1. Podium
2. Wired Internet connection for speakers (conference WiFi for attendees)
3. **PC *(Windows based laptop with current version of MS Office)***
4. Head table with 2 chairs
5. Wired Lavaliere(s)/Handheld mic(s)
6. **Projector — 1280×720 (720p) – HDMI connections available**
7. **Screen — 16:9 aspect ratio**
8. Seating: sessions – theater style
9. Panels will be provided extra seating and handheld mics

Sommario

[1 [INTRODUCTION] 3](#_Toc463299308)

[2 [PRESENTATION] 3](#_Toc463299309)

[3 SLIDE 3: A DIGITAL PLATFORM FOR A SUSTAINABLE MOBILITY 3](#_Toc463299310)

[4 SLIDE 4-5-6 REQUIREMENTS 5](#_Toc463299311)

[4.1 [REQUIREMENTS 1 – FUNCTIONAL ] 5](#_Toc463299312)

[4.2 [REQUIREMENTS 2– NON FUNCTIONAL ] 5](#_Toc463299313)

[4.3 [REQUIREMENTS 3– PROJECT MANAGEMENT ] 5](#_Toc463299314)

[5 [FULFILMENT OF THE REQUIREMENTS] 5](#_Toc463299315)

[6 SLIDE 7: Microservices based architecture – «database per service» pattern 5](#_Toc463299316)

[A microservice architecture with a Database per service pattern will fulfill the non functional requirements 5](#_Toc463299317)

[7 SLIDE 9: Microservices based architecture – «database per service» pattern 5](#_Toc463299318)

[TO BE DEFINED 5](#_Toc463299319)

[8 SLIDE 10: Requirements fullfilment: technology stack 6](#_Toc463299320)

[9 SLIDE 21: Requirements fullfilment: lifecycle process 6](#_Toc463299321)

[9.1 [SYSTEM LANDSCAPE] 7](#_Toc463299322)

[10 SLIDE 30: Development/Unit Test 7](#_Toc463299323)

[11 SLIDE 41: Integration 7](#_Toc463299324)

[12 SLIDE 52: Quality Assurance 7](#_Toc463299325)

[14 SLIDE 123: Conclusion 9](#_Toc463299326)

[15 SLIDE 124: what is next 9](#_Toc463299327)

[~~16~~ ~~Demo~~ 10](#_Toc463299328)

[16.1 [DEVELOPMENT / UNIT TEST ] 11](#_Toc463299329)

[16.1.1 LOCAL H2 12](#_Toc463299330)

[16.1.2 LOCAl MYSQL 12](#_Toc463299331)

[16.1.3 [SCENARIO INTEGRATION TEST/DOCKER] 13](#_Toc463299332)

[16.1.4 [SCENARIO QUALITY ASSURANCE–DOCKER HUB] 15](#_Toc463299333)

[16.1.5 [SCENARIO PRODUCTION–PIVOTAL WEB SERVICES] 16](#_Toc463299334)

[16.1.6 [EVENT DRIVE DEMO] 17](#_Toc463299335)

[17 Wiring microservices - Service Discovery 18](#_Toc463299336)

[17.1.1 [WHAT’S NEXT] 18](#_Toc463299337)

[17.1.2 [CONCLUTION] 18](#_Toc463299338)

# [INTRODUCTION]

Welcome everybody my name is Luigi Bennardis, I’m an information Technology architect with a seniority of about twenty years. Actually I’m in charge of the design and management of the Application Lifecycle platform for the Italian Postal Services.

# [PRESENTATION]

A the title suggest the aim of this project is to provide a digital platform based on a microservices architecture, which will require an application lifecycle management process.

The gathering of the requirements will be the first phase of the process, followed by the choice of the technical design.

The third step is the development of the software and its integration.

The fifth phase is the quality assurance, followed by the sixth and final phase: delivery.

~~architectural (database per service) and technological (Spring Boot) aspects to delivery related scenarios (development, Cloud or dockerized environments), in an ecosystem context where microservices are each other reliable and fault tolerant (Eureka service registry, Ribbon load balancing, Spring Cloud).~~

# SLIDE 3: A DIGITAL PLATFORM FOR A SUSTAINABLE MOBILITY

**[FIRST IMAGE OF THE MAP OF ROME – HISTORICAL DISTRICT]**

This is historical district of Rome.

It is a very ancient area whose urban development dates back to the roman empire, more than  2000 years ago, through the ages, when the city structure has not evolved in a sustainable way.

**[SECOND IMAGE – HISTORICAL DISTRICT OF ROME]**

The blue line surrounds the limited traffic zone .

It is an area of about 10 square kilometers caracterized by a strong concentration of historical monuments, tourism facilities and national institutions, government and parliament among others, that generate a consistent demand for mobility.

As a matter of a fact the urban development was not planned originally planned for the actual need of people and motor vehicle mobility. The street are very often often too narrow in a context of a fragile architectural environment.

So the current mobility situation of this area is unsustainable.

**[THIRD IMAGE – SMART ELECTRIC VEHICLE]**

The solution to this mobility problem is a network of smart electrical vehicles with removal batteries whose organization is powered by a microservices based digital platform.

**[FOURTH IMAGE – SMART ELECTRIC VEHICLE – BATTERY + STATIONS ]**

In this way we could create a modern urban mobility network that is both economically and environment sustainable.

A virtually 24/7 service

# SLIDE 4-5-6 REQUIREMENTS

**INTRODUCTION**

**The gathering of requirements is the first phase of the lifecycle process. Requirements could be classified in Functional, Non Functional and project management related**

**[FOLLOWS SLIDE PRESENTATION 4-5-6] .**

## SLIDE 4 :REQUIREMENTS 1 – FUNCTIONAL

## SLIDE 5 :REQUIREMENTS 2– NON FUNCTIONAL

## SLIDE 6 :REQUIREMENTS 3– PROJECT MANAGEMENT

# [FULFILMENT OF THE REQUIREMENTS]

**INTRODUCTION**

SO LET’S SEE HOW FULFILL ALL THIS REQUIREMTS CHOOSING THE DESIGN PATTERN,THE TECNOLOGY STAK AND THE RIGHT LIFECYCLE PROCESS.

# SLIDE 7: Microservices based architecture – «database per service» pattern

**INTRODUCTION**

## A microservice architecture with a Database per service pattern will fulfill the non functional requirements

**[FOLLOWS SLIDE PRESENTATION 7] .**

# SLIDE 9: Microservices based architecture – «database per service» pattern

**INTRODUCTION**

## TO BE DEFINED

**[FOLLOWS SLIDE PRESENTATION 9] .**

# SLIDE 10: Requirements fullfilment: technology stack

**INTRODUCTION**

THIS CHART SUMMARIZES THE TECHNOLOGY STACK I HAVE INTRODUCED IN THE DEVELOPMENT TO REALIZE THE DESIGN PATTERNS THAT IN MY OPINION COULD ADDRESS THE REQUIREMENTS OF THE PROJECT.

**[FOLLOWS SLIDE PRESENTATION FROM SLIDE 11 TO 18 ]**

# SLIDE 21: Requirements fullfilment: lifecycle process

**INTRODUCTION**

THIS CHART REPRESENTS HOW TECHNOLOGIES MAP THE RELATIONSHIP BETWEEN THE LIFECYCLE AND THE SERVICES NEEDED BY EACH PHASE.

**[FOLLOWS A BRIEF DESCRIPTION OF EACH TECHNOLOGY CORRISPONDING A SERVICE/LIFECYCLE STEP]**

## ~~[SYSTEM LANDSCAPE]~~

# SLIDE 30: Development/Unit Test

**INTRODUCTION**

To be defined.

**[FOLLOWS SLIDES]**

# SLIDE 41: Integration

**INTRODUCTION**

To be defined.

**[FOLLOWS THE EXPOTION OF SLIDES 42 – 43 ]**

# SLIDE 52: Quality Assurance

**INTRODUCTION**

To be defined.

**[FOLLOWS THE EXPOTION OF SLIDES 53 – 61 ]**

# SLIDE 123: Conclusion

**INTRODUCTION**

To be defined.

**[FOLLOWS THE EXPOTION of the slide content ]**

# SLIDE 124: what is next

**INTRODUCTION**

To be defined.

**[FOLLOWS THE EXPOTION of the slide content ]**

# ~~Demo~~

**~~DEVELOPMENT::PRELOAD CHROME PAGES~~**

~~DEV\_LOCAL.bat~~

**~~INTEGRATION::PRELOAD~~**

1. **~~COMMAND LINE DOCKERMACHINE RUN~~**
2. **~~COMMAND LINE ready to run list of container~~**
3. **~~COMMAND LINE READY TO RUN CONTAINERS~~** 
   1. **~~MYSQL~~**
   2. **~~APP WITH LINK~~**
4. **~~CHROME PAGES~~**

**~~DOC\_LOCAL.bat~~**

**~~QUALITY ASSURANCE::PRELOAD CHROME PAGES~~**

~~DOC\_HUB.bat~~

**~~PRODUCTION::PRELOAD CHROME PAGES~~**

**~~EVENT DRIVEN::PRELOAD CHROME PAGES~~**

~~EVENT.BAT~~

**~~CHECK DOCS~~**

## [DEVELOPMENT / UNIT TEST ]

**INTRODUCTION TO THIS PHASE**

Besides the implementation details of the frameworks in this first phase of lifecycle, development,

We see the resolution if the database backing service

Keeping the same coding both for a local in memory He , MySql

* Spring boot and services binding
  + Automatic behaviuor in cloud foundry
  + The goal should be a smooh transition from local execution form developer desktop to production in Cloud Foundry
  + Binding realized with declarative approach
    - Application.properties
    - Yaml file
    - [show the differences]
    - Dual Running: Local with H2/mysql
    - And the in the Cloud with MySQL
    - Configuration
  + Different file
* H2 in memory database
* <https://spring.io/guides/gs/accessing-data-rest/>
* <https://spring.io/guides/gs/accessing-data-jpa/>
* <http://blog.netgloo.com/2014/10/27/using-mysql-in-spring-boot-via-spring-data-jpa-and-hibernate/>

**[SHOW PPT SLIDES WITH IMPLEMENTATION DETAILS]**

### LOCAL H2

00pwd PROFILE h2

Environment information provided by spring boot

<http://localhost:7111/h2-console/>

<http://localhost:7111/trace>

<http://localhost:7111/flyway>

<http://localhost:7111/metrics>

example /flyway number of request

<http://localhost:7111/bookABattery/list>

list of booked batteries – EMPTY

<http://localhost:7111/bookABattery/addBooking/STAZ001/BATT011/ROMA/42.984545454/16.74444/>

add one or more booking

<http://localhost:7111/bookABattery/list>

list of booked batteries – LIST EVIDENCE

<http://localhost:7111/findNearestStation/41.90231/12.4832/0.50/>

find nearest station for an emergency pit stop

### LOCAl MYSQL

00pwd PROFILE mYSQL

[**http://localhost:7111/health**](http://localhost:7111/health)

**show the Jason with information about the disk the up status and the database in this cas MySQL**

<http://localhost:7111/bookABattery/addBooking/STAZ001/BATT011/ROMA/42.984545454/16.74444/>

add one or more booking

<http://localhost:7111/bookABattery/list>

list of booked batteries – LIST EVIDENCE

<http://localhost:7111/findNearestStation/41.90231/12.4832/0.50/>

find nearest station for an emergency pit stop

### [SCENARIO INTEGRATION TEST/DOCKER]

**INTRODUCING**

**now build a Docker image that runs the Spring Boot application.**

**Introducing a scenario of shiiping the application inside a container**

**THE INTEGRATION TEST OF A COMPONENT MUST REFERENCE / USE OTHER COMPONENT S THAT COMES FROM OTHER DEVELOPING LINES**

**SO THE DOCKER REGISTRY SHOUD BE THE HUB OF ALL THE MICROSERVICESOF THIS DIGITAL PLATFORM**

**IT WILL NOT NEE TO MANAGE DEPLOYMENT OF EXECUTALE ARTIFACTS, ONLY PULL THE IMAGE NEEDED AND EXECUTING LINKED TO ONE ON WHICH WE WANT TO CONDUC INTEGRATION TEST**

**Special images with data specific for test**

**Shiiping a inmage rather tha executable software artifact**

**artifact to deployed in qa machine**

**There is no limit to crate dedicated qa/integration machine**

**so it will be superated the sizing and the procurament a priori of machine**

**So let’s go ahead in the develipong process facing the phase of tests of integration test**

**The commit master will have the duty of conduct the integration test against all development done on one or more sevices modified for the new system release or for bug fixing coming from the QA or production environments.**

**THE INTEGRATION TEST WILL BE CONDUCTED IN AN EVIRONMENT WHERE THERE WILL BE AVAILABLE ALL THE SERVICES needed for the system to be consistent.**

**In a simple scenario two or more developers each responsible for the development of a part of a microservice finish with success the unit test**

**Beside the skill of the qa team must be enough to get the application and run it, it is stated in the requirements that in this team there will be no need of technical slìkill that will manage the deployment or other technical tasks.**

**So the commit master in a dedicated branch deliver the work of the developers merging all the contributes and build the microservice**

Docker definition

-I have provided an empty database image that will be load by flyway at start up

Show Dockerfile

Show dockerfile template

Show maven implementation and plug in

Create a local docker image

Run the container + database

Show bindings

**BEFORE FIRST SLIDE**

**I’m going to represent some higlights of configuration, mainly on the maven build plugins and the dockerfile.**

### [SCENARIO QUALITY ASSURANCE–DOCKER HUB]

**INTRODUCTION**

Among the requirements it has been stated that the project’s teams

The progect management requirements shoud accomplish teams geographically distributed.

In such a context the delivery of the application as a docker machine could be realized by means of Docker Hub as a project’s images registry.

So let’s go ahead in the develipong process facing the phase of QUALITY ASSURANCE TEST

The commit master will have the duty of conduct the integration test against all development done on one or more sevices modified for the new system release or for bug fixing coming from the QA or production environments.

THE QA TEST WILL BE CONDUCTED IN AN EVIRONMENT WHERE THERE WILL BE AVAILABLE ALL THE SERVICES needed for the system to be consistent.

In a simple scenario two or more developers each responsible for the development of a part of a microservice finish with success the unit test

Beside the skill of the qa team must be enough to get the application and run it, it is stated in the requirements that in this team there will be no need of technical slìkill that will manage the deployment or other technical tasks.

So the commit master in a dedicated branch deliver the work of the developers merging all the contributes and build the microservice

DOCKER REGISTRY WITH ALL THE IMAGES OF THE MICROSERVICES OF THE DIGITAL PLATFORM

GIT HUB A AS AMAVEN REPOSITORY OF ALL ARTIFACTS EVETUALLY NEEDED BY THE PROJECT

Docker hub /

JenkiNs@openshift –

github AS A MAVEN REPOSITORY

Simplify sistem management using Jenkins on opensstack

### [SCENARIO PRODUCTION–PIVOTAL WEB SERVICES]

**DESCRITPION**

INTRODUCE ANOTHER PAAS PROVIDER FOR THIS DIGITAL PLATFORM

* Pivotal web service presentation
  + My instance
  + Console
  + Application backing services

Quality assureance test has finished successfully so the next of the lifecycle is Production

For the runtime environment it will be used a Cloud PAAS : Pivotal Web Services

Pivotal Web Services Built on Cloud Foundry AN OPEN SOURCE paas sOLUTION

* + Backing services
  + Yml file
  + Create from interface
  + By Manve and then integrate to Jenkins
  + BY cf tool

Database as a Service Pattern implemented with a dedicated instance

Database as service pattern will enforce database per service pattern.  Dedicated instance , dedicated schema and horizontal scale up

Show definition of application .cloudfoundry and its link to application.yml

Deploy by cf tool – maven (only show plugin)

Pws  
Database as service pattern will enforce database per service pattern.  Dedicated instance , dedicated schema and horizontal scale up

Database per service  pattern implementation inside a cloud env is natively implemented  
Segregation is enforced

The databasr is dedicated to the servuce and could be scaled out independently

### [EVENT DRIVE DEMO]

**INTRODUCTION**

Event driven  
Example if extra coding and infrastructure management.   
Show the effective overhead of coding and infra management in a simple example.  
Drawback  
Benefits

# Wiring microservices - Service Discovery

INTRODUCTION

In such an architecture microservices can cooperate for a computing system ecosystem shoud wire each other

Wiring microservices.  
Loosing coupled requirements are not onky those involving tech aspects. In a ecosystem if microservice it is likely thata microservices need each othe wiring.  
Microservices do nita act by themself.  
Si the loosing coupled requirent is completely satisfied implementibg a service discovery a registry where are resolved........

Wiring by resftfull call baut to achieve decouplig it is necessary that service shoud be discocered.

Eureka server is on its turna microservice (run as a spring boot application)

### [WHAT’S NEXT]

### [CONCLUTION]

So that’s all, wishing that this presentation beeing clear enought has catched your , I would like to thanks you of your attention and O’Reilly for the opportunity given to me to take part at such an important event like that