



Politecnico di Milano

Academic Year 2015/2016

Software Engineering 2: “myTaxiService”
Requirement Analysis and Specification
Document

Massimo Schiavo, Marco Edoardo Cittar

November 6, 2015

SUMMARY

1. SECTION 1:

- Description of the given problem
- The identification of the actors
- Our goals

1. SECTION 2:

- Overall Description

1. SECTION 3:

- External interface requirements
- Functional requirements
- The World and the Machine
- Scenarios
- UML Models

1. SECTION 4:

- Alloy

SECTION 1

DESCRIPTION OF THE GIVEN PROBLEM

We will design and implement myTaxiService, a new web and mobile application to optimize the taxi service in big cities. It should simplify the access of passengers to the service and guarantee a fair management of taxi queues. The users will have to register and login in order to use the application, then they can request a taxi and be informed about the code of the incoming taxi and the waiting time.

We suppose the city is divided in zones and every one of them has a queue of available taxis present in the zone, whose position is calculated according to GPS. After a request arrives, the system informs the first taxi in the queue of the zone from which the request came. If the taxi accept the request, the system sends the confirmation to the user. If not, the system will put it at the end of the queue and forward the request to the next available taxi. To accept ride requests, taxi drivers will have to login through the mobile app like normal users. Then they can set themselves as available and receive ride requests. A user can also reserve a taxi by inserting origin, destination and time of the ride. The request must be submitted at least two hours before. The system will allocate a taxi and notify the user 10 minutes before the ride.

SECTION 1

ACTORS:

1. **Guest:** the guests are users who are not registered yet. They must sign themselves up into the system in order to use the features available to registered users.
2. **Registered user:** this type of user, after successful login, has access to all the features of the application as a customer. They can request rides, be them simple or detailed, and receive notifications after a ride has been confirmed.
3. **Driver:** this type of user, after a successful login, can set himself as available and so can be notified when a new ride request arrives and, if he sees it, to accepts it.

SECTION 1

OUR GOALS:

We have divided the goals for every categories of actors we have identified.

Guests should be able to:

[G1] Register themselves into the system

[G2] Log themselves into the system

Users should be able to:

[G3] See number of available taxis of the zone they're in

[G4] Make a request for a simple ride

[G5] Make a request for a detailed ride

SECTION 1

OUR GOALS:

Drivers should be able to:

[G6] Set themselves as available

[G7] Read and accept ride requests

The system should:

[G8] Notify passengers after the confirmation of a simple request

[G9] Notify passenger ten minutes before the ride reserved through a detailed request

[G10] Forward request to the first taxi in queue

[G11] After 30 seconds, forward the request to the second taxi in queue and put the first at the end

SECTION 2

OVERALL DESCRIPTION:

Product perspective:

The application we will project is both a web application and a mobile application. It will not interact with any other existing application or system. It will be user based and we will not provide any internal interface for administration.

User characteristics:

People that will use our application are the ones interested in using the taxi service in the city. They will be able to request a taxi for a ride without any voice call but just with few clicks.

SECTION 2

OVERALL DESCRIPTION:

Assumptions:

- When the app or the web page is closed the user automatically logs out.
- If a driver sees a ride request, he always accepts it.
- Users can't cancel ride requests.
- If the system sends a notification the user will always receive it.
- Users can make an unlimited number of daily ride requests.
- In case of empty queue 10 minutes before a detailed ride, the application will always manage to find one within the time of the ride.
- Users insert existing addresses when making requests.
- The system knows if the credentials inserted during login belong to a user or a driver, so it will display the correct personal page afterwards.

SECTION 3

FUNCTIONAL REQUIREMENTS

- **[G1] Register themselves into the system**
 - [R1] Guests can only see the login page.
 - [R2] Guests can only access to the registration form.
 - [D1] All the fields must be completed in a formal correct way.
- **[G2] Log into the system**
 - [R1] Guests have to provide valid username a password in order to log themselves into the system.
 - [R2] The wrong insertion of one's own credentials will not allow to log himself into the system.
- **[G3] See number of available taxis of the zone he's in**
 - [R1] The system has to visualize on the personal page the number of taxi available in the zone.

SECTION 3

FUNCTIONAL REQUIREMENTS

- **[G4] Make a request for a simple ride**
 - [R1] Users have to insert the correct departure's address.
 - [D1] The inserted address has to be formed by a street and a civic number.
- **[G5] Make a request for a detailed ride**
 - [R1] Users have to insert the correct departure's address.
 - [R2] Users have to insert the correct destination's address.
 - [R3] Users have to insert feasible date and time for the reservation.
 - [D1] The inserted address has to be formed by a street and a civic number.
 - [D2] Date must look like this: DD/MM/YY, and time like this: HH:MM.
- **[G6] Set themselves as available**
 - [R1] The system has to provide a function through which drivers can inform it of their availability.
 - [R2] Once they're available, the system has to insert them into the Taxi Queue of their own zone.

SECTION 3

FUNCTIONAL REQUIREMENTS

- **[G7] Read and accept ride requests**
[R1] The system has to provide a function to read and accept request
- **[G8] Notify passengers after the confirmation of a simple request**
[R1] The system has to notify the user who made a simple request as soon as possible, this means as a taxi is available, informing him about the waiting time.
- **[G9] Notify passengers 10 minutes before the ride reserved through a detailed request**
[R1] The system has to notify the user who reserved a taxi within ten minutes before the time of the ride.

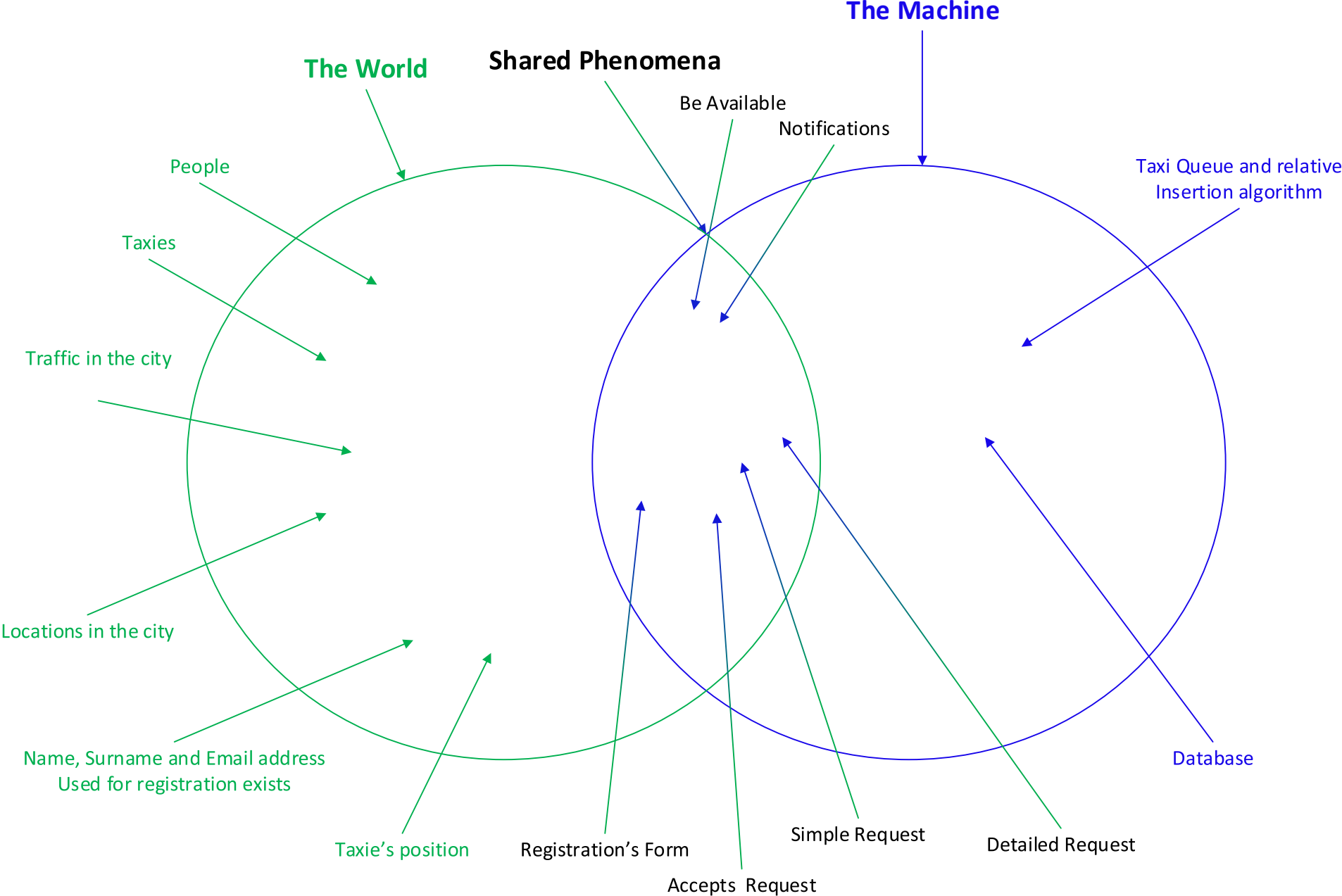
SECTION 3

FUNCTIONAL REQUIREMENTS

- **[G10] Forward requests to the first taxi in queue**
 - [R1] The system has to forward requests to the first taxi in queue allowing him to accept the request.
 - [R2] After a confirmation by a driver the system has to move the driver at the end of the queue and set him as not available.
- **[G11] After 30 seconds, forward the request to the second taxi in queue and put the first at the end**
 - [R1] After a missed confirmation the system has to move the first driver in queue to the last position and forward the request to the second until one driver confirm the request.

SECTION 3

THE WORLD AND THE MACHINE



SECTION 3

SCENARIOS:

Scenario 1

Bob has planned to go to a workshop that will take place on the other side of Milan. He woke up early because he knows that there's always so much traffic in the city. Bob is ready to go with his car when the unexpected happens: the car seems broken. Without wasting any time he opens myTaxi app on his phone. He logs into the system and request a Taxi for a simple ride. John, already logged into the system as driver, sees the simple request on his phone and accepts it. The system immediately notifies Bob telling him John's taxi code and the relative waiting time. Bob has to attend just a few minutes because John's Taxi is only one kilometre away from him. Bob is happy because he will arrive on time despite the unexpected discovery of having his car broken.

SECTION 3

SCENARIOS:

Scenario 2

Bob, during the lunch break at the workshop, realizes that at the end of the event he doesn't have his car because he arrived in taxi. He also realizes that the end of the workshop is scheduled at 6p.m, critical time for traffic in the city with several waiting time for taxis. In order to this consideration, Bob opens myTaxi app and reserves a taxi for 6.15 p.m. to be sure that a taxi will be there at that moment. At 6.05 p.m. Bob's phone rings. A notification with the code of taxi that will arrive to him has just been received from his phone.

SECTION 3

SCENARIOS:

Scenario 3

It's 12a.m. and John, a taxi driver, decides to have a coffee break but he also decides to not log out from the app because he is the third in the Taxi Queue. He thinks he can drink a coffee and smoke a cigarette before he becomes the first of the Taxi Queue. . . but he is wrong. The other two driver that were in list before him log out from the system and John becomes the first in the Taxi Queue. A request arrives but Bob doesn't see it because he is still drinking his coffee. After 30 seconds John hasn't accepted yet and the system puts him into the last position forwarding the request to the second in list.

SECTION 3

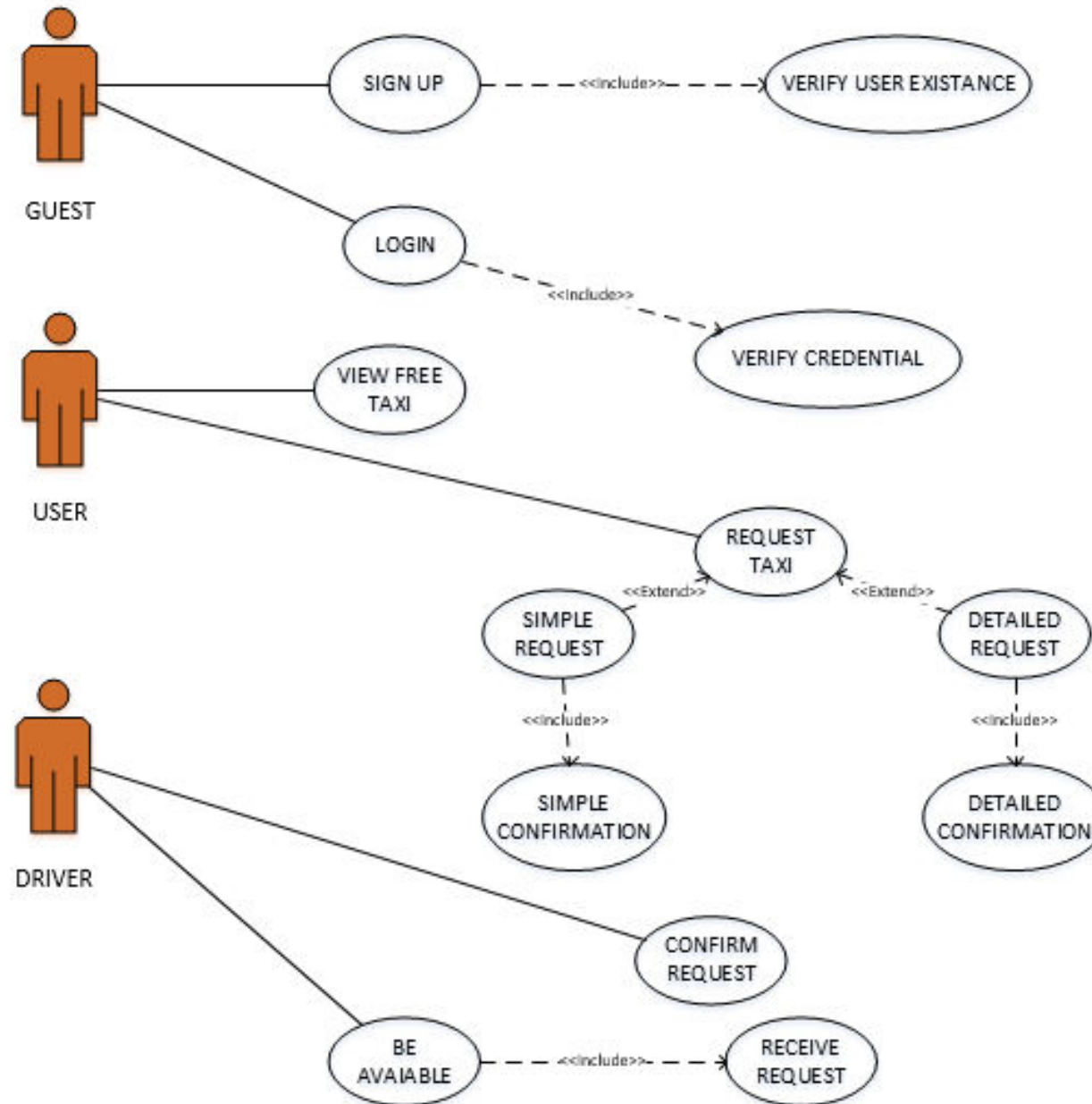
SCENARIOS:

Scenario 4

Giuly decides to go to EXPO in the evening. She knows that the queue to enter is very long so she decides to go there at 5 pm in order to be in a good position when the gate will open. Giuly is scared to take the train to go there because she would be going back home alone. She decides to go there with a taxi and simply she opens myTaxi app and requests a taxi for a simple ride. Unfortunately, the system has detected that there is no one taxi available in her zone at the moment. Giuly immediately receives a notification that informs it of the problem inviting her to attend for the first available taxi.

SECTION 3 - UML MODELS

USE CASE:



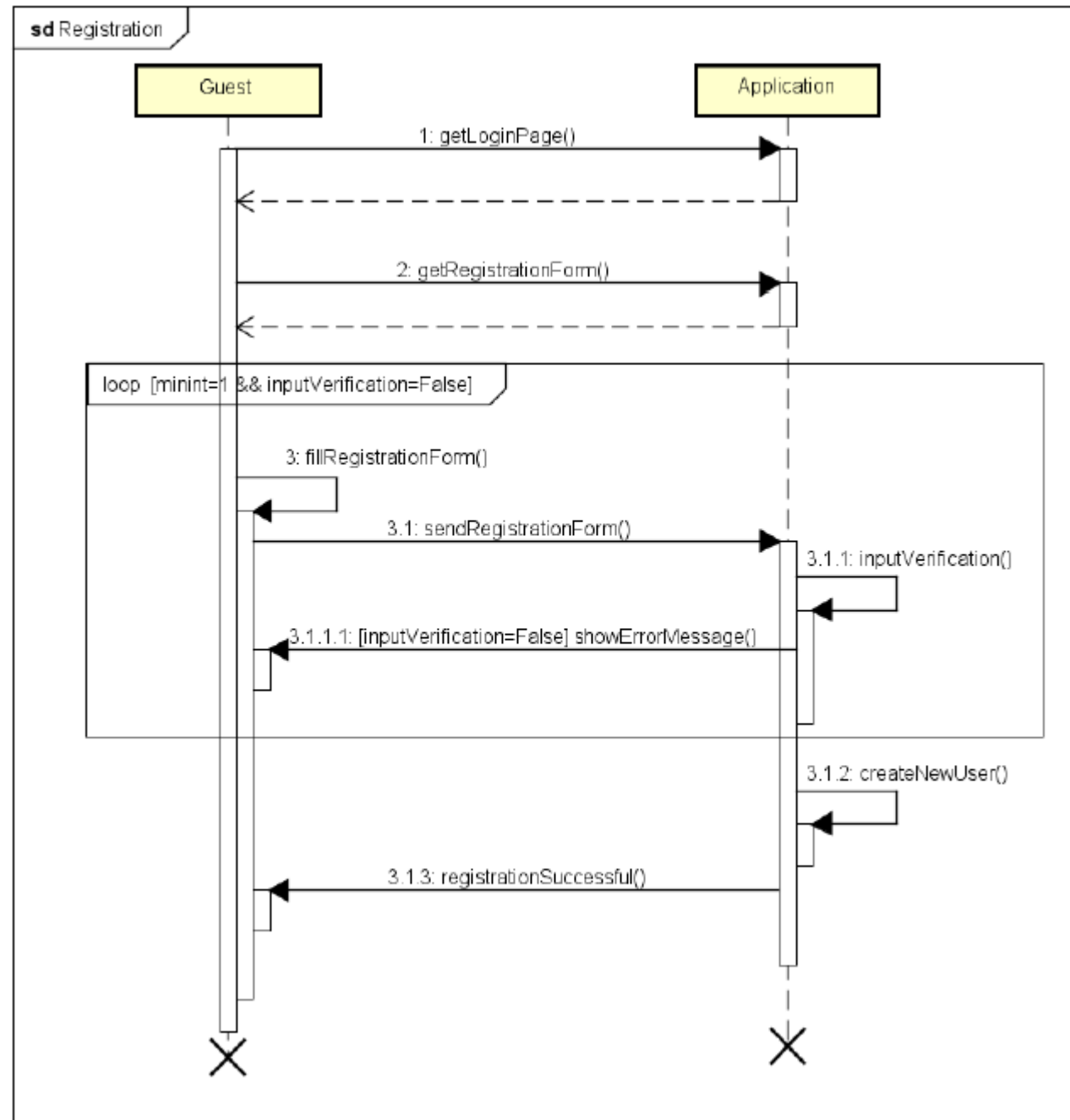
SECTION 3 - UML MODELS

SIGN UP:

Actor	Guest
Goal	[G1]
Entry conditions	NULL
Flow of events	<ol style="list-style-type: none">1. Guest on the login page clicks on the “Register” button to start the registration process.2. Guest fills in at least all the mandatory fields.3. Guest clicks on “Sign up” button.
Exit conditions	Guest successfully ends the registration process. From now on he will be able to log in with the credentials submitted before and start using the application as a User.
Exceptions	<ol style="list-style-type: none">1. Some mandatory field is not filled.2. Chosen username is already taken.3. Email is already in use. <p>The exceptions are handled by showing an error and making the guest repeat the registration process from point 2 of the event flow.</p>

SECTION 3 - UML MODELS

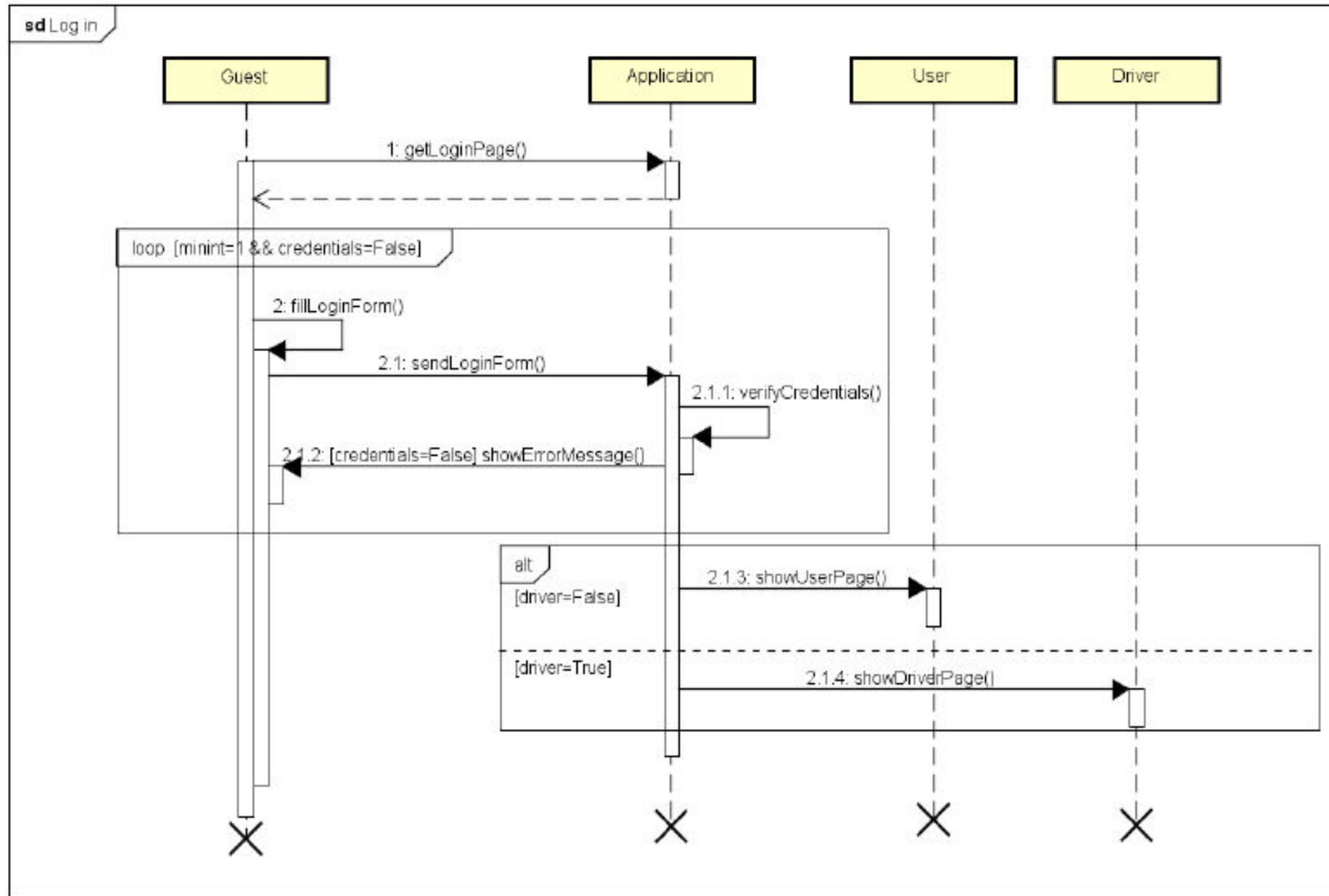
SIGN UP:



Actor	Guest
Goal	[G2]
Entry conditions	Guest must be already registered into the system.
Flow of events	<ol style="list-style-type: none">1. Guest inserts username and password.2. Guest clicks on the “Log in” button.
Exit conditions	The credentials are successfully validated, the application checks if the credentials belong to a user or a driver and shows their respective personal page.
Exceptions	An error is shown when the credentials inserted are not correct, making the user repeat the login process.

SECTION 3 - UML MODELS

LOG IN:



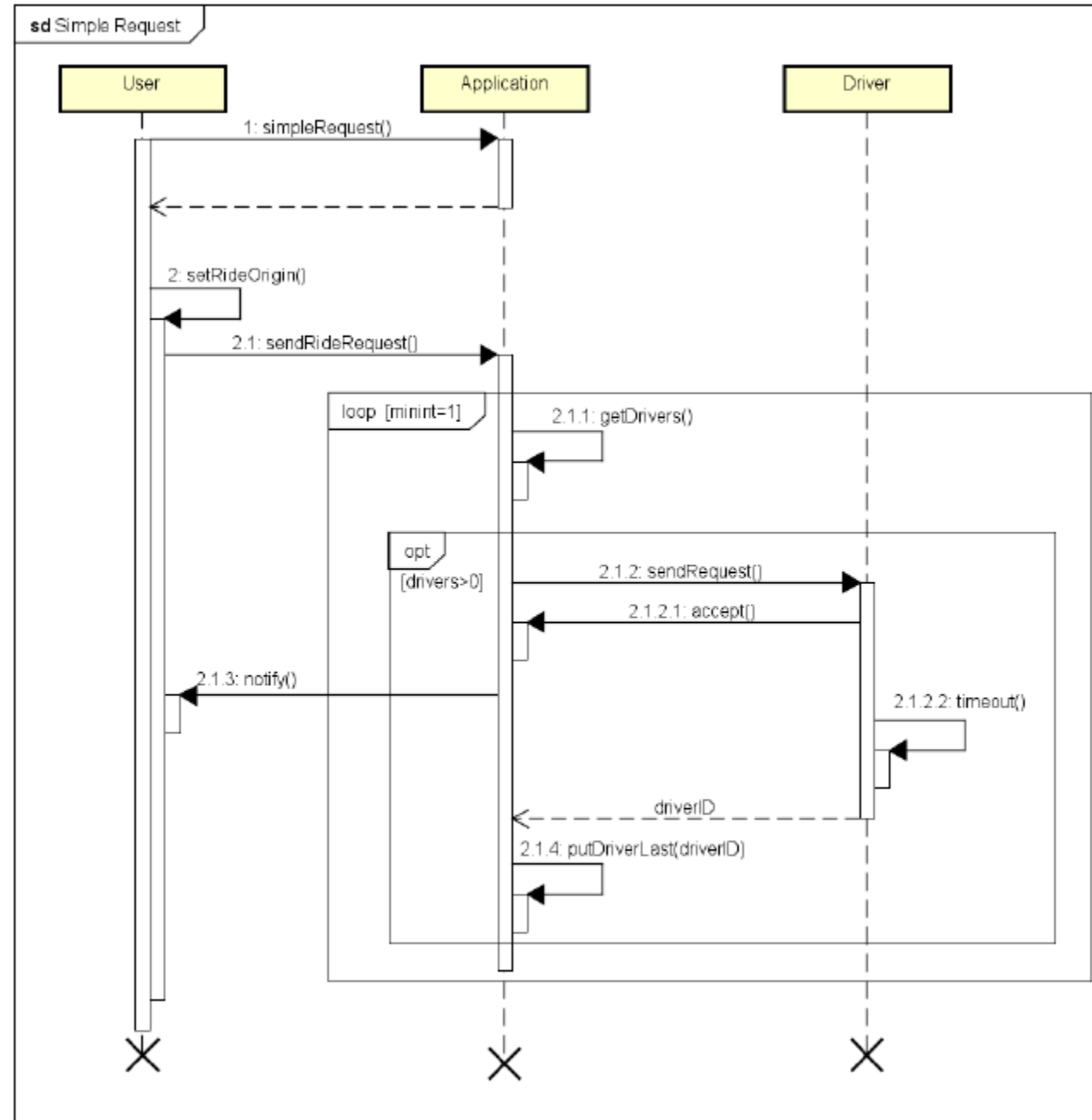
SECTION 3 - UML MODELS

MAKE SIMPLE REQUEST:

Actor	User
Goal	[G4]
Entry conditions	User must be logged in.
Flow of events	1. User clicks on the “Simple request” button.
Exit conditions	User successfully makes a request which is still pending until a driver accepts it.
Exceptions	None. If there’s no taxi, the system waits until one becomes available.

SECTION 3 - UML MODELS

MAKE SIMPLE REQUEST:

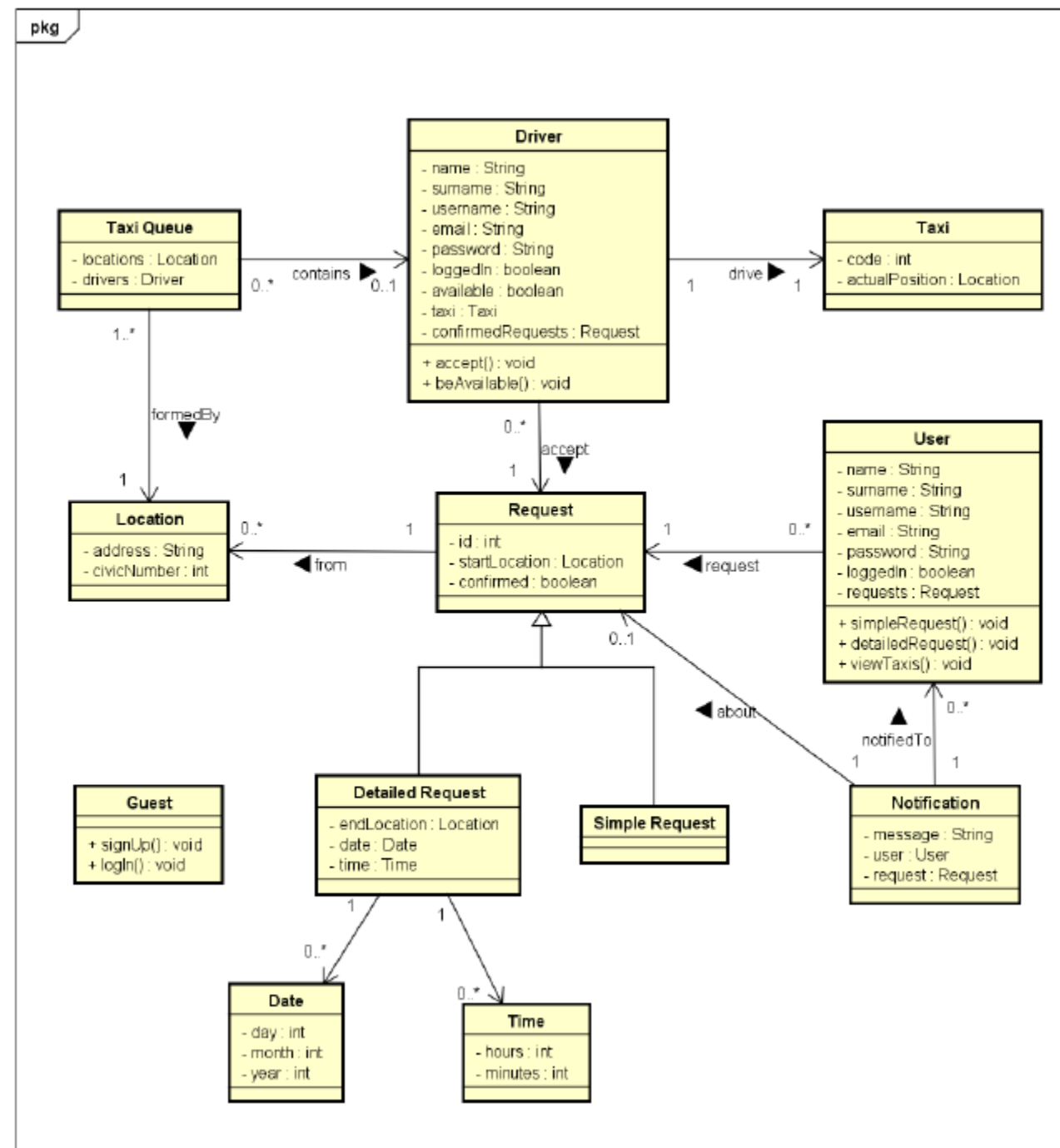


SECTION 3

UML MODELS

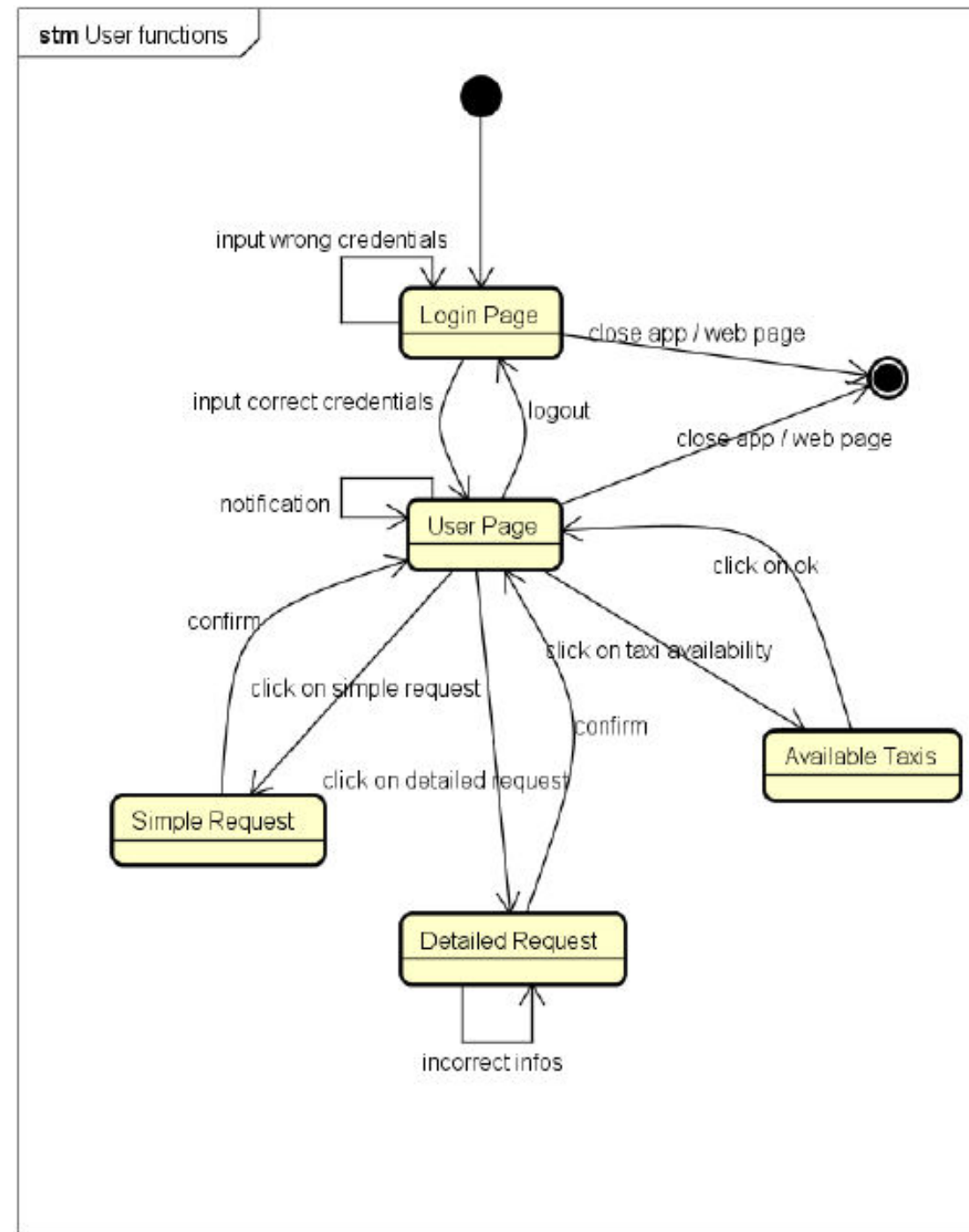
CLASS

DIAGRAM:



SECTION 3

UML MODELS

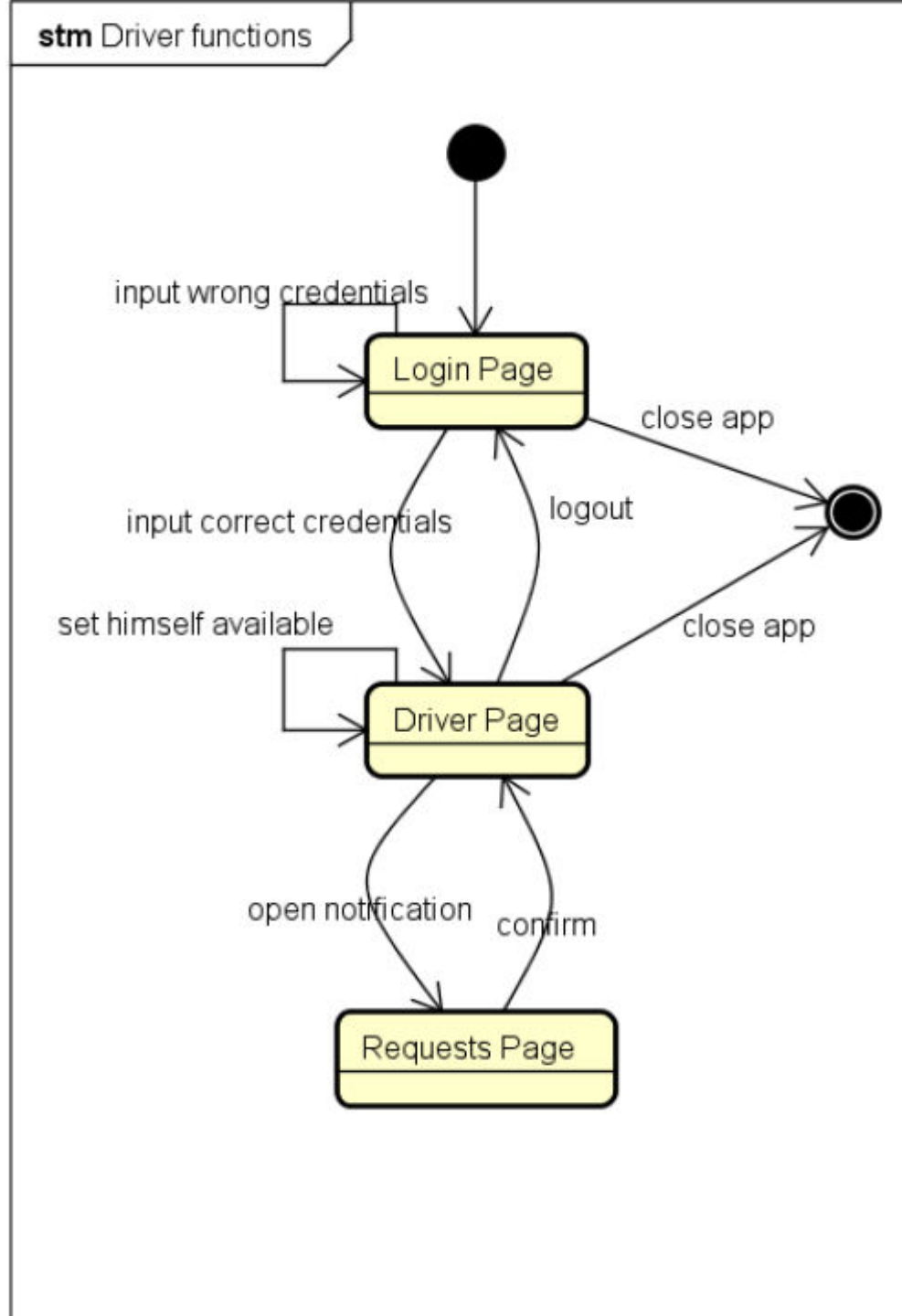


STATE MACHINE DIAGRAM

USER FUNCTIONS:

SECTION 3

UML MODELS

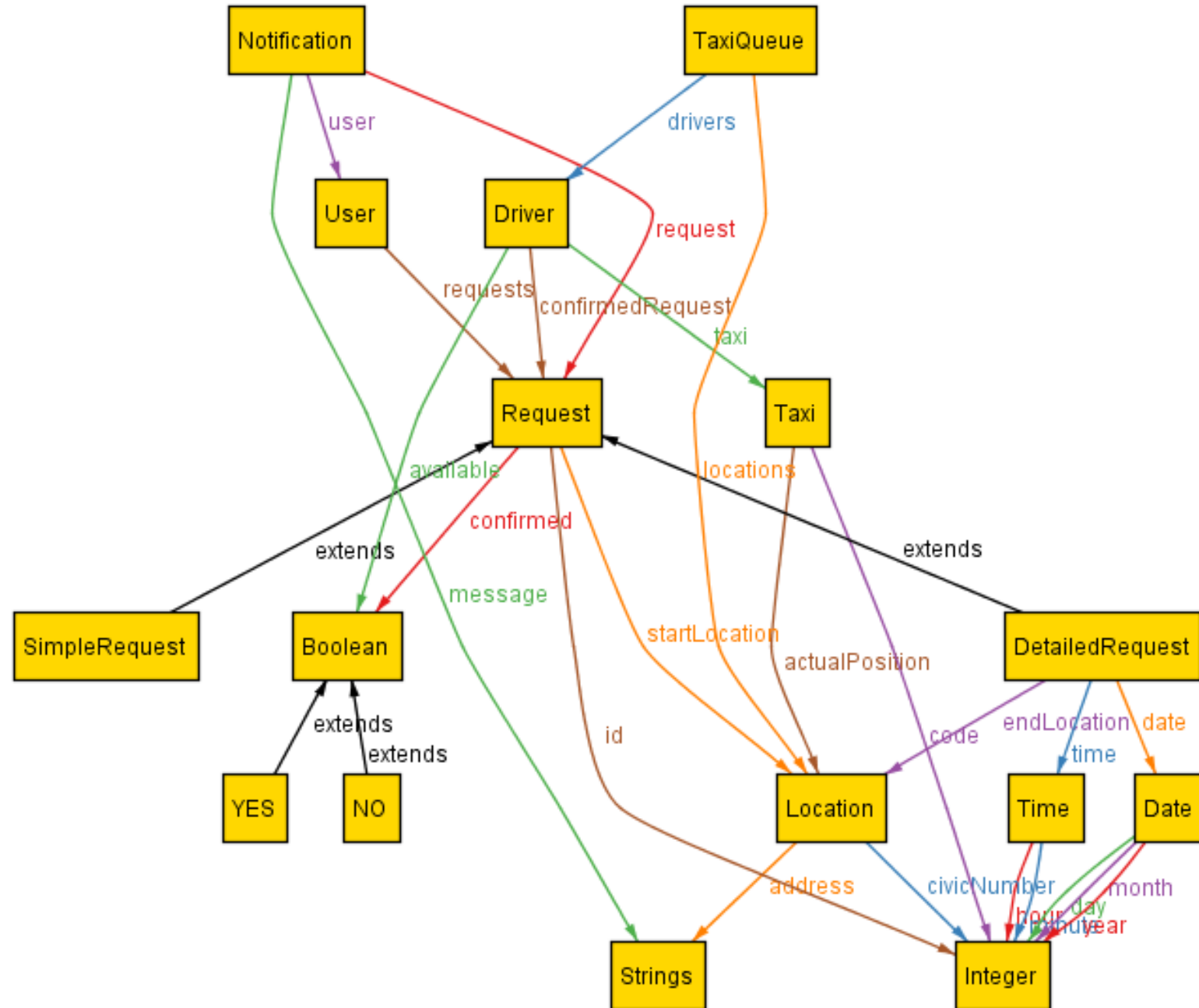


STATE MACHINE DIAGRAM

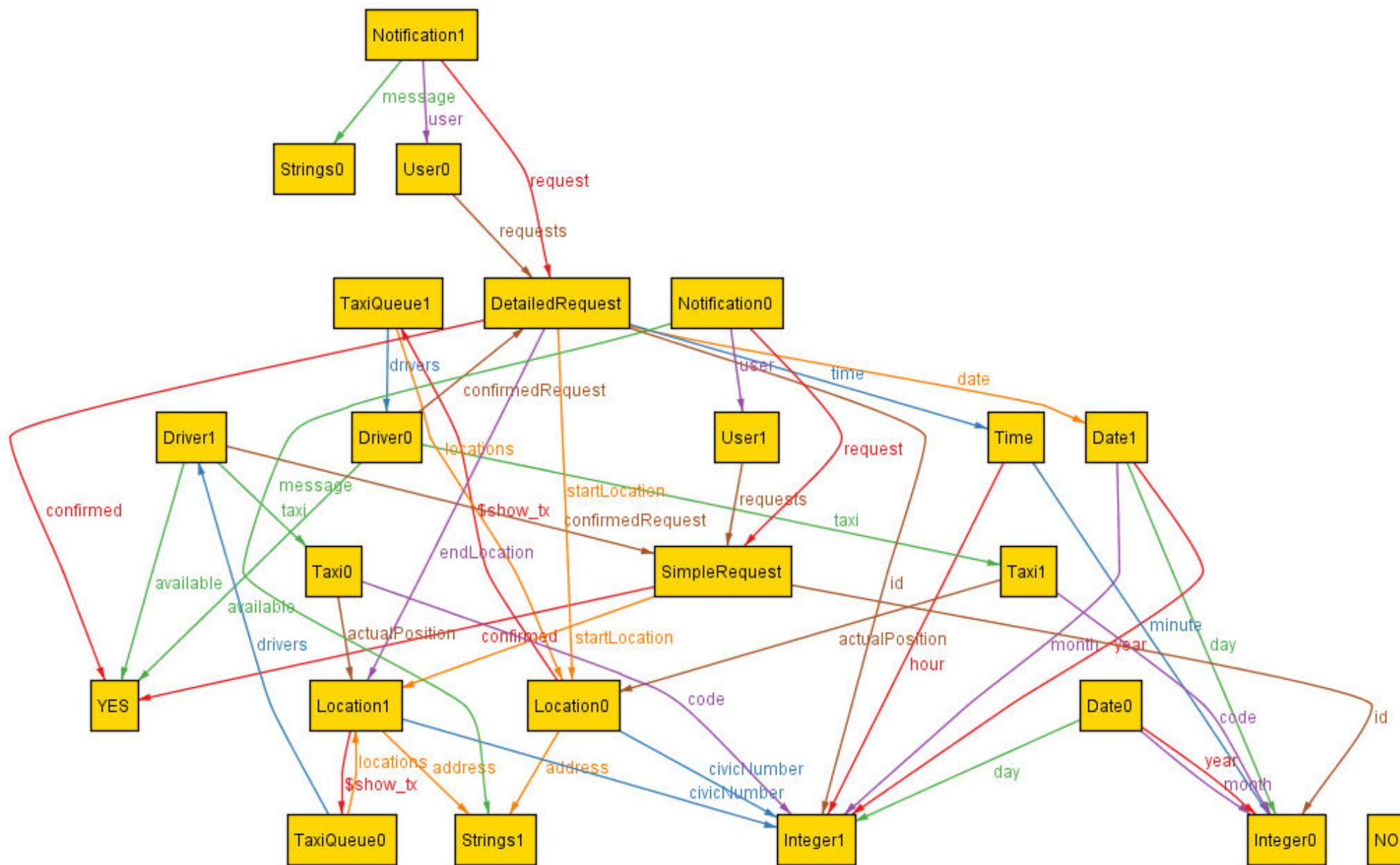
DRIVER FUNCTIONS:

SECTION 4

ALLOY METAMODEL:



EXAMPLE OF GENERATED WORLD:





Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: “myTaxiService”
Design Document

Massimo Schiavo, Marco Edoardo Cittar

December 4, 2015

Summary

- **Section 1:** Introduction
- **Section 2:** Architectural design
- **Section 3:** Algorithm design
- **Section 4:** User interface design
- **Section 5:** Requirements traceability

Introduction

- Purpose
 - This software design specification is made with the purpose of outlining the software architecture and design of myTaxiService application system in detail.
 - The documents will provide to the developers an insight in meeting client's needs efficiently and effectively. Moreover, the document simplifies communication and understanding of the system by providing several views of the system design.

Introduction

- Scope
 - The software design document would demonstrate how the design will accomplish the functional and non-functional requirements defined in the RASD. The document will provide a framework to the programmers by describing the high level components and architecture, sub-systems, interfaces and algorithm design. This is achieved through the use of architectural patterns, sequence diagrams and user interfaces.

Architectural design

- The next chapter of the document describes architectural design of myTaxisystem: the high level components and their interactions, suitable architectural patterns, physical arrangement of components and design decisions applied to the whole system.

High level components

- **AccessManager:** This sub-system consists of all the components responsible of registration and login process. It authenticates users and drivers, allowing them to use their relative functionalities. It's also responsible for guest's registration.
 - Registration
 - Authentication

High level components

- **User:** This sub-system consists of all the components that implement the functions related to the user's operations such as requesting a taxi for a simple ride, making a reservation for a taxi and viewing the number of available taxis in his zone.
 - SimpleRequest
 - DetailedRequest
 - NumberOfTaxi
 - Notification

High level components

- **Driver:** This sub-system consists of all the components that implement the functions related to the driver's operations such as informing about their own availability and accepting an user's request.
 - BeAvailable
 - Conrmation

High level components

- **DataManager:** This is the major sub-system that is responsible for the security of sensible data. It communicates directly with the database providing several operations on it such as adding new users, adding new drivers, checking saved data with data given by the user during the login process.
 - AddData
 - QueryData

High level components

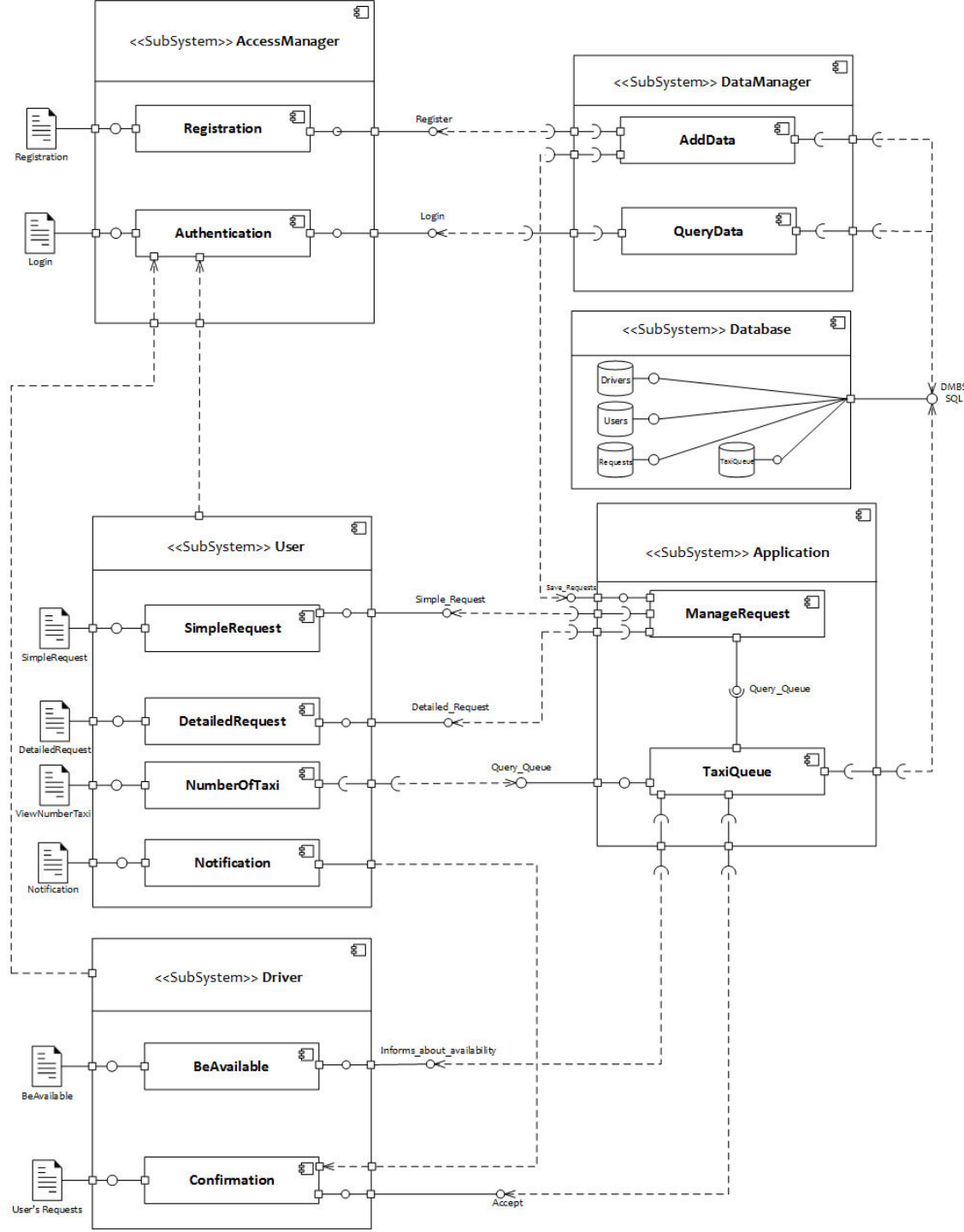
- **Database:** Essentially, it is formed by the databases and give all the functionalities through which operate directly on the database.
 - Drivers
 - Users
 - Requests
 - TaxiQueue

High level components

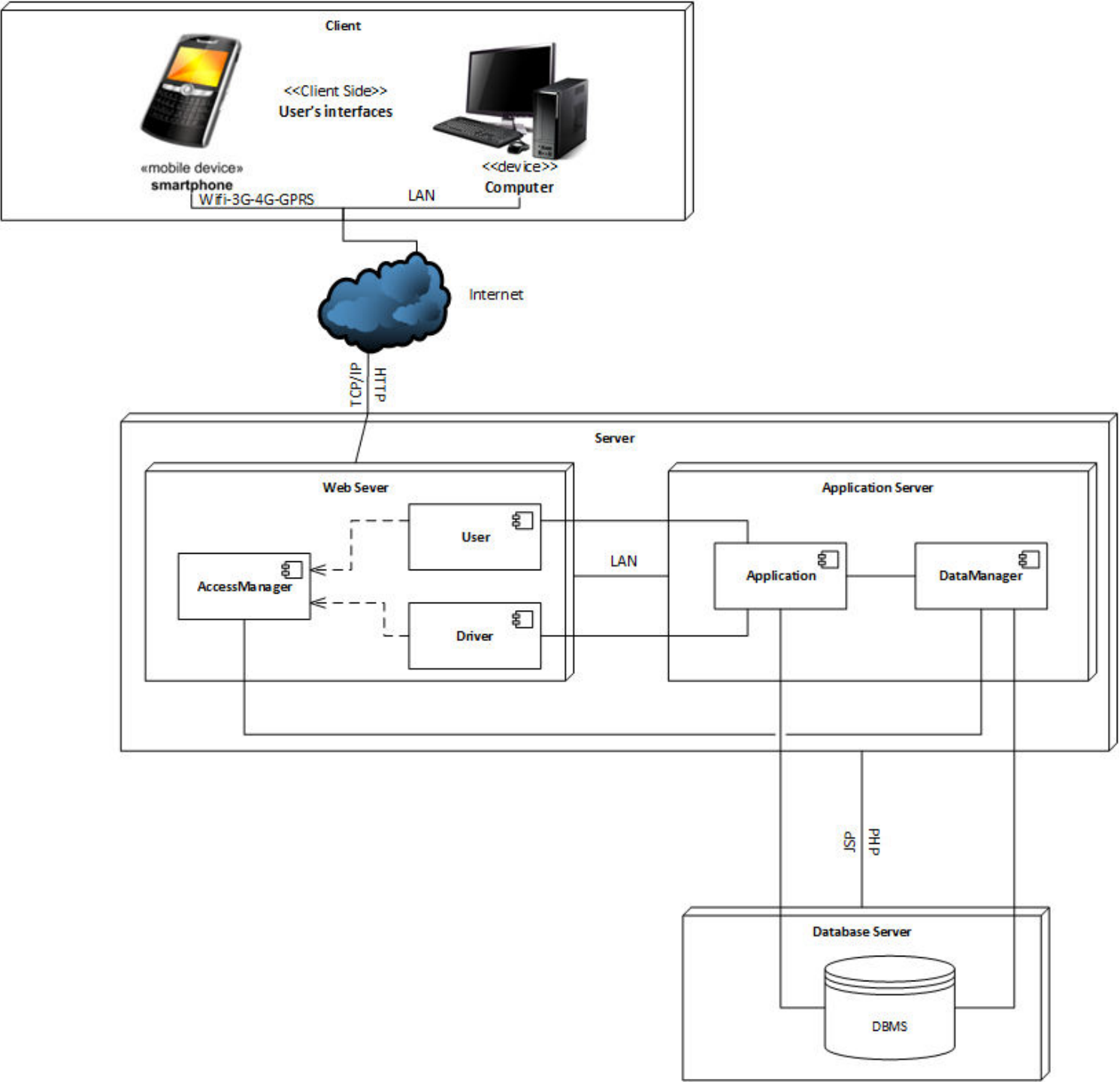
- **Application:** This is the major sub-system that is responsible of the logic of myTaxi app. In particular it manages all the requests done by the users forwarding them to the drivers according to the policy of forwarding.
 - ManageRequest
 - TaxiQueue

Section 2: Architectural design

Component view



Deployment view



Component interfaces

- **AccessManager**
 - **Registration:Register** - This interface allows a guest to register himself into the system by giving the mandatory information.
 - **Authentication:Login** - This is the interface that allows the users/drivers to log into the system. This will guide the user/driver to his relative home page.
- **DataManager**
 - **AddData**: This component uses interfaces provided by other components in order to grant a major grade of security to the data tier interposing between the presentation tier and the data one.
 - **QueryData**: This component, like AddData, communicates with the data tier querying it in order to check user's information given by the user interface.

Component interfaces

- **Database**

- The relevant interfaces given by this tier are the most common known methods through which it's possible to operate, manipulate and querying a database.

- **Driver**

- **BeAvailable:Informs_About_Availability** - This is the interface that allows a driver to inform the system about his current availability. It can be changed by both system and driver according to what is written in the RASD.
- **Confirmation:Accept** - This is the interface through which a driver can accept an incoming user's request forwarded by the system.

Component interfaces

- **User**
 - **SimpleRequest:Simple_Request** - This is the interface that allows the users to ask for a taxi for a simple ride.
 - **DetailedRequest:Detailed_Request** - This is the interface that allows the users to reserve a taxi.
 - **NumberOfTaxi** - This component uses a required interface provided by the TaxiQueue in order to query it and know the number of available taxi in relation to the current position given by the GPS.
 - **Notification**: This is a component that only has a user's interface that depends on the confirmation of the user's own request.

Component interfaces

- **Application**

- **ManageRequest:Save_Request:** This interface is given to the manager data component in order to retrieve information about requests and save them in the database.
- **ManageRequest:Taxi_Queue:** This interface is the major central point of the application because it communicates with the other components whose tasks are to handle requests incoming from the users and forward them correctly to the drivers.
- **TaxiQueue:Query_Queue:** This interface only allows to query the number of available taxis.
- **TaxiQueue:** This component, with the required interfaces it needs, is the central point of our application because, as written before, it is responsible of the correct management of requests, users and drivers on whose the service provided by our application is based.

Section 2: Architectural design

Selected architectural styles and patterns

- myTaxiService will be developed under two main architectural styles/patterns:
 - **MVC** architectural style
 - **3 tier Client/Server** Architecture

Selected architectural styles and patterns

- **MVC Architecture Style (Model - View - Controller)**
 - MVC separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other.
 - 1. The Model component that manages the system data and associated operations on that data.
 - 2. The View component that defines and manages how the data is presented to the user.
 - 3. The Controller component that manages user interaction and passes these interactions to the View and the Model.
 - We will use this MVC Style for myTaxi system because there are multiple ways to view and interact with the data. We have decided to use MVC architectural style to separate the application logic with the interface. The main advantage of this style is that it allows the data to change independently of its representation and vice versa.

Selected architectural styles and patterns

- **Three-Tier Client Server Architecture**

- In a client-server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them. We will use this 3-Tier Client Server Architecture, because data in a shared database has to be accessed from a range of locations, in this way servers can be replicated. In particular, there are three tiers:

- **Data Tier**
- **Application Tier**
- **Client Tier**

Selected architectural styles and patterns

- **Data Tier**

- The data tier maintains the applications data such as Users' data, Drivers' data, Request data, TaxiQueue data and the SQL queries. It stores these data in a relational database management system (RDBMS). All the connections with the RDBMS are managed in this tier.

- **Application Tier**

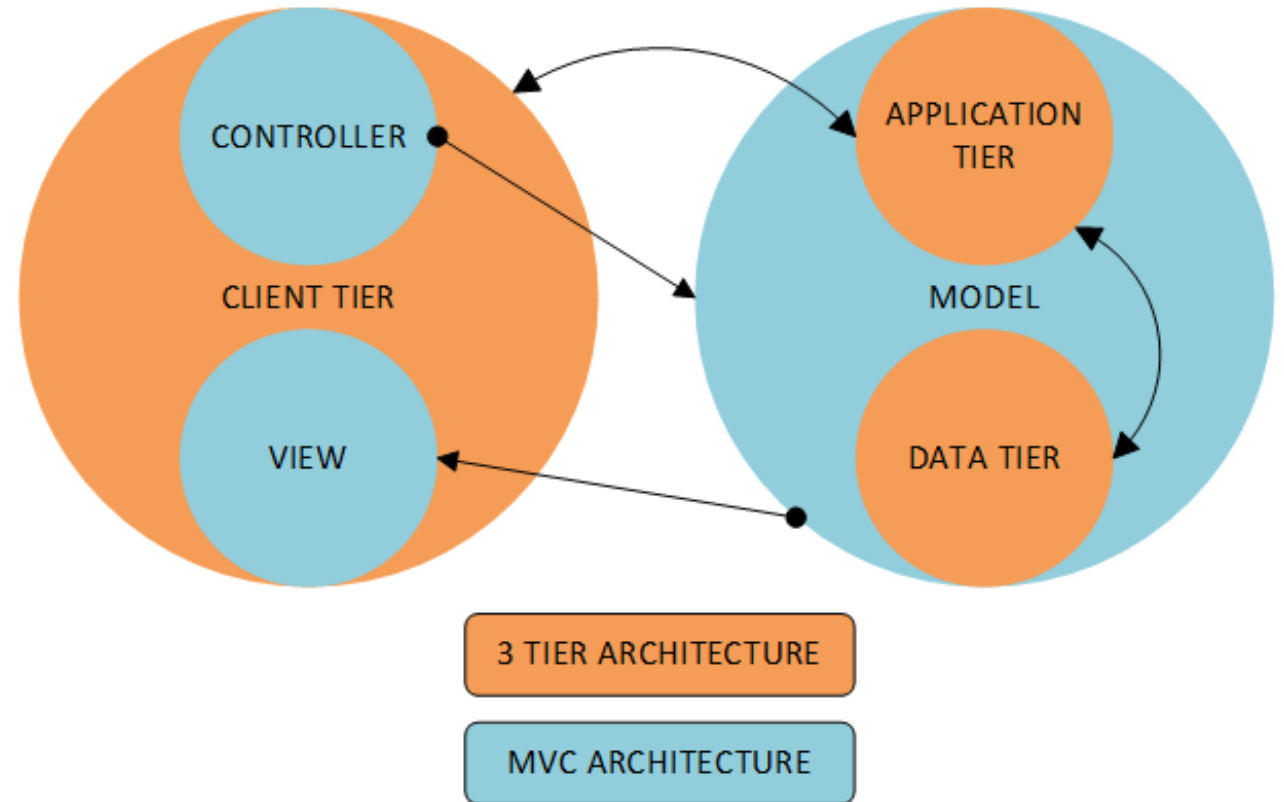
- The application tier (web/application server) implements the business logic, controller logic and presentation logic to control the interaction between the application's clients and data. Business rules enforced by the business logic dictate how clients can't access application data directly and how applications process data.

- **Client Tier**

- The client tier is the application's user interface connecting data entry forms and client side applications. It displays data to the user. User interact directly with the application through user interface. The client tier interacts with the web/application server to make requests and to retrieve data from the database. It then displays to the user the data retrieved from the server.

Section 2: Architectural design

In the picture it's shown how we have combined the two styles/patterns. On the presentation tier we have the view and the controller that invokes methods on the model in which there are the two other tiers. In particular, a change in the model come from the controller to the application tier which then forwards it to the data tier.



Other design decisions

- Object oriented software development methods
 - Improved software maintainability
 - Faster development
 - Lower cost development
 - Improved software development productivity
 - Higher quality software
- Three-tier client server architecture
 - As more users access the system a three-tier solution is more scalable than the other solutions because you can add as many middle tiers as needed to ensure good performance.
 - Security is also the best in the three-tier architecture because the middle layer protects the database tier.

Other design decisions

- MVC Architectural Pattern
 - It should interact with other machines or users effectively.
 - For more efficient interaction, the system should have flexible interfaces.
 - MVC can be taken as for a popular and easy to handle web application development style that has the feature of separating the presentation and intermediate logics.
 - It's easy to code and provides well defined interfaces within each logic.

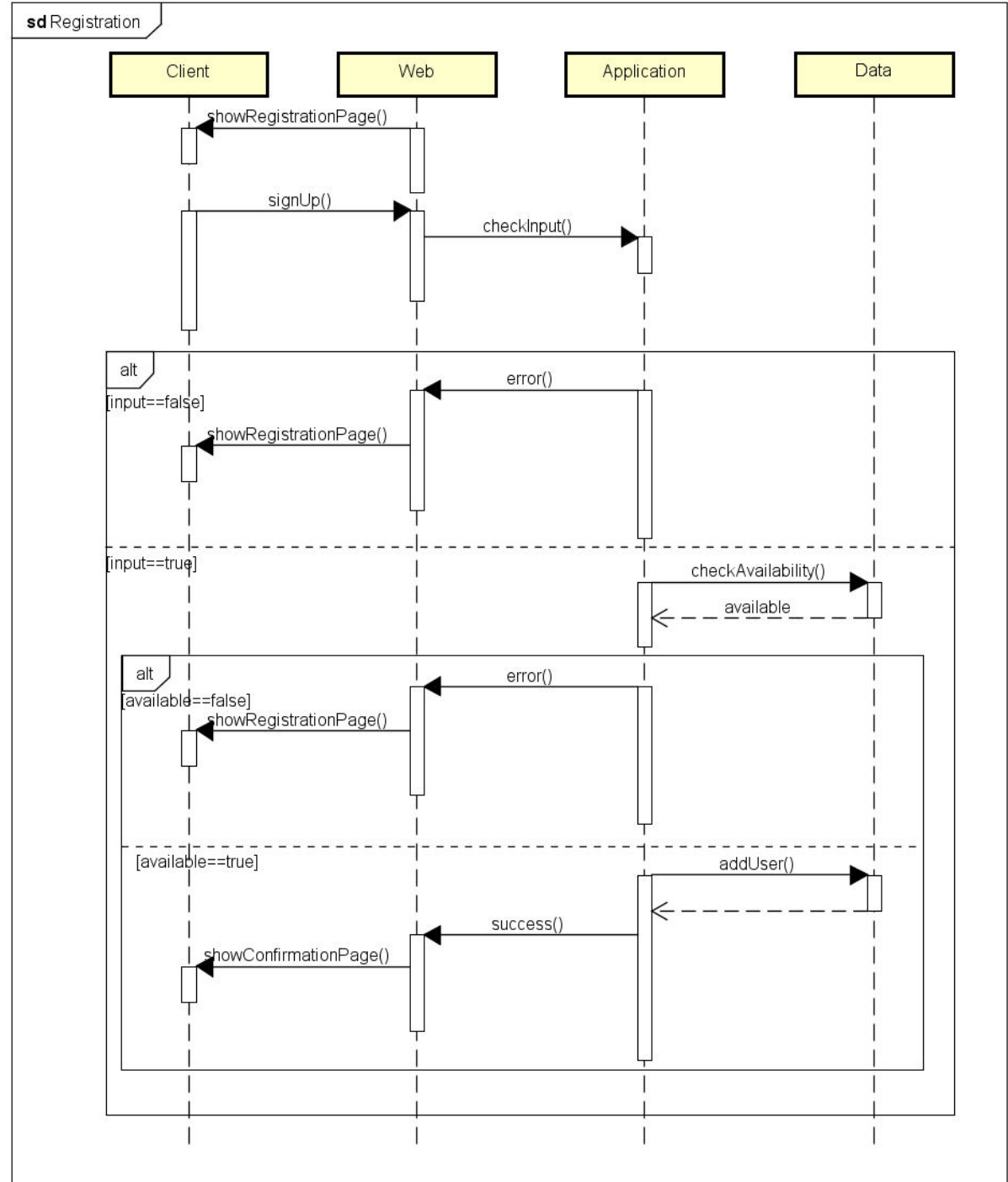
Section 3: Algorithm design

Runtime view

Registration

The algorithm takes in account the completeness, the correctness and the availability of the information inserted. After the guest submits the information the application checks if all the mandatory fields are filled and if the confirmation fields, such as password and email, correspond to the respective previous ones.

If one of the above doesn't apply, the application shows again the registration page to the guest. If they do, then it looks up into the database to verify that the username and email inserted are not already in use. If they don't, then the user is added to the database and is shown a confirmation page, inviting him to log in., otherwise the registration page is shown again.



Section 3: Algorithm design

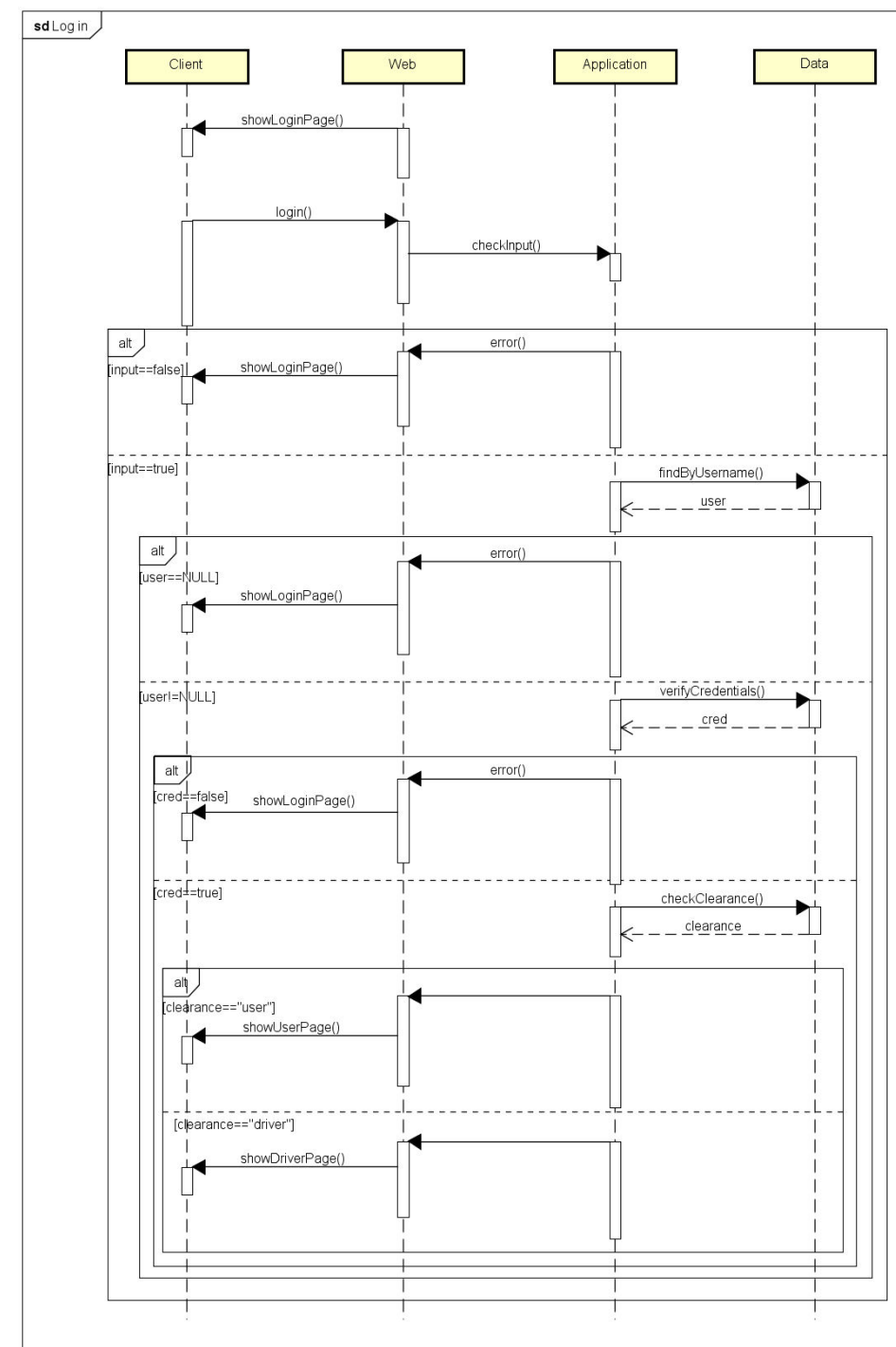
Runtime view

Log in

After the guest inputs his credentials, the application first checks if both fields are filled, then looks for the username in the database to see if it exists. If it does, it checks if the password matches too.

If all of these steps are verified, the guest is logged in and is shown his personal page, otherwise the application shows the login page again.

After the password verification, the application will check the privileges of the account that is about to log in and will consequently show the respective page, be it user's or driver's.



Section 3: Algorithm design

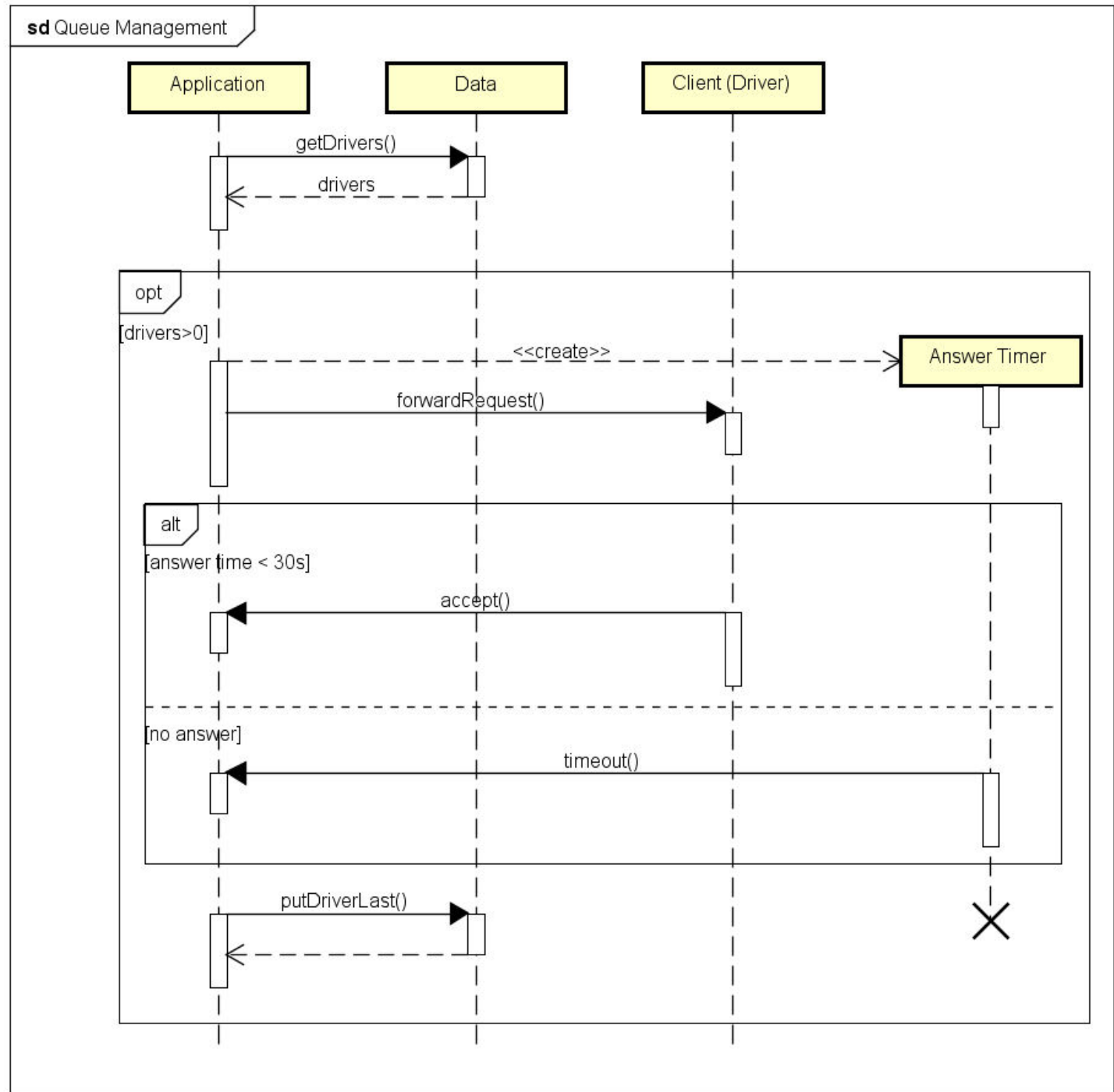
Runtime view

Queue management

This sequence diagram represents how the application manages to forward a request to a driver.

After a user makes a ride request, the application will then begin to allocate the first available taxi, otherwise if there's none it will just wait until one becomes available.

After forwarding the request to the taxi, the system will wait 30 seconds, if the driver answers the call he will be in charge for that request, or else the system forwards the request to the next available taxi in queue. In both cases the driver will be put at the end of the queue.



Section 3: Algorithm design

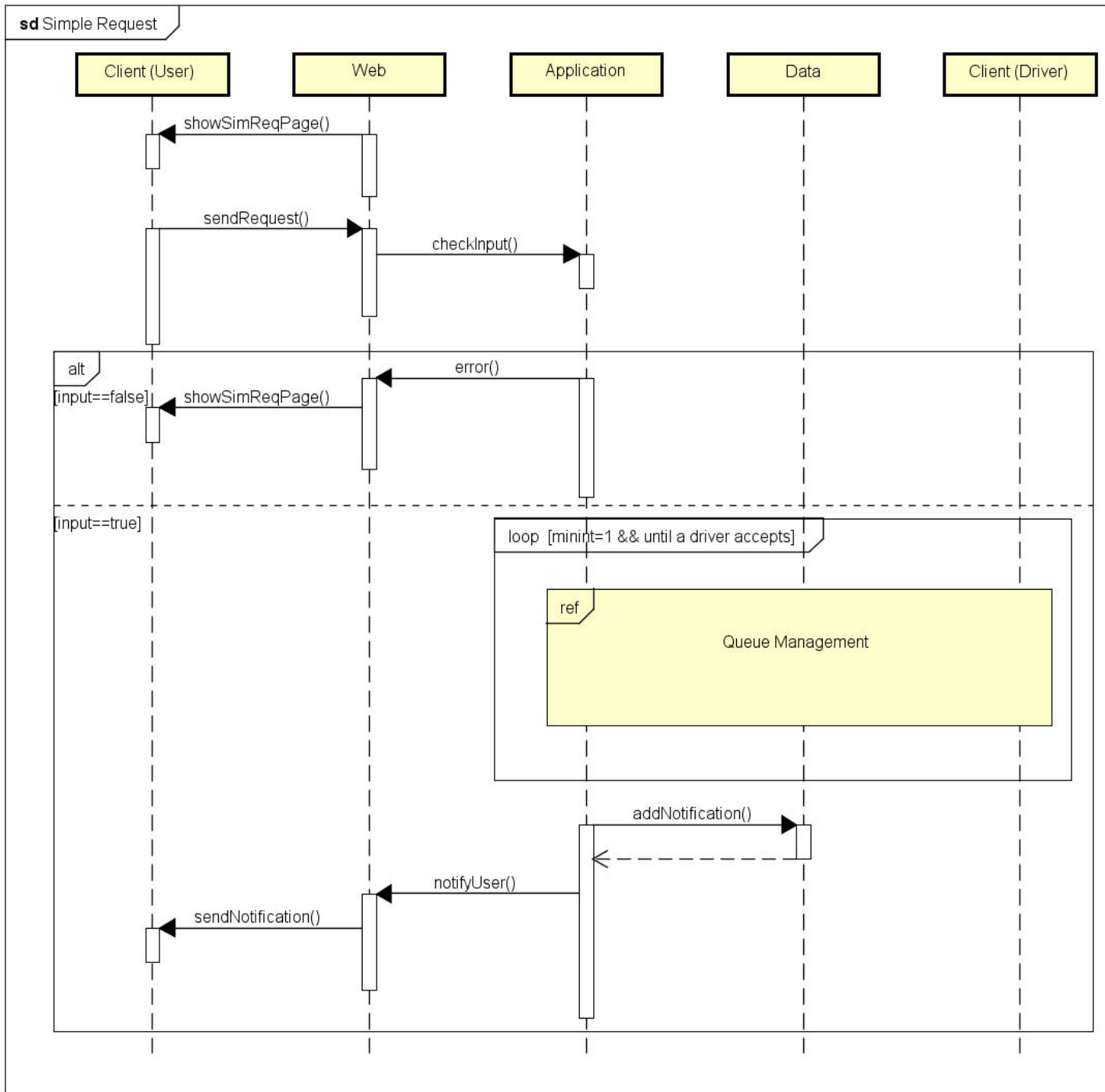
Runtime view

Simple request

After the user clicks on the Simple request button, he is shown the simple request page.

After he inserts the origin of his ride and press the Request button, the application will then begin to a taxi according to the Queue Management algorithm.

Once the taxi is allocated, the user is informed with a notification containing the code of the taxi and the waiting time.



Section 3: Algorithm design

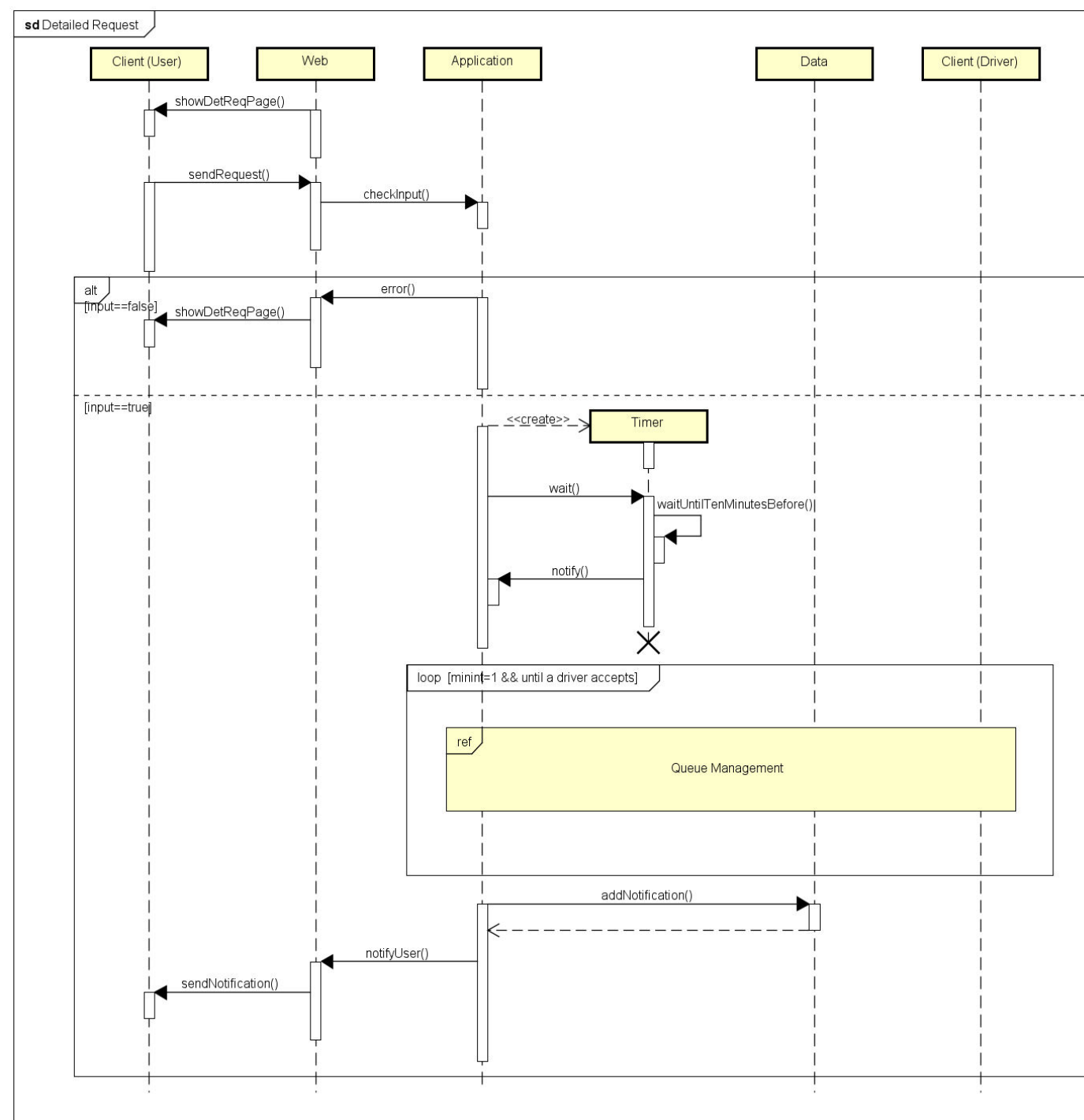
Runtime view

Detailed request

After the user click on the Detailed request button, he is shown the detailed request page.

The user must fill the form and click on the Request button. The application will then check if all the fields are filled and if time and date are formally correct. If that's so, the system will wait until 10 minutes before the ride and then begin to allocate a taxi for the request.

The following taxi allocation, queue management and notification works as the one in the simple request.

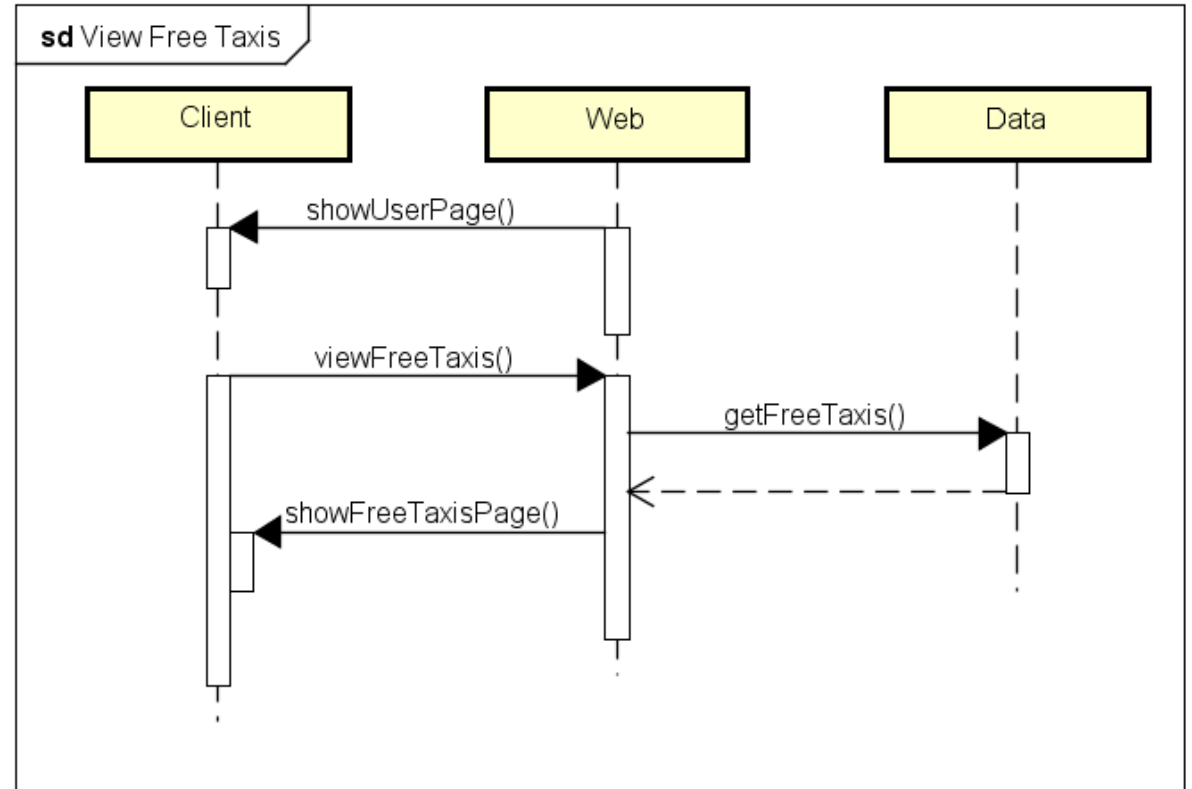


Section 3: Algorithm design

Runtime view

View free taxis

The users can look at the number of available taxis in their zone. By clicking on the View available taxis button, the application will query the database about it, and then return the result to a new page.

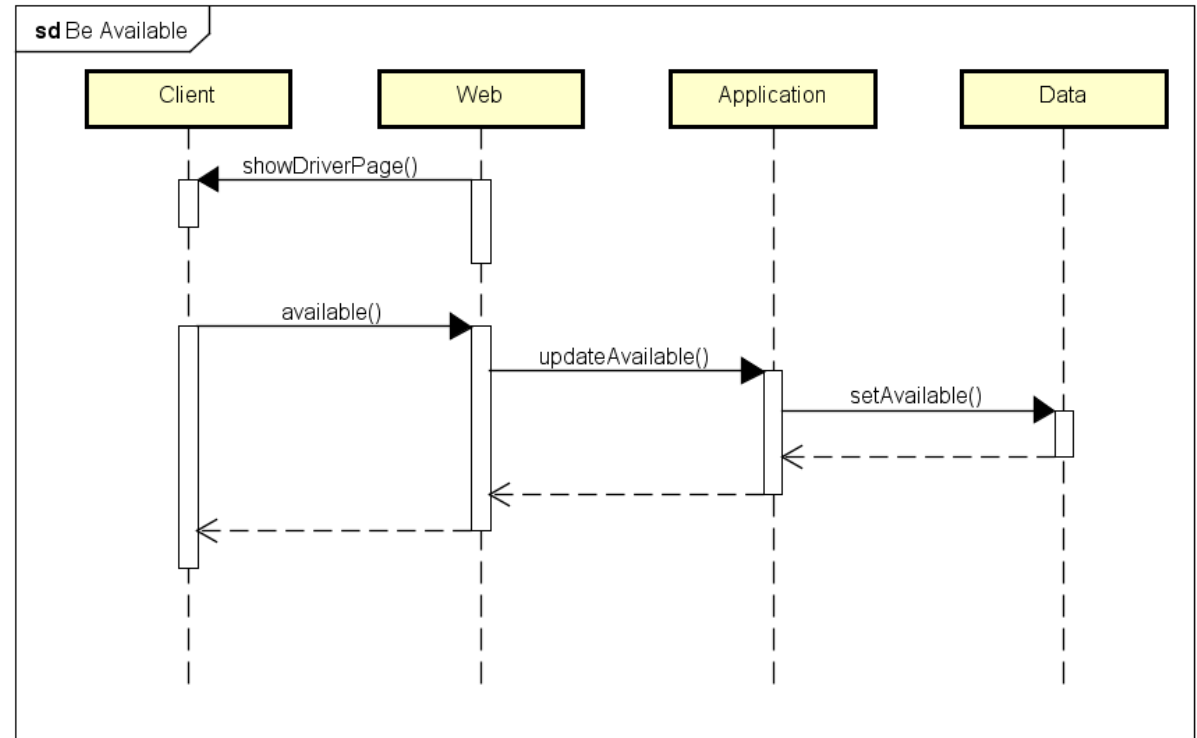


Section 3: Algorithm design

Runtime view

Be available

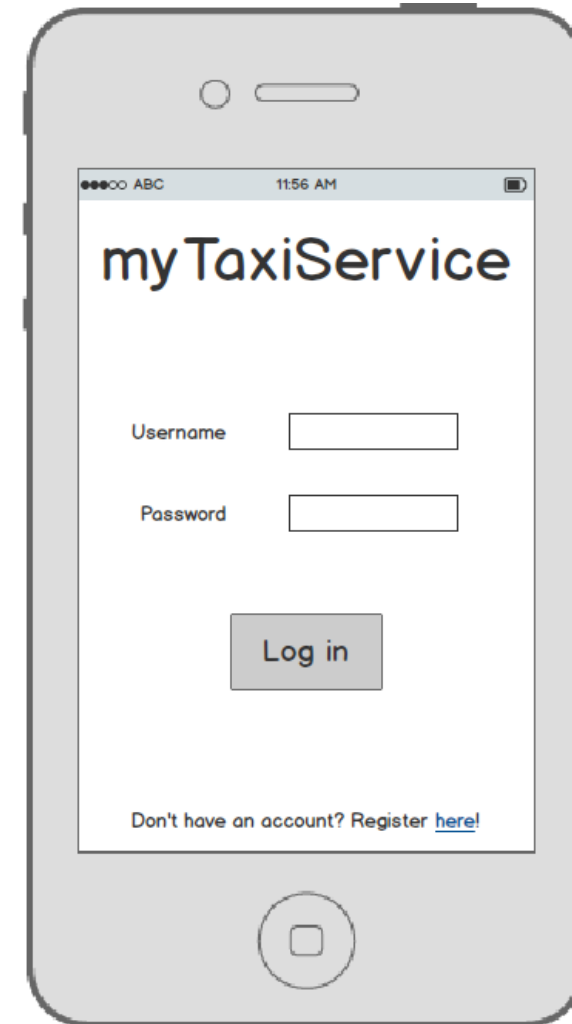
The drivers can use the switch on their personal page to toggle their own availability. After the button is switched, the application changes in the database the available field of the driver from false to true or vice versa.



Section 4: User interface design

Log in page

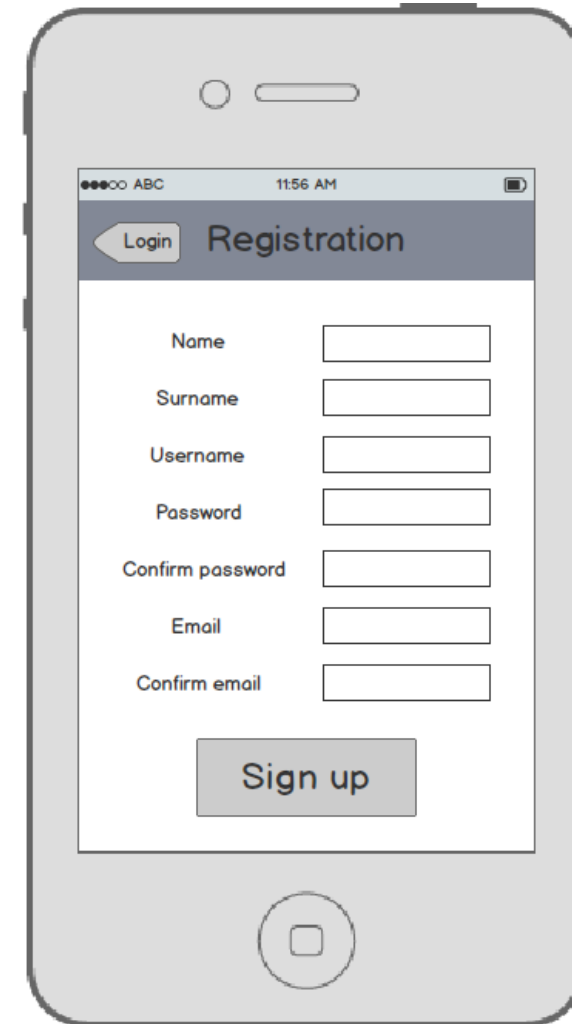
This is the page that shows up when the app is opened. From here guests can log in or register themselves if they don't have an account yet.



Section 4: User interface design

Registration page

Here guests can register themselves by filling all the mandatory fields (the ones shown).



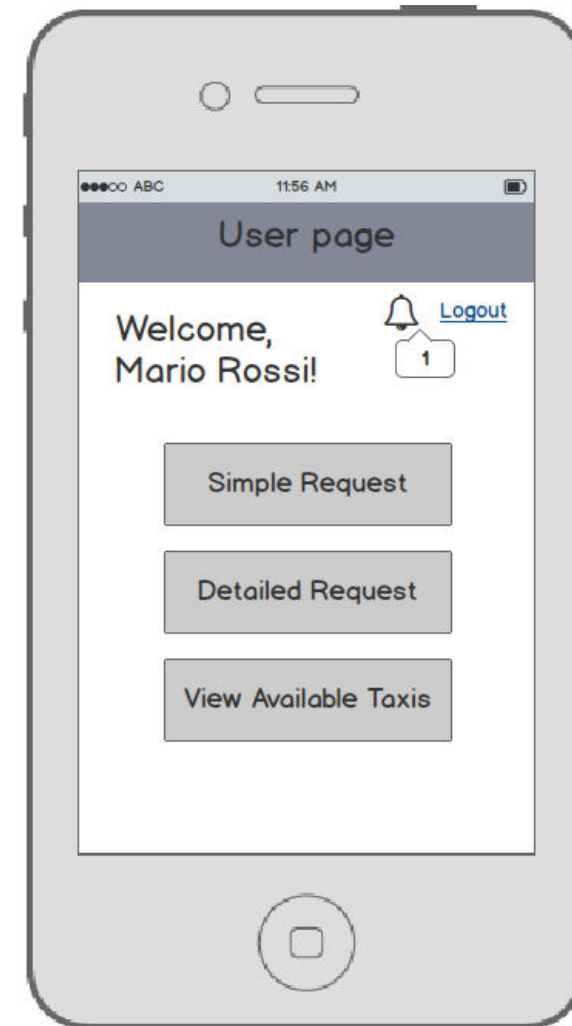
The image shows a mobile application interface for a registration page. The screen is framed by a grey border representing the phone. At the top, there's a status bar with signal strength, 'ABC', '11:56 AM', and a battery icon. Below this is a dark blue header with a 'Login' button on the left and the title 'Registration' on the right. The main content area is white and contains a series of form fields, each with a label to its left and a text input box to its right. The fields are: 'Name', 'Surname', 'Username', 'Password', 'Confirm password', 'Email', and 'Confirm email'. All these fields are outlined in red, indicating they are mandatory. At the bottom of the form is a large, grey 'Sign up' button.

Name	<input type="text"/>
Surname	<input type="text"/>
Username	<input type="text"/>
Password	<input type="text"/>
Confirm password	<input type="text"/>
Email	<input type="text"/>
Confirm email	<input type="text"/>

Section 4: User interface design

User page

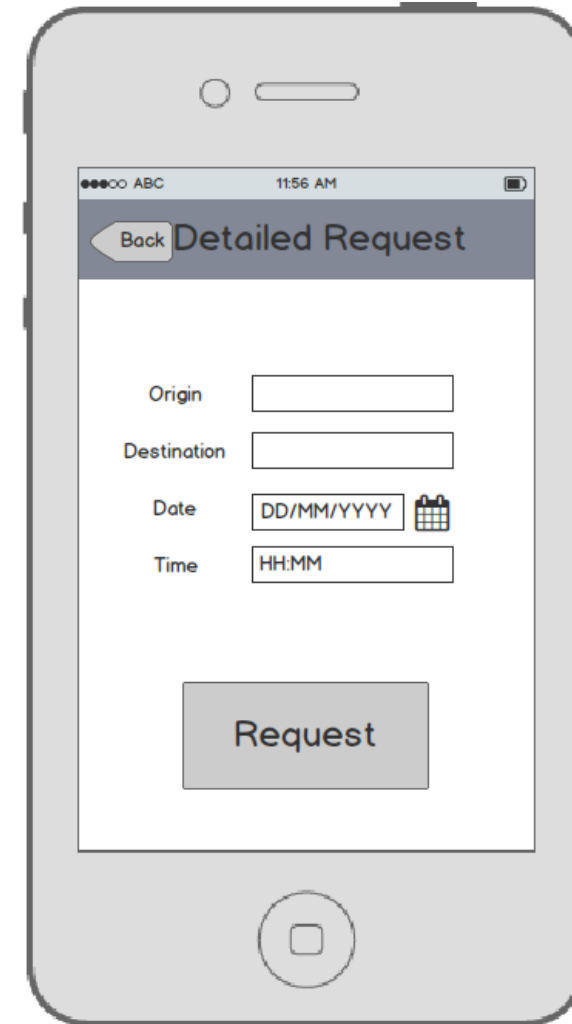
After logging in, users see this page from which they can use all of their functionalities: request a ride, see available taxis in their area and read notifications.



Section 4: User interface design

Detailed request

Users can make a detailed request by filling this form.

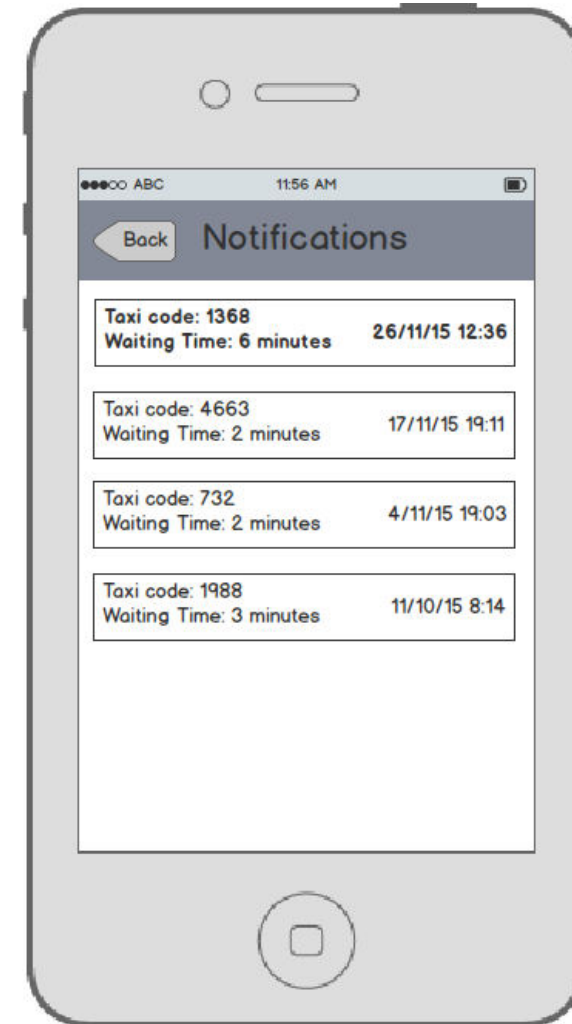


The image shows a mobile application interface for making a detailed request. The screen is displayed on a smartphone with a status bar at the top showing 'ABC' and '11:56 AM'. The app's header is a dark blue bar with a 'Back' button (a left-pointing arrow) and the title 'Detailed Request'. Below the header is a white form area containing four input fields: 'Origin' (a simple text box), 'Destination' (a simple text box), 'Date' (a text box with the placeholder 'DD/MM/YYYY' and a calendar icon to its right), and 'Time' (a text box with the placeholder 'HH:MM'). At the bottom of the form is a large, light gray button labeled 'Request'.

Section 4: User interface design

User notification page

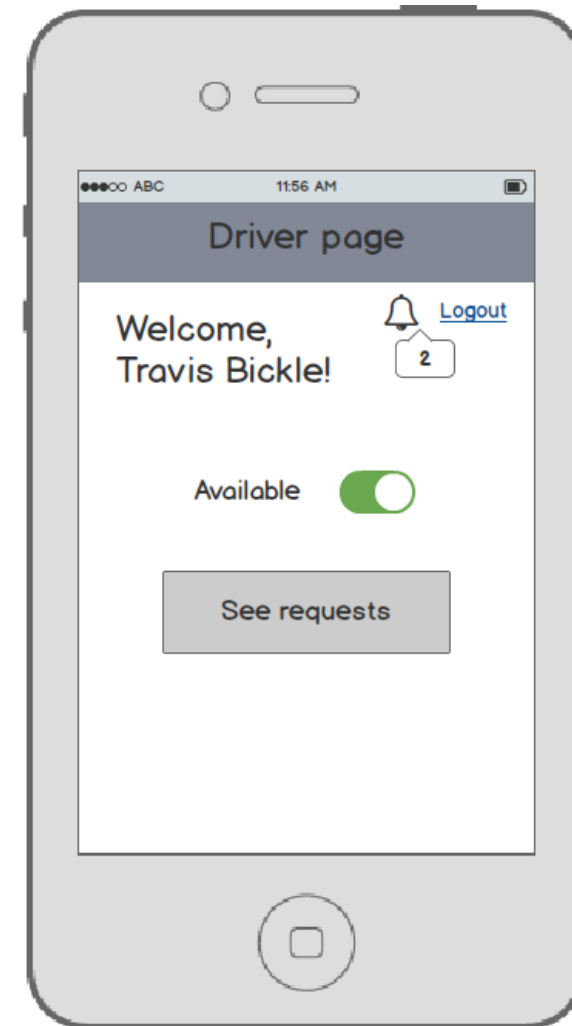
Here users can see notifications about the requests they made. The new ones that hasn't been read yet are written in bold.



Section 4: User interface design

Driver page

This is the page that driver see after they log in. They can set themselves as available and see incoming requests.



Guests requirements

- [G1] Register themselves into the system

Guests can only see the login, registration and confirmation pages, because when they login they are automatically promoted to users or drivers

The filling of the mandatory fields of the form is guaranteed by the algorithm, who discards also if they confirmation fields don't match or if the username or email are already in use.

- [G2] Log into the system

- This too is guaranteed by the algorithm, which checks if the fields are filled, then if the username exists and in the end checks if the username matches the password.

Users requirements

- [G3] See number of taxis of the zone he's in
 - The application is able to guarantee this through the View free taxis function, which queries the database for the information.
- [G4] Make request for a simple ride
 - The algorithm for this function checks if the origin field is empty. If it is, the request can not be fulfilled.
- [G5] Make request for a detailed ride
 - As before, the algorithm makes sure that the input is not empty and formally correct.

Drivers requirements

- [G6] Set themselves as available
 - The driver has this function in the personal page and so he can toggles his availability.
- [G7] Read and accept requests
 - In his personal page the driver can access to the page where he can read the incoming requests and eventually accept them. The application then proceeds to allocate the driver to the ride and set himself as unavailable.

System requirements

- [G8] Notify passengers after the confirmation of a simple request
 - The algorithm of the simple ride request takes care of this by notifying the user as soon as the taxi is allocated.
- [G9] Notify passengers 10 minutes before the ride reserved through a detailed request
 - The algorithm waits until 10 minutes before the ride and then begins to look for a taxi to allocate and notifying the user as soon as it's done.

System requirements

- [G10] Forward requests to the first taxi in queue
 - The application, after a ride request, always forwards it to the first taxi in queue of the user's zone. Then, when a driver is allocated, the application puts him in the last position and sets him as unavailable.
- [G11] After 30 seconds, forward the request to the second taxi in queue and put the first at the end
 - A timer ensures that the driver answers a request within 30 seconds, otherwise he is set as unavailable and put in the last position. The latter will cause the queue to shift by a position and the request will then be forwarded to the driver who is now in the first one.



Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: “myTaxiService”
Test Plan

Massimo Schiavo, Marco Edoardo Cittar

21st January 2016

Summary

- **Section 1:** Introduction
- **Section 2:** Integration strategy
- **Section 3:** Integration tests
- **Section 4:** Stubs and data test required

Introduction

- Purpose and scope

This document describes how to test the integration of the components of the system. Its purpose is to test if the components are correctly integrated between each other, as described in the Design Document.

The scope of the document is to check that the application functionalities are correctly implemented, starting from the login process, to the ride requests to be made by the users and the correct handling of all this through the application logic.

Section 2: Integration strategy

Entry criteria

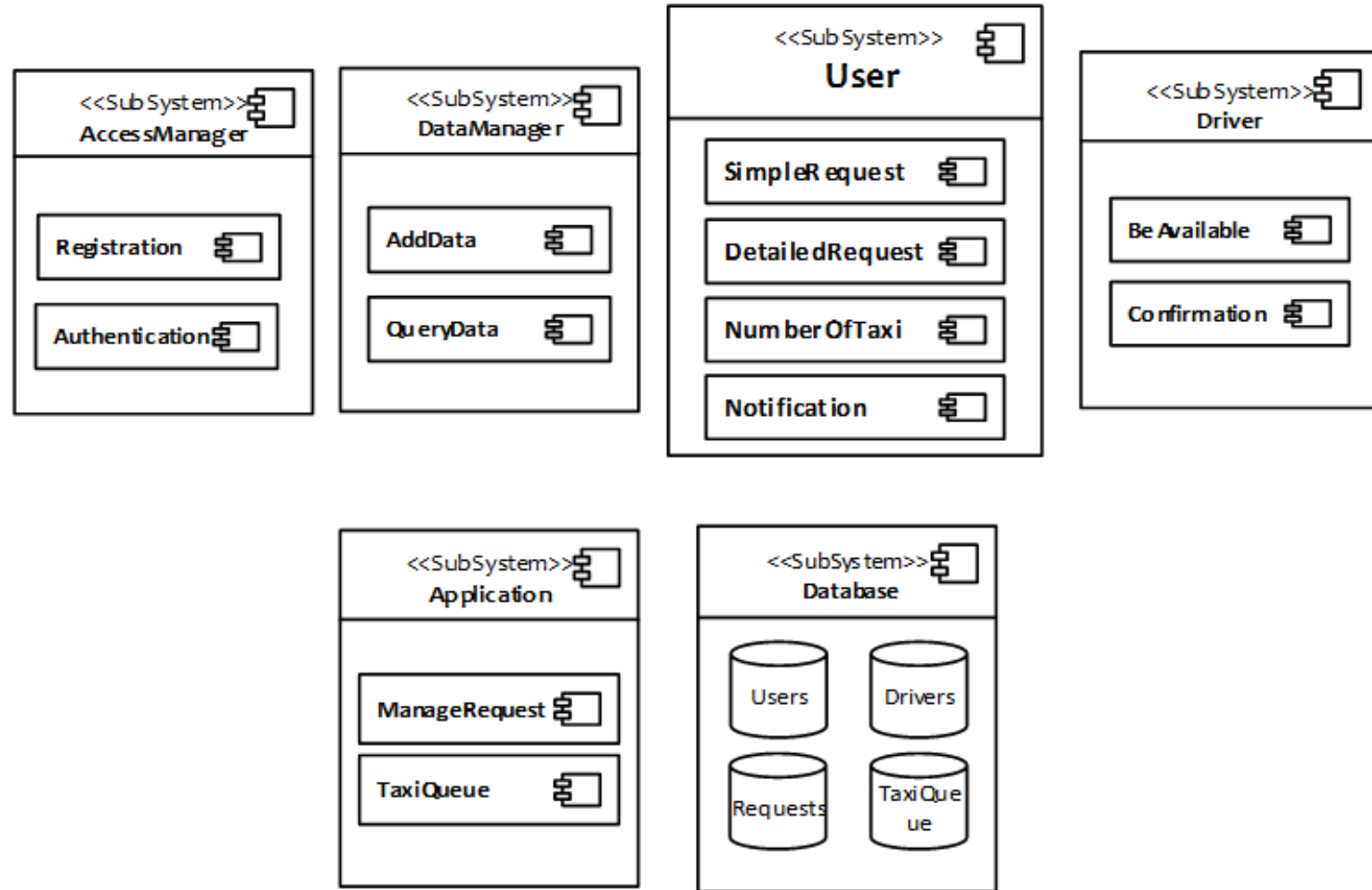
Before starting the integration testing of the subsystems, we recommend to perform a unit test on the Application sub-system in order to check the algorithm correctness, the functional specifications and the structural coverage. This is the major class that contains the logic of myTaxiService. In this class will be implemented, according to the document design already provided, the algorithm of management of the queues of taxis and the algorithm responsible for the forwarding of the requests arrived from the user to the taxi drivers and that's why it's important to perform a unit test on this class before proceeding to the integration test.

The documentation about what is needed to perform this kind of test is in the fourth section of this presentation.

Section 2: Integration strategy

Elements to be integrated

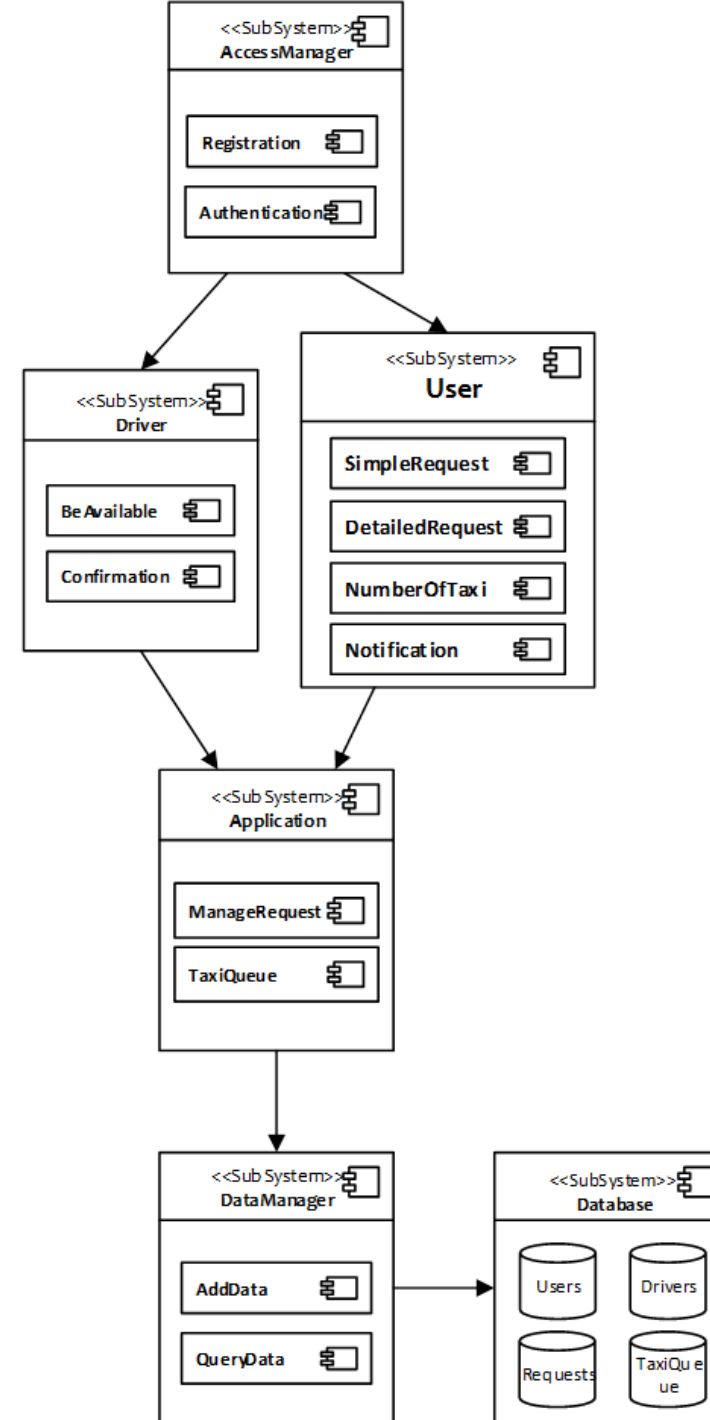
The items to be tested consists of the integration of all the subsystem developed that form myTaxiService.



Section 2: Integration strategy

Integration testing strategy

For testing, we choose the top-down approach. In this approach testing is conducted from the high level interfaces to the core of our application. It's advantageous because the major flaws occur toward the top of the program. It's also a way to simplify the readability of the results of the test case because once the I/O functions are added, representation of test cases is easier. Due to these considerations, in the picture is shown through the arrows the order of integrations between the subsystems.



Section 3: Integration tests

Integration test I1

Integration Test Identifier	I1T1
Test item(s)	This test procedure verifies the correct integration between AccessManager and the User and Driver components.
Purpose	The test must check if the AccessManager can correctly handle users' input.
Preconditions	A Database and a DataManager stub must be created to simulate the login process.
Procedure Steps	Some input tries will be performed, including the login of a User and a Driver. If the components are correctly integrated, the AccessManager should be able to differentiate between the two types of login and show the correct personal page.

Section 3: Integration tests

Integration test I2

Integration Test Identifier	I2T1
Test item(s)	This test procedure verifies the correct integration between the User and the Application components.
Purpose	The test must check if the User component can correctly dispatch users' requests.
Preconditions	I1T1 must have been already correctly executed.
Procedure Steps	The two main functionalities of this component are tested, that are SimpleRequest and DetailedRequest. If the components are correctly integrated the Application should correctly receive the requests from the User and will then proceed to manage them.

Section 3: Integration tests

Integration test I2

Integration Test Identifier	I2T2
Test item(s)	This test procedure verifies the correct integration between the Driver and the Application components.
Purpose	The test must check if the Driver component can correctly receive users' requests and reply to them.
Preconditions	I2T1 must have been already correctly executed. The stubs described in the 4.3 section must have been created.
Procedure Steps	At first it must check that the Driver, by setting himself as available, is correctly inserted in a queue. Then a simulated request from the stub is sent to the Driver which then should act properly, either if he answers the request or not.

Section 3: Integration tests

Integration test I3

Integration Test Identifier	I3T1
Test item(s)	This test procedure verifies the correct integration between the Application and the DataManager.
Purpose	The test must check if the Application component can correctly make query and modification requests of the Database through the DataManager.
Preconditions	A Database stub is needed.
Procedure Steps	The Application should try dispatching the query and add calls to the DataManager. Some examples of these calls can be the query for the first available driver in a queue and the modification of the order of a taxi queue after the Driver's answer to a request.

Section 3: Integration tests

Integration test I4

Integration Test Identifier	I4T1
Test item(s)	This test procedure verifies the correct integration between the DataManager and the Database components.
Purpose	The test must check if the DataManager can correctly query and modify the Database.
Preconditions	I3T1 must have been already correctly executed.
Procedure Steps	The DataManager should be able to receive query and modification requests from the logic (Application) of the system, correctly execute them and eventually return the desired data.

Stubs and data test required

- At the first step of our integration, a stub that simulates the database response correlated to a logging or registering action is required.
- A stub that simulate operations on a database is also needed.
- At the second step of the integration test different stubs are required:
 - A stub for drivers which by clicking on `be_available` gains the right to enter in a queue list.
 - A stub that simulate a pending request.
 - A stub that simulate the timing out of the response time with the relative consequences.
- By integrating the remaining components there are no stubs to add.



Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: “myTaxiService”
Project Plan

Massimo Schiavo, Marco Edoardo Cittar

1st February 2016

Purpose

In the first part of this document we are going to show through two different procedures, the size of our app and the estimated cost.

In the second part instead we are going to show the project tasks and their relative division during the project.

In the last part we will show our risk project management.



Functional Points

A **function point** is a "unit of measurement" to express the amount of business functionality an information system provides to a user. Function points measure software size. The table below shows the weights values that we have used to calculate the FP value.

Function Types	Weight		
	Simple	Medium	Complex
N. Inputs	3	4	6
N. Outputs	4	5	7
N. Inquiry	3	4	6
N. ILF	7	10	15
N. EIF	5	7	10

We have used this table to find the functional points of our app.

In particular:

We have taken into consideration these aspects:

Internal Logic Files (ILFs)

It's a user identifiable group of logically related data that resides entirely within the application.

External Logic Files (ELFs)

It's a user identifiable group of logically related data that is used for reference purposes only.

External Inputs (EIs)

It is an elementary process in which data crosses the boundary from outside to inside.

External Inquiries (EIQs)

It's an elementary process with both input and output components that result in data retrieval from one or more internal logical files and external interface files.

External Outputs (Eos)

It is an elementary process in which derived data passes across the boundary from inside to outside.

Aspect	Total cost	Description
Internal Logic File	56	The application stores information about: Users, Drivers, Requests, TaxiQueue, Taxis and Locations.
External Logic File	10	The application has to manage the position of each taxis from an external service based on GPS locations
External Inputs	26	The application has to manage all the interactions between users and driver.
External Inquiries	13	The application allows a user to view the number of taxi available in his zone according to his phone GPS location and the application allows a driver to view the pending user's requests in order to confirm them.
External Outputs	0	There is no external output
TOTAL	105	we can hypostasize the size of the project in terms of lines of code. LOC = 105 * 46 = 4830 Lines Of Code.

COCOMO II

COnstructive **CO**st **MO**del II is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. .

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC SF_j:	thoroughly unprecedented 6.20	largely unprecedented 4.96	somewhat unprecedented 3.72	generally familiar 2.48	largely familiar 1.24	thoroughly familiar 0.00
FLEX SF_j:	rigorous 5.07	occasional relaxation 4.05	some relaxation 3.04	general conformity 2.03	some conformity 1.01	general goals 0.00
RESL SF_j:	little (20%) 7.07	some (40%) 5.65	often (60%) 4.24	generally (75%) 2.83	mostly (90%) 1.41	full (100%) 0.00
TEAM SF_j:	very difficult interactions 5.48	some difficult interactions 4.38	basically cooperative interactions 3.29	largely cooperative 2.19	highly cooperative 1.10	seamless interactions 0.00
PMAT SF_j:	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower 7.80	SW-CMM Level 1 Upper 6.24	SW-CMM Level 2 4.68	SW-CMM Level 3 3.12	SW-CMM Level 4 1.56	SW-CMM Level 5 0.00

We have used this table for the first part and all the other table relative to the different aspects found in the manual of this model.

In particular:

Aspect	Incidence	Value	Description
PRECEDENTNESS	Very low	6,20	It reflects the previous experience in past project like this. For us, this kind of project is the first in our life we are doing and that's why this value will be very low.
DEVELOPMENT FLEXIBILITY	Very high	1,01	It reflects the degree of flexibility in the development process. The professor left us a large space of flexibility without forcing us with too much details, that's why this value is going to be very high.
RISK RESOLUTION	Very high	1,41	According to our project risk management.
TEAM COHESION	Very high	1,10	It reflects how well the development team know each other and work together. At the beginning of the project we didn't know each other and both of us did not know how the other worked. Although this aspect, we hadn't any problems on work's organization and division of tasks. Due to these considerations, this value will be very high.
PROCESS MATURITY	High	3,12	There are two ways of rating Process Maturity. We have chosen the second that s organized around the 18 Key Process Areas (KPAs) in the SEI Capability Maturity Model. We can consider this value as high.
TOTAL		12,84	

Driver factor	Incidence	value
Required software reliability	Very high	1.26
Database size	High	1.00
Product complexity	Nominal	1.00
Required reusability	High	1.07
Documentation match to life-cycle needs	Nominal	1.00
Execution time constraint	Very low	n/a
Main storage constraint	Very low	n/a
Platform volatility	Low	0.87
Analyst capability	Very high	0.71
Personnel continuity	Very low	1.29
Application experience	Low	1.10
Platform experience	-	-
Programmer capability	-	-
Language and tool experience	-	-
TOTAL (PRODUCT)		1.18

Section 1

Effort estimation

The final equation gives us the effort estimation measured in Person-Months (PM)

$$\text{Effort} = A * \text{EAF} * \text{KSLOC}^E$$

The values of A , B , C , and D in the COCOMO II.2000 calibration are:

$$A = 2.94 \quad B = 0.91$$

$$C = 3.67 \quad D = 0.28$$

EAF is the product of all the cost drivers that is equal to: **1.18**

KSLOC represents the estimated lines of code obtained from the FP analysis: **4830**

E is the exponent derived from the Scale Drivers with the equation below:

$$B + 0.01 * \sum\{i\} SF[i] = 0.91 + 0.01 * 12.84 = \mathbf{1.0384}$$

With all of these parameters we can calculate the final effort:

$$\text{Effort} = 2.94 * 1.18 * 4.830^{1.0384} = \mathbf{17.8008 \text{ PM}}$$

Section 1

Shedule estimation

We are going to use this formula to compute the estimated duration:

$$\text{Duration} := 3.67 * \text{Effort}^F$$

$$\text{Where } F = 0.28 + 0.2 * (E-B) = 0.28 + 0.2 * (1.0384-0.91) = \mathbf{0.3057}$$

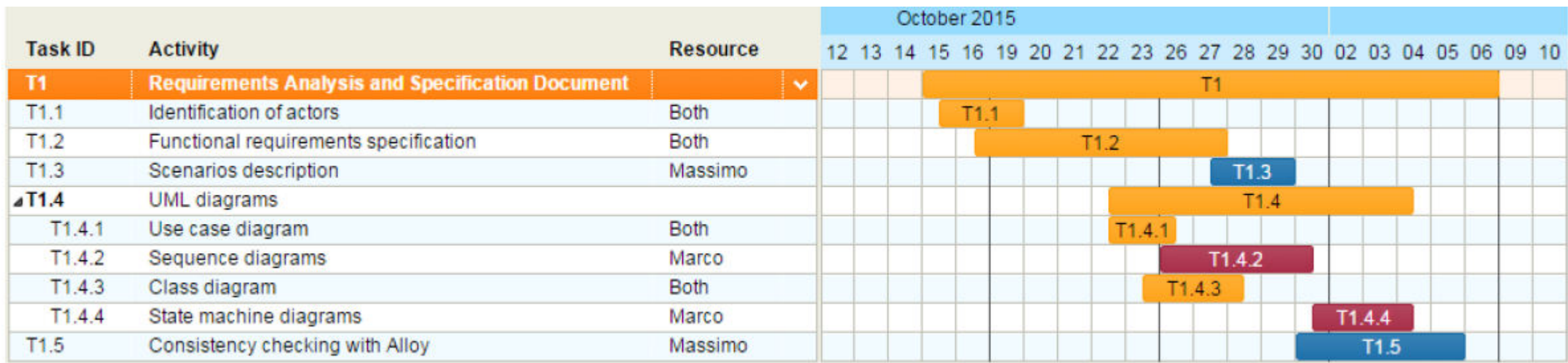
$$\text{So } \rightarrow \text{Duration} = 3.67 * 17.8008^{0.3057} = \mathbf{8.84 \rightarrow 9}$$

The duration estimated by these computations is not similar to how the reality is. It is also truth that in our project we had just to do the documentation. Probably if we were to do also the implementation and development of the entire application, the duration of the global project could be about 9 months.

$$P = \text{Effort} / \text{Duration} = 17.8008 / 9 = \mathbf{1.98 \rightarrow 2}$$

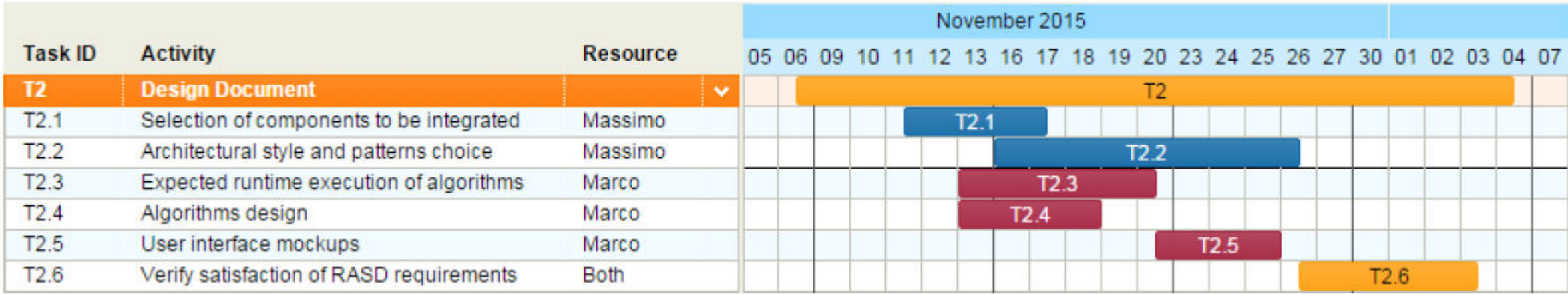
Task chart diagram

On the right part of the diagram we can find a list of task for the Requirements Analysis and Specification Document, on the left it's shown the «work in progress» during the project period.



Task chart diagram

On the right part of the diagram we can find a list of task for the Design Document, on the left it's shown the «work in progress» during the project period.



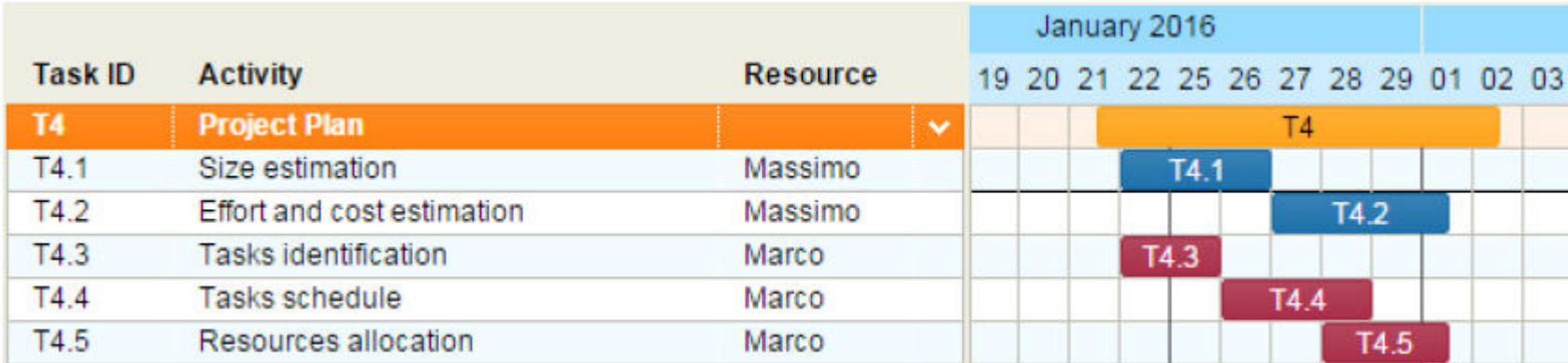
Task chart diagram

On the right part of the diagram we can find a list of task for the Test Plan, on the left it's shown the «work in progress» during the project period.



Task chart diagram

On the right part of the diagram we can find a list of task for the Project Plan, on the left it's shown the «work in progress» during the project period.



Project risk management

Risk	Probability	Effect	Strategy
The database used in the system cannot process and manage all the transactions per second as expected	Moderate. It depends on how much the application will be used in the market.	Serious. The system can go down and cannot process some requests from the users	Alert the customer when the capacity of database is running out fast. Two alternatives: Delete the oldest data. Adding a new database to the system.
The server has a data overload.	Low. This risk has a pre-strategy used to manage it because tests about stress were done.	Catastrophic. If this case occur, the entire system goes down not only by processing the last requests but it is also impossible to access into the system while the system is still down.	This can happen principally during the rush hour. It is useful to increase the server capability during these hours.

Risk	Probability	Effect	Strategy
A user's request considered by him as sent, it is actually never been sent due to a network error.	Low. This can happen only if people are in a place with little cellular coverage.	Serious. A user believes he has sent a request, but in reality, it is not true.	Force a minimum cellular coverage to all the users for using correctly the application.
Market risk. This is meant as the possibility to not have an expected number of downloads due to the age of the population in relation to those who use the service. The teenager and the part of the population that is young, use much more the railways services than the taxi.	Moderate. This risk can be taken into consideration based on what the city is smart. Integrate a service like this in a city whose population is old would make little sense.	Moderate. Probably the money coverage spent on the project will has much time to be regained.	It is useful to study accurately the target population before staring to develop the app. It is a good thing make a survey by which obtain the habits of the citizens.

Risk	Probability	Effect	Strategy
<p>People risks.</p> <p>This is associated with the availability, skill level, and retention of the people on the development team.</p> <p>Probably it could be useful to have a programmer and a designer in our team.</p>	High.	<p>Low.</p> <p>Probably the absence of these two figures mean a low palatability level of the application in terms of design. The functional use of the application is anyway guaranteed.</p>	<p>The customer can inform about these two figures to include in the project team only during the development part according of course to the estimated cost.</p>