POLITECNICO DI MILANO
Computer Science and Engineering
Software Engineering 2 Project

Tesla Car Sharing

Integration Test Plan Document

Danchenko Elena, 874840
Cilloni Stefano, 880924

January 15th, 2017

Document version: 1.0

Contents

1. Introduction

1.1. Purpose and Scope

1.1.1. Purpose

This document represents the integration test plan for software system of Testa Car Sharing software.
The integration test plan document describes the integration strategy as sequence of connections between components to be tested, integration test cases which are planned to be made during the test phase, the interpretation of results of the test cases and tools to be used.

1.1.1. Scope

Software system of Testa Car Sharing service aims to provide the possibility for clients of car sharing service to use the service though their computer or mobile phone, as well as for car sharing service employees to manage system.

1.2. List of Definitions and Abbreviations

- **DD:** Design Document
- **ITPD:** Integration Test Plan Document (this document)
- **Datastore:** It is the No-SQL (non relational) distributed database management system by the GCP. It is a DBMS (Database Management System) for cloud infractures). It is well integrated with the App Engine cloud environment through a large set of APIs.
- **Application layer:** it is the core layer of the whole application which implements through software logics and procedures that allow software to run.
- **API:** Application Programming Interface is a set of subroutine definitions, protocols, and tools for building application software.
- **REST:** Representational state transfer are one way of providing interoperability between computer systems on the Internet. REST-compliant web services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations.
- **HTTPS:** is is a protocol for secure communication over a computer network which is widely used on the Internet.

- **Functional module:** Like a Java package it groups classes, functions and programming code. In the Atuin Framework it has a semantical meaning that help developer to well structure code.
- **Administration Submodule:** It is the logical set of the controllers that in a functional module are dedicated to implement administrative works.
- **Client Submodule:** It is the logical set of the controllers that in a functional module are dedicated to implement works more strictly related to the clients of the Tesla Car Sharing project.

1.3. List of Reference Documents

This document refers to the following documents:

- Specification document: Assignments AA 2016-2017
- Requirement Analysis and Specification Document of the Tesla Car Sharing
- Design Document of the Tesla Car Sharing project
- Example document: Integration Test Plan of Spin Grid project

2. Integration Strategy

2.1. Entry Criteria

The prerequisites that system should meet before integration testing starts are the following:

1. All components for integration testing are available.
2. All components are unit tested and all unit test bugs are fixed.
3. For all components code inspection was performed.

The following documents must be prepared before integration testing:

1. Requirement Analysis and Specification Document
2. Design Document
3. Integration Testing Plan Document (this document)

2.2. Elements to be Integrated

The main high level components of the system are the following:

- **Datastore:** this data layer provides data management functions like transactions, atomicity, data reliability and scalability.
- **Application server:** this layer implements all the application logic for the system. Search algorithms, optimization, data management and formatting, policies, user permissions are performed here.
- **Mobile application:** it is a presentation layer with few computational capabilities mostly used to represent raw data in UIs. It connects directly to API provided by the application server to obtain information.
- **Web browser:** this presentation layer let the user asks for web pages provided by the application server. When web pages runs on the user browser they may ask directly to the application server information through API calls in an asynchronous way.

Considering system architecture integration testing will be performed for integration between high level components (subsystems) and also for integration between components of application layer..

2.3. Integration Testing Strategy

For integration testing of the software system was chosen bottom-up approach. This decision was made based on prerequisite requiring that all the components of the system are fully coded and unit tested before integration testing start. This approach makes localization of errors much easier but requires the implementation of drivers calling the tested modules.
But because of cyclic dependency among several modules in the tested system we will also use stubs when necessary.

The client applications and the application server are connected through REST API available over the HTTP protocol. Hence it will not be difficult to integrate the client part in the system, testing it separately.

Our system will use a non-relational database based on object database interaction model. It uses programming object to manage and operate on remote data adopting the API of the Google Datastore. For this reason we will just unit test the database entities (i.e. Task, Car, etc...) to assure that they interact well with the Datastore and then we will do integration testing just with tested entities.

The PermissionController is not included in the integration test diagram because since it follows the *decorator* pattern it wraps each controller but does not interact with them. So it is enough to test it just with unit tests.

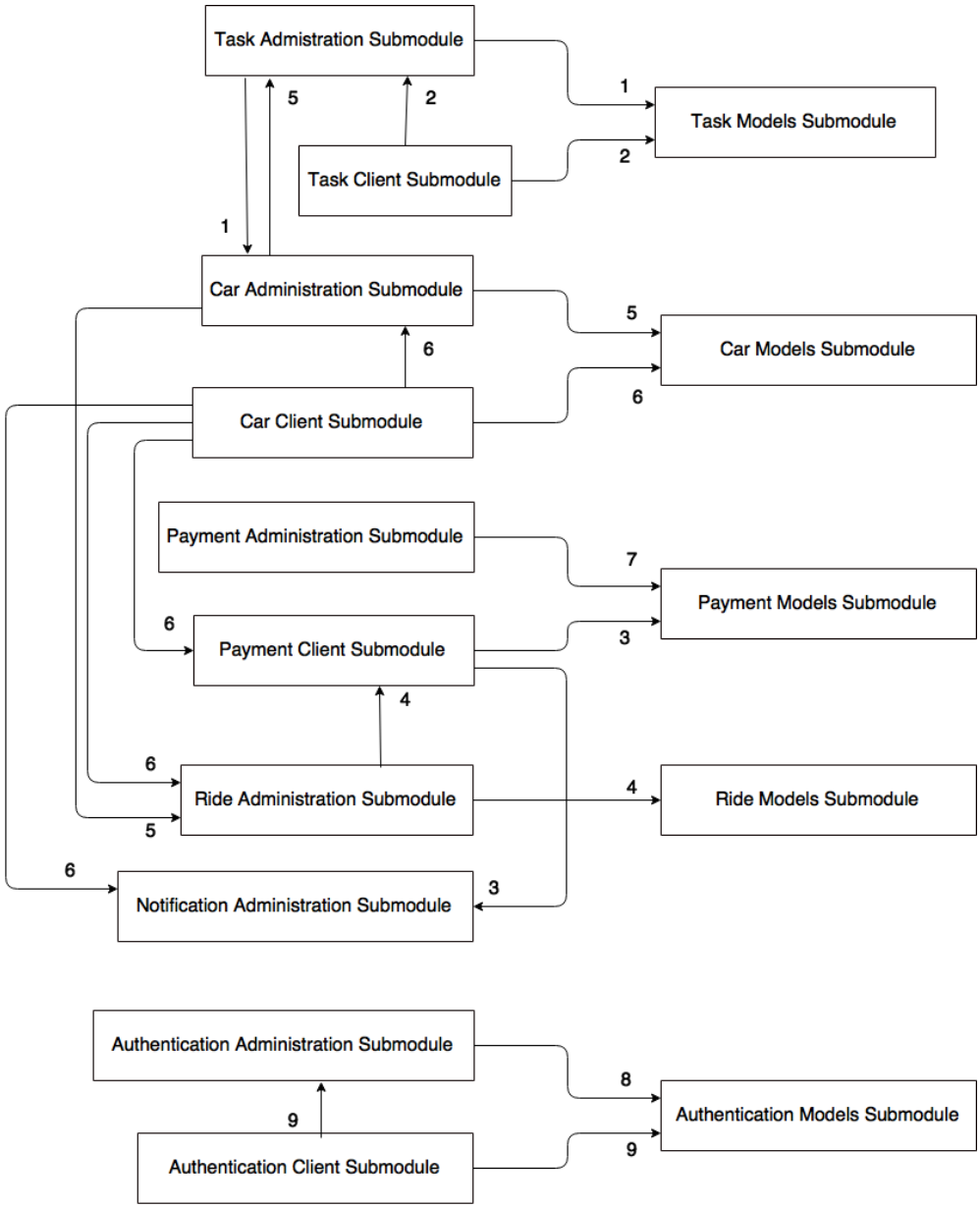

Figure 2.1. Diagram of the submodules integration

2.4. Sequence of Components/Function Integration

2.4.1. Software Integration Sequence

The table 2.1. and the Figures 2.1. and 2.2 show the integration sequence that we will use to test our system.
The order of testing single items is chosen in order to test before the most independent items going to the least independent ones. This strategy is useful to avoid the implementation of useless stubs because once the most independent components are tested they can be exploited to make tests over the less independent components.

N.	Subsystem	Component	Integrates with
I1T1	Application server	Task Administration Submodule	Task Models Submodule
I1T2	Application server	Task Administration Submodule	Car Administration Submodule
I2T1	Application server	Task Client Submodule	Task Models Submodule
I2T2	Application server	Task Client Submodule	Task Administration Submodule
I3T1	Application server	Payment Client Submodule	Payment Models Submodule
I3T2	Application server	Payment Client Submodule	Notification Administration Submodule
I4T1	Application server	Ride Administration Submodule	Ride Models Submodule
I4T2	Application server	Ride Administration Submodule	Payment Client Submodule
I5T1	Application server	Car Administration Submodule	Car Models Submodule
I5T2	Application server	Car Administration Submodule	Task Administration Submodule
I5T3	Application server	Car Administration Submodule	Ride Administration Submodule
I6T1	Application server	Car Client Submodule	Car Models Submodule
I6T2	Application server	Car Client Submodule	Car Administration Submodule
I6T3	Application server	Car Client Submodule	Notification Administration Submodule
I6T4	Application server	Car Client Submodule	Ride Administration Submodule
I6T5	Application server	Car Client Submodule	Payment Client Submodule
I7T1	Application server	Payment Administration Submodule	Payment Models Submodule
I8T1	Application server	Authentication Administration Submodule	Authentication Models Submodule
I9T1	Application server	Authentication Client Submodule	Authentication Models Submodule

I9T2	Application server	Authentication Client Submodule	Authentication Administration Submodule

Table 2.1: Integration of the system components.

2.4.2. Subsystem Integration Sequence

The integration sequence of the subsystems is described in Table 2.2 and in Figure 2.2.

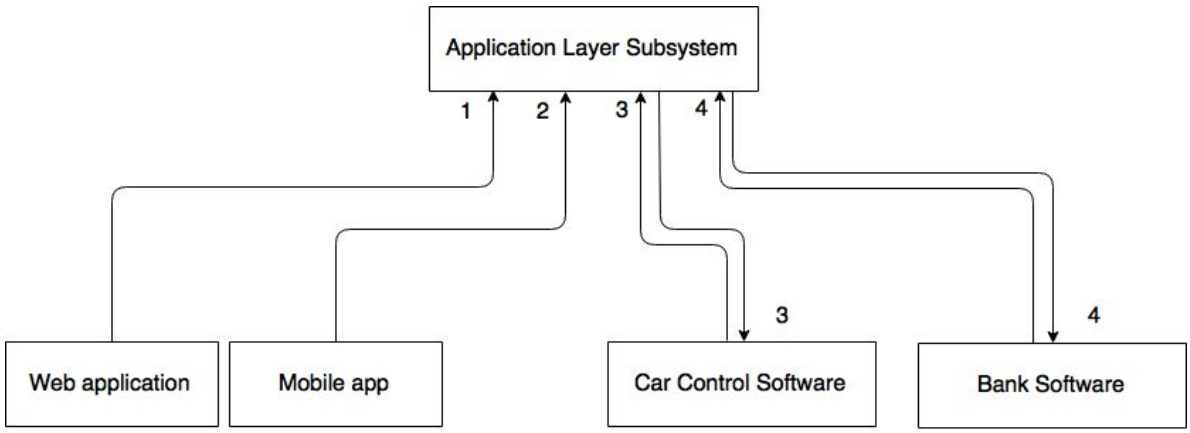

Figure 2.2. Diagram of the subsystems integration

N	**System**	**Integrates with**
SI1T1	Web Application	Application Layer Subsystem
SI2T1	Mobile Application	Application Layer Subsystem
SI3T1	Car Control Software	Application Layer Subsystem
SI3T2	Application Layer Subsystem	Car Control Software
SI4T1	Bank Software	Application Layer Subsystem
SI4T2	Application Layer Subsystem	Bank Software

Table 2.2: Integration of the system components.

3. Individual Step and Test Description

The following code samples aim to give an idea of how algorithms in the project will be implemented.

3.1. Integration tests for application layer modules

3.1.1. Integration test case I1: Task Administration Submodule

Test Case Identifier	I1T1
Tested Integration	Task Administration Submodule → Task Models Submodule
Input	Typical calls of Task Administration Submodule that interact with the Task Models Submodule to create or delete tasks in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Task Administration Submodule driver.
Description	The driver calls the proper methods of the Task Administration Submodule components to ensure that they interact correctly with Task Models.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I1T2
Tested Integration	Task Administration Submodule → Car Administration Submodule
Input	Typical calls to the methods of Task Administration Submodule that interact with Car Administration Submodule to check the state of the car.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Car Administration Submodule Stub. Task Administration Submodule driver.

Description	The driver calls the proper methods of the Administration Submodule components to ensure that they interact correctly with the Car Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.2. Integration test case I2: Task Client Submodule

Test Case Identifier	I2T1
Tested Integration	Task Client Submodule → Task Models Submodule
Input	Typical calls of Task Client Submodule that interact with the Task Models Submodule to modify tasks in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Task Client Submodule driver
Description	The driver calls the proper methods of the Task Client Submodule components to ensure that they interact correctly with Task Models.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I2T2
Tested Integration	Task Client Submodule → Task Administration Submodule
Input	Typical calls to the methods of Task Client Submodule that interact with Task Administration Submodule in order to change the task status.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Task Client Submodule driver. I1 succeeded.
Description	The driver calls the proper methods of the Task Client Submodule components to ensure that they interact correctly with the Task Administration Submodule components

	methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.3. Integration test case I3: Payment Client Submodule

Test Case Identifier	I3T1
Tested Integration	Payment Client Submodule → Payment Models Submodule
Input	Typical calls of Payment Client Submodule that interact with the Payment Models Submodule to get information about charges and manage information about payments in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Payment Client Submodule driver
Description	The driver calls the proper methods of the Payment Client Submodule components to ensure that they interact correctly with Payment Models.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I3T2
Tested Integration	Payment Client Submodule → Notification Administration Submodule
Input	Typical calls to the methods of Payment Client Submodule that interact with Notification Administration Submodule in order to request sending of messages to users.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Payment Client Submodule driver.

Description	The driver calls the proper methods of the Payment Client Submodule components to ensure that they interact correctly with the Notification Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.4. Integration test case I4: Ride Administration Submodule

Test Case Identifier	I4T1
Tested Integration	Ride Administration Submodule → Ride Models Submodule
Input	Typical calls of Ride Administration Submodule that interact with the Ride Models Submodule to manage entities of rides in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Ride Administration Submodule driver
Description	The driver calls the proper methods of the Ride Administration Submodule components to ensure that they interact correctly with Ride Models Submodule.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I4T2
Tested Integration	Ride Administration Submodule → Payment Client Submodule
Input	Typical calls to the methods of Ride Administration Submodule that interact with Payment Client Submodule in order to request calculation of the payment.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Ride Administration Submodule driver. I3 succeeded.
Description	The driver calls the proper methods of the Ride Administration Submodule components to ensure that they

	interact correctly with the Payment Client Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.5. Integration test case I5: Car Administration Submodule

Test Case Identifier	I5T1
Tested Integration	Car Administration Submodule → Car Models Submodule
Input	Typical calls of Car Administration Submodule that interact with the Car Models Submodule to manage entities of cars in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Car Administration Submodule driver
Description	The driver calls the proper methods of the Car Administration Submodule components to ensure that they interact correctly with Car Models.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I5T2
Tested Integration	Car Administration Submodule → Task Administration Submodule
Input	Typical calls to the methods of Car Administration Submodule that interact with Task Administration Submodule in order to request creation or deletion of the service task.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Car Administration Submodule driver. I1 succeeded.
Description	The driver calls the proper methods of the Car Administration Submodule components to ensure that they interact correctly with the Task Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I5T3
Tested Integration	Car Administration Submodule → Ride Administration Submodule
Input	Typical calls to the methods of Car Administration Submodule that interact with Ride Administration Submodule in order to request creation of the new ride.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Car Administration Submodule driver. I4 succeeded.
Description	The driver calls the proper methods of the Car Administration Submodule components to ensure that they interact correctly with the Ride Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.6. Integration test case I6: Car Client Submodule

Test Case Identifier	I6T1
Tested Integration	Car Client Submodule → Car Models Submodule
Input	Typical calls of Car Client Submodule that interact with the Car Models Submodule to get information about cars and modify car status in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Car Client Submodule driver
Description	The driver calls the proper methods of the Car Client Submodule components to ensure that they interact correctly with Car Models.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I6T2
Tested Integration	Car Client Submodule → Car Administration Submodule
Input	Typical calls to the methods of Car Client Submodule that interact with Car Administration Submodule to request information from the car.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Car Client Submodule driver. Car Control Software Stub. I5 succeeded.
Description	The driver calls the proper methods of the Car Client Submodule components to ensure that they interact correctly with the Car Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I6T3
Tested Integration	Car Client Submodule → Notification Administration Submodule
Input	Typical calls to the methods of Car Client Submodule that interact with Notification Administration Submodule in order to request sending of messages.
Output	The actions expected to be done by calling methods are well performed.
Environmental requirements	Car Client Submodule driver.
Description	The driver calls the proper methods of the Car Client Submodule components to ensure that they interact correctly with the Notification Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I6T4
Tested Integration	Car Client Submodule → Ride Administration Submodule
Input	Typical calls to the methods of Ride Administration Submodule to trigger the start of the ride.
Output	The actions expected to be done by calling methods are well performed.
Environmental requirements	Car Client Submodule driver.
Description	The driver calls the proper methods of the Car Client Submodule components to ensure that they interact correctly with the Ride Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I6T5
Tested Integration	Car Client Submodule → Payment Client Submodule
Input	Typical calls to the methods of Car Client Submodule that interact with Payment Client Submodule to apply the fee in case of reservation expiration.
Output	The actions expected to be done by calling methods are well performed.
Environmental requirements	Car Client Submodule driver.
Description	The driver calls the proper methods of the Car Client Submodule components to ensure that they interact correctly with the Payment Client Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.7. Integration test case I7: Payment Administration Submodule

Test Case Identifier	I7T1
Tested Integration	Payment Administration Submodule → Payment Models Submodule
Input	Typical calls of Payment Administration Submodule that interact with the Payment Models Submodule to manage information about charges in the Datastore and to submit payment to Bank Software.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Payment Administration Submodule driver. Bank Software stub.
Description	The driver calls the proper methods of the Payment Administration Submodule components to ensure that they interact correctly with Payment Models and with the Bank Software.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.8. Integration test case I8: Authentication Administration Submodule

Test Case Identifier	I8T1
Tested Integration	Authentication Administration Submodule → Authentication Models Submodule
Input	Typical calls of Authentication Administration Submodule that interact with the Authentication Models Submodule to manage accounts entities in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Authentication Administration Submodule driver
Description	The driver calls the proper methods of the Authentication Administration Submodule components to ensure that they interact correctly with Authentication Models Submodule components.

Method	Automated by Pytest through the use of driver generated by Mockito.

3.1.9. Integration test case I9: Authentication Client Submodule

Test Case Identifier	I9T1
Tested Integration	Authentication Client Submodule → Authentication Models Submodule
Input	Typical calls of Authentication Client Submodule that interact with the Authentication Models Submodule to get information about accounts entities in the Datastore.
Output	The requested actions on the model are performed. Entities in the Datastore are modified.
Environmental requirements	Authentication Client Submodule driver
Description	The driver calls the proper methods of the Authentication Client Submodule components to ensure that they interact correctly with Authentication Models Submodule components.
Method	Automated by Pytest through the use of driver generated by Mockito.

Test Case Identifier	I9T2
Tested Integration	Authentication Client Submodule → Authentication Administration Submodule
Input	Typical calls to the methods of Authentication Client Submodule that interact with Authentication Administration Submodule in order to request creation or deletion of the account.
Output	The actions expected to be done by calling methods are well performed. Expected data is received as response.
Environmental requirements	Authentication Client Submodule driver. I8 succeeded.
Description	The driver calls the proper methods of the Authentication

	Client Submodule components to ensure that they interact correctly with the Authentication Administration Submodule components methods.
Method	Automated by Pytest through the use of driver generated by Mockito.

3.2. Subsystems integration test cases

3.2.1. Integration test case SI1

Test Case Identifier	SI1T1
Tested Integration	Web Application → Application Layer Subsystem
Input	Typical and well-formed HTTPS requests. Erroneous requests.
Output	Application layer should respond in a proper way according to the requests. In case of erroneous requests it should response with proper error messages.
Environmental requirements	Application layer is fully implemented and tested. Web application driver. API client tester.
Description	To check the correctness of the received data, the response will be compared with the expected output for it.
Method	Automated by Pytest through calls to the web application driver and to the API client tester.

3.2.2. Integration test case SI2

Test Case Identifier	SI2T1
Tested Integration	Mobile Application → Application Layer Subsystem
Input	Typical and well-formed HTTPS requests. Erroneous requests.
Output	The application layer should respond in a proper way

	according to the requests. In case of erroneous requests it should response with proper error messages.
Environmental requirements	Application layer is fully implemented and tested. API client tester.
Description	To check the correctness of the received data, the response will be compared with the expected output for it.
Method	Automated by Pytest through calls to the API client tester.

3.2.3. Integration test case SI3

Test Case Identifier	SI3T1
Tested Integration	Application Layer Subsystem → Car Control Software
Input	Typical HTTPS requests to the Car Control Software.
Output	The Car Control Software should respond in a proper way according to the requests
Environmental requirements	Application layer is fully implemented and tested.
Description	To check the correctness of the received data, the response will be compared with the expected output for it.
Method	Automated by Pytest calling the methods that interact with the Car Control Software.

Test Case Identifier	SI3T2
Tested Integration	Car Control Software → Application Layer Subsystem
Input	Typical HTTPS requests to the Application Layer Subsystem.
Output	Requests should call the correct methods, are called in the Application Layer Subsystem.
Environmental requirements	Existing and working driver to emulate the Car Control Software behaviour in order to test how the Application Layer behaves when HTTPS calls are received. Application layer is fully implemented and tested.
Description	To check if the sent data by the Car Control Software is correctly interpreted and correct methods are called.
Method	Automated by Pytest using the driver to issue the requests.

3.2.4. Integration test case SI4

Test Case Identifier	SI4T1
Tested Integration	Application Layer Subsystem → Bank Software
Input	Typical HTTPS requests to the Bank Software.
Output	The Bank will show the payment page to the user.
Environmental requirements	Application layer is fully implemented and tested.
Description	To check the correctness of the request submitted to the bank.
Method	Automated by Pytest.

Test Case Identifier	SI4T2
Tested Integration	Bank Software → Application Layer Subsystem
Input	Typical HTTPS requests to the Application Layer Subsystem.
Output	The Application Layer Subsystem understands the request and the correct methods are called.
Environmental requirements	Existing and working driver to emulate the Bank Software behaviour in order to test how the Application Layer behaves when HTTPS calls are received. Application layer is fully implemented and tested.
Description	To check if the sent data by the Bank Software is correctly interpreted and correct methods are called.
Method	Automated by Pytest using the driver to issue the requests.

4. Tools and Test Equipment Required

We suppose to use the following testing tools for this plan fulfillment:

1. **Mockito for Python**. Mockito is the open source testing framework for creating stubs and drivers.
2. **Pytest**. Pytest makes easy to write small tests but it is also useful to write complex test like integration tests. We will use it for integration testing.

5. Program Stubs and Test Data Required

To fulfil the integration tests in a feasible way we have to develop some stubs and some drivers to substitute the parts of the software that are not yet implemented while running tests.

5.1. Drivers

Drivers for Submodules. These drivers will be implemented with the use of Mockito for Python.
They will substitute missing Submodules of the application layer.

API client tester. Using the python module called “reqeusts” we will implement a light API client tester. We want create this particular driver in order to test the REST API of the Applciation Layer without the actual client application. This driver needs to be scriptable to allow tests automation with Pytest.

5.2. Test data

Test data used to run tests will be stored only in the development instance of the datastore. The development instance is generated by the Google App Engine SDK in the development enviroment. It allows to have a local and working version of the Datastore that can be filled with sample data.
Some CSV files, containing test data, will be used as sources to fill the development version of the Datastore through dedicated scripts that developers can run each time they want to rebuild a fresh copy of sample data.
The test data structure will be the same of the future real data and this one is described in the Entity-Relation diagram of the Design Document which is reported in Figure 5.1.
We want to remember that the relations shown in the diagram do not really exists in the Datastore. They are implemented through the application server logic programmatically since the Datastore is a non-relational DBMS.

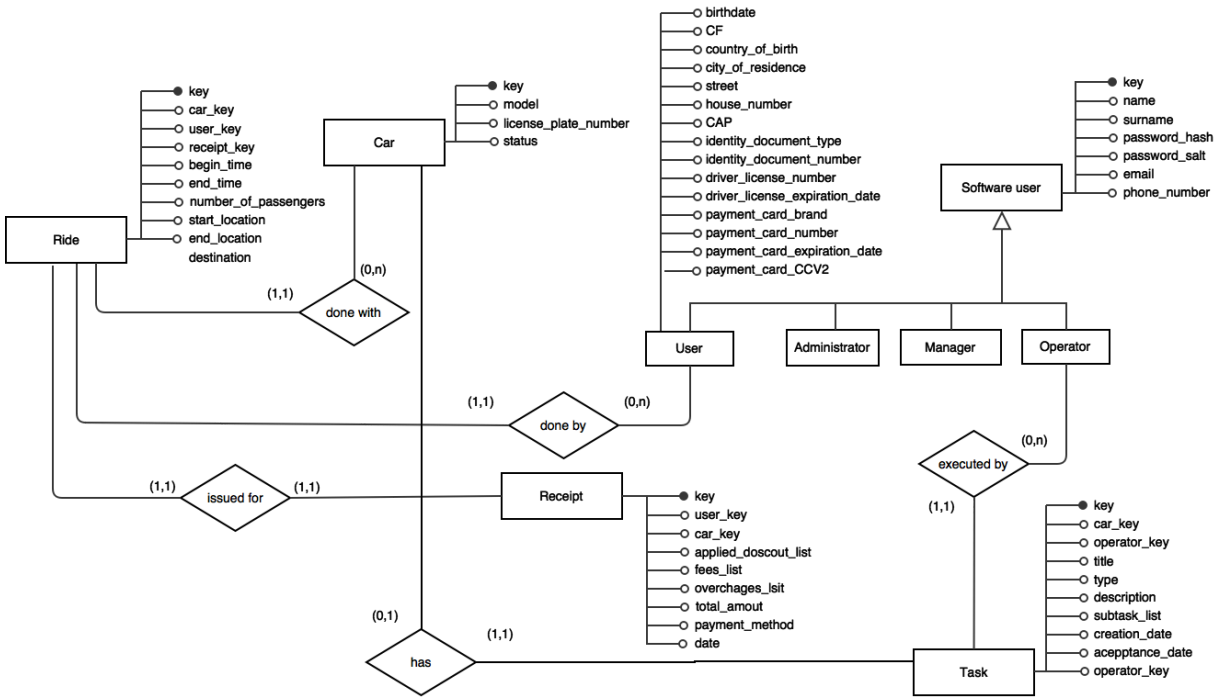

Figure 5.1. Entity-Relationship diagram.

6. Hours of Work

6.1. Elena Denchenko

- 22/12/2016: 3h
- 02/01/2017: 4h
- 05/01/2017: 1h 30min
- 09/01/2017: 2h
- 12/01/2017: 1h
- 15/01/2017: 2h 30min

6.2. Stefano Cilloni

- 22/12/2016: 3h
- 02/01/2017: 4h
- 07/01/2017: 2h
- 09/01/2017: 2h
- 14/01/2017: 1h
- 15/01/2017: 2h

7. Changelog