



# Politecnico di Milano

Computer Science and Engineering

## Project of Software Engineering 2: "*my Taxi Service*"

Reference professor: Raffaella Mirandola

**AUTHOR:** ANDREA MAIOLI (MAT. 852429)



# Requirements Analysis and Specifications



# The Given Request

- ▶ Design an application that is able to manage taxi requests.
- ▶ **Given constraints:**
  - ▶ Fair management of taxi queues.
  - ▶ Service accessible from both mobile and web application.
  - ▶ Passenger must be informed with code of incoming taxi and waiting time.
  - ▶ Taxi driver can accept/decline requests via mobile application.
  - ▶ Division of the city in zones, and for each zone one queue.
  - ▶ The system must provide an API to enable development of future additional services.

# Assumptions (Main points)

- ▶ There is an existing call center infrastructure that will not be discontinued.
- ▶ Drivers and Call Center operators are directly hired from the government and are considered public employees.
- ▶ If a driver is not assigned to a request, he/she can pick up a passenger (as taxi normally does).
- ▶ City: Milan (3.2 Millions of people).
- ▶ Number of drivers: 5.000 (due to last Corriere Della Sera analysis).
- ▶ The number of requests never saturate the available taxies.

# Actors: who will use the new system?

- ▶ **Guest:** a non register user.
- ▶ **Passenger:** registered user who will use the system for requesting a ride.
- ▶ **Call Center Operator:** special registered user who will handle the request coming from the telephone and instert them into the system.
- ▶ **Taxi Driver:** special registered user who will bring to completion the requests coming from passengers.
- ▶ **Administrator:** is a special user with the right to modify the information stored inside the system.

# Goals of the new system

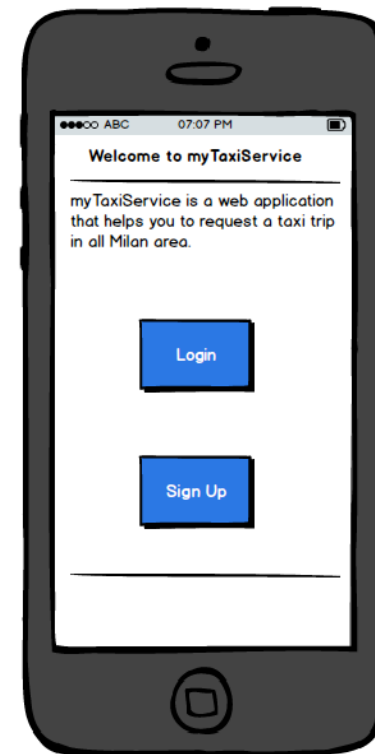
1. Simplify the access of passengers to the taxi service.
2. Guarantee a fair management of the taxi queue.
3. Simplify the communications between drivers and call center.
4. Simplify the methods of access to the requests for the taxi drivers.
5. Allow the system to manage events and requests.
6. Allow the administrator to handle area and user's informations.

# Future Implementations

- ▶ Creation and tuning of a specific algorithm for **dynamic taxi allocation**
  - ▶ After a year of data collection will be possible to analyze the relation between the *number of request* in a specific hour of a specific day with:
    - ▶ hosted events
    - ▶ the weather type
    - ▶ the day of the week.
  - ▶ Will help predict the estimated density of request per area, and accordingly increase/decrease the number of taxi presents in all the areas.

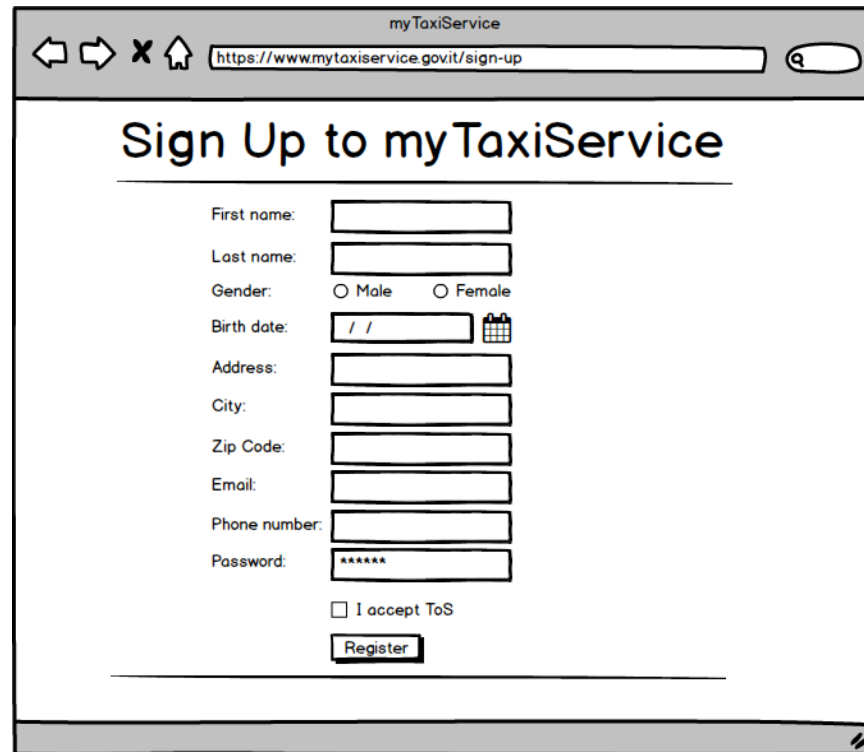


# User Interface – Home Page





# User Interface – Sign Up Page



myTaxiService


https://www.mytaxiservice.govit/sign-up

## Sign Up to myTaxiService

First name:

Last name:

Gender: ☐ Male ☐ Female

Birth date:  /  /  

Address:

City:

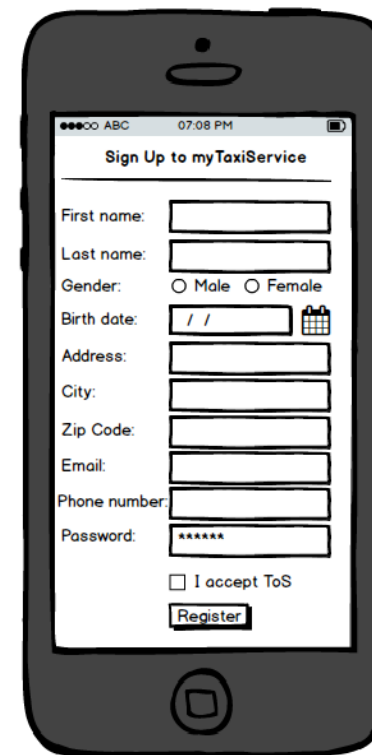
Zip Code:

Email:

Phone number:

Password:

☐ I accept ToS




ABC 07:08 PM

## Sign Up to myTaxiService

First name:

Last name:

Gender: ☐ Male ☐ Female

Birth date:  /  /  

Address:

City:

Zip Code:

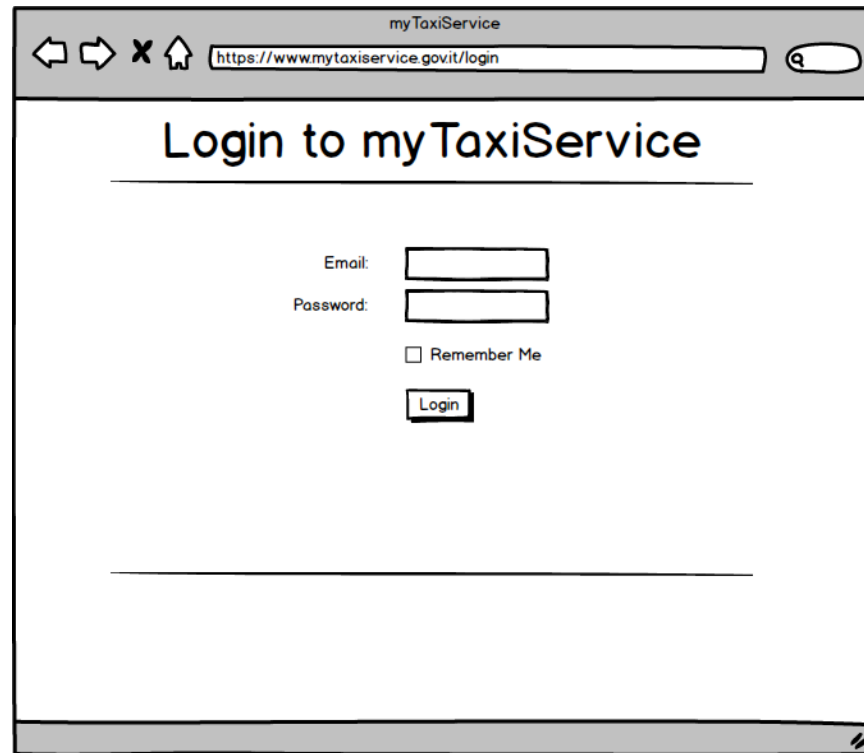
Email:

Phone number:

Password:

☐ I accept ToS

# User Interface – Login Page



myTaxiService

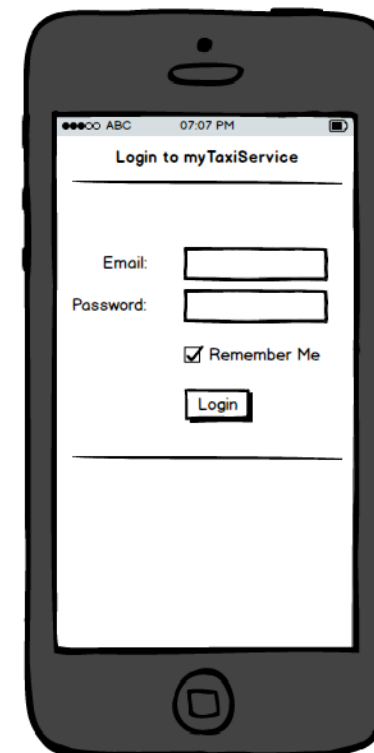
https://www.mytaxiservice.govit/login

## Login to myTaxiService

Email:

Password:

☐ Remember Me



ABC 07:07 PM


## Login to myTaxiService

Email:

Password:

☒ Remember Me

# User Interface – Administrator Login Page



myTaxiService

https://www.mytaxiservice.govit/login

## Login to myTaxiService

**Second Factor Authentication**

Welcome Administrator! Please provide the security code to continue:

Send



ABC 11:08 PM

## Login to myTaxiService

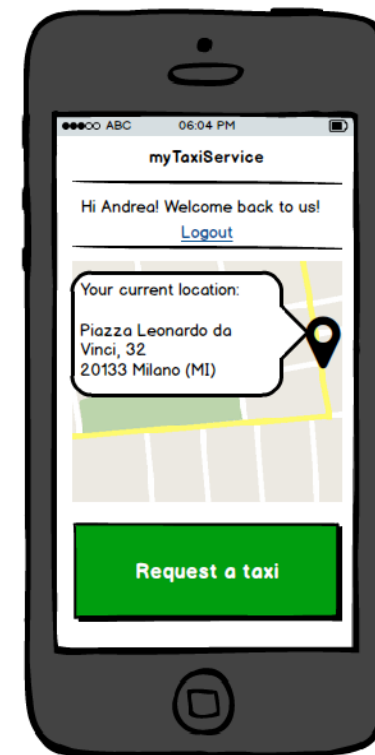
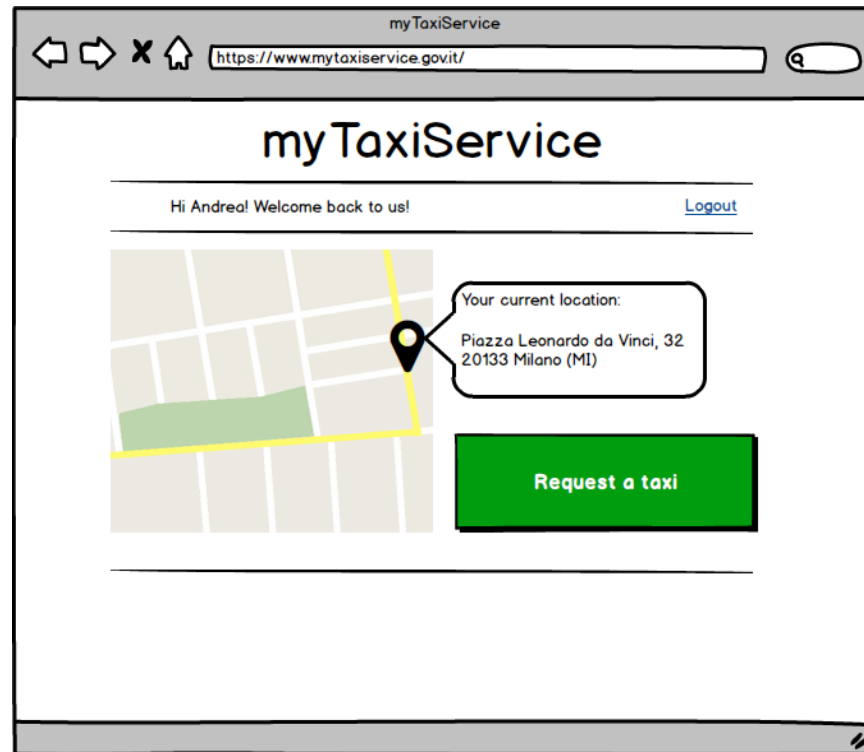
**Second Factor Authentication**

Welcome Administrator! Please provide the security code to continue:

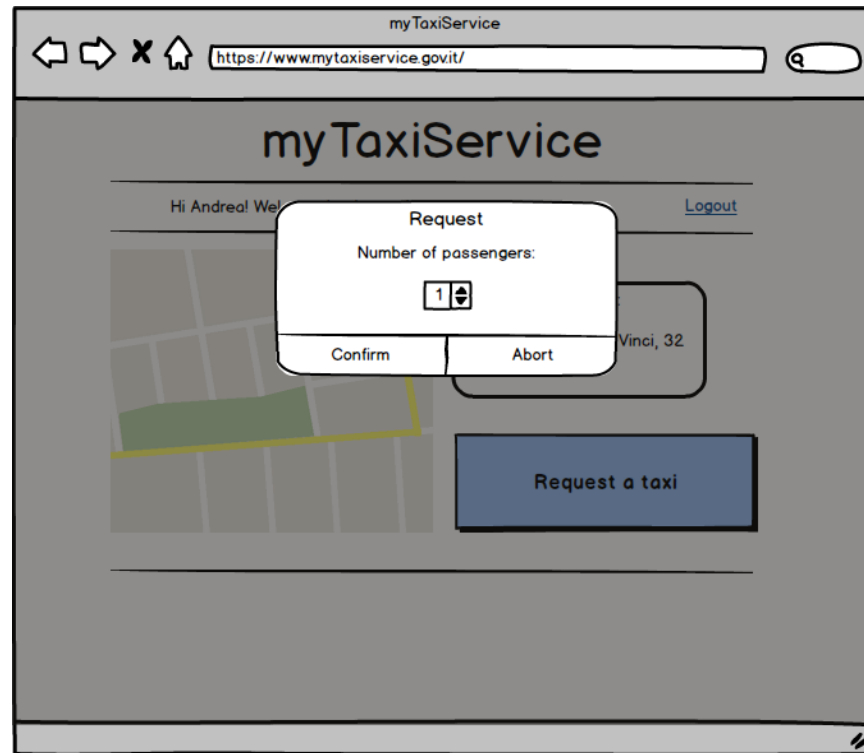
Send



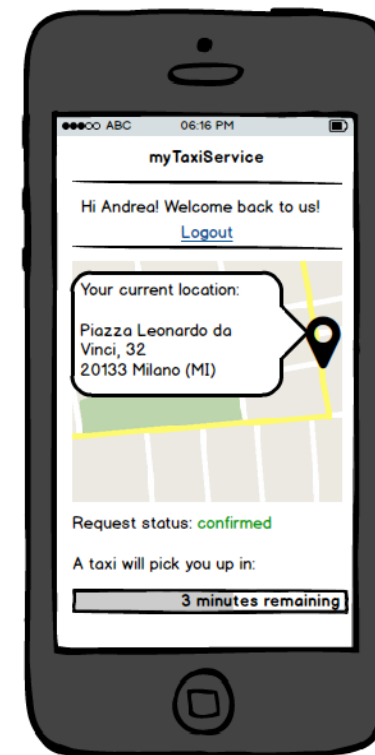
# User Interface – Passenger Home Page



# User Interface – Taxi Request

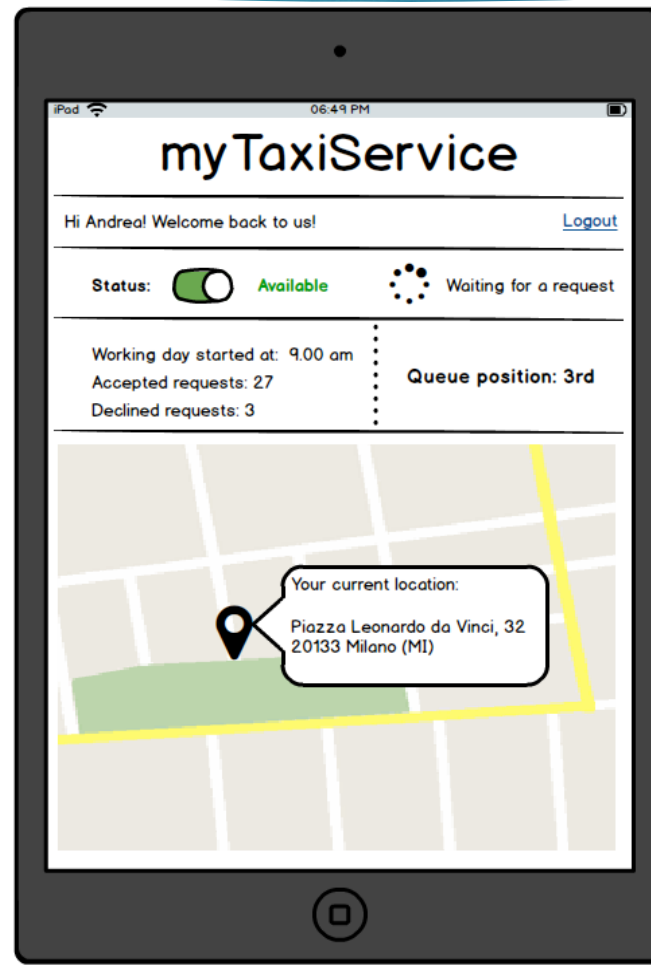


# User Interface – Active Request Page

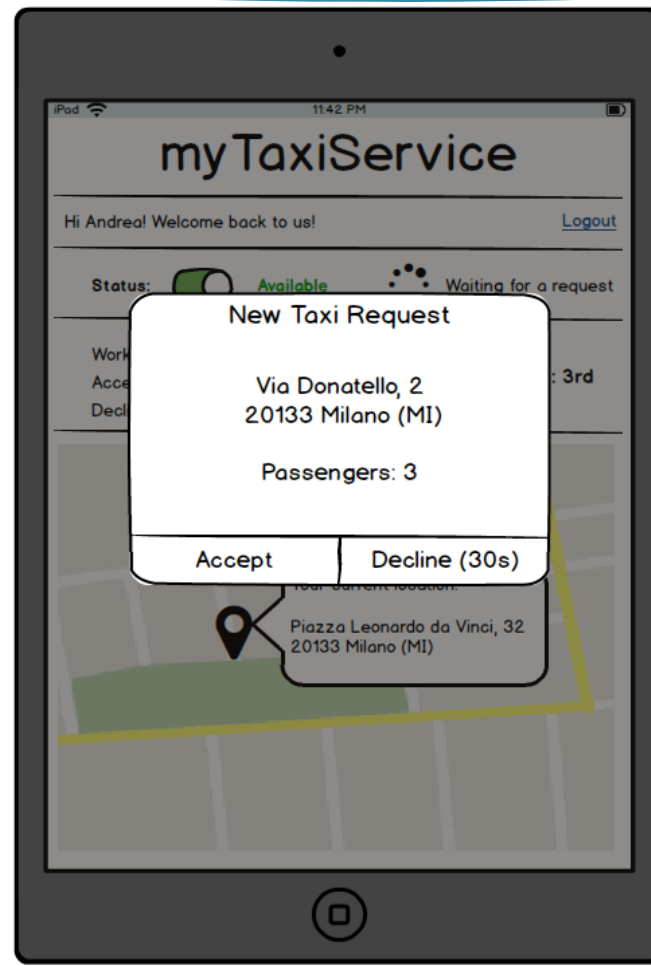




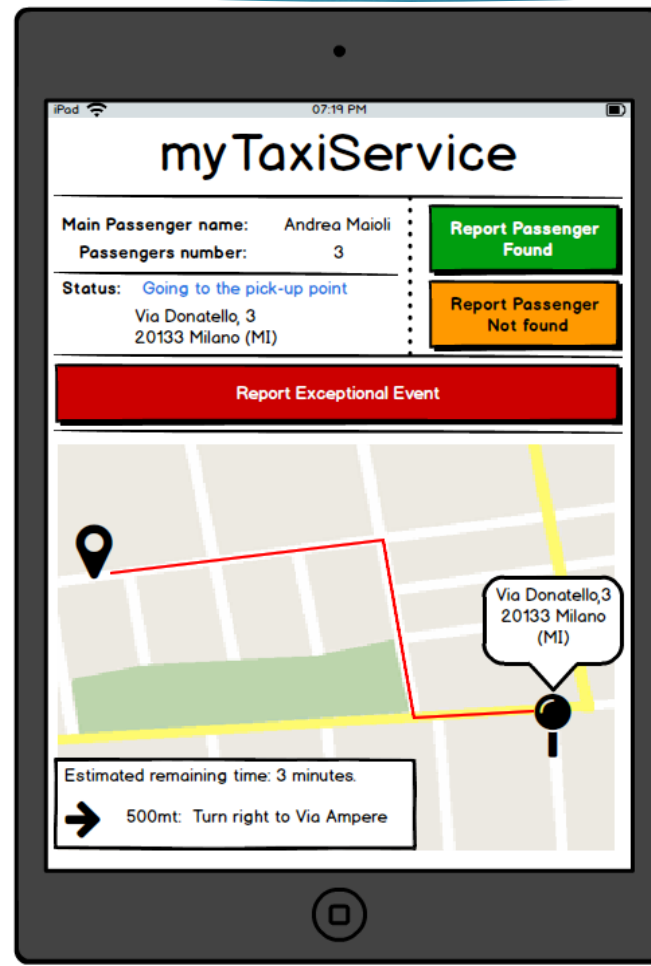
# User Interface – Driver Home Page



# User Interface – Incoming Request

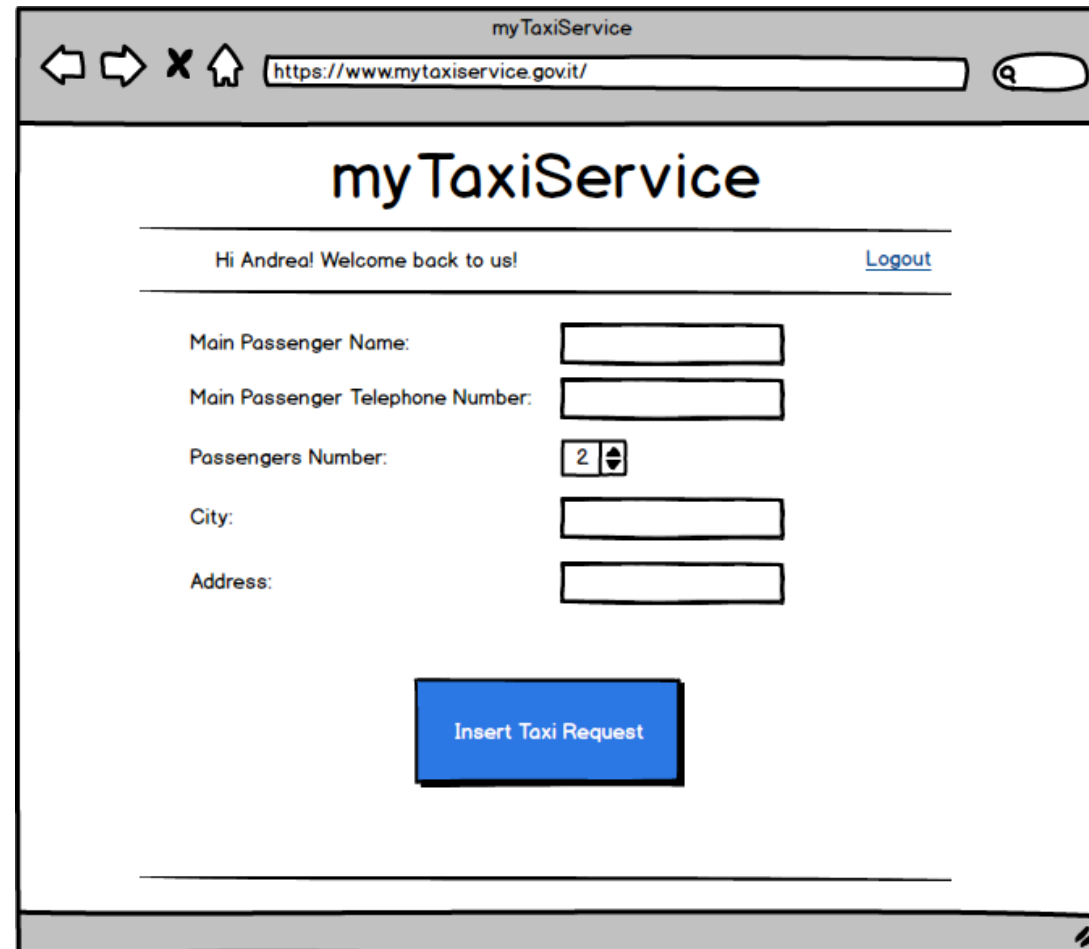


# User Interface – Driver Active Request Page





# User Interface – Operator's Home Page



The screenshot shows a web browser window titled "myTaxiService" with the address bar displaying "https://www.mytaxiservice.gov.it/". The page content includes a welcome message, a "Logout" link, and a form for inserting a taxi request. The form fields are: "Main Passenger Name" (text input), "Main Passenger Telephone Number" (text input), "Passengers Number" (spinner box with "2"), "City" (text input), and "Address" (text input). A blue "Insert Taxi Request" button is positioned below the form fields.

myTaxiService

https://www.mytaxiservice.gov.it/

## myTaxiService

---

Hi Andrea! Welcome back to us! [Logout](#)

---

Main Passenger Name:

Main Passenger Telephone Number:

Passengers Number:

City:

Address:

[Insert Taxi Request](#)

# Functional Requirements

- ▶ **Guest** can:

- ▶ Sign Up

- ▶ **Passenger** can:

- ▶ Login
- ▶ Modify profile information
- ▶ Request a taxi
- ▶ Get the status and ETA of an active request
- ▶ Logout

# Functional Requirements

- ▶ **Driver can:**
  - ▶ Login
  - ▶ Update the status
  - ▶ View queue position and workday statistics
  - ▶ Receive and respond to taxi request
  - ▶ Report if a passenger is found or not
  - ▶ Report an exceptional event which prevent him/her to get to the pick-up point
  - ▶ Get ETA to the pick-up point and indications
  - ▶ Logout



# Functional Requirements

- ▶ **Call Center Operator** can:
  - ▶ Login
  - ▶ Insert in the system a taxi request
  - ▶ Get the status and ETA of the inserted request
  - ▶ Logout

# Functional Requirements

- ▶ **Administrator** can:
  - ▶ Login on the web application
  - ▶ Create, delete or modify an area
  - ▶ Modify or delete an user
  - ▶ Modify the user level (can be driver, call center operator)
  - ▶ Logout

# Functional Requirements

- ▶ **The system** must be able to:
  - ▶ Assign a request to the first taxi in the queue.
  - ▶ Organize drivers in queue.
  - ▶ Forward a request to the second driver in the queue if the first driver decline it.
  - ▶ Assign a new driver to a request, if the assigned driver report an exceptional events.
  - ▶ If no driver is available in a queue, the system must be able to find the driver that will arrive to a fixed pick-up point in the less possible time.
    - ▶ The driver can be one from another area, or one that will finish a ride in the request's area.

# Performance Requirements

- ▶ The system must be able to compute the response to each request in no more than 0.5s.
- ▶ The system must be able to serve simultaneously at least:
  - ▶ 1600 taxi drivers
  - ▶ 50 call center operators
  - ▶ 110 passengers

With a total response time for each action lower than 3 seconds.



# Software System Attributes

- ▶ **Availability:**

- ▶ 24 hours per day, 7 days per week.

- ▶ **Security:**

- ▶ User's credential stored using a hashing function.
  - ▶ User's password must a combination of letters and numbers and at lest 8 characters long.
  - ▶ The Application Server must be the only machine able to communicate with the Database Server.
  - ▶ All the user inputs must be filtered in order to prevent SQL-Injection, Cross Site Scripting and other type of attacks.
  - ▶ HTTPS must be enable on the web server and all the connection coming from the HTTP protocol must be redirected to the HTTPS one.

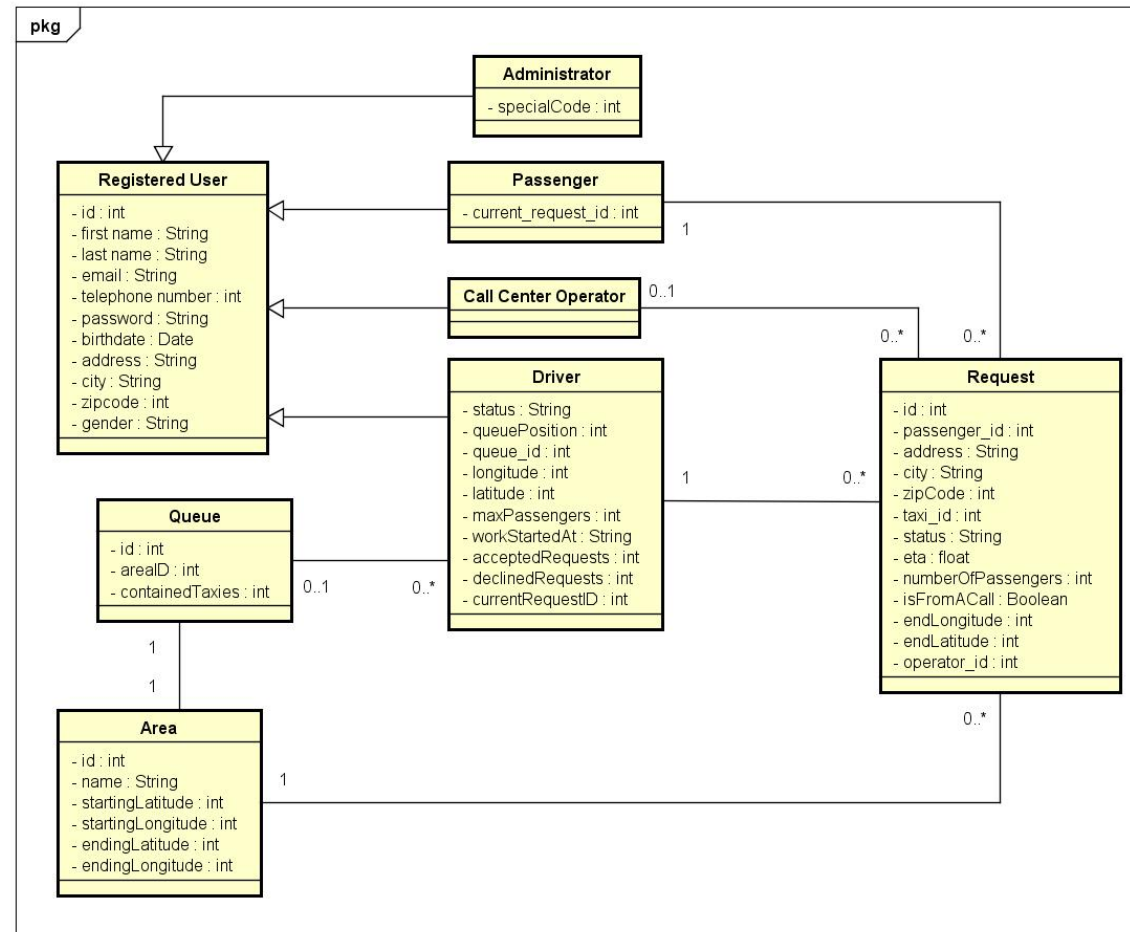
# Software System Attributes

## ► Maintainability:

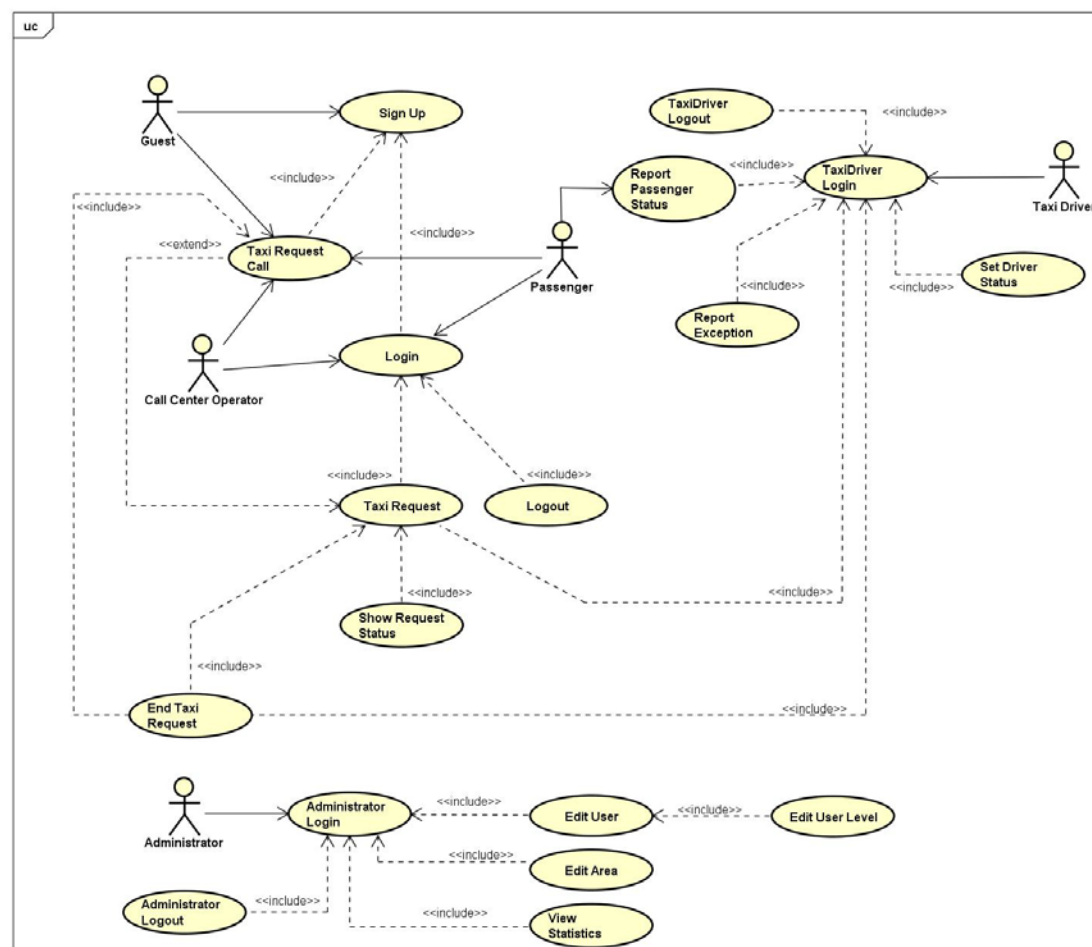
- Each function and class of the application must be well commented in order to allow future developers to understand and modify the code.
- Each update or modification to the application or the infrastructure must be traceable and well documented.
- Each solution to every future problem must be well documented, in order to know how it was solved.
- Each method provided by the API must be well documented and some example must be provided.

(as ITIL suggests)

# Class Diagram



100





100



*Alloy code is available in the documentation, with tests results.*

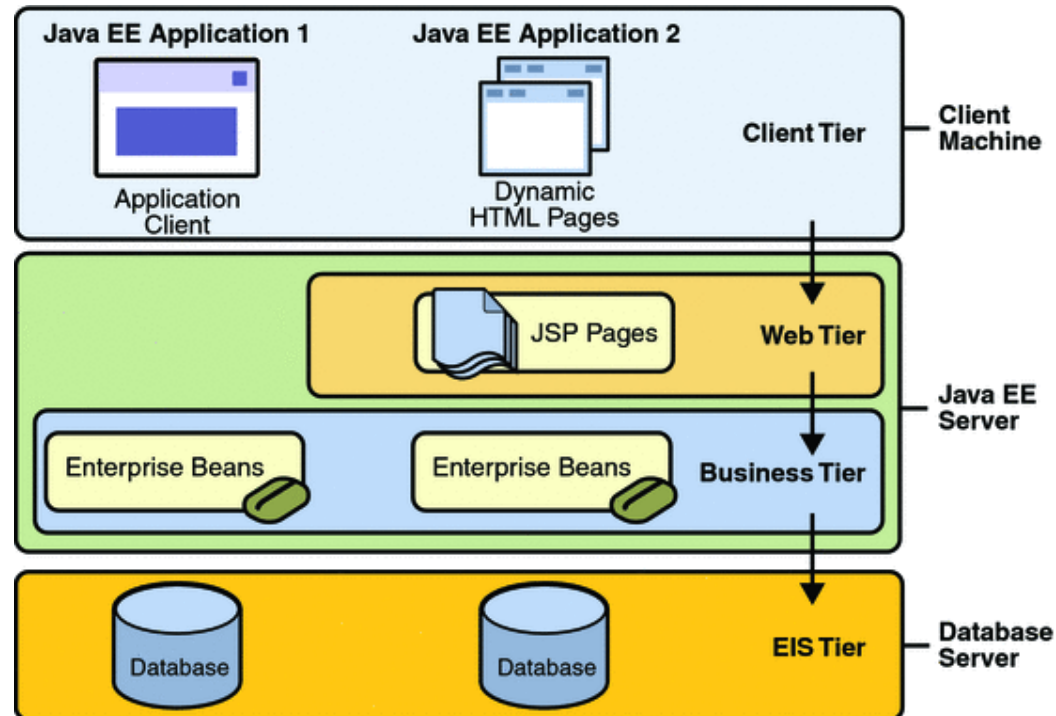


# Design



# Architectural Design – Overview

- The project is based on the **Java Enterprise Edition** (JEE) architecture.



# Architectural Design – High Level Components

## ▶ Database Server:

- ▶ Contains the application data and information used by the entire system.
- ▶ Will consist in a DBMS which is **MySQL**.

## ▶ Application Server:

- ▶ Contains all the application logic of the system and manages all the actions inside it.
- ▶ Will use Java Persistent API (**JPA**) and **Java Entity Beans**.
- ▶ Run by **Glassfish Server**.



# Architectural Design – High Level Components

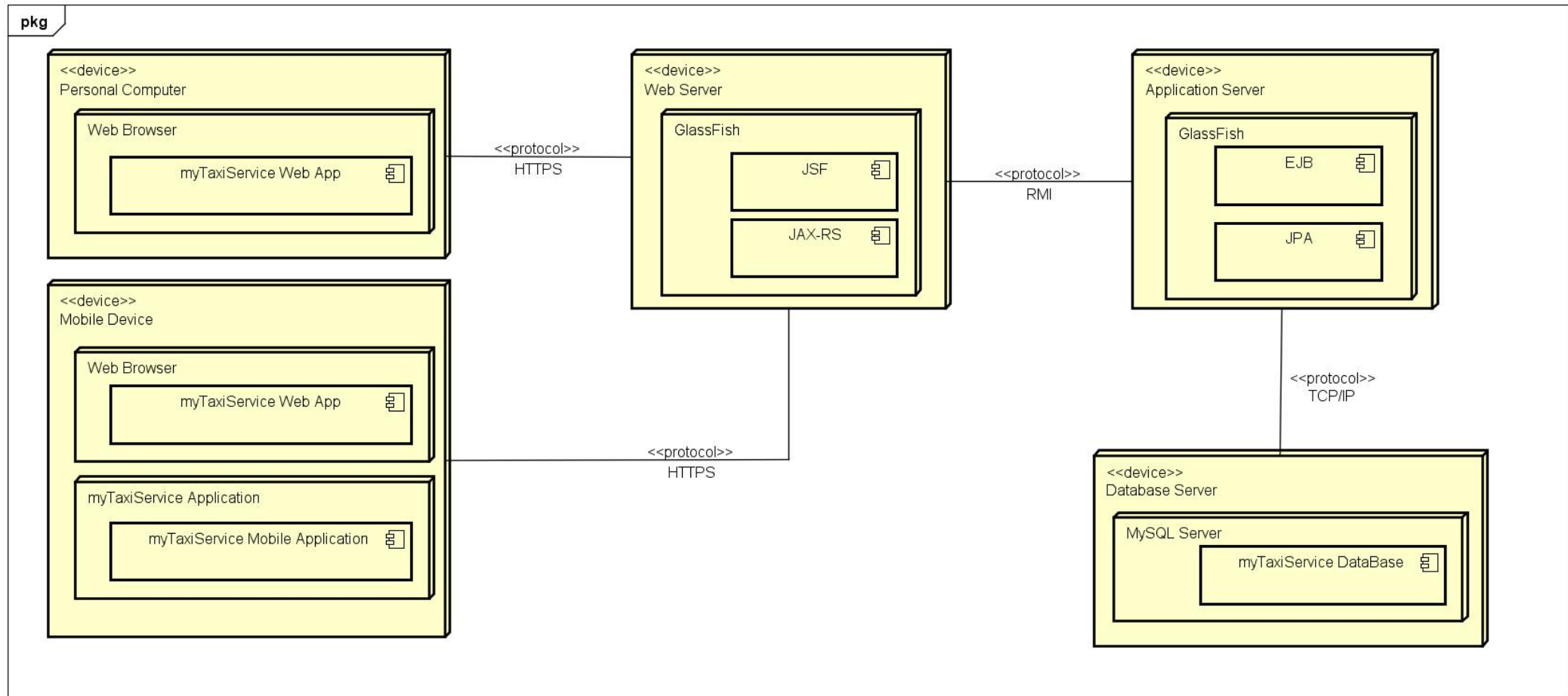
## ▶ Web Server:

- ▶ Provides a web interface to access the system using a web browser and exposes a RESTful API used by the mobile application to interact with the system.
- ▶ Will use **JAX-RS** to provide the RESTful API and Java Server Faces (**JSF**) to provide a web interface.
- ▶ Run by **Glassfish Server**.

## ▶ Client:

- ▶ Represented by a web browser and the mobile application (available on Android, iOS and Windows Phone).

# Architectural Design – Deployment Diagram



# Architectural Design – Software Components

## ► Application Server Components:

- ***Request Manager***: provides all the functionalities required to create and manage taxi requests.  
(e.g.: create a request)
- ***Queue Manager***: provides all the functionalities required to manage a queue.  
(e.g.: remove a taxi from the queue)
- ***Account Manager***: provides all the functionalities required to manage an account.  
(e.g.: register, login)
- ***Location Manager***: provides all the functionalities required for handling geographic coordinates and taxi's areas.  
(e.g.: get area of a location, computing the time required to arrive to a place)
- ***Taxi Manager***: provides all the functionalities required to manage a taxi.  
(e.g.: update its status)

# Architectural Design – Software Components

- ▶ **Web Server Components:**

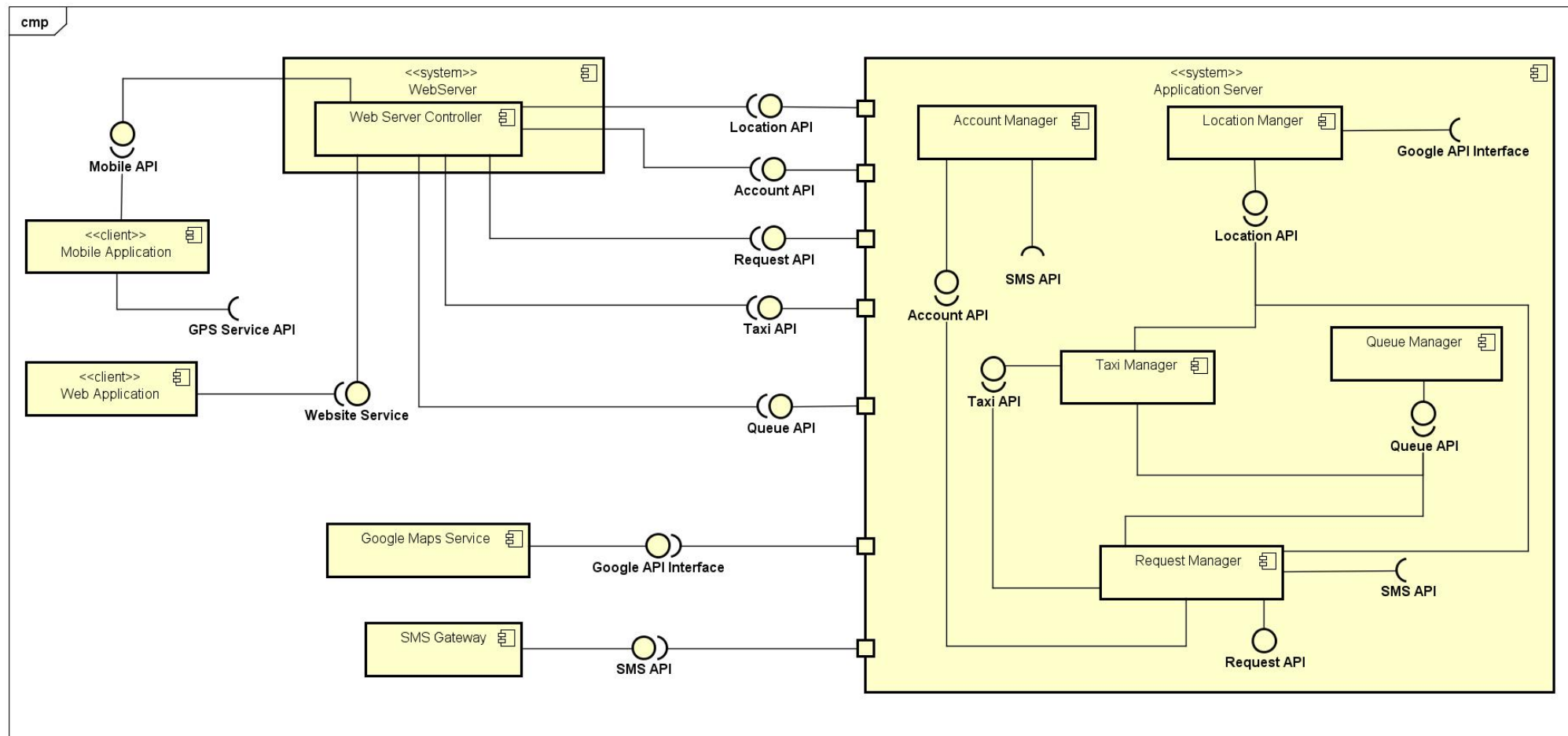
- ▶ **Web Server Controller:** is in charge of the invocation of the methods provided by the API exposed by the Application Server.

This component will also manage both the request through the provided RESTful API and the Website.

- ▶ The **Database Server** does not implement any application logic.



# Architectural Design – Components View



# Architecture Styles and Patterns

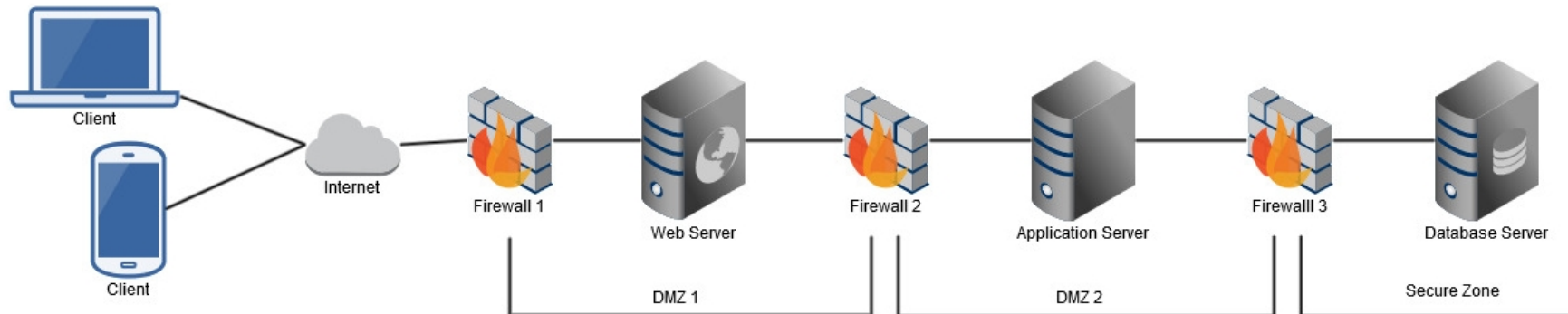
- ▶ Model View Controller
- ▶ **Client-Server** is applied in communication between components.
- ▶ REST API

# Other Design Decisions

- ▶ **Amazon EC2 with Elastic Load Balancing:**
  - ▶ Automatic scalable and load-balancing based cloud computing, resulting in a ***better flexibility, lower costs that are directly related to usage, better availability*** and ***better fault tolerance***.
- ▶ **Polling Requests from Clients**
  - ▶ The client will continuously send a request to the web server in order to receive and update informations.
- ▶ **InnoDB as storage engine for MySQL**
  - ▶ ***More resistant to table corruption*** than other storage engines.
  - ▶ Grant an ***higher level of concurrency*** (better performance).

# Other Design Decisions: Security

- ▶ **Two Factor Authentication for Administrators Login**
- ▶ **Slow Hashing Function and Salt for Password**
  - ▶ Slow down brute force attack
  - ▶ Usage of pre-build hashing tables not possible.
- ▶ **Firewalls**
  - ▶ Configured on a default deny base.







# Integration Testing Plan



# Integration Testing Strategy

- ▶ **Two phases:**

- ▶ 1) Integration between components that compose the same subsystem.
- ▶ 2) Integration of different subsystems

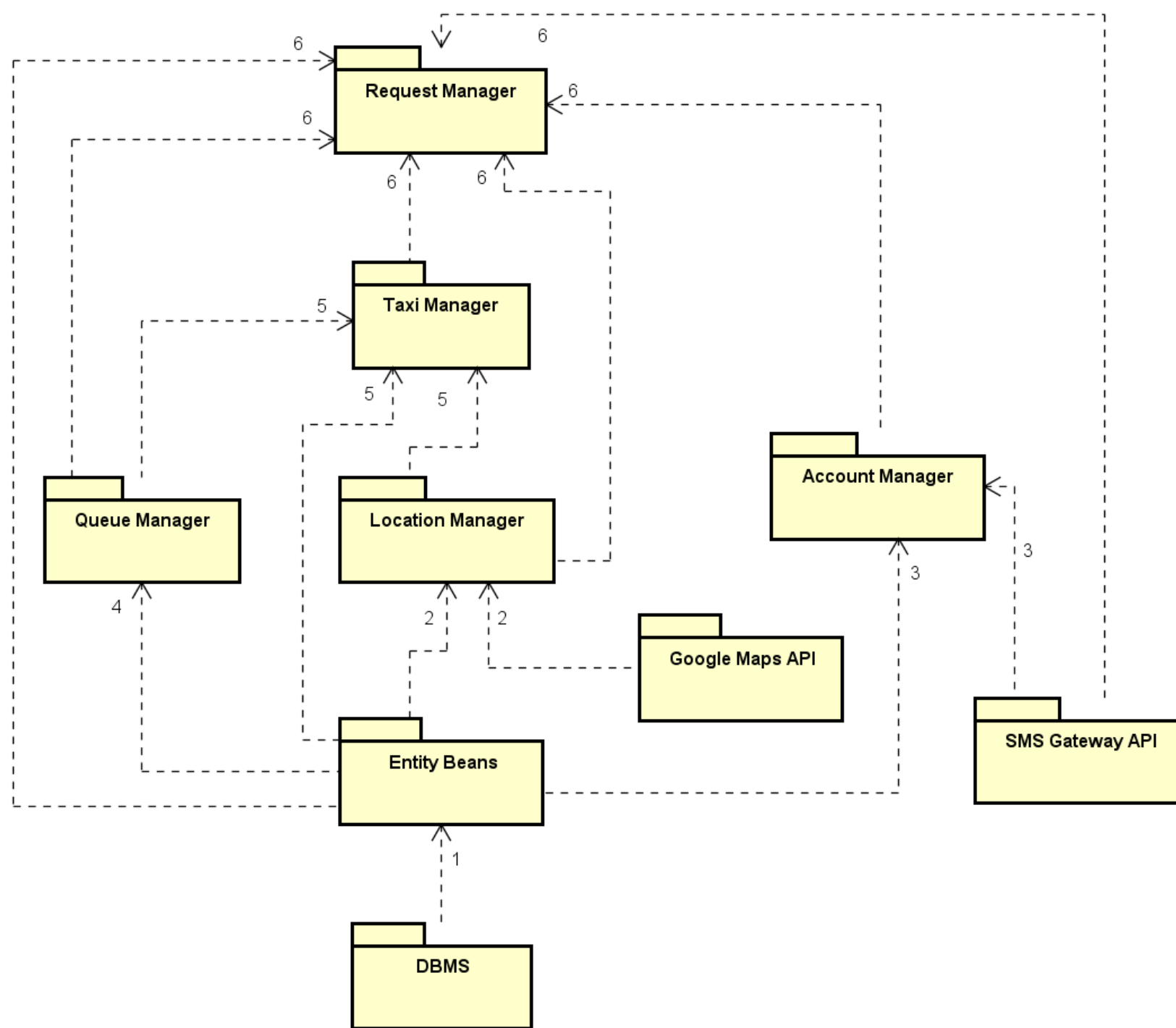
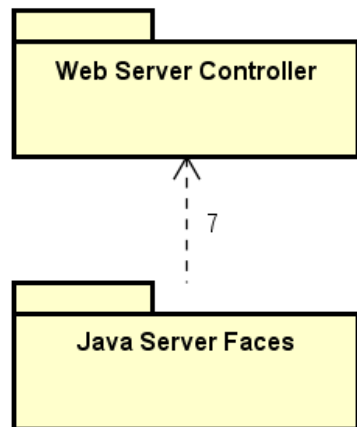
- ▶ **Bottom-up approach**

- ▶ Nature of the system: different components relies on other.
- ▶ No useless stubs.

- ▶ Integration will start from the components with the minimum number of dependencies.

# Integration Sequence

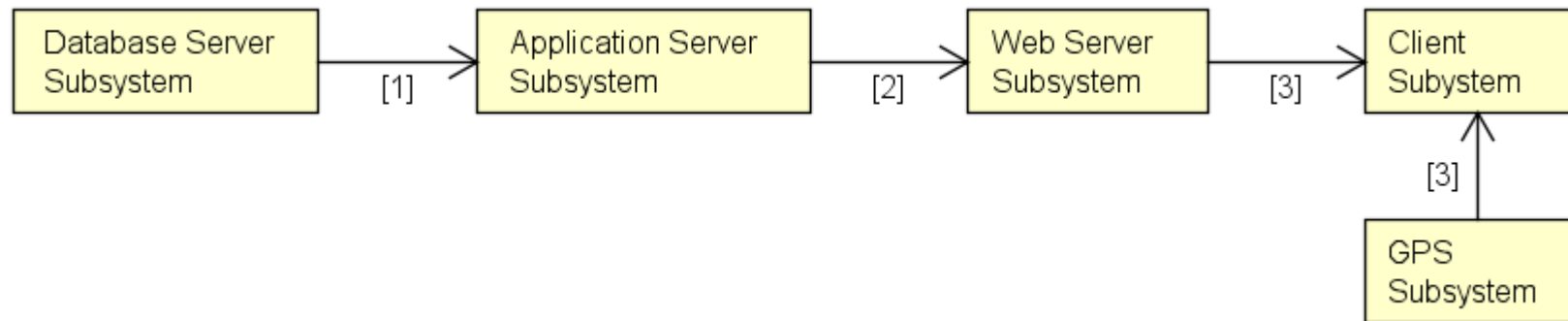
Software



# Integration Sequence

Subsystems

# Integration Sequence - Subsystems





# Example Of Test Cases

Test Case Identifier	I1T1
Test Item(s)	DBMS → Entity Bean "Area"
Input Specification	Typical query on the table "Area".
Output Specification	All the requested operations are made on the table and all the expected data is returned from the query.
Environment Needs	Testing Database, Glassfish Server, Driver for the Entity Bean
Purpose	This test check if the called methods of the Entity Bean "Area" execute the expected query on the DBMS.

Test Case Identifier	SI2T1
Test Item(s)	Application Server Subsystem → Web Server Subsystem
Input Specification	The Web Server Subsystem calls the Remote EJB on the Application Server Subsystem.
Output Specification	The action seen on the system is the expected one.
Environment Needs	Glassfish Server, Application Server's software components must be completed, driver for the Web Server Subsystem (can be a Browser)
Purpose	This test checks if the Web Server and the Application Server can properly communicate without errors, and verify if the expected actions are performed on the system.

*All test cases are available in the documentation.*



# Project Plan

# Function Points Analysis

FP Type	FP Count
Internal Logic Files	46
External Interface Files	20
External Inputs	90
External Outputs	17
External Inquiries	16
Total:	189
SLOC = 53 * FP_Count:	10017

*The complete analysis of the functional points is available in the documentation.*

► Java Multiplier for SLOC is 53



# COCOMO II – Scale Drivers

Code	Name	Factor	Value
PREC	Precedentedness	Low	4.96
FLEX	Development Flexibility	High	2.03
RESL	Architecture / Risk Resolution	High	2.83
TEAM	Team Cohesion	Very High	1.10
PMAT	Process Maturity	High	3.12
$E = 0.91 + 0.01 * \sum_i SD(i)$			1.0504

- ▶ **PREC** is set to "Low" because there is no enough experience with the used technology and the design skills achieved until now are not enough.
- ▶ **FLEX** is set to "High" because there are some goals and some general key-point defined, but there is also a good level of flexibility.
- ▶ **RESL** is set to "High" due to the risks analysis done.
- ▶ **TEAM** is set to "Very High" because the hired developers will have a good level of experience in working in team.
- ▶ **PMAT** is set to "High" because the correspondent CMM level is 3.
- ▶ *The complete analysis of the drivers is available in the documentation.*



# COCOMO II – Cost Drivers

Code	Name	Factor	Value
<b>Product</b>			
RELY	Required Software Reliability	Nominal	1.00
DATA	Data Base Size	Nominal	1.00
CPLX	Product Complexity	Nominal	1.00
RUSE	Developed for Reusability	Nominal	1.00
DOCU	Documentation Match to Lifecycle Needs	Nominal	1.00
<b>Platform</b>			
TIME	Execution Time Constraint	Nominal	1.00
STOR	Storage Constraint	Nominal	1.00
PVOL	Platform Volatility	Low	0.87
<b>Personnel</b>			
ACAP	Analyst Capability	Nominal	1.00
PCAP	Programmer Capability	High	0.88
PCON	Personnel Continuity	Very High	0.81
APEX	Application Experience	Very Low	1.22
PLEX	Platform Experience	Very Low	1.19
LTEX	Language and Toolset Experience	Low	1.09
<b>Project</b>			
TOOL	Use of Software Tools	Low	1.09
SITE	Multisite Development	Extra High	0.80
SCED	Required Development Schedule	Nominal	1.00
<b>EAF = <math>\prod_i CD(i)</math>:</b>			0.8557

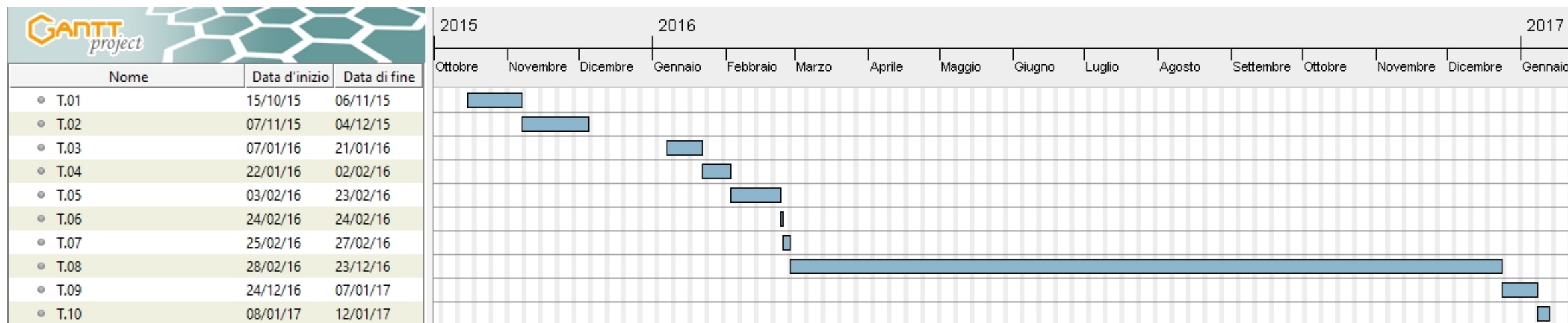
# COCOMO II – Results of Analysis

- ▶  **$KSLOC = 10.017$**
- ▶  **$EAF = \prod_i CD(i) = 0.8557$**
- ▶  **$E = 0.91 + 0.01 * \sum_i SD(i) = 0.91 + 0.01 * 14.04 = 1.0504$**
- ▶  **$SE = 0.28 + 0.2 * (E - 0.91) = 0.28 + 0.2 * (1.0504 - 0.91) = 0.28 + 0.02808 = 0.30808$**
- ▶  **$Effort = 2.94 * EAF * KSLOC^E = 2.94 * 0.8557 * 10.017^{1.0504} = 28.3037 \text{ person/months}$**
- ▶  **$Duration = 3.67 * Effort^{SE} = 3.67 * 27.3412^{0.30808} = 10.279 \text{ months}$**
- ▶  **$N_{people} = \frac{Effort}{Duration} = \frac{28.3037}{10.279} = 2.7535 \text{ people}$**
- ▶  **$EffettiveDuration = \frac{Effort}{Actual Memebtrs} = \frac{28.3037}{3} = 9.4346 \text{ months} = 284 \text{ days}$**

# Tasks and Schedule

Task ID	Task Description	Starting Date	Deadline	Duration	Status
T.01	Creation of the Requirement Analysis and Specification Document (RASD)	15/10/2015	06/11/2015	23 Days	Done
T.02	Creation of the Design Document (DD)	07/11/2015	04/12/2015	28 Days	Done
T.03	Creation of the Integration Testing Plan Document (ITPD)	07/01/2016	21/01/2016	15 Days	Done
T.04	Creation of the Project Plan Document (PPD)	22/01/2016	02/02/2016	12 Days	Done
T.05	Creation of the slides for the Project Presentation	03/02/2016	23/02/2016	21 Days	To Be Done
T.06	Presentation of the slides to the stakeholders	24/02/2016	24/02/2016	1 Days	To Be Done
T.07	Apply changes to the project documentation, if required	25/02/2016	27/02/2016	3 Days	To Be Done
T.08	Implementation	28/02/2016	23/12/2016	300 Days	To Be Done
T.09	Integration Testing	24/12/2016	7/01/2017	15 Days	To Be Done
T.10	Deployment	8/01/2017	12/01/2017	5 Days	To Be Done

# Gantt Diagram





# Risks Analysis

## ► Project Risks:

- Delays over the planned deadlines (High Probability, Medium Effect)
  - *Contromeasure*: release application without less essential features.
- Requirements change (Very Low Probability, Medium Effect)
  - *Contromeasure*: the developed code must be easily extensible.
- Lack of Experience (High Probability, Medium Effect)
- Temporary Unavailability of personnel involved in critical tasks (Medium Probability, Medium Effect)
  - *Contromeasure*: reorganize the resource allocation

## ► Technical Risks:

- Data loss (Very Low Probability, Very High Effect)
  - *Contromeasure*: backup of all data into a dislocated region.
- Data Leaks and security issues (Very Low Probability, Very High Effects)
  - *Contromeasure*: identify and fix the component that causes the data leaks; Also verify the configuration of each service.

# Risks Analysis

- ▶ **Business Risks:**

- ▶ Bad estimation of Project Costs (Medium Probability, Medium Effect)

- ▶ *Contromisure*: allocate more economical resources to the project or downscale the project size.



Questions?





Thank You