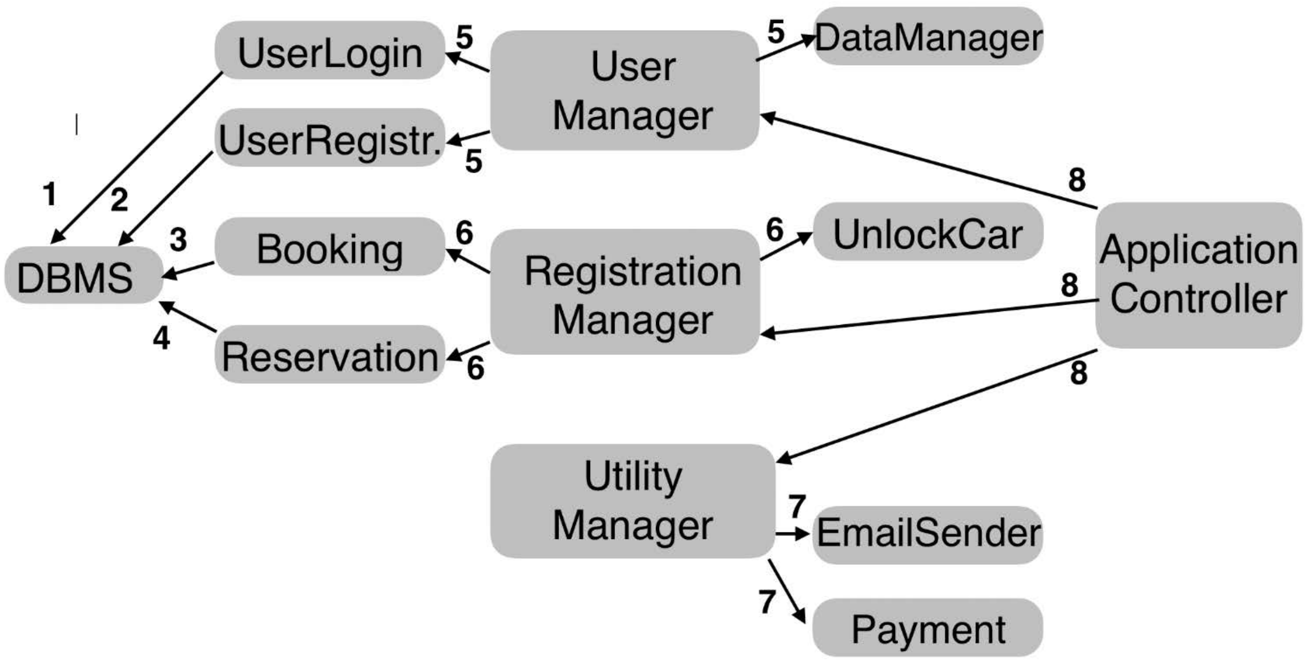


Figure 1

Mobile Application and car computer component integration:

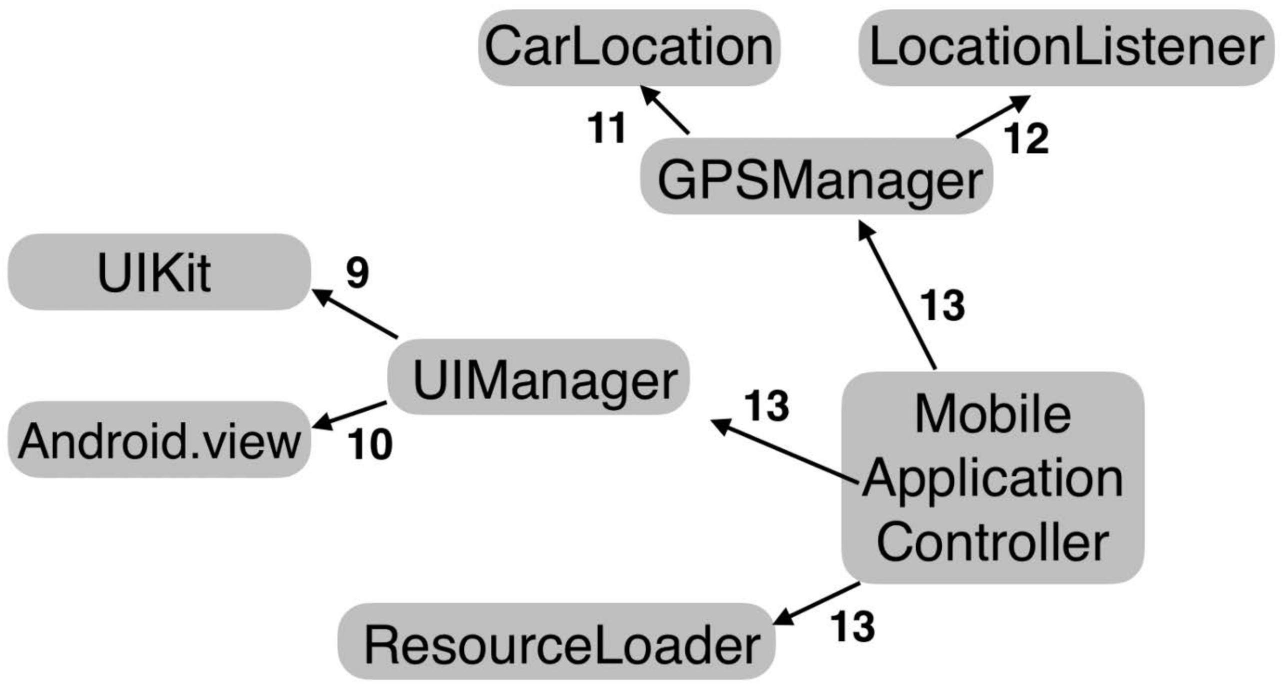


Figure 2

Third Party System component integration:

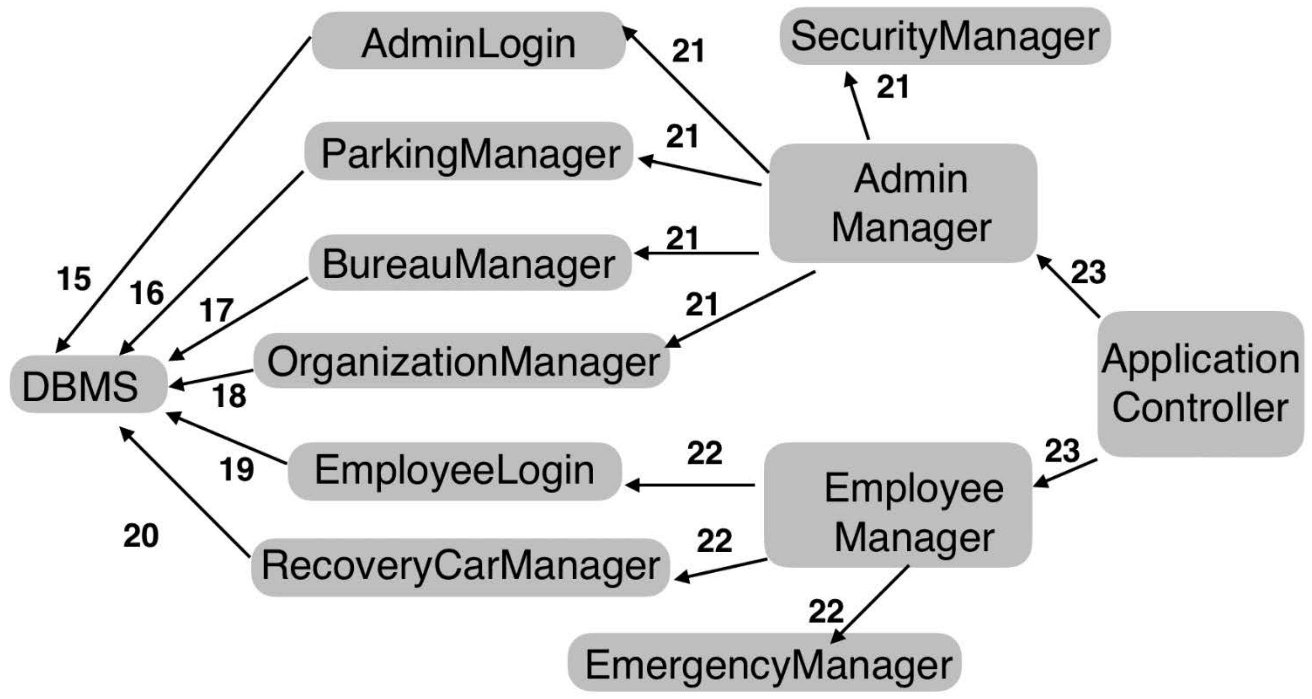


Figure 4

Web Application component integration:

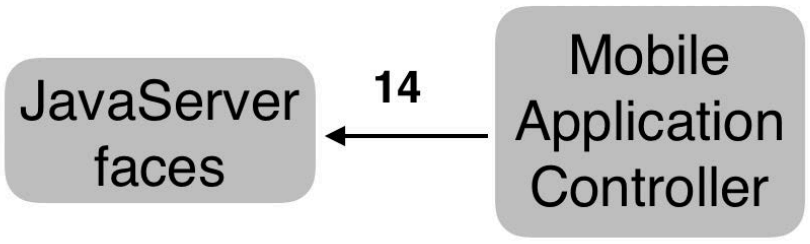


Figure 3

Integration of the system component:

N	SUBSYSTEM	COMPONENT	INTEGRETES WITH
I1	Data Management, Logic	UserLogin	DBMS
I2	Data Management, Logic	UserRegistration	DBMS
I3	Data Management, Logic	Booking	DBMS
I4	Data Management, Logic	Reservation	DBMS
I5	Logic	UserManager	UserLogin UserRegistration DataManager
I6	Logic	ReservationManager	Booking Reservation UnlockCarManager
I7	Data Management, Logic	UtilityManager	EmailSender PaymentManager
I8	Data Management, Logic	ApplicationController	UserManager Reservation Manager UtilityManager
I9	User client	UIManager	UIKit
I10	User client	UIManager	Android.view

N	SUBSYSTEM	COMPONENT	INTEGRETES WITH
I11	User client	GPSManager	CarLocator
I12	User client	GPSManager	LocationListener
I13	User client	MobileAppController	UIManager GPSManager Resource Manager
I14	Administration	WebAppController	JavaServerFaces
I15	Data Management, Logic	AdminLogin	DBMS
I16	Data Management, Logic	Parking Manager	DBMS
I17	Data Management, Logic	Bureaucracy Manager	DBMS
I18	Data Management, Logic	Organization Manager	DBMS
I19	Data Management, Logic	Employee Login	DBMS
I20	Data Management, Logic	RecoveryCar Manager	DBMS
I21	Logic	AdminManager	AdminManager ParkingManager BureauManager OrganizationManager SecurityManager
I22	Logic	EmployeeManager	EmployeeLogin CarRecoveryManager EmergencyManager
I23	Logic	ApplicationController	AdminManager EmployeeManager

2.4.2 Subsystem Integration Sequence

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

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

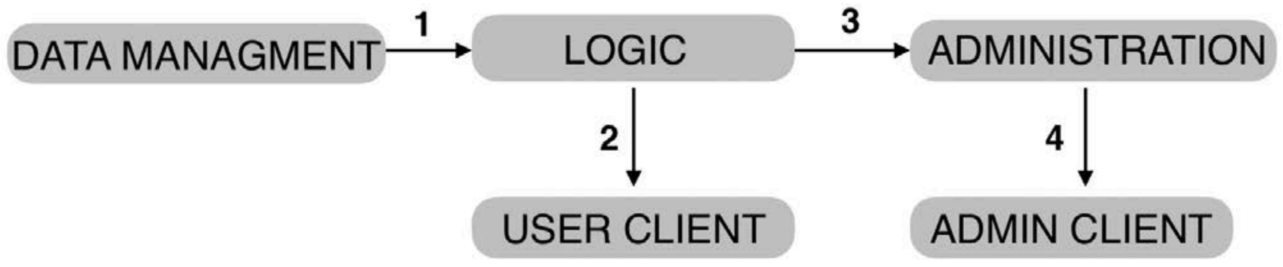


Figure 5

Integration order of the subsystems:

N	SUBSYSTEM	INTEGRETES WITH
SI1	LOGIC	DATA MANAGEMENT
SI2	USER CLIENT	LOGIC
SI3	ADMINISTRATION	LOGIC
SI4	ADMIN CLIENT	ADMINISTRATION

3 INDIVIDUAL STEPS AND TEST DESCRIPTION

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

3.1 Integration test case SI1

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

3.2 Integration test case SI2

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

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

3.3 Integration test case SI3

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

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

3.4 Integration test case SI4

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

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

3.5 Integration test case I1

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

3.6 Integration test case I2

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

3.7 Integration test case I3

Test Case Identifier	I3T1
Test Item(s)	Booking → DBMS
Input Specification	Typical queries on the Parking table in the database
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.8 Integration test case I4

Test Case Identifier	I4T1
Test Item(s)	Reservation → DBMS
Input Specification	Typical queries on the Reservation table in the database
Output Specification	The queries insert the correct tuple and return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.9 Integration test case I5

Test Case Identifier	I5T1
Test Item(s)	UserManager → UserLogin, UserRegistration, DataManager
Input Specification	Methods call from UserManager to UserLogin, UserRegistration, DataManager to manage and update the information of the User.
Output Specification	The User information must be correct and up-to-date.
Environmental Needs	Tomcat Server.
Test Description	Verify that the information is correctly updated and that it refers to the correct user. Control that the user's information is persistently updated.
Testing Method	Automated with JUnit.

3.10 Integration test case I6

Test Case Identifier	I6T1
Test Item(s)	ReservationManager → Booking, Reservation, UnlockingCar
Input Specification	Methods call from ReservationManager to Booking, Reservation, UnlockingCar to update Reservation and Bookings' status and to find an available car in a specified parking, and to unlock a car.
Output Specification	The reservation must be correctly updated without duplicating elements and the correct first available Car must be returned and removed from the ones available. The Booking status must be updated and the information of the user must be correctly checked when he tries to unlock a car.
Environmental Needs	Tomcat Server.
Test Description	The test aims to verify that the ReservationManager requests are correctly satisfied by Booking, Reservation, UnlockingCar.
Testing Method	Automated with JUnit.

3.11 Integration test case I7

Test Case Identifier	I7T1
Test Item(s)	UtilityManager → EmailSender, PaymentMethod
Input Specification	Methods call from UtilityManager to the EmailSender in order to guarantee a right email authentication process. and also Method call from UtilityManager to the PaymentMethod in order to guarantee a right payment.
Output Specification	The email authentication process and payment process must be correctly handled.
Environmental Needs	Tomcat Server, e-mail sender and receiver and payment tools.
Test Description	Assure that a user can properly verify his/her email address in order to start using the system functionalities. and Assure that a payment is properly and automatically done when a user finish to use the car.
Testing Method	Automated with JUnit.

3.12 Integration test case I8

Test Case Identifier	I8T1
Test Item(s)	ApplicationController → UserManager, ReservationManager, UtilityManager
Input Specification	Request from ApplicationController to UserManager, ReservationManager, UtilityManager for the functionalities offered.
Output Specification	The concurrency between the request must be properly managed and the ApplicationController has to be able to provide the right functionality carrying out the proper request.
Environmental Needs	Tomcat Server.
Test Description	Multiple requests for the UserManager, ReservationManager, UtilityManager SessionBeans have to be simultaneously carried out, in order to ensure that the users have no concurrency trouble.
Testing Method	Automated with JUnit.

3.13 Integration test case I9

Test Case Identifier	I9T1
Test Item(s)	UIManager → UIKit
Input Specification	Methods call from UIManager to the UI elements, to display output data and change their status.
Output Specification	The view shall change accordingly and display the output data.
Environmental Needs	Xcode, iOS Simulator.
Test Description	Verify that the bindings of the view items are correctly set in the controller and that the view actually changes and responds to method calls. Check that the output is displayed correctly.
Testing Method	Automated (iOS testing suite), manual testing on physical devices.

Test Case Identifier	I9T2
Test Item(s)	UIManager → UIKit
Input Specification	Perform (or simulate) gestures on the UI elements.
Output Specification	The controller shall receive the actions and log them.
Environmental Needs	Xcode, iOS Simulator.
Test Description	Check that the gestures perform the correct actions on the controller.
Testing Method	Automated (iOS testing suite), manual testing on physical devices.

3.14 Integration test case I10

Test Case Identifier	I10T1
Test Item(s)	UIManager → android.view
Input Specification	Methods call from UIManager to the UI elements, to display output data and change their status.
Output Specification	The view shall change accordingly and display the output data.
Environmental Needs	Android Emulator.
Test Description	Verify that the bindings of the view items are correctly set in the controller and that the view actually changes and responds to method calls. Check that the output is displayed correctly.
Testing Method	Automated (Android testing suite), manual testing on physical devices.

Test Case Identifier	I10T2
Test Item(s)	UIManager → android.view
Input Specification	Perform (or simulate) gestures on the UI elements.
Output Specification	The controller shall receive the actions and log them.
Environmental Needs	Android Emulator.
Test Description	Check that the gestures perform the correct actions on the controller.
Testing Method	Automated (Android testing suite), manual testing on physical devices.

3.15 Integration test case I11

Test Case Identifier	I11T1
Test Item(s)	GPSManager → CarLocation
Input Specification	Calls to the CarLocation framework methods to get location data of the car.
Output Specification	Car location data or a meaningful error status shall be returned.
Environmental Needs	Xcode, iOS Simulator.
Test Description	The purpose of the test is to check that our controller (GPSManager) can correctly get the position from the corresponding iOS API. Error statuses shall also be checked.
Testing Method	Automated (iOS testing suite).

3.16 Integration test case I12

Test Case Identifier	I12T1
Test Item(s)	GPSManager → LocationListener
Input Specification	Calls to the Android Location framework methods to get location data of the user.
Output Specification	User location data shall be returned, or a meaningful error status.
Environmental Needs	Android Emulator.
Test Description	The purpose of the test is to check that our controller (GPSManager) can correctly get the position from the corresponding Android API. Error statuses shall also be checked.
Testing Method	Automated (Android testing suite).

3.17 Integration test case I13

Test Case Identifier	I13T1
Test Item(s)	MobileApplicationController → UIManager, GPSManager, ResourceLoader
Input Specification	Calls to GPSManager methods to get the user's location. Load application resources (images, sounds, data) from ResourceManager.
Output Specification	The location data shall be returned from GPSManager in a suitable format, or an exception shall be raised if the location data is not available. ResourceManager should provide the required resources without errors.
Environmental Needs	Xcode, iOS Simulator, Android Emulator.
Test Description	GPSManager should be able to return the correct GPS data in a universal and consistent format independently from the architecture (iOS or Android). ResourceLoader is responsible for the retrieval of the resources stored into the application bundle. This test aims to assessing that all the resources can be accessed without errors by the mobile
Testing Method	Automated (Android and iOS testing suites).

3.18 Integration test case I14

Test Case Identifier	I14T1
Test Item(s)	WebApplicationController → JavaServerFaces
Input Specification	WebController is given the typical output to be displayed on the web page.
Output Specification	JavaServerFaces shall display the required output in a correct way.
Environmental Needs	Tomcat Server, Stub of the Logic subsystem of the third part system to provide the output data.
Test Description	The purpose of this test case is to check if JSF can communicate correctly with the WebApplicationController bean.
Testing Method	Automated with JUnit.

3.19 Integration test case I15

Test Case Identifier	I15T1
Test Item(s)	AdminLogin → DBMS
Input Specification	Typical queries on the User table in the database
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.20 Integration test case I16

Test Case Identifier	I16T1
Test Item(s)	ParkingManager → DBMS
Input Specification	Typical queries on Parking, Parking Slot and Plug Slot tables in the database
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.21 Integration test case I17

Test Case Identifier	I17T1
Test Item(s)	BureauManager → DBMS
Input Specification	Typical queries on the Auto table in the database
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.22 Integration test case I18

Test Case Identifier	I18T1
Test Item(s)	OrganizationManager → DBMS
Input Specification	Typical queries on database tables
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.23 Integration test case I19

Test Case Identifier	I19T1
Test Item(s)	EmployeeLogin → DBMS
Input Specification	Typical queries on the User table in the database
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.24 Integration test case I20

Test Case Identifier	I20T1
Test Item(s)	RecoveryCarManager → DBMS
Input Specification	Typical queries on the Auto table in the database
Output Specification	The queries return the correct results.
Environmental Needs	Tomcat Server, Test Database, driver for the JavaEntity Beans.
Test Description	The purpose of these tests is to check that the correct methods of the Entity Beans are called, and that they execute the correct queries to the DBMS.
Testing Method	Automated with JUnit.

3.25 Integration test case I21

Test Case Identifier	I21T1
Test Item(s)	AdminManager → SecurityManager, AdminLogin, ParkingManager, BureauManager, OrganizationManager.
Input Specification	Methods call from AdminManager to Manage, Security service, Parking distribution, employee tasks, software maintenance to update software and security service, to administrate parking and employee task and to check for insurance and other car bureaucracy.
Output Specification	The administrator procedure must be correctly updated without duplicating elements and the correct first available parking information must be right and updated, the employee tasks must be updated and the bureaucracy information must be right.
Environmental Needs	Tomcat Server
Test Description	The test aims to verify that the AdminManager requests are correctly satisfied by SecurityManager, AdminLogin, ParkingManager, BureauManager, OrganizationManager.
Testing Method	Automated with JUnit.

3.26 Integration test case I22

Test Case Identifier	I22T1
Test Item(s)	EmployeeManager → EmployeeLogin, RecoveryCarManager and EmergencyManager.
Input Specification	Methods called from EmployeeManager to Manage cars recovery and emergencies.
Output Specification	The car recovery information must be correctly updated without duplicating elements and the information of emergency location and problem must be correct too.
Environmental Needs	Tomcat Server
Test Description	The test aims to verify that the EmployeeManager requests are correctly satisfied by EmployeeLogin, RecoveryCarManager and EmergencyManager.
Testing Method	Automated with JUnit.

3.27 Integration test case I23

Test Case Identifier	I23T1
Test Item(s)	ApplicationController → EmployeeManager, AdminManager
Input Specification	Request from ApplicationController to EmployeeManager, AdminManager for the functionalities offered.
Output Specification	The concurrency between the requests must be properly managed and the ApplicationController has to be able to provide the right functionality carrying out the proper request.
Environmental Needs	Tomcat Server
Test Description	Multiple requests for the EmployeeManager and AdminManager SessionBeans have to be simultaneously carried out, in order to ensure that the users have no concurrency troubles.
Testing Method	Automated with JUnit.

4 TOOLS AND TEST EQUIPMENT REQUIRED

4.1 Tools

Following the bottom-up approach, we will use different frameworks to test all the components in each phase of testing.

- Manual testing: necessary to verify immediately the correctness of the code and very useful to identify the test data need for the next testing;
- JUnit: this component was designed mainly for unit testing and we will use it at the end of the implementation of each component to ensure that all the methods are consistent and logically correct. It allows to discover implementation errors, unhandled exceptions and also interaction with other components
- Arquillian: this component was designed mainly for integration testing and allows to try test cases against a container and allows to check if the surroundings if it work in an appropriate way. It is also useful for dependency injection check and to verify if the components integrate with each other in the planned way.
- Apache JMeter: this component allows to test the system with different sizes of load in order to verify if it responds in the correct way and time to a client request.

4.2 Test Equipment

All the testing activities have to be performed in a dedicated environment, for this reason we need proper equipment to perform it.
On front-end side, we need:

- At least 2 Android smartphone with at least Android 5.0 Lollipop installed;
- At least 2 Android tablets with at least Android 5.0 Lollipop installed;
- At least 2 iPhones of each member of the family with the latest possible software installed;
- At least 2 iPads of each member of the family with the latest possible software installed.

We need at least two items of each mobile device in order to test requests for resources that have been reserved by another client to check how to system manages an invalid request (verified also inserting the correct input test data). As we mentioned in section 1.8 of the RASD, we will develop firstly the client app for Android and iOS since they are the most spread out mobile operating systems, and then we will develop it for Windows Phone, even if it has the 1.76% of the market share of mobile operating systems on December 2016 (source: NetMarketShare), so for that we will need, as for the other two operating systems, 2 devices for both phone and tablet with Windows Phone installed.

On back-end side we will test the system on a cloud server that will have installed a software set similar to the final one installed in the definitive server. For this reason, we will use:

- Windows Azure for the infrastructure;
- Red Hat Enterprise Edition as operating system;
- MySQL as DBMS;
- Tomcat as Web Server;
- Java Enterprise Edition;
- Apache JMeter;

5 PROGRAM STUBS AND TEST DATA REQUIRED

5.1 Program Stubs and Drivers

According to the section 2.3 of this document, we will approach the testing with a bottom-up strategy; this means that we are going to need drivers to mock the behavior of the dependencies involved in the testing of each subcomponent. The drivers that we will need to test each subcomponent are:

- **User Authentication Driver** to test the login and registration functions of the Main System. It will be used to test **userLogin**, **userRegistration** and **emailSender** components with their functions and essentially it will call the functions sending them fake credentials to verify the consistency of the code;
- **User Data Driver** to test the communication of the Main System with the database component. It will be used to test the **personalDataManagement** component and it will input data that have to be stored in the database;
- **User Action Driver** to test the booking and the retrieval of cars. It will call all the functions that allow user to reserve a car, to open it or any other action that the user could do through the app;
- **Payment Driver** to test the functionality of the **paymentManager** that include the payment of a bill and the verification of a payment method;
- **User Input Driver** used to test the integration between **APIController** and **ApplicationController** components these components have to cooperate in the right way to handle correctly the request of the user. Also the **Algorithm Controller** will be involved in this test;
- **Car Mobile Application Driver** to call the functions of the **SensorManager** subsystem that includes a component that handles the GPS signal and different components to handle the input from the different sensors of the car;
- **User App Driver** to test the functions of the internal components of the User App for the correct rendition of the system output. This driver has a different task from the stub that we intend to use because it test the correctness of the User App itself testing its internal components.

Even if the bottom-up strategy does not usually involve the use of stubs, we need to create one for testing the full Main System in order to know if it will respond correctly to the client requests and also to check if it can handle a big amount of them at the same time. The reason why we need to introduce a User App stub is that the Main System will be developed before the client and also because the core and the client have a mutual dependency and the both need strongly the output of the other.

5.2 Test Data

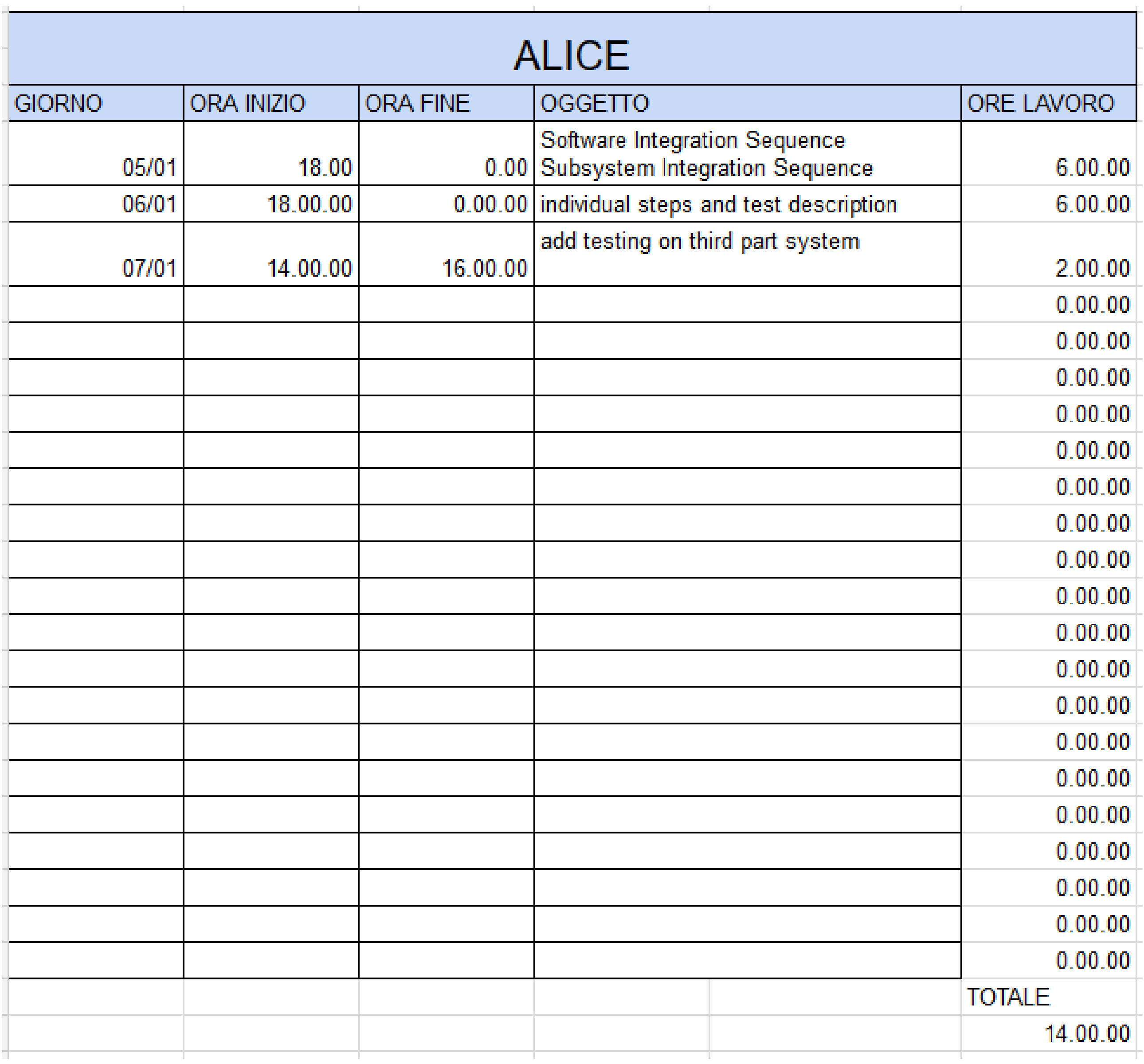
The input data test that we need to perform the tests are:

- A set of valid and invalid user credentials to test registration, login, credential verification and data storage to the database;
- A set of valid and invalid GPS data to simulate every possible case of reservation, including instances for testing of:
    - Reservation from position without safe area;
    - Unlock request from non-matching position;
- A set of valid and invalid payment methods to test payment management;
- A set of valid and invalid reservations to test the correctness of the booking, reservation and Main System application controller components (this last component has to verify if the reservation exists and according to the check, open or not the car).

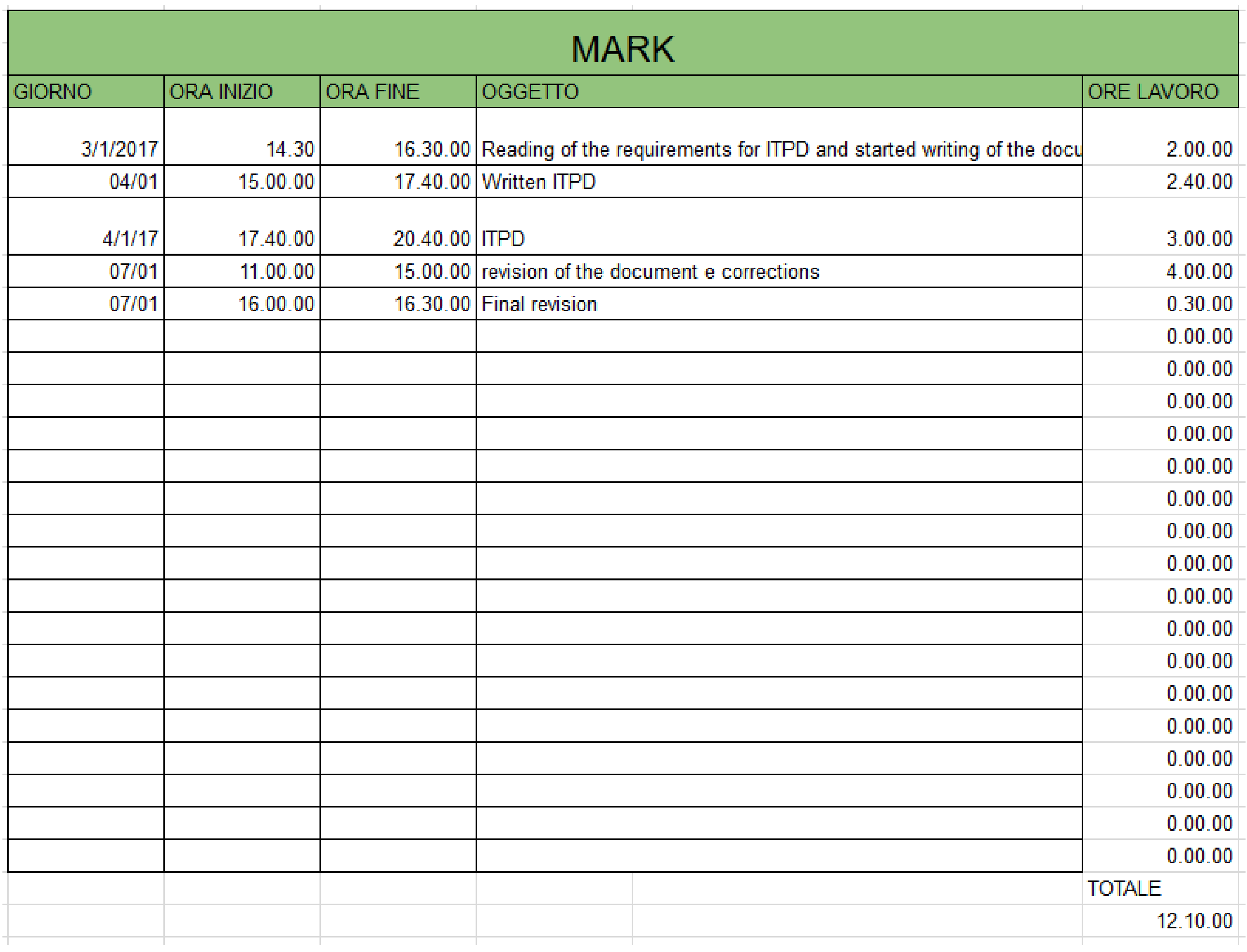
All these sets of data must contain these particular instances:

- Null object;
- Null fields.

6 EFFORT SPENT



ALICE				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
05/01	18.00	0.00	Software Integration Sequence Subsystem Integration Sequence	6.00.00
06/01	18.00.00	0.00.00	individual steps and test description	6.00.00
07/01	14.00.00	16.00.00	add testing on third part system	2.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				14.00.00



MARK				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
3/1/2017	14.30	16.30.00	Reading of the requirements for ITPD and started writing of the docu	2.00.00
04/01	15.00.00	17.40.00	Written ITPD	2.40.00
4/1/17	17.40.00	20.40.00	ITPD	3.00.00
07/01	11.00.00	15.00.00	revision of the document e corrections	4.00.00
07/01	16.00.00	16.30.00	Final revision	0.30.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				12.10.00

DAVIDE				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
3/1	14.30.00	16.30.00	Started writing the document	2.00.00
04/01	15.00.00	17.40.00	Written ITPD	2.40.00
07/01	1.30.00	2.30.00	Fixed ITPD layout	1.00.00
07/01	13.10.00	14.30.00	Fixed ITPD tables layout	1.20.00
07/01	17.00.00	18.00.00	Final adjustments	1.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				8.00.00