Politecnico di Milano

2015-2016

Software Engineering 2: “MyTaxiService”
Integration Test Plan Document

version 1.0

Author: Nenad Petrovic

21st January 2016

Contents

1 Introduction 3

1.1 Revision history 3

1.2 Purpose and scope 3

1.3 Definitions, acronyms, abbreviations 3

1.4 Reference documents 7

2 Integration strategy 7

2.1 Entry criteria 7

2.2 Elements to be integrated 11

2.2.1 Server side elements 12

2.2.2 Client side elements 14

2.3 Integration testing strategy 16

2.4 Sequence of component/function integration 19

3 Individual steps and test description 35

3.1 Test case specifications 36

3.2 Test procedures 59

4 Tools and test equipment required 62

4.1 Software tools 62

4.2 Equipment 62

5 Program stubs/drivers and test data required 64

5.1 Stubs and drivers 64

5.2 Test data 66

6 Software and tools used for document creation 68

7 Hours of works 68

8 References 68

1 Introduction

1.1 Revision history

Title	Version	Author	Date	Summary
Integration Test Plan Document	1.0	Nenad Petrovic	13-01-2016	The first version of the document

1.2 Purpose and scope

The Integration Test Plan Document (ITPD) that is going to be presented describes the plans for testing the integration of the created components.

This document is supposed to be written before the integration test happens, and takes the architectural description from Design Dociment as a starting point.

The purpose of this document is to test the interfaces between the components as described in Design Document.

This software implements a new online taxi scheduling service that would help the government of a large city to simplify the accesss of passengers to the taxi drivers, and would also guarantee a fair management of taxi queries at the same time by offering applications to both user and taxi-driver side with corresponding functionalities for each of them.

This document aims every team member who cooperates in the integration test, and should be read by them.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- Starting point: is a location where the drive should start from, determined by its GPS coordinates.
- Ending point: is a location where the current drive stops, determined by its GPS coordinates. At this point, taxi driver's availability changes automatically to available.
- Request: is a message which consists of user's desired destination for a taxi drive and, optionally, maximum waiting time . The user himself is a „sender" of a request, while the taxi driver who receives the request is called „receiver". If there is no taxi driver who can receive the request, the receiver part of the message is empty (user receives a message stating that there is no taxi available and gives him/her an option to send the same request again or change the desired destination), and in case of forwarding the message to another taxi driver, system changes the receiver to the taxi driver polled from a queue. The request constains starting point-determined

by sender's GPS location and ending point- which is, in fact the desired destination selected by user.

- Response: is a message which contains „y"-yes in case where taxi driver accepts the request or „n"-no in case where taxi driver rejects the request by user. This message also contains the estimated time needed for taxi driver to come to the requested starting point and estimated price. In this case, taxi driver has a role of sender, while the user who sent the request is receiver. Receiver part of this message can't be blank. User can accept or reject the drive offer.

- Report:is a message written by user or taxi driver during the drive event, in order to mention bad behaviour of the other side. There is no strict definition for „bad behaviour", so the one who reports has to write reason and describe the situation itself as close as possible. After that, administrators can view the reports and decide to delete user from system or not. User that is banned from system is prevented from using it and can't login or register once again, because his/her fiscal code is on the „black list". Single user can receive many reports. Reports are stored in database and could be viewed by administration.

- Taxi zone: a part of city (approximately $2km^2$), which is defined by its center point, and its boundaries are calculated and managed by the system. Each taxi zone has its taxi queue, which consists of car identification numbers of available taxi drivers whose current location belongs to its boundaries. The city has at least one taxi zone (in case of a very small town).

- Availability: could have value „y"(yes)- which means „available" or „n"(no)- which means „unavailable". If taxi driver is available, he/she is placed in taxi queue related to his/her taxi zone and is able to respond to requests belonging to this taxi zone. If taxi driver is unavailable, he/she can't receive requests. When taxi driver accepts the request, his availability switches to „unavailable". After finishing the drive, the availability switches to „available" and is placed in a queue belonging to his/her current taxi zone. In certain cases, when something goes wrong (traffic rush, accident etc.),taxi driver can manually switch to „unavailable". When taxi driver sends S.O.S signal, his/her status changes to „unavailable" automatically.

- Drive event: is a system abstraction of a taxi drive, and consists of request by user, driver's response and reports during a drive. There could be many reports, or no reports at all. Administrations can browse the database of drive events and see the actors of a drive event related to request and response. During active drive event, user can report driver, or driver can report user for bad behavior by describing the reason of report as close as possible. Drive event starts after the user accepts the offer by taxi driver. Sometimes, it is called simply "drive" in this text.

- Estimated waiting time: time needed for a taxi driver to come to the user's current location.

- Maximum waiting time: maximum time that is user ready to spend waiting for a taxi to come.

- Bad behaviour: aggresion, avoiding payment, offensive beahviour etc.

- Black-box testing: assumes that the structure of the software being tested is unknown. Tests are based on knowledge of the system specification.

- White-box testing: based on the knowledge of the software and attemps to identify test sets that cause execution of all the instructions or all the branches or specfic paths in system.

1.3.2 Acronyms

- API: Application Programming Interface.
- DD: Design Document.
- DBMS: DataBase management system.
- DB: DataBase.
- EJBs: Enterprise JavaBeans (EJB) is a managed, server software for modular construction of enterprise software, and one of several Java APIs. EJB is a server-side software component that encapsulates the business logic of an application. The EJB specification is a subset of the Java EE specification.
- GUI: Graphical User Interface
- HTTPS (HTTP over SSL, and HTTP Secure) is a protocol for secure communication over a computer network which is widely used on the Internet. HTTPS consists of communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security or its predecessor, Secure Sockets Layer. The main motivation for HTTPS is authentication of the visited website and protection of the privacy and integrity of the exchanged data.
- JPA:The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.
- JAX-RS: Java API for RESTful Web Services (JAX-RS) is a Java programming language API that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern.
- JEE: Java Enterprise Solution
- JSF:

Java Server Faces (JSF) is a Java specification for building component-based user interfaces for web applications.[1] It was formalized as a standard through the Java Community Process and is part of the Java Platform, Enterprise Edition.

JSF 2 uses Facelets as its default templating system. Other view technologies such as XUL can also be employed. In contrast, JSF 1.x uses JavaServer Pages (JSP) as its default templating system.

- JVM: Java Virtual Machine
- OS: Operating System.
- MVC: Model View Controller
- REST (REpresentational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services.

- Enterprise Bean[1]

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called checkInventoryLevel and orderProduct. By invoking these methods, clients can access the inventory services provided by the application.

Benefits of Enterprise Beans

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services such as transaction management and security authorization.

Second, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant Java EE server provided that they use the standard APIs.

When to Use Enterprise Beans

You should consider using enterprise beans if your application has any of the following requirements:

The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.

Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.

The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

Types of Enterprise Beans

Table summarizes the two types of enterprise beans. The following sections discuss each type in more detail.

Table Enterprise Bean Types

Enterprise Bean Type	Purpose

[1] http://docs.oracle.com/javaee/5/tutorial/doc/bnblt.html

Session	Performs a task for a client; optionally may implement a web service
Message-Driven	Acts as a listener for a particular messaging type, such as the Java Message Service API

1.4 Reference documents

This document referecnces to previously created document – RASD and DD.

- Requirements and Analysis Specification Document („RASD 1.4.pdf“).
- Design Document („DD.pdf“).
- Integration Test Plan Document („ITPD.pdf“) [this document]

2 Integration strategy

2.1 Entry criteria

In order to begin integration test, it is necessary to provide:

-Requirements Analysis and Specification Document

-Design Document, which includes both architectural design and detailed design.

-Perform unit tests of functions that belong to corresponding components (unit test plan)[2].

-Integration test plan (this document)

Now, a few words about unit tests.

When it comes to unit tests, systematic testing is going to be used, that uses characteristics of the software artifacts and information about the beahviour of the system.

In what follows, the functions that are going to be part of unit testing are listed for each of the subsystems:

[2]I assume that Unit Test Plan is already done, because UTPD (Unit Test Plan Document) was not previously developed as an assignment or part of any assignment.

Administrator

1.1 Browse users

void onClickBrowse(Event e)

Users[] browseUsers(string username, string firstName, string lastName, string fiscalCode)

1.2 View reports

void onSelectUser(Event e)

Report[] viewReports (string userid)

1.3 View drives

void onSelectUser(Event e)

Drive[] viewReports (string userid)

1.4 View drive and view report

void onSelectDrive(Event e)

void onSelectReport(Event e)

void viewDrive(string driveId)

void viewReport(string reportId)

Drive viewDrive (string userid)

Report viewReport (string userid)

1.5 Delete user

bool deleteUser(string userid)

1.6 Promote/downgrade

void onUpdateUser(Event e)

bool updateUser(userid,carId,carModel,licenseNumber)

Bool isEligible(string carId, string licenseId)

bool downgradeUser(userid)

bool isValidCarId(string carId)

bool isValidLicenseNumber(string licenseNumber)

2. Guest

2.1 Register

void onClickRegister(Event e)

User registerUser(string username, string firstName, string lastName, string fiscalCode, string password, string confirmPass, string picturePath, character Gender, string mobilePhone)

bool isValidFiscalCode(string fiscalCode)

void createSession(string username)

2.2 Log in

void onClickLogin(Event e)

User login(username, password)

void createSession(string username)

3. Guest

3.1 Edit profile

void onClickModify(Event e)

User ModifyUser(user.username, firstName, lastName, fiscalCode, password, confirmPass, picturePath, Gender, mobilePhoneNumber)

3.2 Report

void onClickReport(Event e)

Report newReport(string username1, string username2, string reason)

TaxiDriver getTaxiDriver(string username)

3.3 Log out

void onClickLogout(string username)

void destroySession(string username)

3.4 Request taxi

void onClickSubmitRequest(Event e)

Request makeRequest(User u, String maxWaiting, Location statpoint, Location endpoint)

3.5 Respond offer

Void onRespondOffer(Event e)

Drive handleResponse(Response response)

4. ITaxi

4.1 Change availability

void onAvailabilityChange(Event e)

bool changeAvailability(string taxiId)

4.2 Respond request

bool onClickButton(Event e)

TaxiDriver handleTaxiResponse(Response response)

float estimatePrice (startpoint, endpoint, timestamp)

time estimatedWaitingTime(startpoint, endpoint)

Response makeOffer(TaxiDriver taxidriver, User user)

6. GPS-related functions

float distance(location startpoint, location endpoint)

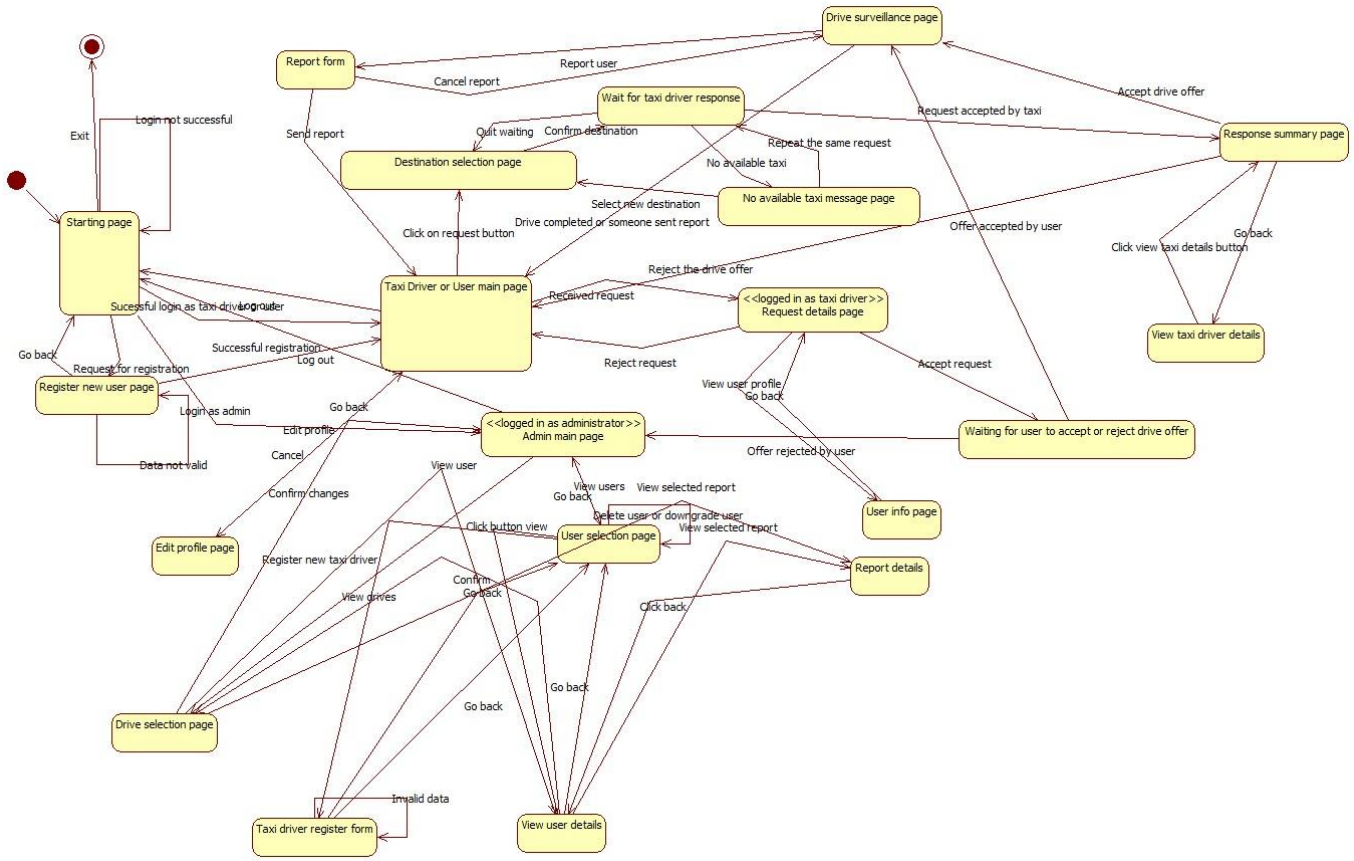
Location getUserLocation(string id)

Time estimateTime(Location startpoint, Location endpoint)

There are two main classes of test sets: white-box and black-box.

For unit testing, it is more common to choose the first approach, beacause covering of small portions of code is possible, while, for integration testing, it is more suitable to use black-box method, because it helps to identify the missing funcionalities.

As a guide, the state diagram below is used:



The steps in deriving test cases are:
1.Analyze a state diagram
2. Identify test cases
3. Check that test cases fulfill a coverage criterion
4. Execute test cases on the actual systems

All states and transitions should be covered.
State coverage means that every state in the model should be visited by at least one test case.Transition coverage means that every transition between states should be traversed by at least one test case. This is the most commonly used criterion. A transition can be thought of as a (precondition, postcondition) pair.

2.2 Elements to be integrated

This part of the document deals with elements that are going to be integrated and references to Design Document. These elements are described in details in Design Document and are going to be listed here, with brief overview.

In general, the system MyTaxiService consists of several subsystems:

-DataLayer subsystem

-Session manager subsystem

-Guest subsystem

-Scheduler subsystem

-User subsystem

-Taxi driver subsystem

-Admin subsystem

-Developer API subsystem

Some of the subsystems have components on both client and server side, which interact in order to achieve their goals.

The set of components is split into two subsets – the first one consists of all the elements from the server side, while another one is related to the elements contained on the client side and they are going to be presened in this order.

2.2.1 Server side elements

First, let's take a look at the elements from the server side:

1.UnregisteredManager is an element which gives ability unregistered users to register or log in. This part of the server uses external city government databases through REST service in order to check the validity of fiscal code. It also uses DataLayer to check if there is same user already in a database or the user is banned and can't be registered. In case of successful registration, this component uses DataLayer to add new users to database once they are registered. This element, as it can be seen, doesn't use the scheduler, as unregistered users can't make taxi requests.

It has two smaller components:

-register component

-login component

2.RegisteredUserManager is an element which enables users to logout , receives all the messages from users - drive requests, drive offer responses, reports, SOS signal, modify their profile, log out and all other functions that are present in RASD usecase diagrams. This component can access GPS coordinates service in order to determine the users location when necessary. In order to acomplish all of the operations, this component interacts with the Datalayer and visualises all the necessary elements when it is needed (maps, etc.). Also, this component uses Scheduler in order to serve the users in requests related to taxi drives.

Considering further, this component could be split into smaller components:

-profile management component, which includes functions related to profile editing

-taxi request component, which includes functions related to making a taxi request

-respond offer component, which is a part of negotiations between taxi driver and user

-report component, responsible for making reports and writing related reason during taxi drive

3.TaxiDriverManager is similar to previous one, but also includes some additional options, like accepting or rejecting a drive request and changing the availability status, in order to support TaxiDriver clients.

Considering further, this component could be split into smaller components:

-profile management component, which includes functions related to profile editing

-taxi request component, which includes functions related to making a taxi request

-respond offer component, which is a part of negotiations between taxi driver and user

-report component, responsible for making reports during taxi drive and writing related reason

-drive accept/reject component, which is the one of the component where drivers differ from typical users, and gives them ability to accept or reject drive

-availability change component, which is used for managing taxi driver availability

4.AdminManager is supporting AdminClients, and gives them ability, using the DataLayer, to browse users, read reports, promote users to taxi drivers, prevent some users from being drivers, delete users in case of bad behaviour, and log out. Notice that Admin manager doesn't use Scheduler, as it isn't necessary (Administrators can't make taxi requests).

It could be split into smaller components:

-browsing and viewing users

-viewing drives related to users

-viewing reports

-deleting users

-promoting users to taxi drivers

5.Scheduler is the core element of the system that cross-references the location data and the system messages in order to dispatch a taxi from the corresponding taxi zone using the DataLayer. It uses GPS coordinates service in order to determine the taxi zone of the current user, deals with taxi queues and forwards the request to another taxi in current taxi zone if there is available taxi. It also

receives the drive offer responses from uses and creates drive events. Scheduler uses results from time and cost calculator when makes drive offers to users.

In fact, this is an element used by three subsystems: user, taxi driver, developer.

Scheduler has three smaller components:

-Time and cost calculator. It estimates the time the time and price for a taxi drive. It uses data from DataLayer in order to have enough information to do its part of job. By the way, the Scheduler uses this component when it makes a drive offer to user- so it makes this system fair. Users can accept or rejecet the offer if the price is too high.

-Taxi queue manager – component which deals with avaiable taxi drivers and dispatching, adding and removin them from queue

-Zone determination – component that determines the taxi zone corresponding to a given location.Uses maps external GPS and Maps API.

6.DataLayer encapsulates the operations related to persistence of entities relevant to the system – report, drive, different types of users – User, Taxi Driver, Administrator and deals with their storage into database. It consists of:

-query constructor

-persistence

-database operations

Datalayer is an element used by many subsytems.

7.DevelopmentManager is a part of the system which offers access to basic system functions to developers that are outside the system via some kind of serivce (REST, for example) in order to give them ability to embed or extend MyTaxiService in their applications. They can externally request a taxi, for example by API or register or log in from external aplplication (for example, Facebook). This element uses all other components in order to achieve its functionalities.

8.SessionManager, a part of the system that is responsible for creating and destroying sessions (log in- create session, log out-destroy session). It is necessary to have this part of the system included, because it distincts logged in and logged out state.

This is an element used by many subsystems.

2.2.2 Client side elements

Now, let's take a look at the elements which belong to the client side:

-GuestClient is a client component used by users before they register or log in. On the server side, this client type communicates with UnregisteredManager.

-UserClient is a client component used by users that have been already registered and logged in. This client communicates with RegisteredUserManager on the server side, which gives them a set of options – to select destination and make a taxi request, accept o reject taxi drive offer, modify

their profiles, report another user during the taxi drive and all the functionalities previously mentioned.

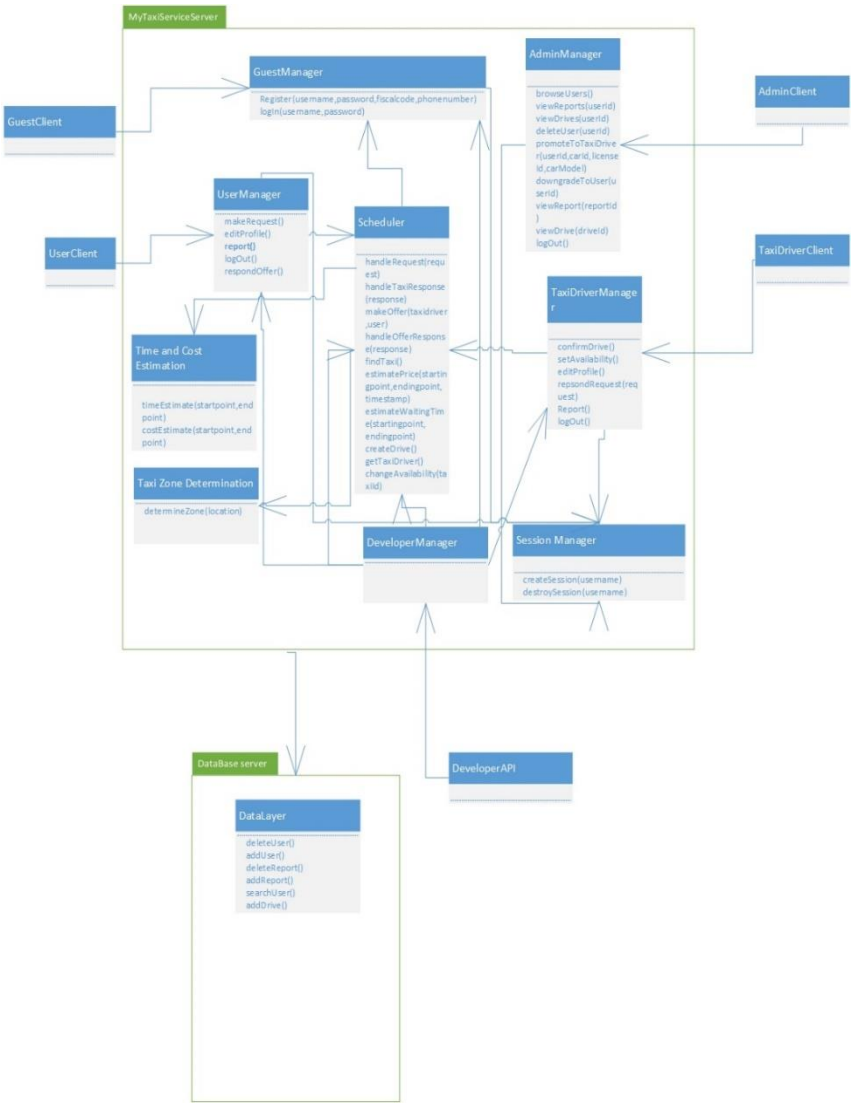
-TaxiDriverClient is client component used by users that have already been registered, logged in, and also promoted to taxi drivers by the administration. They have all the same functionalities and options as users, but also can accept or reject request for a taxi drive and change their availability status manually in case that some unexpected event occurs (accident,breakdown etc.). Taxi driver manager is a part of the server side which corresponds to this client.

-AdminClient is a client component used by the administration. This client offers them to browse users, promote them to taxi drivers by inserting the necessary data (license id, car id) , take their taxi driver privilegues in certain cases, read reports and block users. On the server side, we have AdminManager which deals with requests sent from AdminClient.

-DeveloperAPI is programmatic inteface component that gives ability to developers to use all the functionalities of MyTaxiSevice application, in order to make their own applications or integrate it into some bigger system. This element communicates with all of the server side elements, except the AdminManager, because it would be dangerous for service integrity and consistency.

As it could be seen, detailed descriptions of the smaller components are omitted here, because they are already mentioned in the part related to server side, and each of the client side components has its counterpart component on the server side, which is used to deal with the actions from the client side.

Once again, let us take a look at a system.



2.3 Integration testing strategy

Topic of this part of the document is integration testing startegy selection. Possible approaches are going to be discussed and selected to most appropriate one, according to the specific application, its requirements and architecture.

In general, there are two different orientations:

1.Structural orientation, where modules are constructed,integrated and tested based on a hierarchical project structure. Here, we have the following approaches:

-top-down

-bottom-up

-sandwich

-backbone

2.Functional orientation, where modules are integrated according to application characteristics or features:

-threads

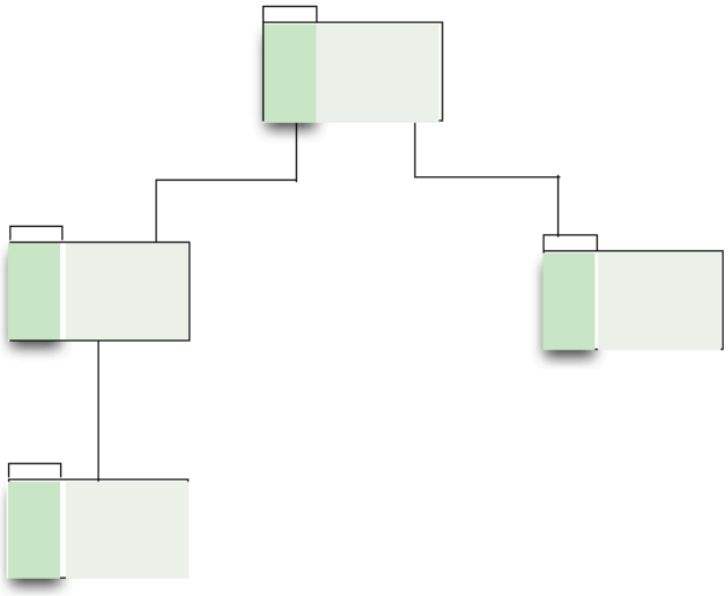
-critical module

Structural methods are more often used for simpler and smaller subsystems.

Combinations of thread and critical modules integration testing are often preferred for larger subsystems, and provide better process visibility. On the other side, structural methods are simpler.

In this case, a combination of different methods is going to be used – mainly a combiantion of thread and critical modules, when it comes to more complex components integration.

A thread is partion of several modules that together provide a user-visible feature. An illustration is displayed below:



Critical module strategy starts with riskiest modules in order to construct parts of one module to test funcionality in other.

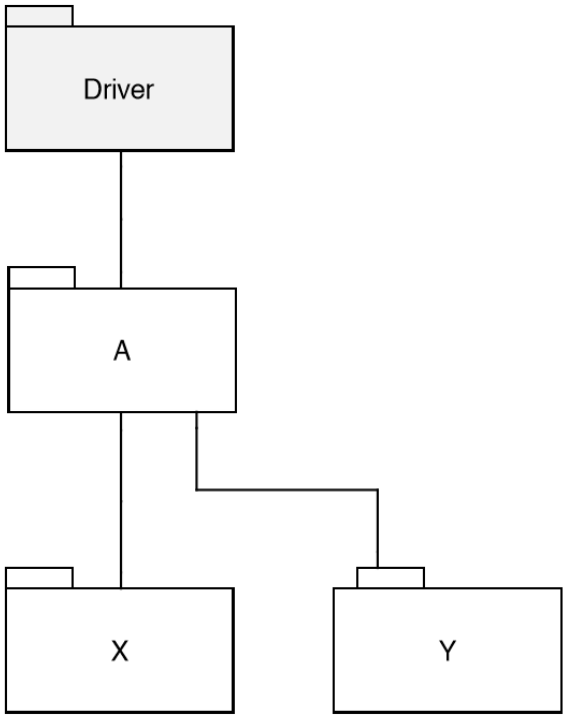
This startegy (Critical module) is going to be used for following modules: DataLayer and Session Manager, because they are the modules which are the necessary for integration of other subsystems of MyTaxiService system, and without them, it is not even possible to develop other functionalities and integrate the elements properly. For example, without DataLayer it is not possible to login or register users. Without Session Manager, it is possible to register users, but not to log them in.

For simpler integrations, a variant of bottom-up approach is going to be used – it is going to be started from the smaller components (bottom-level) , going towards bigger components (top-level). A subset of integrated simpler components is changed with more complex one that could be further integrated (driver).

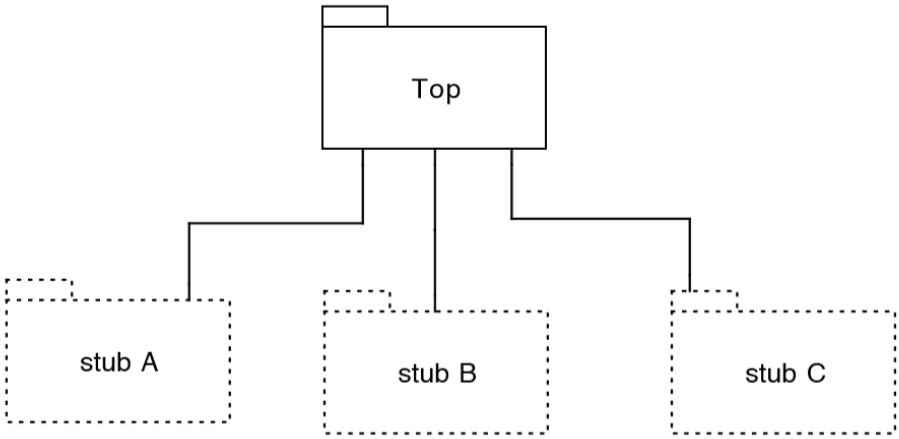
Threads strategy is going to be used for testing the subsystem related to specific users, which include a set of components and their interaction in order to achieve the functionality that is visible to the user. But, before that it is necessary to make sure that cirtical modules are working properly.

So, in general, it could be said that for more complex parts related to user-visible featues, when it comes to integration - threads (functional) strategy is going to be used, while, for smaller and simpler parts, bottom-up approach is going to be used (structural) while for GUI and API integration we will have top-down approach.

Bottom-up



Top-down: will be used for GUI and API integrations



2.4 Sequence of component/function integration

In what follows, a sequence of integration is going to be presented in the following order. For each subsystem the integration order is enumerated by I1,I2...IK. I1 stands for the first integration in a sequence, while IK stands for the last integration in this sequence.

2.4.1 Datalayer integration

First, the Datalayer subsystem is going to be integrated. It is used by all other subsystems in MyTaxiService and is the most critical part (critical module). Datalayer itself is not very complex component in terms of integration, so it could be integrated using some of the structural strategies (bottom-up).

The bottom component is the query execution subcomponent. It includes the functions related to CRUD operations.

Next level (upper level) is the component that deals with persistence – making objects using the query results.

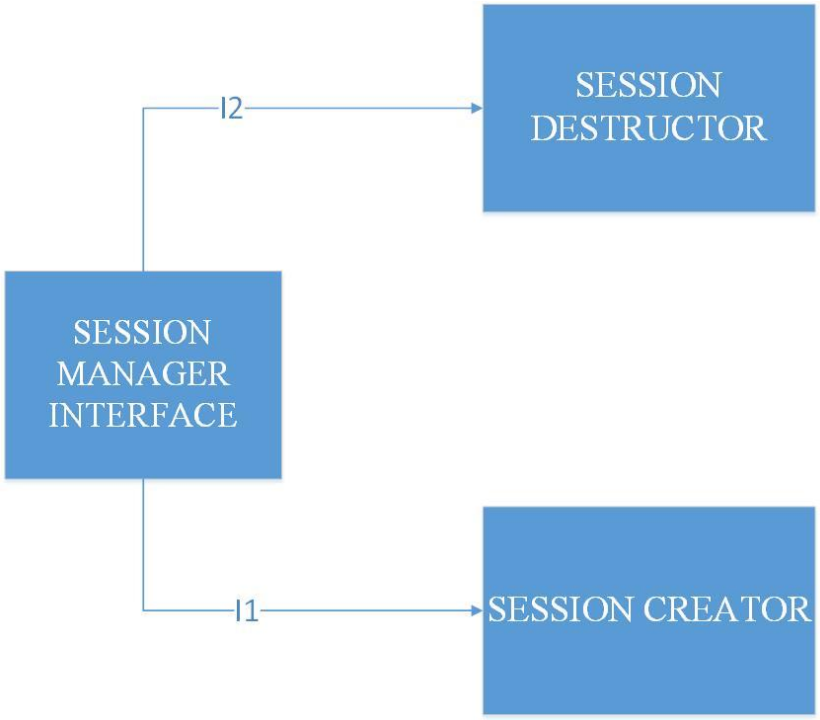
And, on the top, is the component that deals with query constructor accroding to the actions taken or data required by client or server side of the system.

2.4.2 Session manager integration

It is a very simple element that deals with creating and destroying sessions, which enables users to log in and log out.

So, functions that are necessary for these operations are going to be integrated into this element, as it is one of the cirtical modules.

The integration itself is simple and consists of integration of session manager interface with modules related to creating and destroying sesssions.



Next step is to integrate this element with other, more complex elements, in order to make subsystems for different types of users (passengers, taxi drivers, administrators, developers, guests).

2.4.3 Guest subsystem integration

This is the first of the user-oriented subsystem that is going to be integrated.

Without this subsystem, it is not possible to register and log in users, so it is also very cirtical module, when it comes to overall MyTaxiService system.

As it is told previously, this subsystem consists of two important components – one for logging in, and one for registration.

Each of these parts is going to be integrated using threads strategy – parts of different modules are going to be integrated in order to achieve functionality that is visible to user (functional approach).

It is also necessary to integrate the components related to these functionalities with previous two in order to achieve the goal.

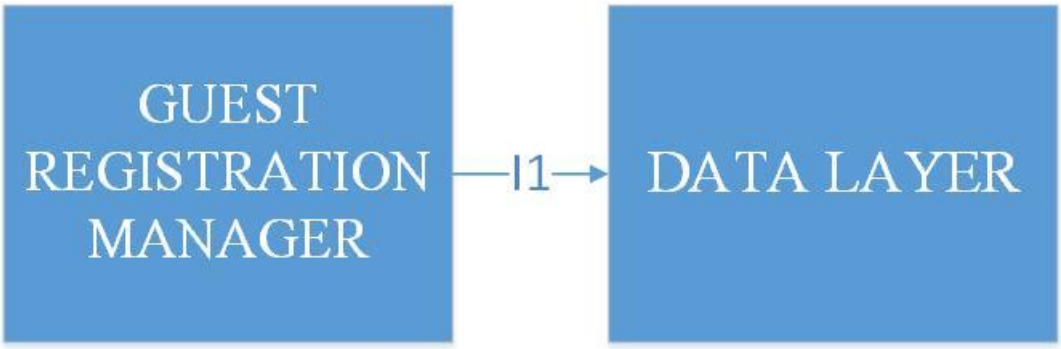
2.4.3.1 Guest registration integration

The first step for the potential users is to make profile.

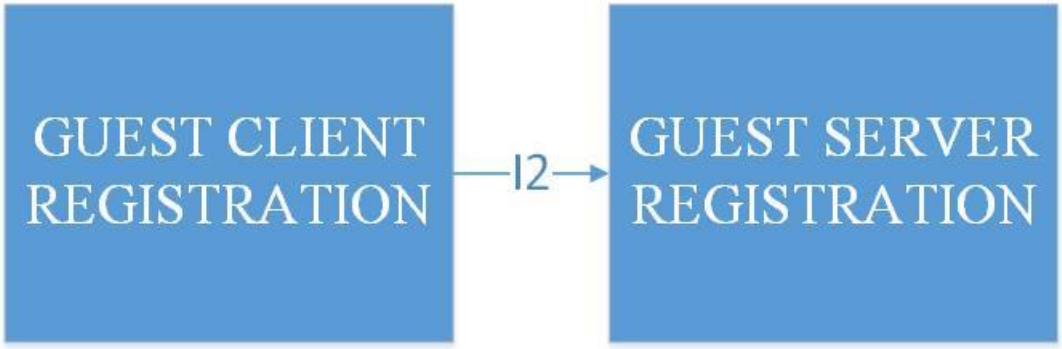
So, guest registation is going to be the first user-oriented function that is going to be integrated.

On the client side we have a component that accepts user input. On the server side, we have component connected with previous one which gives response to user actions using the DataLayer .

First, the server-side component is going to be integrated with the DataLayer (I1) in order to get module that can perform database operations on the server side.

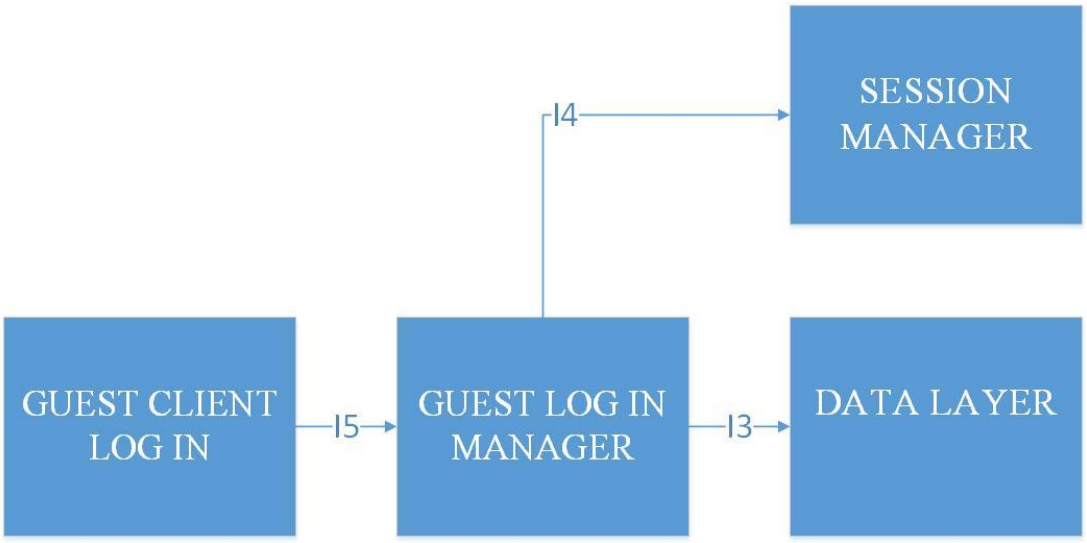


After that, user client is going to be integrated with this, bigger module from the server side(I2, bottom-up), so the clients can interact with the server side.



2.4.3.2 Guest log in integration

For log in integration, we use the same approach as for registration (bottom up in combination with functional). First, the modules of the server side are integrated – guest manager with datalayer (I3), and after that, with session manager (I4).

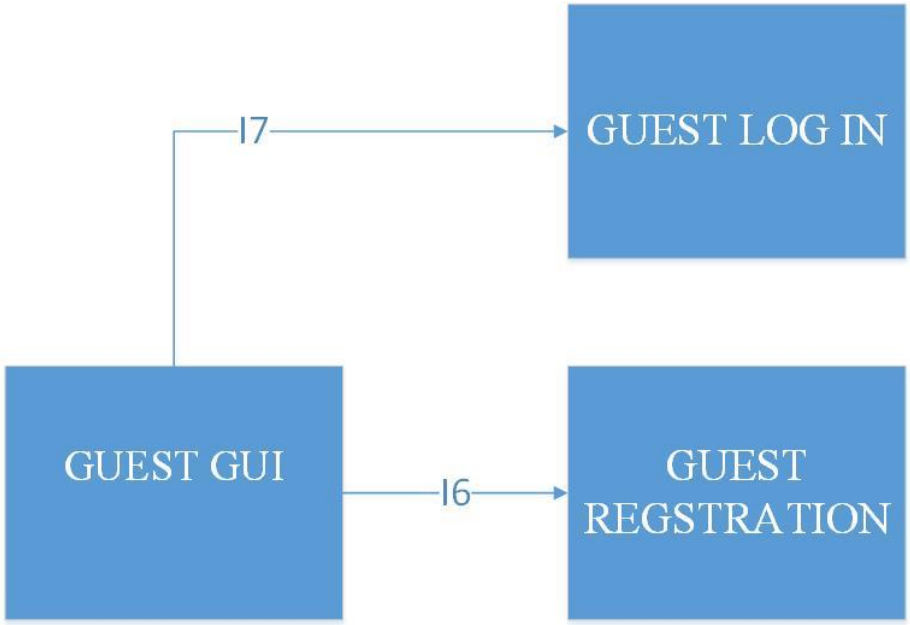


After that, the integration of these components is going to be performed with GUI component of the client side.

It is first necessary to register in order to log in, so, the sequence of integration is the following.

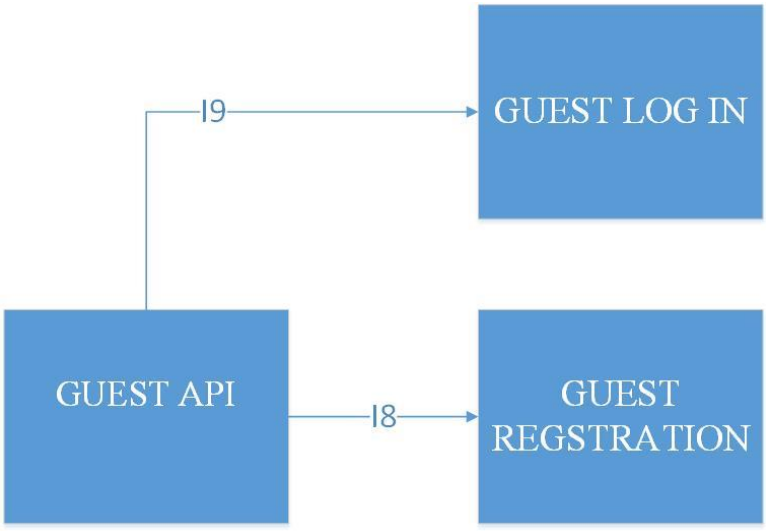
So, we first get the registration driver, and after that, we integrate it with log in module.

And now, we finally have completed the integration of Guest subsystem. Top-down strategy is applied here.



For this part of the system, Developer API is needed, so one more integration is going to be Performed (top-down):

--

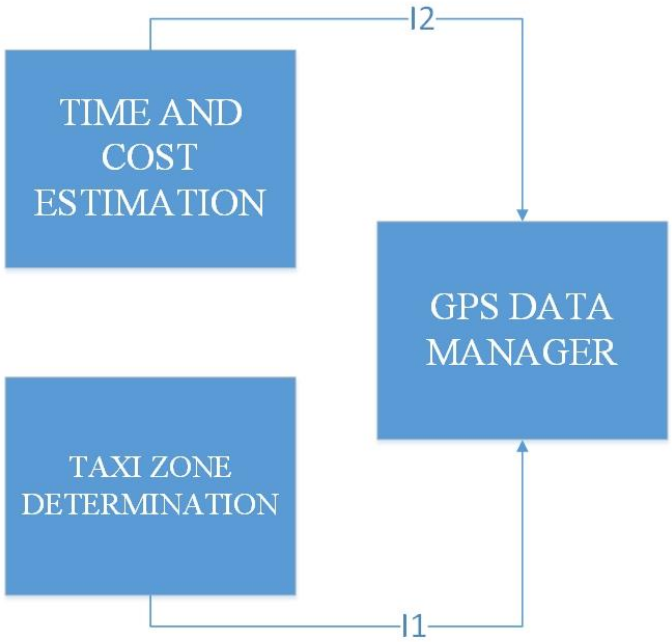


2.4.4 Scheduler integration

Again, one of the critical modules is going to be integrated. Scheduler is the core part of the system and is necessary for any type of system users in order to make taxi requests, respond offers, make estiamtions related to time and cost.

So ,the next step is to integrate the Scheduler. A combination of bottom-up and top-down integration is going to be used for integration of this element.

The first parts that ae going to be integrated are parts related to coordinates and taxi zone determination. No estimation is possible without having coordinates module. After that, coordinates module is going to be integrated with part of the scheduler that deals with time and cost estimation.



Here, the bottom-up approach is used. Now, we can change the integrated module with bigger module (stub), that could be further integrated with part related to taxi queue management.

And, finally (bottom-up strategy here), it is going to be integrated with taxi queue manager in order to achieve the complete Scheduler that can deal with taxi requests (I3), and with component that gives an external interface to the rest of the system (I4).

2.4.5 Registered User and Taxi driver subsystem integration

For integration of this part of the system, threads approach is going to be used. Components are going to be integrated in such order, that each subset of integrated modules gives a user-visible functionlaity.

User-visible functionalities, related to this part of the system could be devided into several parts:

-profile management component, which includes functions related to profile editing

-taxi request component, which includes functions related to making a taxi request

-respond offer component, which is a part of negotiations between taxi driver and user

-report component, responsible for making reports and writing related reason during taxi drive

When it comes to taxi drivers, their subsystem is the same, but offers few more features:

-drive accept/reject component, which is the one of the component where drivers differ from typical users, and gives them ability to accept or reject drive

-availability change component, which is used for managing taxi driver availability

So, each of them is going to be integrated, and, after that, they are going to be integrated together into the user subsystem. For integration of bigger elements, a kind of structural strategy is going to be used.

2.4.5.1 Profile management integration

First, on the server side, Registered User manager is going to be integrated with the DataLayer in order to enable operations related to profile modifications (updates). For profile management, the part of the client related to these operations is going to be integrated with the corresponding part on the server side after that. The integration is going to be performed in a following sequence:

After this, we have profile editing feature completely available and integrated.

The integration is the same for both taxi drivers and users.

It is needed to perform taxi driver integration in parallel with user integration, in order to make request feature integration possible. In order to make taxi requests, it is necessary to have taxi drivers.

2.4.5.2 Taxi request integration

This is the key part of the whole MyTaxiService system. Functional strategy is going to be used for integration of this part. The goal is to enable users to request taxi (user-visible feature).

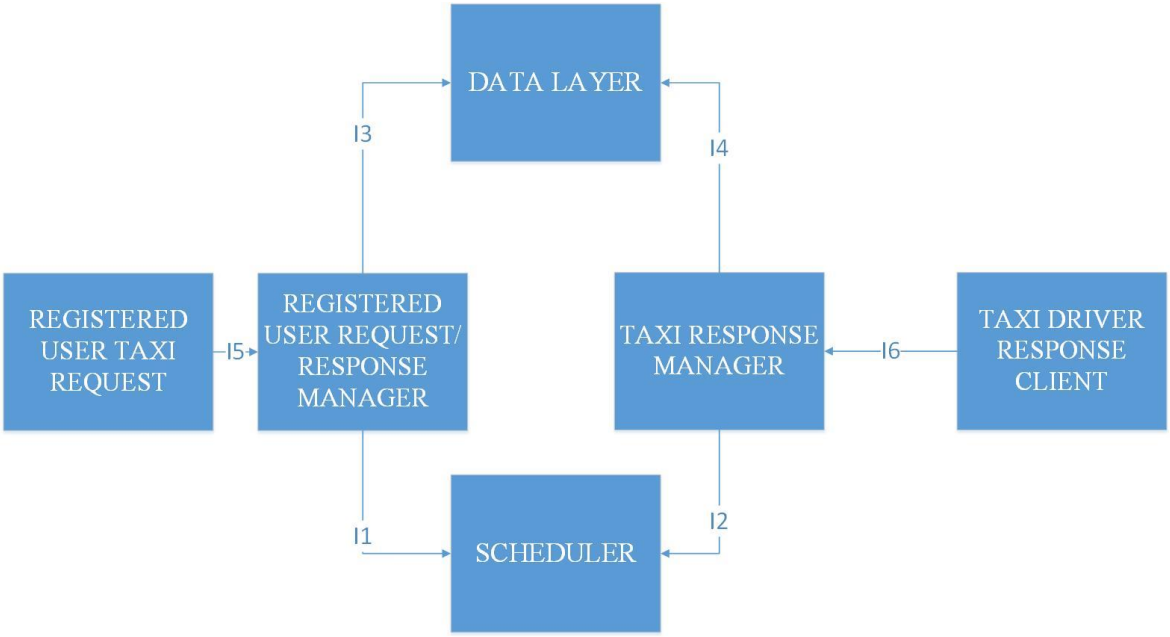
This is the most complex integration in the whole system, and is going to involve the integration of both client and server sides of Registered Users and Taxi drivers.

First, the request/response module on the server side of the users is going to be integrated with the scheduler. Scheduler is necessary for estimations and zone determinations, so making a request does not have any sense without usage of this module. The next step would be to do the same on the taxi driver side. After that, we can make taxi request, and respond them from the taxi side.

Further, the integration with DataLayer is performed in order to be able to keep track of drive events and store them into database.

And, finally, the integration with client side of the corresponding user types is performed (I5 and I6).

After performing these steps of integration, it is possible to state that we have a user-visible feature (taxi request) completely integrated.

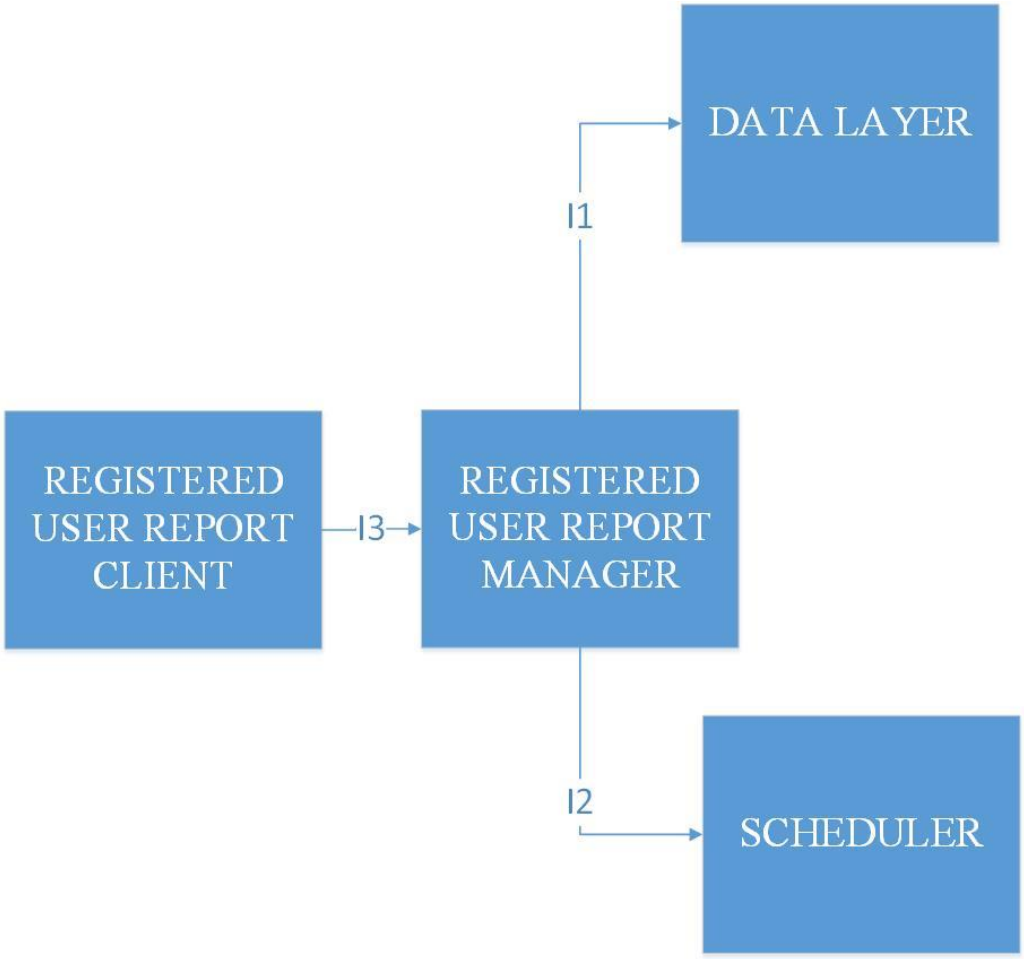


2.4.5.3 Report module integration

This part of system gives ability to users to make reports against other users (user to taxi driver or vice-versa).

This part of the system is symmetric for user and taxi driver. It is necessary in order to complete the taxi driver availability component, because the report event could affect taxi driver availability.

First, the server side report manager is integrated with DataLayer (I1) which is responsible for storage of the reports. After that, the manager on the server side is integrated with Scheduler, because writing reports affects the taxi driver availability and queue state. Finally, client is integrated with the server side part.



2.4.5.4 Taxi driver status management integration

The next feature that is going to be integrated is the module responsible for taxi driver status change. They can change their availability manually, or it can be changed automatically in case of certain events (report, end of the drive event).

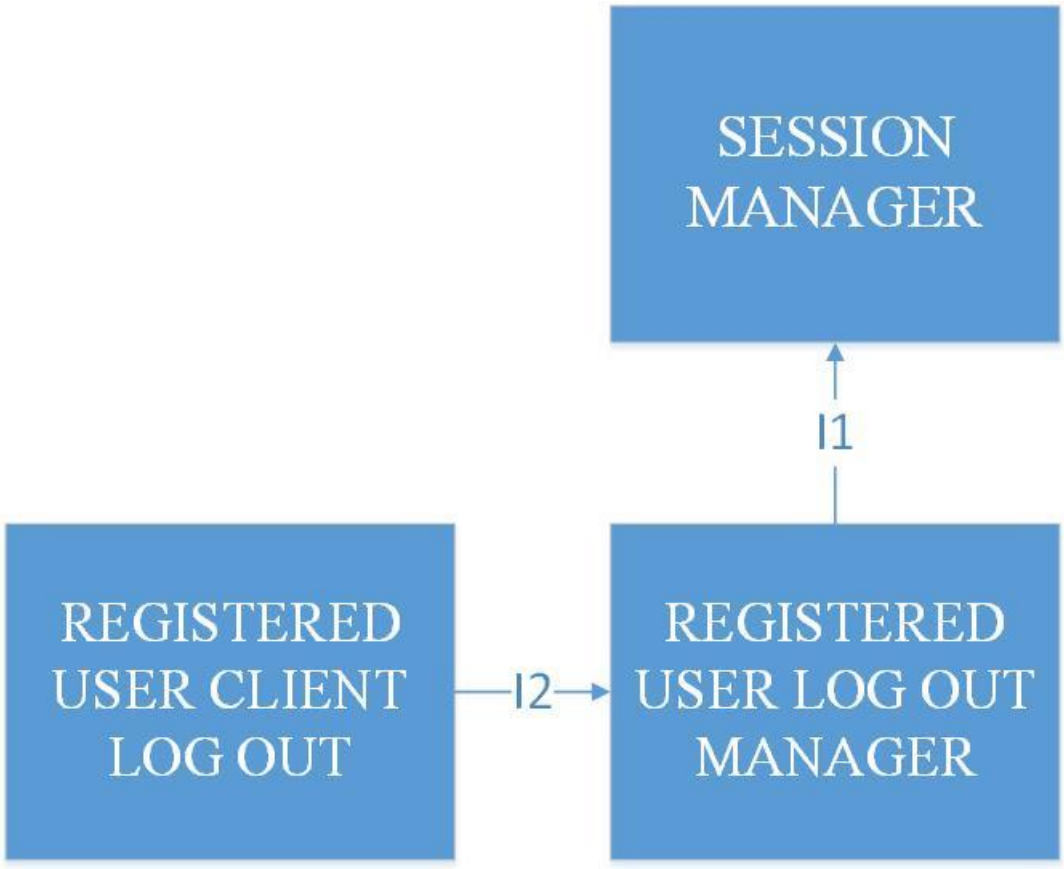
First, the server-side components are integrated (I1), and after that, client component is integrated with server component (I2). Once again, we are integrating a user-visible feature from simpler module to more complex one.

2.4.5.5 Log out integration

Log out feature is going to be achieved by integrating components from both client and server side, similar to log in which is previously mentioned.

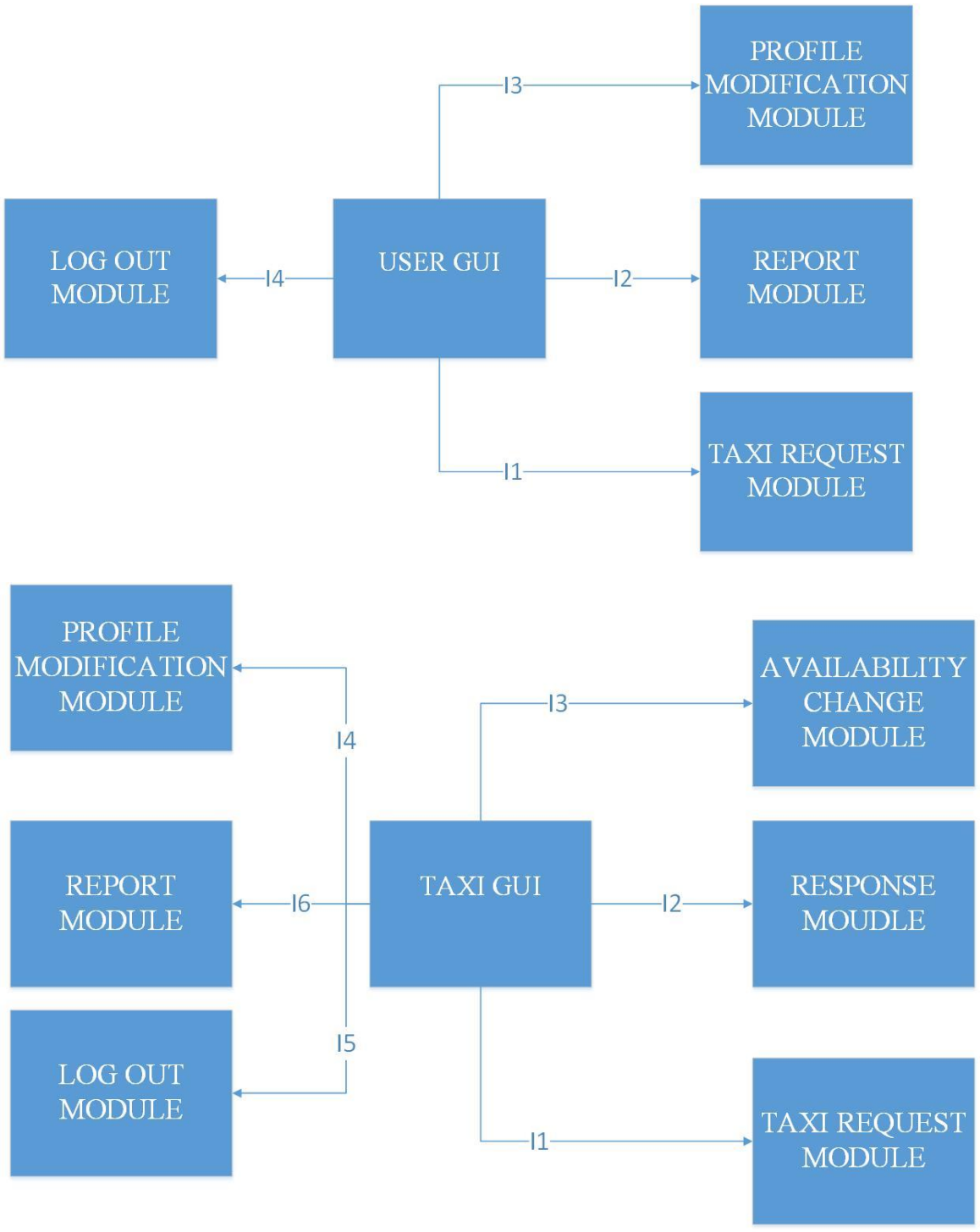
It is performed the same way for both user and taxi driver.

First, the server side-components are connected, and after that, the driver is connected with client component (I2).

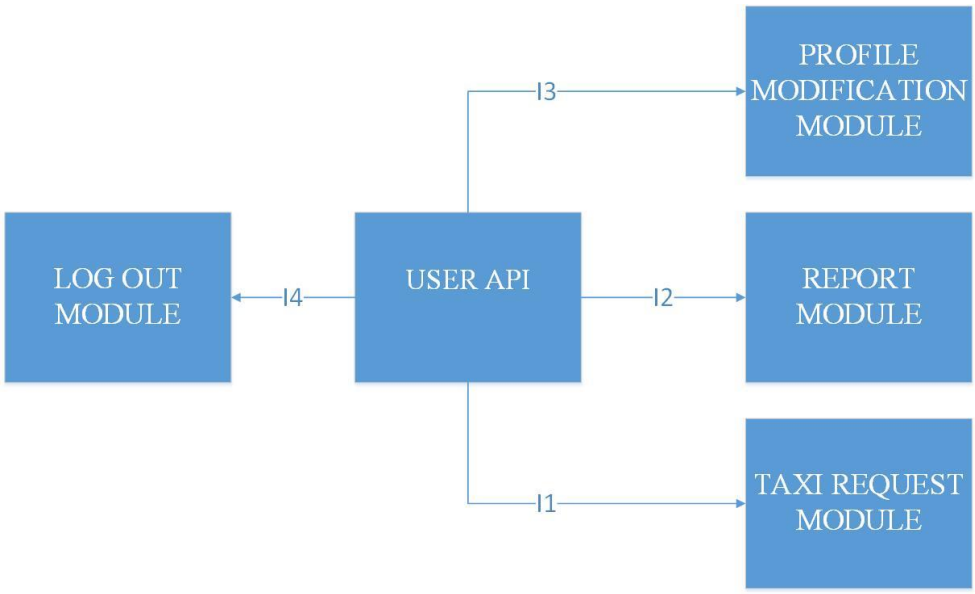


2.4.5.6 User/taxi driver Interface integration

And, finally the integration with GUI is performed in following order (for user and taxi driver) using top-down strategy:



Developer API will involve usage of User Subsystem, so the programmatic interface is also going to be constructed.



2.4.6 Admin subsystem integration

In what follows, admin subsystem is going to be integrated.

It offers the following features:

1.browsing and viewing users

2.viewing drives related to users

3.viewing reports

4.deleting users

5.promoting users to taxi drivers

6.log out

Features 1,2,3,4,5 are going to be integrated into data manipulation module, because CRUD operations are behind them, so heavy use of DataLayer is expected in this case.

Log out is going to be performed the same way as for users and taxi drivers.

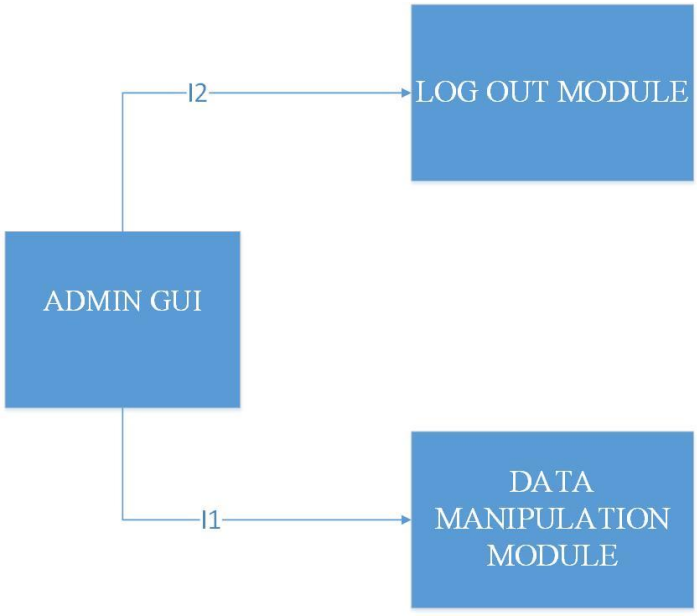
2.4.6.1 Data manipulation module

This module deals with browsing, editing, viewing and deleting data.

First, the server side part that deals with data manipulation is going to be integrated with data layer (I1). After that, client-side component is going to be connected with server side components.

2.4.6.2 Admin GUI module

And finally, admin GUI is going to be integrated with other modules (top-down).



2.4.7 Developer API integration

The API that is going to be created will combine various functionalities that are performed by guests and users together with scheduling.

-everything related to guests

-everything related to users

-scheduling

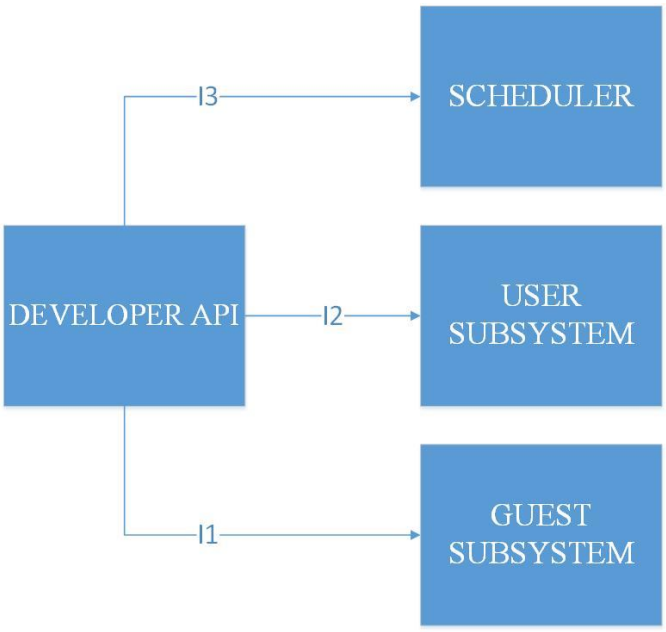
It is decided, that all other features are omitted for first version of the system, in order to avoid misuse of the API.

In further versions, it is expected to add taxi driver features for developers.

The integration is going to be performed in following sequence, when it comes to Developer API

(the picture below).

Here, a top-down integration is used.



3 Individual steps and test description

In this chapter, individual steps in testing ae going to be described.

In order to perform a specific integration test, it is necessry to:

-Design the integation test

-Design a driver (if it was not made at the unit test)

-Design input test data (if it was not made at the unit test)

-Setting up a system, the components involved, the driver and the input test data

-Performing the integration test

In what follows, test cases are going to be specified.

3.1 Test case specifications

In what follows, test case specifications are given by tables with following structure:

1.Test case identifier - used to identify a test dase

2.Test case items – interface betweeen which components is tested. Arrow C1 -> C2 means that interface from C1 to C2 is tested.

3.Input specification – what is going to be input in this test

4.Output specification – output expected for given input

5.Environmental needs – which test cases need to be performed

3.1.1 Integration test case I1 - DataLayer

Test case identifier	I1T1
Test case items	Persistence Manager -> Query Executor
Input specification	Create typical Persistance Manager input that performs queries: -view user -add users -delete users -update users

	-view drive -add drive -view report
Output specification	Check if correct functions are called inside Query Executor: -corrrsponding INSERT queries constructed for viewing -corresponding DELETE queries properly costructed for deleting -corresponding UPDATE queires properly costructed for updates
Environmental needs	Persistence manager driver

Test case identifier	I1T2
Test case items	Query Constructor -> Persistence Manager
Input specification	Create typical Query Constructor input: -construct insert query -construct delete quer -construct update query
Output specification	Check if correct functions are called inside Persistence Manager: -new objects created -objects deleted -objects changed
Environmental needs	I1T1 succeeded; Query constructor driver

3.1.2 Integration test case I2 – Session Manager

Test case identifier	I2T1

Test case items	Session Manager Interface -> Session Creator
Input specification	Create Session Manager input that involves usage of Session Creator: -request new session
Output specification	Check if correct functions are called inside Session Creator and if new session is created: -new session created
Environmental needs	Guest client driver;

Test case identifier	I2T2
Test case items	Session Manager Interface -> Session Destructor
Input specification	Create Session Manager input that will involve Session Destructor: -request end of session
Output specification	Check if correct functions are called inside Session Destructor and if session is destroyed.
Environmental needs	User client driver

3.1.3 Integration test case I3 – Guest Registration

Test case identifier	I3T1
Test case items	Guest Registration Manager-> Data Layer
Input specification	Create typical Guest Regsistration Manager input: -insert new user
Output specification	Check if correct functions are called inside Data Layer:

	-check if user already exists -check if data is valid -check if user is banned -add user to database if data is valid, user not banned and does not exist in database
Environmental needs	I1T1 succeeded; I1T2 suceeded; Guest Registration driver

Test case identifier	I3T2
Test case items	Guest Registration Client-> Guest Manager
Input specification	Create typical Guest Regsistration Manager input: -create registration request
Output specification	Check if correct functions are called inside Guest Registration Manager: -registration request handler
Environmental needs	I3T1 succeeded;

3.1.4 Integration test case I4 – Guest Log in

Test case identifier	I4T1
Test case items	Guest Log in Manager-> Data Layer
Input specification	Create Guest Log in Manager input that involves usage of Data Layer: -select user by user name and password
Output specification	Check if correct functions are called inside Data Layer: -check if such user exists by select query and compare password to check if the right one is entered.

Environmental needs	I1T1 succeeded; I1T2 suceeded;Guest Login driver

Test case identifier	I4T2
Test case items	Guest Log In Manager-> Session Manager
Input specification	Create typical Guest Log in Manager input: -create new log in request with correct username and password
Output specification	Check if correct functions are called inside Session Manager and if new session is created:
Environmental needs	I4T1 succeeded; I2T1 succeeded; I2T2 succeded

Test case identifier	I4T3
Test case items	Guest Log In Client-> Guest Log in Manager
Input specification	Create typical Guest Log in Manager input: -create login request
Output specification	Check if correct functions are called inside Guest Log in Manager: -login request handler
Environmental needs	I4T3 succeeded;

3.1.5 Integration test case I5 – Guest GUI

Test case identifier	I5T1
Test case items	Guest GUI-> Guest Registration Manager

Input specification	Create typical Guest GUI input that involves Guest Registration Manager: -fill in the necessary fields with data using GUI -click register button
Output specification	Check if correct functions are called inside Guest Registration Manager: -handler function for registration request of new user should be called -new user added to database
Environmental needs	I3T1 succeeded; I3T2 succeeded; Guest registrataion stub

Test case identifier	I5T2
Test case items	Guest GUI-> Guest Log In Manager
Input specification	Create typical Guest GUI input that involves Guest Log In Manager: -fill in username and password fields -click log in butttom
Output specification	Check if correct functions are called inside Guest Log in Manager: -log in event handler should be called -user is logged in
Environmental needs	I4T1 succeeded; I4T2 suceeded; I4T3 succeeded; I5T1 succeded ;Guest log in stub

3.1.6 Integration test case I6 – Scheduler

Test case identifier	I6T1

Test case items	Taxi Zone Determination Module-> GPS Data Manager
Input specification	Create typical Taxi Zone Determination Module Input: -determine taxi zone of a given coordinates
Output specification	Check if correct functions are called inside GPS Data Manager
Environmental needs	GPS Data manager functions unit tests succeeded

Test case identifier	I6T2
Test case items	Estimator-> GPS Data Manager
Input specification	Create typical Estimator input: -insert startpoint and endpoint
Output specification	Check if correct functions are called inside GPS Data Manager: -calculation of distance between two points is performed
Environmental needs	GPS Data manager functions unit tests succeeded

Test case identifier	I6T3
Test case items	Scheduler Submodule-> Taxi Queue Manager
Input specification	Create typical Scheduler Submodule input: -add taxi to queue -remove taxi from queue -forward request to another taxi from same queue -taxi changes availability -taxi driver is reported
Output specification	Check if correct functions are called inside Taxi Queue Manager and as a result: -taxi is added to queue

	-removed from queue -request is forwarded to another taxi from same queue
Environmental needs	I6T1 succeeded; I6T2 succeeded; Scheduler driver; Taxi Driver driver

Test case identifier	I6T4
Test case items	Scheduler Interface-> Scheduler Submodule
Input specification	Create typical Scheduler Interface input: -request a taxi
Output specification	Check if correct functions are called inside Scheduler Submodule -taxi request handler
Environmental needs	-I6T1 succeeded; I6T2 succeeded; I6T3 succeeded; Scheduler Submodule Stub

3.1.7 Integration test case I7 – Profile editing

Test case identifier	I7T1
Test case items	User Profile Editing Manager -> Data Layer
Input specification	Create typical User Profile Editing Manager input: -update user information
Output specification	Check if correct functions are called inside Data Layer: -update query is executed and user data is modified
Environmental needs	I1 succeeded; User Profile Editing driver

Test case identifier	I7T2

Test case items	User Profile Editing Client -> User Profile Editing Manager
Input specification	Create typical User Profile Editing Client input: -request user information update
Output specification	Check if correct functions are called inside User Profile Editing Manager -user information update request handler
Environmental needs	I7T1 succeeded

3.1.8 Integration test case I8 – Taxi Request

Test case identifier	I8T1
Test case items	Registered User Request/Response Manager -> Scheduler
Input specification	Create typical Registered User Request/Response Manager input that involves Scheduler -request a taxi -respond the offer
Output specification	Check if correct functions are called inside Scheduler: -determine taxi zone of the user -get taxi from current taxi zone -estimation
Environmental needs	I6 succeded; Taxi Driver driver

Test case identifier	I8T2
Test case items	Taxi Response Manager -> Scheduler
Input specification	Create typical Taxi Response Manager input: -in one case, accept request

	-in another case – reject it
Output specification	Check if correct functions are called inside Scheduler: -in first case, estimation should be called -in second case, scheduler should forward the request to another taxi from same queue
Environmental needs	I6 succeeded; I8T1 suceeded; Taxi Response Driver

Test case identifier	I8T3
Test case items	Registered User Request/Response Manager -> Data Layer
Input specification	Create typical Registered User Request/Response Manager input that involves Data Layer: -user accepts drive offer
Output specification	Check if correct functions are called inside Data Layer: -new drive event is created and inserted into database
Environmental needs	I1 succeded; I8T1 succeeded; I8T2 succeeded; Taxi Driver driver

Test case identifier	I8T4
Test case items	Taxi Response Manager -> Data Layer
Input specification	Create typical Taxi Response Manager input that involves Data Layer: -user accepts offer -taxi driver accepts drive request
Output specification	Check if correct functions are called inside Data Layer: -new drive event inserted into database in case if user accepts the offer

Environmental needs	I1 succeeded; I8T1 suceeded; I8T2 suceeded; I8T3 succeeded; Taxi Driver driver; User driver

Test case identifier	I8T5
Test case items	Registered User Request/Response Client -> Registered User Request/Response Manager
Input specification	Create typical Registered User Request/Response Client input: -request a taxi -respond the offer
Output specification	Check if correct functions are called inside Registered User Request/Response manager: -request handler -response handler
Environmental needs	I8T3 succeded;

3.1.9 Integration test case I9 – Report module

Test case identifier	I9T1
Test case items	Registered User Report Manager -> Data Layer
Input specification	Create typical Registered User Reporrt Manager input that involves Data Layer: -make a report with reason incuded
Output specification	Check if correct functions are called inside Data Layer: -query that inserts report into database is executed
Environmental needs	I1 succeded;

Test case identifier	I9T2
Test case items	Registered User Report Manager -> Scheduler
Input specification	Create typical User Report Manager input: -report a driver
Output specification	Check if correct functions are called inside Scheduler: -taxi zone determination for a taxi driver against who the report is written -taxi added into correspondig queue of the taxi zone that is previously determined
Environmental needs	I6 succeeded; I9T1 suceeded

Test case identifier	I9T3
Test case items	Registered User Report Client -> Registered User Report Manager
Input specification	Create typical Registered User Report Client: -make a report request
Output specification	Registered User Report Manager calls the correct functions: -report request handler
Environmental needs	I9T1 succeeded; I9T2 succeeded

3.1.10 Integration test case I10 – Status change

Test case identifier	I10T1
Test case items	Taxi Driver Status Manager -> Scheduler
Input specification	Create typical Taxi Driver Status Manager Input: -report taxi driver

	-manually change availability: 1.from available to unavailable 2.from unavailable to available
Output specification	Check if correct functions are called inside Scheduler: -first, the taxi zone determination is performed -taxi added to queue if it is reported -1.removed from queue, 2. added to queue
Environmental needs	I6 succeeded; Taxi Driver driver

Test case identifier	I10T2
Test case items	Taxi Driver Status Client ->Taxi Driver Status Manager
Input specification	Create typical Taxi Driver Status Client input: -request statis change
Output specification	Check if correct functions are called inside Taxi Driver Status Manager -status change request handler
Environmental needs	I10T1 succeeded

3.1.11 Integration test case I11 – Guest Log out

Test case identifier	I11T1
Test case items	Guest Log Out Manager-> Session Manager
Input specification	Create typical Guest Log out Manager input: -call log out for some user
Output specification	Check if correct functions are called inside Session Manager:

	-destroy session for that user
Environmental needs	I2 succeeded;

Test case identifier	I11T2
Test case items	Guest Log Out Client-> Guest Log Out Manager
Input specification	Create typical Guest Log Out Manager input: -request a log out
Output specification	Check if correct functions are called inside Guest Log out Manager: -log out request handler
Environmental needs	I11T2 succeeded;

3.1.12 Integration test case I12 – User GUI

Test case identifier	I12T1
Test case items	User GUI -> Taxi Request Module
Input specification	Create typical User GUI input that involves usage of Taxi Request Module: -select desired destination -input maximum waiting time -make a taxi request -respond to offer
Output specification	Check if correct functions are called inside Taxi Request Module: -taxi request event handler -offer response event handler
Environmental needs	I8 succeded; Taxi request stub

Test case identifier	I12T2
Test case items	User GUI -> Report Module
Input specification	Create typical User GUI input that involves usage of Report Module: -request a report -drive reason -submit report
Output specification	Check if correct functions are called inside Report Module -report form is created -report submit event handler is called after submission
Environmental needs	I9 suceeded; Report stub

Test case identifier	I12T3
Test case items	User GUI -> Profile Editing Module
Input specification	Create typical User GUI input that involves Profile Editing Module: -click on profile modify button -fill in the necessary fields -submit changes
Output specification	Check if correct functions are called inside Profile Editing Module -profile modification event handler
Environmental needs	I7 succeded

Test case identifier	I12T4
Test case items	User GUI->Log out Module

Input specification	Create typical User GUI input that involves Log out module: -request a log out
Output specification	Check if correct functionss are called inside Log out module: -log out event handler
Environmental needs	I11 suceeded

3.1.13 Integration test case I13 – Taxi Driver GUI

Test case identifier	I13T1
Test case items	Taxi Driver GUI -> Taxi Request Module
Input specification	Create typical Taxi Driver GUI input that involves usage of Taxi Request Module: -the same as I12T1
Output specification	Check if correct functions are called inside Taxi Request Module: -the same as I12T1
Environmental needs	I8 succeded; Taxi request stub

Test case identifier	I13T2
Test case items	Taxi Driver GUI -> Response Module
Input specification	Create typical Taxi Driver GUI input that involves usage of Response Module: -respond the taxi drive request
Output specification	Check if correct functions are called inside Response Module: -taxi response event handler
Environmental needs	I8 suceeded;

Test case identifier	I13T3
Test case items	Taxi Driver GUI -> Availability Change Module
Input specification	Create typical Taxi Driver GUI input that involves usage of Availability Change Module: -change availability
Output specification	Check if correct functions are called inside Availability Change Module: -availability change event handler
Environmental needs	I10 succeded

Test case identifier	I13T4
Test case items	Taxi Driver GUI -> Profile Editing Module
Input specification	Create typical Taxi Driver GUI input that involves Profile Editing Module: -the same as I12T3
Output specification	Check if correct functions are called inside Profile Editing Module: -the same as I12T3
Environmental needs	I7 succeded; Profile editing stub

Test case identifier	I13T5
Test case items	Taxi Driver GUI->Log out Module
Input specification	Create typical Taxi Driver GUI input that involves Log out module: -request a log out
Output specification	Check if correct functionss are called inside Log out module:

	-log out event handler
Environmental needs	I11 suceeded; Log out stub

Test case identifier	I13T6
Test case items	Taxi driver GUI -> Report Module
Input specification	Create typical User GUI input that involves usage of Report Module: -request a report -drive reason -submit report
Output specification	Check if correct functions are called inside Report Module -report form is created -report submit event handler is called after submission
Environmental needs	I9 suceeded; Report module stub

3.1.14 Integration test case I14 – Admin Integration

Test case identifier	I14T1
Test case items	Admin Data Maniulation Manager -> Data Layer
Input specification	Create typical Admin Data Manipulation input
Output specification	Check if correct functions are called inside Data Layer
Environmental needs	I1 succeded; Admin Driver

Test case identifier	I14T2

Test case items	Admin Data Manipulation Client -> Admin Data Manipulation Manager
Input specification	Create typical Admin Data Manipulation Client Input:
Output specification	Check if correct functions are called inside Admin Data Manipulation Manager.
Environmental needs	I1T4 suceeded;

Test case identifier	I14T3
Test case items	Admin Client GUI -> Data Manipulation Module
Input specification	Create typical Admin Client GUI input that involves use of Data Manipuation Module: -search user -delete user -update user to taxi driver -withdraw taxi driver privilegues -view report -view drive
Output specification	Check if correct functions are called inside Data Manipulation Module: -search user event handler -delete user event handler -update to taxi driver event handler -downgrade driver event handler -view report event handler -slected drive display event handler
Environmental needs	I14T2 succeeded succeded;Admin Data Manipulation stub

Test case identifier	I14T5
Test case items	Admin Client GUI -> Log Out Module
Input specification	Create typical Admin Client GUI input that involves Admin Log Out -click log out botton
Output specification	Check if functions are called inside Admin Log Out Module -log out event handler
Environmental needs	I11 succeded

3.1.15 Integration test case I15 – Guest API

Test case identifier	I15T1
Test case items	Guest API-> Guest Registration Manager
Input specification	Create typical Guest API input that involves Guest Registration Manager: -register new user function call
Output specification	Check if correct functions are called inside Guest Registration Manager -registration handler
Environmental needs	I3T1 succeeded; I3T2 succeeded;

Test case identifier	I15T2
Test case items	Guest API-> Guest Log In Manager
Input specification	Create typical Guest API input that involves Guest Log In Manager: -log in user function call

Output specification	Check if correct functions are called inside Guest Log in Manager: -log in handler
Environmental needs	I4T1 succeeded; I4T2 suceeded; I4T3 succeeded; I5T1 succeded

3.1.16 Integration test case I12 – User API

Test case identifier	I16T1
Test case items	User API -> Taxi Request Module
Input specification	Create typical User API input that involves usage of Taxi Request Module: -request taxi function call
Output specification	Check if correct functions are called inside Taxi Request Module: -taxi request handler
Environmental needs	I8 succeded;

Test case identifier	I16T2
Test case items	User API -> Report Module
Input specification	Create typical User API input that involves usage of Report Module: -report function call
Output specification	Check if correct functions are called inside Report Module -report handler
Environmental needs	I9 suceeded;

Test case identifier	I16T3
Test case items	User API -> Profile Editing Module

Input specification	Create typical User API input that involves Profile Editing Module: -update user function call
Output specification	Check if correct functions are called inside Profile Editing Module: -update user handler
Environmental needs	I7 succeded

Test case identifier	I16T4
Test case items	User API->Log out Module
Input specification	Create typical User API input that involves Log out module
Output specification	Check if correct functionss are called inside Log out module
Environmental needs	I11 suceeded

3.1.17 Integration test case I17 – Developer API Integration

Test case identifier	I17T1
Test case items	Developer API -> Guest Subsystem
Input specification	Create typical Developer API input that involves use of Guest subsystem : -register function call -log in function call
Output specification	Check if correct functions are called inside Guest Subsystem: -register event handler -log in event handler
Environmental needs	I15 succeded; Guest stub

Test case identifier	I17T2
Test case items	Developer API -> User Subsystem
Input specification	Create typical Developer API input that involves use of User Subsystem: -update profile function call -request taxi functio ncall -report function call
Output specification	Check if correct functions are called inside User Subsystem: -update profile event handler -request taxi event handler -report event handler
Environmental needs	I16 suceeded; User stub

Test case identifier	I17T3
Test case items	Developer API -> Scheduler
Input specification	Create typical Admin input that involves use of Scheduler: -determine taxi zone -add taxi to queue -remove taxi from queue -cost and time estimation -get available taxi
Output specification	Check if correct functions are called inside Scheduler: -taxi zone determination event handler -add taxi to queue event handler -remove taxi from queue event handler
Environmental needs	I6 succeeded; Scheduler stub

3.2 Test procedures

In what follows, test procedures are going to be presented.

Each of the test procedure has its identifier, purpose and procedure steps. Purpose explains what is going to be tested in the concrete procedure, while procedure steps are in fact, sequences of test cases that are going to be performed.

Test procedure identifier	TP1
Purpose	This test procedures verifies whether the guest software can: -register new users -log in already registered users
Procedure steps	Execute I5 after I1-I4

Test procedure identifier	TP2
Purpose	This test procedures verifies whether the user software can: -rmodify user profile -request a taxi -respond to an offer -write report to other side -log out
Procedure steps	Execute I12 after I6-I11

Test procedure identifier	TP3

Purpose	This test procedures verifies whether the taxi driver software can: -modify user profile -request a taxi -respond to a drive request -write a report against the opposite side -change availability status -log out
Procedure steps	Execute I13 after I6-I12

Test procedure identifier	TP4
Purpose	This test procedures verifies whether the admin application can: -browse users -view taxi drives -read reports -promote users to taxi drivers -delete users -log out
Procedure steps	Execute I14 after I13

Test procedure identifier	TP5

Purpose	This test procedures verifies whether the developer api can: -register new user -log in -rmodify user profile -request a taxi -write reports against the opposite side -respond to a drive offer -log out -get the available taxi for given taxi zone -estimate cost of a drive between two given points -estimate time between two given points
Procedure steps	Execute I17 after I5-16

4 Tools and test equipment required

In this chapter, tools and equiment required for testing are going to be described and their usage explained.

4.1 Software tools

Tools from Verification tools [2] lecture are considered, and their possible usage in testing explained (since the software is not going be implemented and tested in reality).

Considering the purpose of the software tools presented during lecture, the tools used to automate the integration testing could be the following:

-JUnit is the most used framework for unit testing in Java. Since Java implementation is planned it is going to be used for unit tests of the single components (not covered by this document, stated in *2.1 Entry condition chapter*), but is also going to be used for integration testing in combination with Mockito and Arquillian. So, this tool is not going to be used only r to satisfy the entry criteria. but also in integration testing.

-Arquillian is a test framework which can also manage the test of the containers and their integration with JavaBeans (dependency injection). It will be mainly used it for that purpose (since Design Document is based on JEE architecture).

-Mockito is an open-source test framework useful to generate mock objects, stubs and drivers. It will be used it in several test cases to mock stubs and drivers for the components to test. In different parts of the system, different strategies are going to be used, so there will be different drivers and stub present in integration tests (as enivornmental needs).These drivers and stubs are going to be described in next chapter and their purpose explained *(5.1 Stubs and drivers).*

It is also going to be mentioned that final GUI integration test is going to be performed manually, in order to ensure that right user experience is achieved after integration.

4.2 Equipment

In order to perform testing, a reduced set of devices is going to be used in order to get at least minimal version of the system that could be tested.

Considering the RASD (constraints and all other requirements) and DD (architecture), the following devices are necessary dor testing :

Device type	Number	Role
Server computer	1	This one would have role of: Web, Application and Database server at the same time
Personal computer	3	1. Admin client test

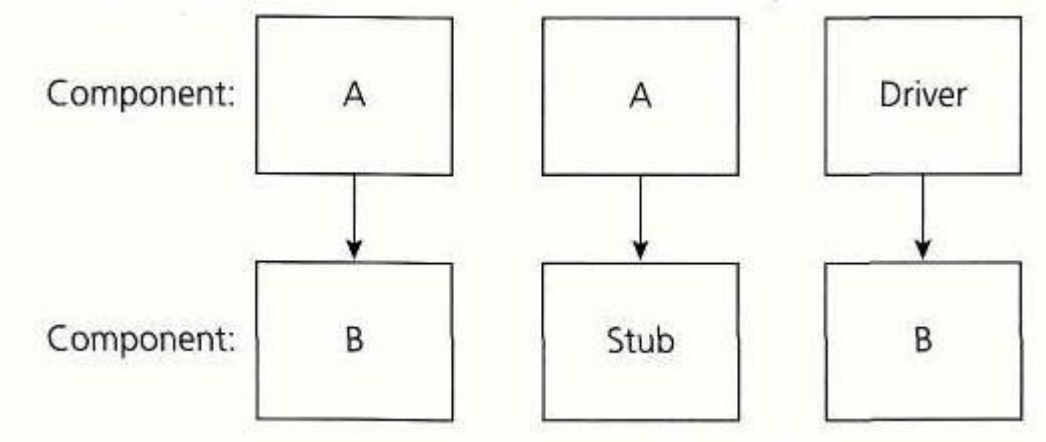
		2. User client test 3. Developer API test
Android smartphones	2	One will be used by user, while another will be used by taxi driver.

5 Program stubs/drivers and test data required

5.1 Stubs and drivers

In order to make integration test possible without developing the complete system first, it is necessary to use stubs and drivers that take role of the part of the system that is not completed at the moment of performing the test.

Illustration below shows the concept:



Considering all the tests that are going to be performed, in what follows, the most important drivers and stubs for specific tests are going to be presented and their role described.

Again, it is mentioned that for final API and GUI integrations, top-down approach is used, so stubs are present, while, in other situations, bottom-up is used, so drivers for these integrations are described.

Test	**Driver/stub required**
Taxi request	Taxi Driver driver component:considering this test from user's point of view, it is needed to have a taxi driver in a protocol where he/she could accept or reject drive request, so driver component should be able to accept or reject the request.
Taxi drive offer response	Taxi Driver driver component:considering this test from user's point of view, it is needed to have a taxi driver in a protocol where he/she could accept or reject drive request, so driver component should be able to accept or reject the request. In case when it comes to taxi drive offer

	response, it is necessary to previously accept the request. User driver component: from taxi driver's point of view, we need User driver wich can accept or reject the offer.
Scheduler	Taxi Driver driver: it is used to simulate different events: availability change and report, rejecting the drive request in order to lead to changes in taxi queue Scheduler submodule stub – consists of Taxi zone determinator and Estimator stub
DataLayer	Query constructor driver: to create queries Persistence driver: to deal with objects and their storage into database
Developer API	Guest API stub User API stub Taxi Driver API stub Scheduler API stub
User GUI/API	Registration stub Profile edit stub Taxi request stub Log out stub Respond offer stub Report stub
Admin GUI/API	Log out stub Data manipulation stub
Taxi driver GUI/API	Profile edit stub Taxi request stub Log out stub Respond offer stub Report stub

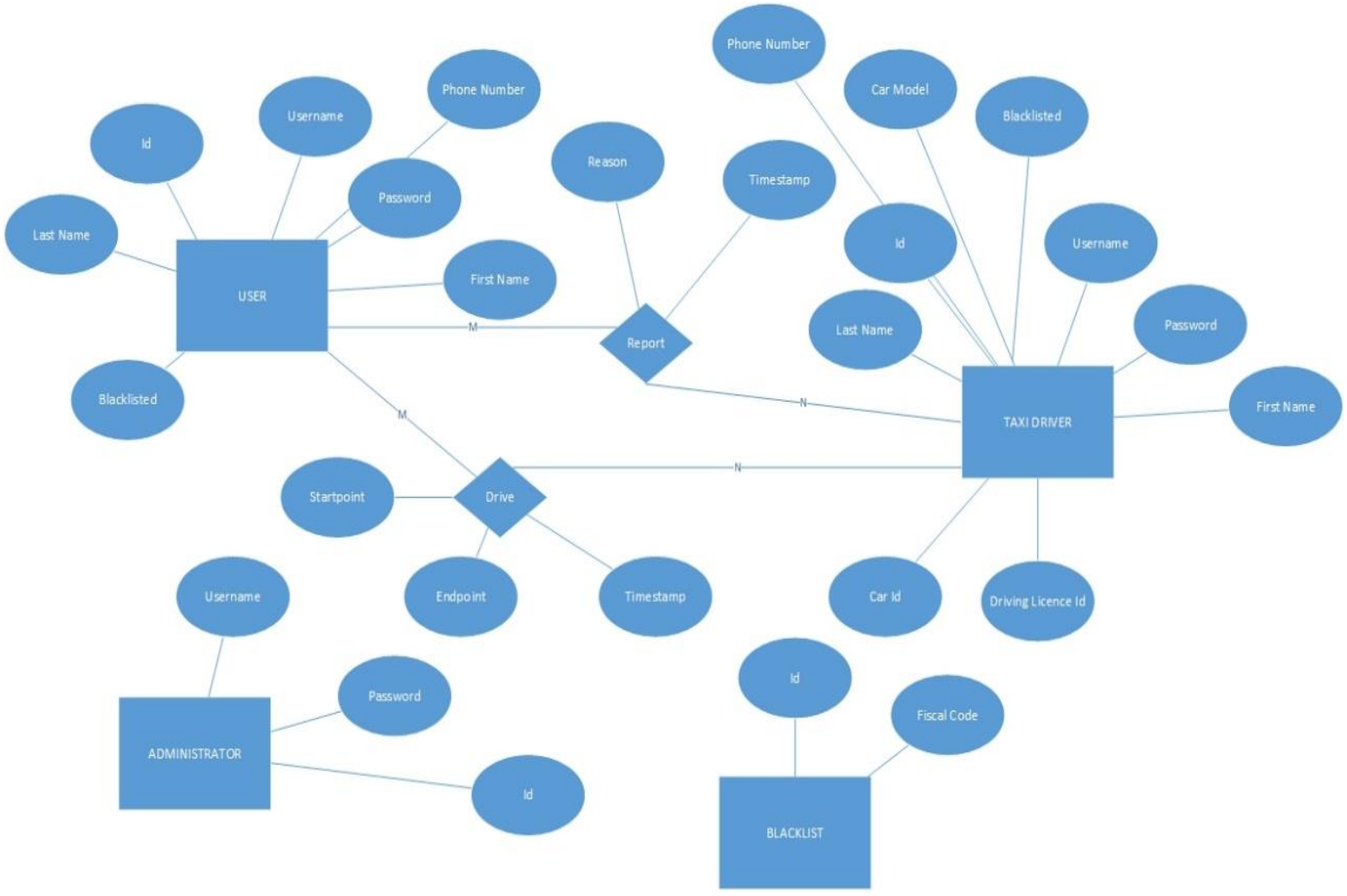
	Availability change stub Respond request stub

5.2 Test data

It is also necessary to provide test data, which could contain files, databases and other type of data organization which contains the necessary data and is required in order to perform a specific test.

In this case, test database is required and its usage in tests and structure are going to be described in what follows.

Database is going to be developed, according to Entity-Relation diagram below.



The test database contains a reduced set of instances. It is necessary for tests which include interaction with database.

For each of the test that need interaction with database, required test data in database is described in following table.

On the left side, tests are listed, while, on the right side, the data required for each of the tests is listed.

Test	Data required
User registration	User table in database
User log in/log out	User table in database with at least one row
User profile modification	User table in database with at least one row
Taxi request/Offer response/Taxi request response	User table and taxi driver tables in database with at least one row each of them.
Reporting	Report table in database
Delete user	Admin table with at least one row and user or taxi driver table wth at least one row
View/search users	User table with at least one row and Admin table with at least one row
View/search taxi drivers	Taxi driver table with at least one row and Admin table with at least one row
Promote to taxi driver	User table with at least one row and Admin table with at lest one row
Downgrade driver	Taxi driver table with at least one row and Admin table with at least one row
View report	Report table with at least one row, Taxi driver table with at least one row, User table with at least one row and Admin table with at least one row.
View drive	Drive table with at least one row, Taxi driver table with at least one row, User table with at

	least one row and Admin table with at least one row.
Scheduler	Drive table in database.
DataLayer	Database with all tables.

6 Software and tools used for document creation

- Microsoft Office Word 2016: To create and redact this document
- Microsoft Visio 2016

7 Hours of works

The time spent:

- Nenad Petrovic: ~40 hours.

8 References

1. Verificaion second part (Softwaree Engineering 2 course, Politecnico di Milano)-verification second lecture.pdf
2. Verification Tools (Software Engineering 2 course, Politecnico di Milano)- Verif-Tools.pdf