POLITECNICO
MILANO 1863

Software Engineering 2 Project: myTaxiService

Project Presentation

Damiano Binaghi, Giovanni Zuretti

RASD

The purpose of our project:

The project aims to create a new application for the management of taxi service for helping both customers, in order to make them book easily their rides, and taxi drivers, in order to let them manage their work in a more e efficient and organized way.

The purpose of our Requirement Analysis and Specification Document:

The purpose of this document is to collect and describe both functional and non-functional requirements of the Taxi Service application which we were asked to design.

MAIN GOALS AND FUNCTIONAL REQUIREMENTS

- **Allow users to sign-up, to log-in and to manage their own profile.**
  - The system must provide sign-up, log-in and profile modification functionalities.
  - The system check the correctness of the inserted data.
  - The system send a mail when user sign-up to verify the correctness of email.
- **Allow data managers to register, modify and delete taxi driver profiles.**
  - The system must provide the following functionalities: insertion, deletion and modification of taxi profile.
  - The system must generate a password for the new taxi user.

MAIN GOALS AND FUNCTIONAL REQUIREMENTS

- **Allow customer to real-time book a taxi ride and notify the assigned taxi driver.**
  - The system must provide an input form which allows the customer to insert the information about his position.
  - The system must assign from the taxi queue of the zone in which the customer is (or, if no taxis in that zone are free, from the other closest zone) a taxi cab to the request.
- **Allow customer to book in advance a taxi ride (advance-booking) and notify the assigned taxi driver.**
  - The system must provide an input form which allows the customer to insert information about the starting point, the destination point and the time at which he will need the taxi.
  - 10 minutes before the time specified in the reservation, the system must allocate a taxi to answer the request.

MAIN GOALS AND FUNCTIONAL REQUIREMENTS

- **Allow fair management of the queues.**
  - Whenever a driver ends a ride or changes zone (automatically detected by gps device) the system must insert him in the queue of the zone where he is.
  - The system must place as first in the queue of each zone the driver that in that zone has the highest absolute waiting time (time passed from the last end ride or log-in operation).
- **Allow taxi drivers to either accept or refuse the assignation and notify customer (in case of acceptance).**
  - The system must provide an interface to allow the taxi driver to either accept or refuse an assignation.
  - The system must send a notification to the customer related to the request when a taxi diver accept the assignation.
  - The system must reinsert the driver as last in the queue if the request is rejected.

MAIN GOALS AND FUNCTIONAL REQUIREMENTS

- **Allow taxi drivers to report fake-users or other problems (accident or mechanical failure).**
  - The system must provide an interface for the taxi drivers to cancel an assignation in case of accidents, fake users or other problem.
  - The system must automatically search for a new taxi to answer the request of the customer and notify them.
- **Do not allow customer to take the wrong taxi or taxi driver to pick up the wrong customer.**
  - With the notification of assignation the system must also send to the customer a unique code associated to the taxi which is going to provide the ride.
  - The system must check that the code inserted by the taxi driver matches the code generated by the system itself for that particular ride.

MAIN GOALS AND FUNCTIONAL REQUIREMENTS

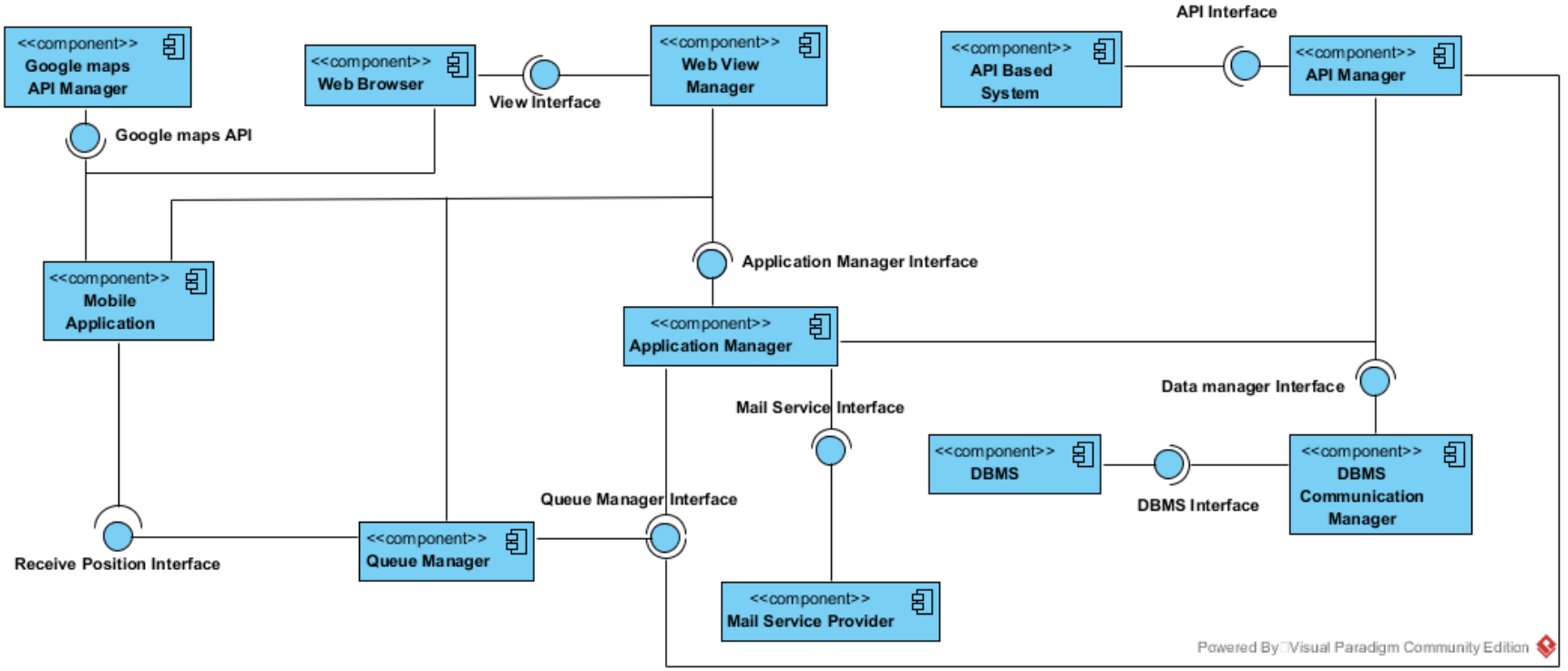
- **Allow the development of applications that need to interface with myTaxiService through APIs.**
  - The system must provide programmatic interfaces to allow other application to access user data (only read, no modi cation) and user's functionalities (e.g booking taxis).
  - The system must provide programmatic interfaces to allow users to use the same log-in credentials in other systems which interface with our system.

DD

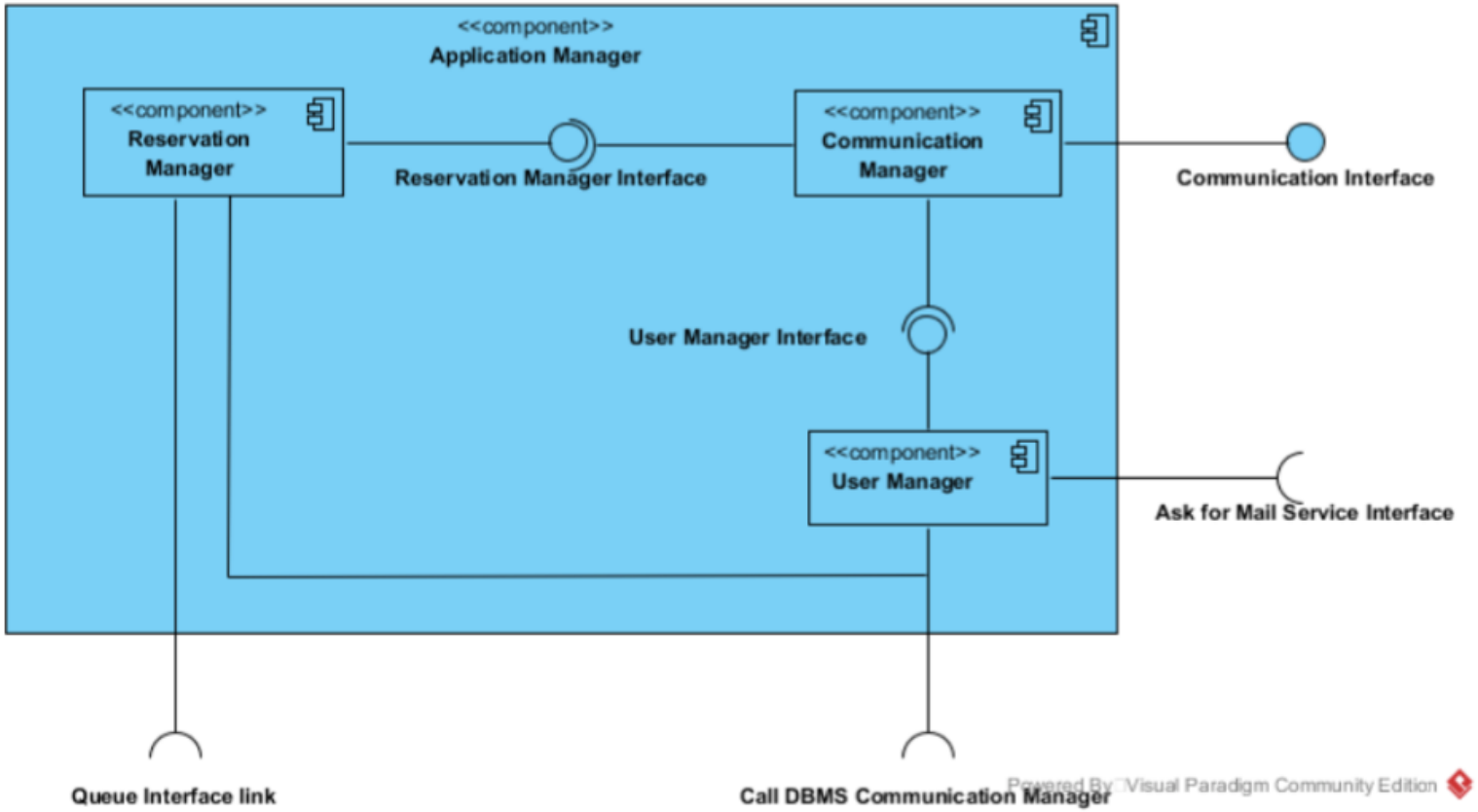
Purpose of Design Document:

This document identifies which are the best choices to take in order to meet all the requirements specified in the Requirement Analysis and Specification Document. For choices we mean architectural choices, which will involve both the software and the hardware (e.g. the physical tier division of the application) that will build the actual system which will provide the real service.

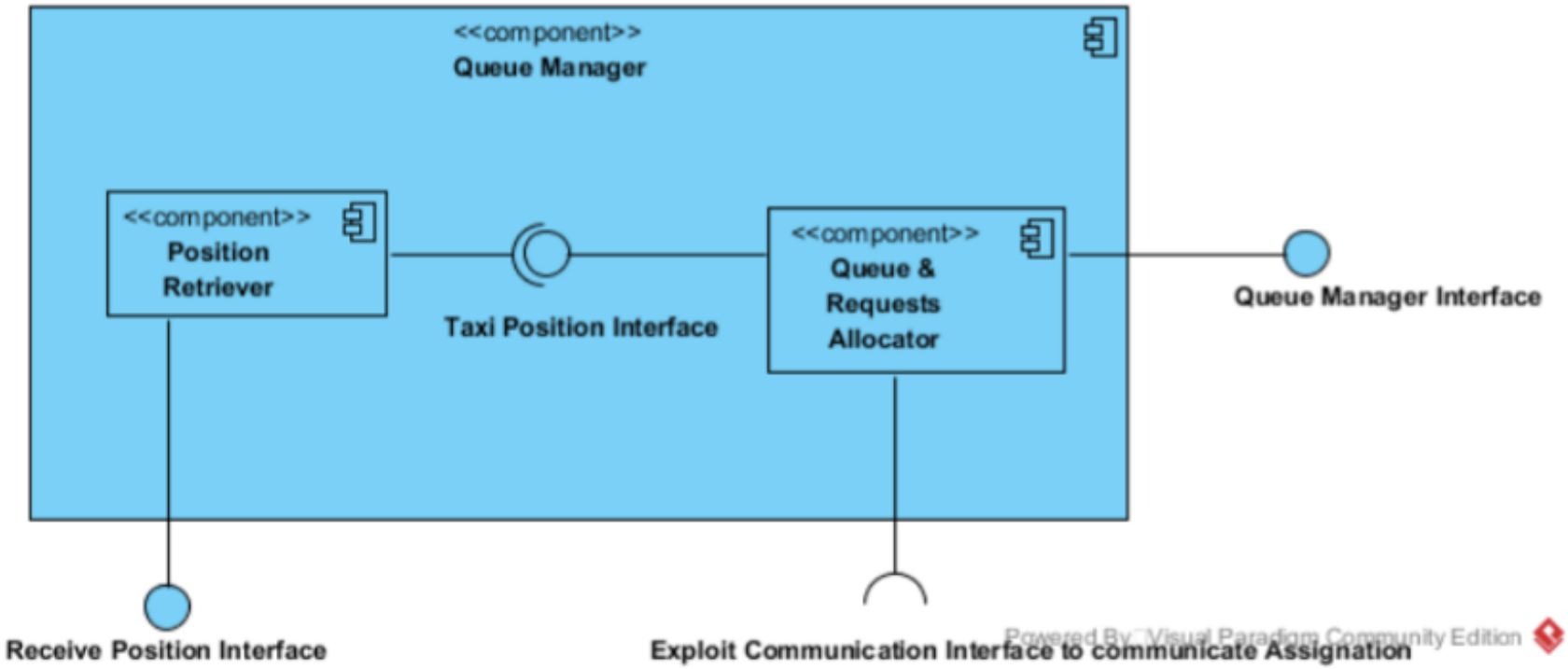
HIGH LEVEL COMPONENT VIEW



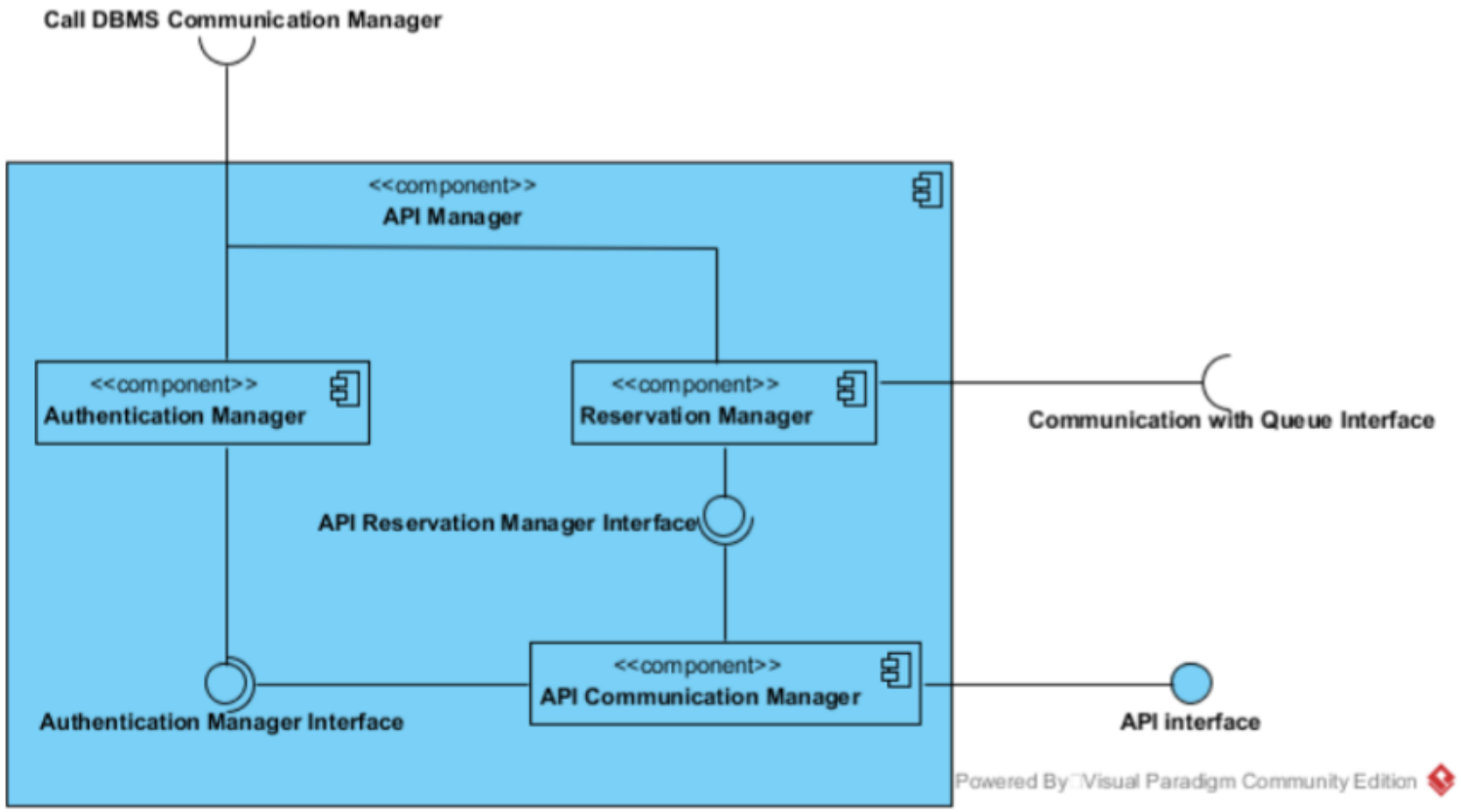
APPLICATION MANAGER



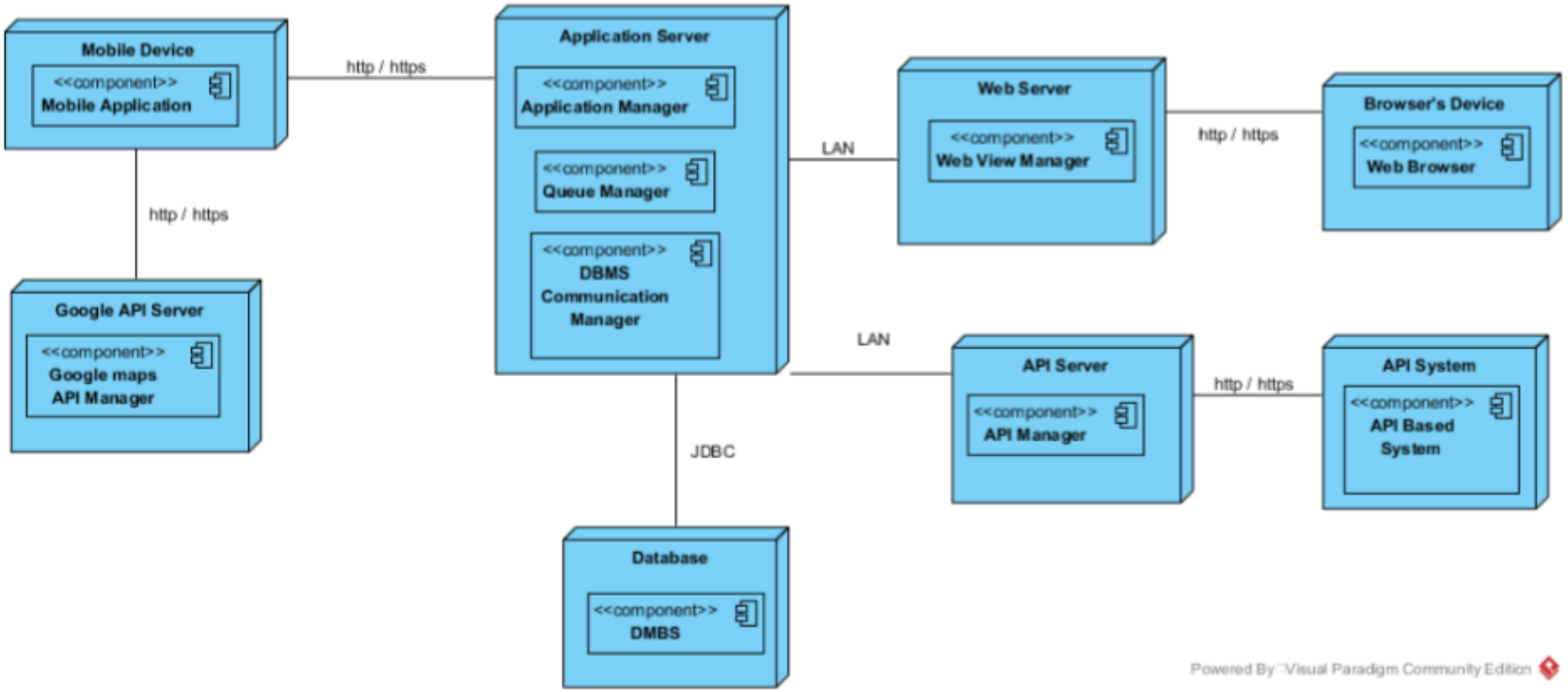
QUEUE MANAGER



API MANAGER


Call DBMS Communication Manager
<<component>>
API Manager
<<component>>
Authentication Manager
<<component>>
Reservation Manager
Communication with Queue Interface
API Reservation Manager Interface
<<component>>
API Communication Manager
Authentication Manager Interface
API interface
Powered By Visual Paradigm Community Edition

DEPLOYMENT VIEW



SELECTED ARCHITECTURAL STYLES

- **Service Based Application**
  - MyTaxiService is a Service Based Application because both it uses and it provides some sort of services, but we cannot say it is service oriented because it is not true that each component of the application can be seen as a service and can be externally used as a service.
- **Event Based Application**
  - MyTaxiService is an Event Based Application because the request of booking a new ride made by a customer is managed as an event by the system, where drivers and customers are the subscriber of such events.
- **Client-Server Application**
  - MyTaxiService is also a Client-Server Application because the user profile and the authentication are managed in a typical Client-Server style.

SELECTED ARCHITECTURAL STYLES

- **MVC Pattern in our Application**
  - MyTaxiService is based on the Model-View-Controller (MVC) pattern. We decide to use such pattern for the development of the application because we have different type of views (Mobile Application and Web Application).
- **Multi-Tier Pattern**
  - This pattern was selected to divide the parts of our software which accomplish different tasks.
  - We selected a 4-tier architecture for the Web Application (client, web server, application server, database) which does not require the installation of any software client side,
  - We selected a 3-tier architecture for the mobile application (client, application server, database).

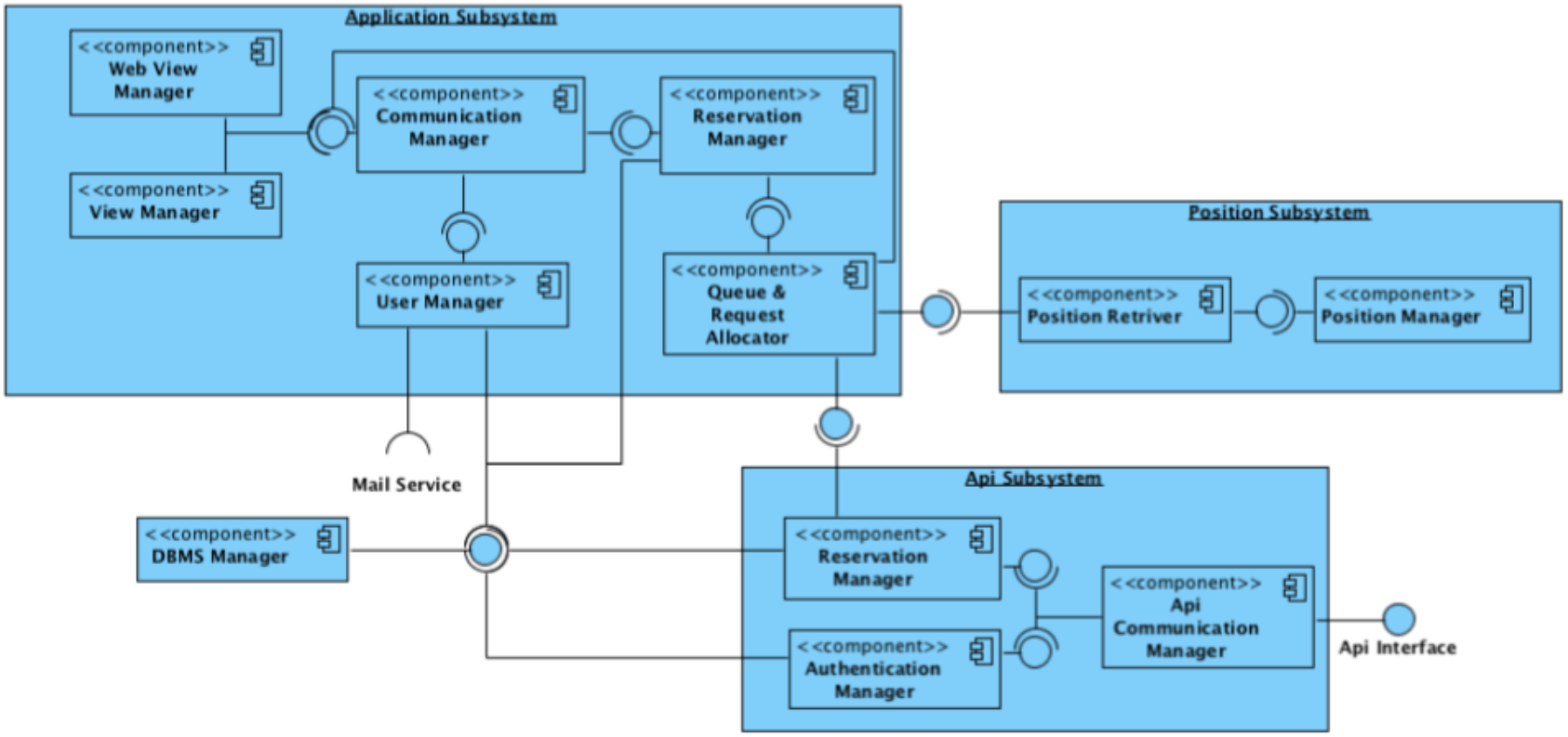
USER INTERFACE

INTEGRATION TEST PLAN

Purpose of Integration Test Plan:

The purpose of this document is to explain and plan how the integration among the software modules of our application should be done.

ELEMENTS TO BE INTEGRATED



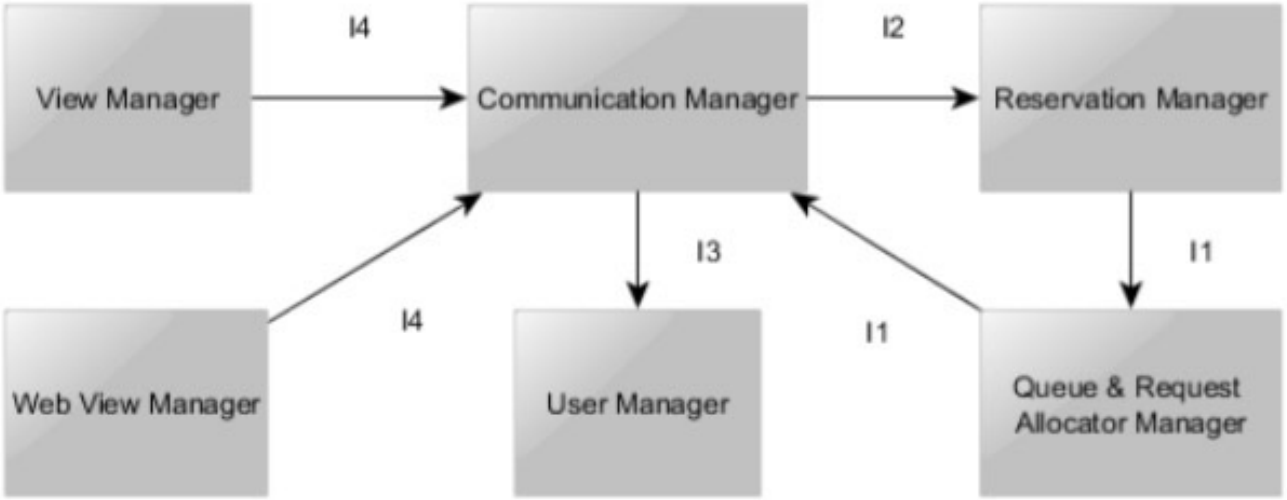
INTEGRATION TEST STRATEGY

We will use a bottom-up approach:

We chose such approach because we thought was the best for the type and the complexity of the application we have designed. We thought, indeed, that the thread and the critical modules approach were too complex for our system which is rather simple.

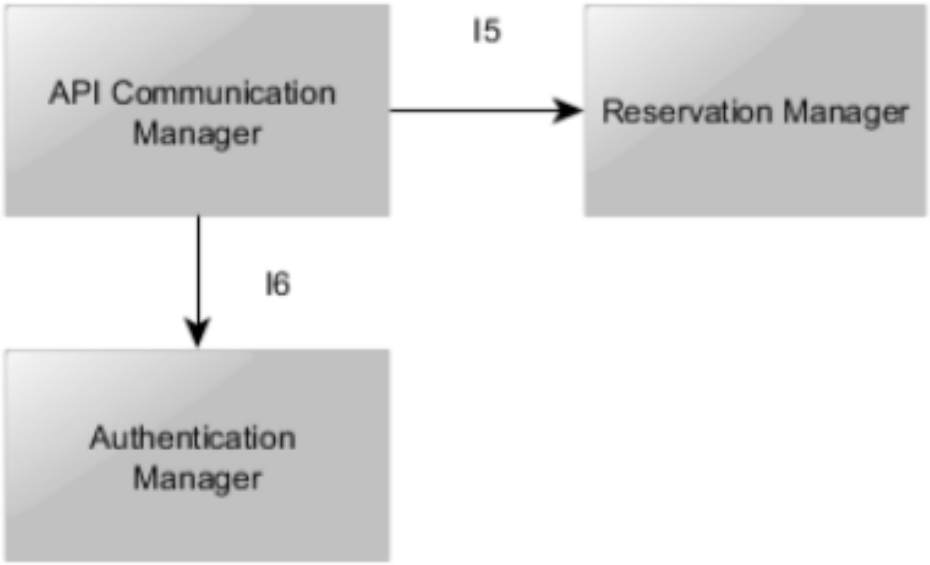
A sandwich approach could be a little confusing, while we chose bottom-up over top-down because we have more confidence with it and because it is easier to test leaves components in such away.

INTEGRATION OF APPLICATION SUBSYSTEM



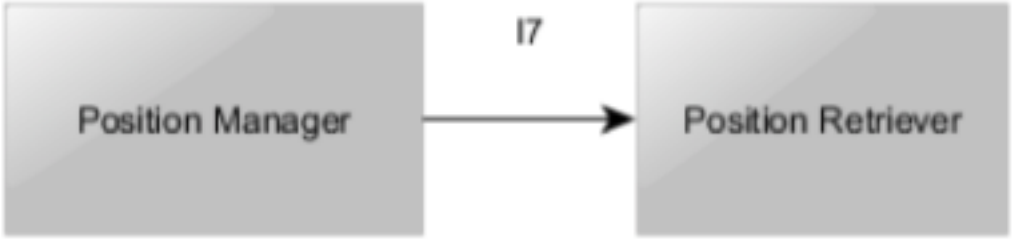
ID	Integration Test
I1	Reservation Manager → Queue & Request Allocator Manager → Communication Manager
I2	Communication Manager → Reservation Manager
I3	Communication Manager → User Manager
I4	View Manager, Web View Manager → Communication Manager

INTEGRATION OF API SUBSYSTEM



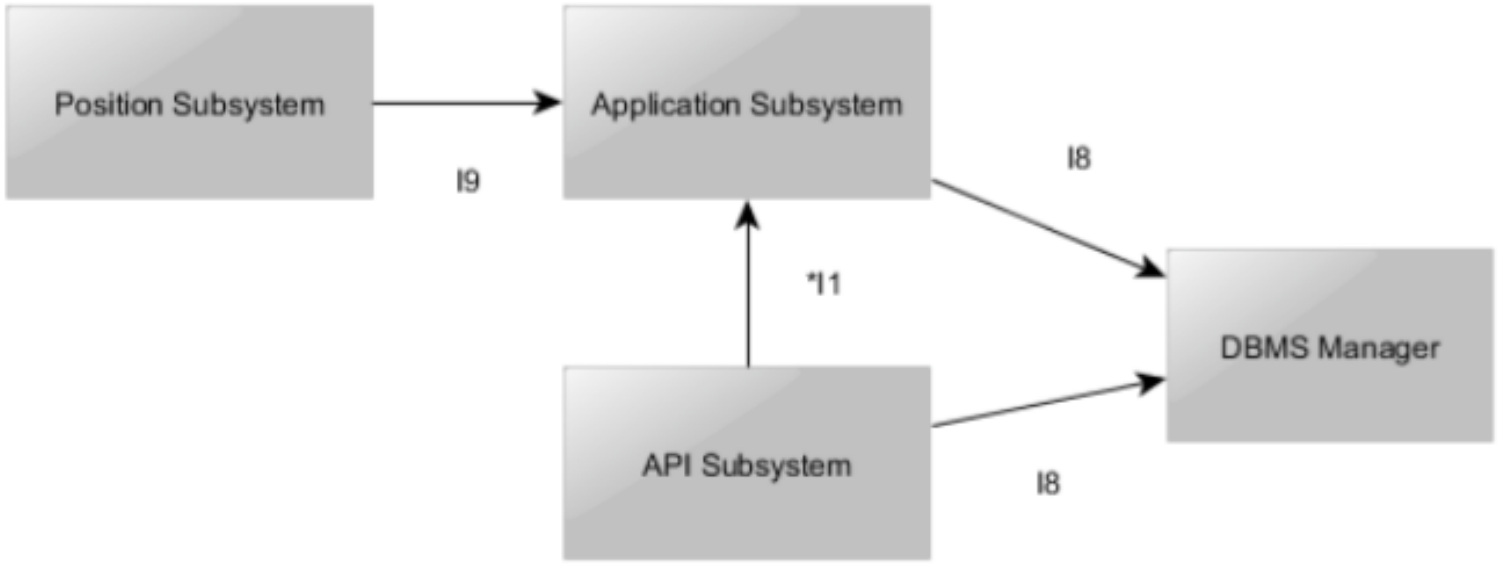
ID	Integration Test
I5	API Communication Manager → Reservation Manager
I6	API Communication Manager → Authentication Manager

INTEGRATION OF POSITION SUBSYSTEM



ID	Integration Test
I7	Position Manager → Position Retriever

SUBSYSTEM INTEGRATION SEQUENCE



ID	Integration Test
*I1	API Subsystem → Application Subsystem
I8	Application Subsystem, API Subsystem → DBMS Manager
I9	Position Subsystem → Application Subsystem

EXAMPLE OF INTEGRATION TEST CASE

INTEGRATION TEST CASE I1

Test Items: Reservation Manager, Queue & Request Allocator Manager, Communication Manager

Input Specification: Create multiple instance of an instant-booking request by different users in different zones. Create also different type of answers (accept or refuse) by taxi drivers.

Output Specification: Check that a taxi driver is assigned to the requests.

Objects and Methods: It will be used the method newRequest (user, position) in the Queue & Request Allocator Manager, an the chain of calls for assigning a taxi driver between the Queue & Request Allocator Manager and the Communication Manager.

Environmental Needs: Request Manager Driver, which creates the instant- booking requests, and Communication Manager Stub, which receives the communications of assignation of a driver and replies to them. Furthermore, the Queue & Request Allocator must be initialized with different queues and with different taxi drivers in each of them.

TEST PROCEDURE

Integration test procedure TP1

Purpose: This procedure verifies that

- The system manages a booking request by the user.
- The system correctly communicates with the assigned taxi.
- The system retrieves and generates the correct information about the booking request and that it communicates them to the interested parts.
- The system handles correctly the generation of request from an advance booking instance stored in the DB.

Procedure steps: I1, I2, I4, I8

Integration test procedure TP2

Purpose: This procedure verifies that

- The system manages correctly the registration of a new user.
- The system manages correctly the login procedure.
- The system manages correctly modi cation of user information.

Procedure steps: I3, I4, I8

TEST PROCEDURE

Integration test procedure TP3

Purpose: This procedure verifies that

- The system manages correctly the updating of the position of the taxis Procedure steps: I7, I9

Integration test procedure TP4

Purpose: This procedure verifies that

- The system manages correctly the login of external application through APIs.
- The system manages correctly all functionalities provided through APIs. Procedure steps: I1,I5, I6, I8

PROJECT PLAN DOCUMENT

Purpose of Project Plan Document:

The purpose of this document is to describe the project plan for the myTaxiService Application.

This document was done afterwards some activities and tasks we will identify were already completed, for example the requirement analysis and the design process. However they will still be taken into account for the planning.

FUNCTION POINTS APPROACH

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

INTERNAL LOGIC FILES

The application include a number of Logic Files used to store information about:

- Customers (simple)
- Taxi Drivers (simple)
- other type of Users (Taxi Data Managers and the system administrator) (simple)
- Booking Requests (medium)

FP = (3 x 7) + (1 x 10) = 31

EXTERNAL INPUTS

- Sign up (simple)
- Login (simple)
- Logout (simple)
- Edit Personal Data (simple)
- Insert new Taxi Driver (simple)
- Instant Booking Request (simple)
- Advance Booking Request (medium)
- Delete a request (simple)
- Send Position (simple)
- Confirm/Refuse Assignation (simple)
- Report Fake (simple)
- Report Accident (simple)
- Verification Code Insertion (simple)
- Start Ride (simple)
- End Ride (simple)
- API Authentication (simple)
- Instant Booking Request From API (simple)
- Advance Booking Request From API (medium)

FP = (16x 3) + (2 x 4) = 56

EXTERNAL OUTPUT

- When a new request must be handled, the system should assign a driver to the request notifying him of the assignation. (simple)

- When a driver is assigned successfully to a request, the system should notify the waiting customer that his request has been handled communicating the waiting time for the taxi to arrive. This must done both for the normal system and the API service offered by the system itself. (2 x medium)

  . $FP = (1 \times 4) + (2 \times 7) = 18$

EXTERNAL INQUIRIES

We have identified the following inquiries:

- Show booking history to customer (simple)
- Show fake reports to admin (simple)
- Show accidents report to admin (simple)

FP = 3 x 3 = 9

FINAL CALCULATION

ILF	31
External Inputs	56
External Outputs	18
External Inquiries	9
Total	**114**

LOC = 114 x 46 = 5244

COCOMO II APPROACH

Scale Driver	*Factor*	*Value*
Precedentedness	Low	4.96
Development Flexibility	High	2.03
Risk Resolution	Nominal	4.24
Team Cohesion	Very High	2.19
Process Maturity	Low	6.24
Total		19.66

COCOMO II APPROACH

Cost Drivers	*Factor*	*Value*
Required Software Reliability	High	1.10
Data Base Size	Low	0.9
Product Complexity	High	1.17
Developed for Reusability	Nominal	1.00
Documentation Match to Life-Cycle needs	Nominal	1.00
Execution Time Constraint	Low	n/a
Main Storage Constraint	Low	n/a
Platform Volatility	Low	0.87
Analyst Capability	High	0.85
Programmer Capability	High	0.88
Personnel Continuity	Extra High	n/a
Application Experience	Low	1.10
Platform Experience	Nominal	1.00
Language and Tool Experience	Nominal	1.00
Usage of Software Tools	Very High	1.00
Multisite Development	Extra High	0.8
Required Development Schedule	Nominal	1.00
Product		0.66

EFFORT ESTIMATION

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

- A = 2.94 (Constant provided by COCOMO II)
- EAF = CostDriverFactor = 0.66
- KSLOC= 5.244
- E = 0.91 + ScaleDriverFactor * 0.01 = 0.91 + 19.66 * 0.01= 1.1066 (again 0.91 is a constant de ned in COCOMO II)

$$Effort = 2.94 * 0.66 * 5.244^{1.1066} = 12.14\ Person/Month$$

$$Duration = 3.67 * Effort^{F}$$

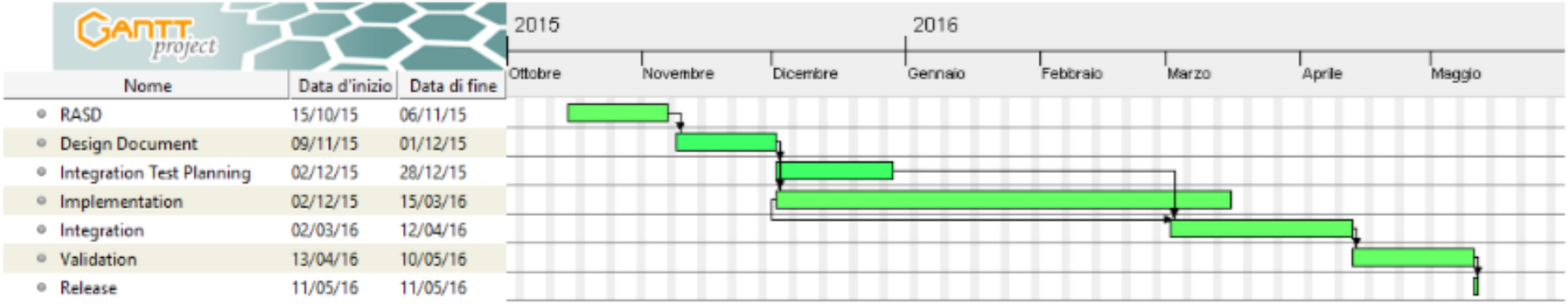
$$F = 0.28 + 0.2 * (E - 0.91) = 0.31932$$

$$Duration = 3.67 * 12.14^{0.31932} = 8,14 \sim 8\ Months$$

$$NumberOfPeople = Effort/Duration = 12.14/8.14 = 1.49 \sim 2\ People$$

TASKS IDENTIFICATION AND SCHEDULING

- Requirement Analysis and Specification Document
- Design Document
- Integration Test Planning
- Implementation
- Integration
- Validation
- Release



Nome	Data d'inizio	Data di fine
RASD	15/10/15	06/11/15
Design Document	09/11/15	01/12/15
Integration Test Planning	02/12/15	28/12/15
Implementation	02/12/15	15/03/16
Integration	02/03/16	12/04/16
Validation	13/04/16	10/05/16
Release	11/05/16	11/05/16

CODE INSPECTION

ASSIGNED CLASS:

- AnnotationProcessorImpl.java

ASSIGNED METHODS:

- process (ProcessingContext ctx, Class c)
- processAnnotations (ProcessingContext ctx, AnnotatedElement element)
- process (ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result)

FUNCTIONAL ROLE OF THE CLASS

- The AnnotationProcessorImpl class implements the AnnotationProcessor interface which is the core for processing annotations in Glassfish 4.1.
- The AnnotationProcessorImpl class is responsible for maintaining a list of annotations handlers per annotation type.
- Annotation handlers are managed as a stack.
- A processing context is needed in order to have access to class instances.

process (ProcessingContext ctx, Class c)

- Private method called by the following public methods: process(ProcessingContext ctx), which processing all classes in the given ctx context, and process(ProcessingContext ctx, Class[] classes), which will process a particular set of classes
- Checks if package annotation is already present
- Does not abort the whole annotation process in case the class is not present in the context
- Calls the method process(ProcessingContext ctx, AnnotatedElement element) in order to process all the elements type (CLASS, FIELD, CONSTRUCTOR, METHOD, PARAM)
- Checks for possible superclasses involved in class annotations
- Returns a ProcessingResult set, which contains all the new annotations processed

processAnnotations (ProcessingContext ctx, AnnotatedElement element)

- Private method called by process(ProcessingContext ctx)
- This method scans all the annotations for the element which is passed as a parameter
- For each annotation a new AnnotationInfo object, called subElement, is initialized
- If subElement has not been processed yet, the method process(ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result) is called, else the current methods logs that subElement has already been processed
- Returns the set of all the processed annotations of element

process (ProcessingContext ctx, AnnotationInfo element, HandlerProcessingResultImpl result)

- Private method called by process(ProcessingContext ctx, AnnotatedElement element)
- Logs the annotations that is going to be processed and marks it as UNPROCESSED, and calls the handler for the annotation
- If the handler is found, checks for possible dependencies between annotations and process those dependencies which were not processed yet
- Annotations are processed calling the processAnnotation(AnnotationInfo element), which may return an AnnotationProcessorException. If the exception is fatal, the methods returns throwing the same exception. Otherwise an error counter is incremented. If the counter is grater than 100 the annotation process is abort.
- If the method returned an exception, but no one of the previous “early-termination” condition was verified, the ResultType for the annotation is set to FAILED. In this case and in the case that the processing was successful, the processingResult are added to the result passed as parameter.
- If no handler was found for the given annotation type, it is searched in the delegate of the AnnotationProcessorImpl. If there is no delegate an error is reported.

END

THANK YOU FOR ATTENTION!!!