|  |  |
| --- | --- |
|  |  |
|  |  |

IT Technical Architectural Specification

for Travel Simple Site

**Document Revisions**



**WORKING GROUP**

List of persons involved in drawing up the document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Git Username** | **Function** | **Date** | **Signature** |
| Luca MARTINELLI | Martins96 | IT Architect / Developer | 24/03/2022 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

REFERENCE DOCUMENTS

|  |  |
| --- | --- |
| **Code – Version** | **Document Title** |
|  |  |
|  |  |
|  |  |

TERMS AND DEFINITIONS

|  |  |
| --- | --- |
| **BL** | Business Logic |
| **BE** | Back-End |
| **FE** | Front-End |
| **DB** | Database |
| **WS** | Web Service |
| **OC** | Openshift Client (CLI) |
| **S2I** | Source to Image |
|  |  |
|  |  |

**TABLE OF CONTENTS**

[1. Purpose 5](#_Toc100936681)

[2. Validity 5](#_Toc100936682)

[3. License 5](#_Toc100936683)

[4. Architectural Diagram 6](#_Toc100936684)

[5. Component-Module Matrix 7](#_Toc100936685)

[6. Interfaces 8](#_Toc100936686)

[7. Logical/Physical Data Model 9](#_Toc100936687)

[8. Description of Components 11](#_Toc100936688)

[8.1 Introduction 11](#_Toc100936689)

[8.2 Module architecture 11](#_Toc100936690)

[8.2.1 Maven 11](#_Toc100936691)

[8.2.2 Code tree 11](#_Toc100936692)

[8.2.3 Openshift installation scripts 11](#_Toc100936693)

[8.2.4 Database access 12](#_Toc100936694)

[8.2.5 Database image creation 12](#_Toc100936695)

[8.2.6 Properties configuration 12](#_Toc100936696)

[8.3 Module description 13](#_Toc100936697)

[8.3.1 Home Module 13](#_Toc100936698)

[8.3.1.1 Use cases 13](#_Toc100936699)

[8.3.1.2 Structure 13](#_Toc100936700)

[8.3.1.3 Java backend code 15](#_Toc100936701)

[8.3.1.4 Angular frontend code 17](#_Toc100936702)

[8.3.1.5 Building phase 18](#_Toc100936703)

[9. Nonfunctional Aspects 20](#_Toc100936704)

# Purpose

The purpose of this document is to describe the technical implementation for this Web Site, a *travel organization* expose some hotels in this site and the users can register an account and booking hotels.

# Validity

This document has a validity only for track the architecture and the technology used, all code developed are made for study / curiosity / fun, and this is an open source and free project under GPL license.

**Conventions:**

Within this document:

* in pink draft/work-in-progress contribution to be definitively defined in a later release
* in turquoise TBD/TBC information already required.

# License

This project is free and use the GPL open-source license.

# Architectural Diagram

|  |  |
| --- | --- |
|  | Standard implementation module with backend and frontend |
| [IMG HERE] | [Description here] |

# Component-Module Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Component** | **Home** | **Hotel** | **Login** | **Registration** | **User Management** |
| *Front end* | X | X | X | X | X |
| *Back end* | X | X | X | X | X |
| *DB Connection* | X | X | X | X | X |
| *Message Queues* |  |  |  |  |  |
| *Batch execution* |  |  |  |  |  |
| *Mail deamon* |  |  |  | X |  |
|  |  |  |  |  |  |

# Interfaces

All interfaces used in the FE – BE communication are RESTful.

Backend use Quarkus – Resteasy component for expose the endpoints. For convention the data are serialized in JSON object structures.

# Logical/Physical Data Model

The database used is a simple MySQL 8.0.

***Database***:

**Name**: travel\_site

***Tables:***

**Name**: hotels

**Field**:

`HOTEL\_ID` - varchar(*50*) - *NOT NULL AND UNIQUE*

`TITLE` - tinytext

`LONG\_DESCRIPTION` - longtext

`SHORT\_DESCRIPTION` - text

`LOCATION` - varchar(*255*)

`STARS` - tinyint(*1*) - *DEFAULT '1'* - *'From 1 to 5'*

`PREVIEW\_IMAGE` - varchar(*256*) - *DEFAULT NULL*

**Primary Key:**

HOTEL\_ID

\_\_\_\_\_\_\_\_\_\_\_

**Name**: hotels\_images

**Field**:

`HOTEL\_ID` - varchar(*50*) - *NOT NULL*

`IMAGE\_SRC` - varchar(*256*) - *DEFAULT NULL*

**Primary Key:**

*None*

\_\_\_\_\_\_\_\_\_\_\_

**Name**: offer\_products

**Field**:

`HOTEL\_ID` - varchar(*50*) - *NOT NULL*,

`BASIC\_PRICE` - double - *DEFAULT NULL*,

`OFFER\_PRICE` - double - *DEFAULT NULL*

**Primary Key:**

*None*

\_\_\_\_\_\_\_\_\_\_\_

**Name**: rooms

**Field**:

`HOTEL\_ID` - varchar(*50*) - *DEFAULT NULL*

**Primary Key:**

*None*

\_\_\_\_\_\_\_\_\_\_\_

**Name**: showcase\_products

**Field**:

`HOTEL\_ID` - varchar(*50*) - *NOT NULL AND UNIQUE*

`BASIC\_PRICE` - double - *DEFAULT NULL*

**Primary Key:**

HOTEL\_ID

\_\_\_\_\_\_\_\_\_\_\_

**Name**: users

**Field**:

`USERNAME` - varchar(*100*) - *NOT NULL AND UNIQUE*,

`PASSWORD` - varchar(*512*) - *NOT NULL*,

`GROUPS` - varchar(*512*) - *DEFAULT NULL*,

`NAME` - varchar(50) - *DEFAULT NULL*,

`SURNAME` - varchar(50) - *DEFAULT NULL*,

`EMAIL` - varchar(128) - *DEFAULT NULL*,

`BIRTHDATE` - varchar(10) - *DEFAULT NULL*,

`COUNTRY` - varchar(64) - *DEFAULT NULL*,

`AVATAR` - mediumtext

**Primary Key:**

HOTEL\_ID

\_\_\_\_\_\_\_\_\_\_\_

# Description of Components

## Introduction

The present document represents a guide, from the architectural point of view, in order to perform changes and to introduce new structures involving the GitHub project. Nevertheless, the AS document does not provide the following information:

* Java syntax
* How to use design patterns
* Any explanation about the logic of the frameworks used
* Any explanation about the external usage (i.e. Openshift capabilities, Docker concepts, ecc…)

Names of Java classes, interfaces methods, jsp name and all other resources are always suggested and not mandatory.

## Module architecture

### Maven

All modules use Maven as build engine; definitions of each module are described inside the *pom.xml* file. The core of module framework uses Quarkus, a subatomic supersonic cloud ready framework by Red Hat. All components are loaded in the dependencies based on “*quarkus-universe-bom*” platform artifact-id.

Maven model version is actual set to *4.0.0*.

The root for this project group name is: ***com.lucamartinelli.app.travelsite***.

In addition, the Maven’s pom include: *quarkus-maven-plugin* for manage Quarkus build, environment and runtime; and *maven-surefire-plugin* for manage unit tests.

### Code tree

The project’s structure is divided in multiple part, we have the test inside the ***src/test*** folder, where we can find the test code inside ***java*** folder and the resources like properties file inside the ***resources*** folder. The main code is placed inside the directory ***src/main***, where the folder with name ***resources*** contains the configuration files, the folder ***docker*** have the Dockerfiles generated by Quarkus project generator, the ***angular*** folder contains the angular project for the FE part that will populate the static exposed file with angular build (*ng build*) or (*ng build -c production*), and finally, ***java*** folder carries all java source codes for BE logics and the REST exposition.

### Openshift installation scripts

For Quarkus projects, the root folder presents a script file (shell sh) with the instructions for compile, package and deploy on Openshift platform, all you need is execute the authentication login inside your Openshift platform and start the sh script.

**Attention**: in order to execute this script, you must have NodeJS installed on your machine (the Angular build trigger is available only via NodeJS Package Manager command - *npm*).

The Quarkus S2I technology for Openshift deployment is based on quarkus-openshift plugin.

|  |
| --- |
| **<dependency>**  **<groupId>io.quarkus</groupId>**  **<artifactId>quarkus-openshift</artifactId>**  **</dependency>** |

This is defined inside the *pom.xml* file and inherit the version for Quarkus BOM.

### Database access

The database access is managed by Agroal Datasource imported by Quarkus BOM. This allows to configure the connection pool ad close the idle connections without too much code effort.

The configurations can be found inside the application.properties file, the root property key is *quarkus.datasource.* and the credentials are loaded from environment variables, for this reason the module required a secrets map (if you run it in an Openshift environment) or a configmap/env variable startup configurated for each module that use the DB connections. The Database address will be loaded automatically in an Openshift Environment through the Service Environment Variables generated from the system after the DB services’ s creation.

### Database image creation

This project use MySQL as Database, and provides the information and commands for create, load a startup config, and run an image of this database. All information and command are contained inside the db-scripts project. With this project you can build an image in Openshift and Docker; and, also, push it in a registry like quay.io.

As is, the yaml file contains the instructions for build the image using the Dockerfile placed in the Git Repository .

### Properties configuration

All configurations (configmaps and secrets) are included inside the config-doc project (except the SQL scripts used for populates the DB with starting data).

The command for create the configs from file on Openshift is:

|  |
| --- |
| oc create secret generic *name-of-secret* --from-file=*secret-file-name.yml* |
| oc create configmap *name-of-configmap* --from-file=*configmap-file-name.yml* |

The italic grey color indicates the placeholder that will be changed with the configmap/secrets name as described in the README.md files.

**Attention**: the --from-file parameter does not accept only yml files, but you can load any simple textual file you want.

**BE CAREFULL**: the config map contains actual name for a specific project and the routes generated by Quarkus should be different, please check and update the routes name based on your environment.

### Angular rest endpoint resolver

In all Angular projects there are a logic in order to resolve automatically the endpoint in DEV environment or in PROD. In all module are present classes with name: rest-service.ts and rest-service-env-url.ts for manage the REST calls

Endpoints are loaded in standard Angular way, using the Environment class replaced by *ng build* command.

#### Endpoints class

For test using *ng serve* running command, the endpoints will be loaded from class: environment.ts

For running in production mode (files built with *ng build* command), the endpoint will be loaded from class: environment.prod.ts  
This class start a call for each endpoint requested by an app name, the BE layer will send a correct endpoint configured in the configmap loaded on Openshift, this allow the flow to change dynamically and to be configurable. For avoid multiple and redundant calls for endpoints, the class *env-singleton.ts* will store values after the first call and it will reduce the calls and time in execution, in less words it will caching the endpoints.

#### rest-service.ts

Used only in dev runtime, loaded automatically by Angular.

In this class there are multiple methods for call via REST the external endpoints; the RAW call means the response will be full and it will contain: headers, HTTP status code and payload. The response type will be: *HttpResponse<?>*

All methods will return an Observable class, in order to manage synchronous or asynchronous directly in the code.

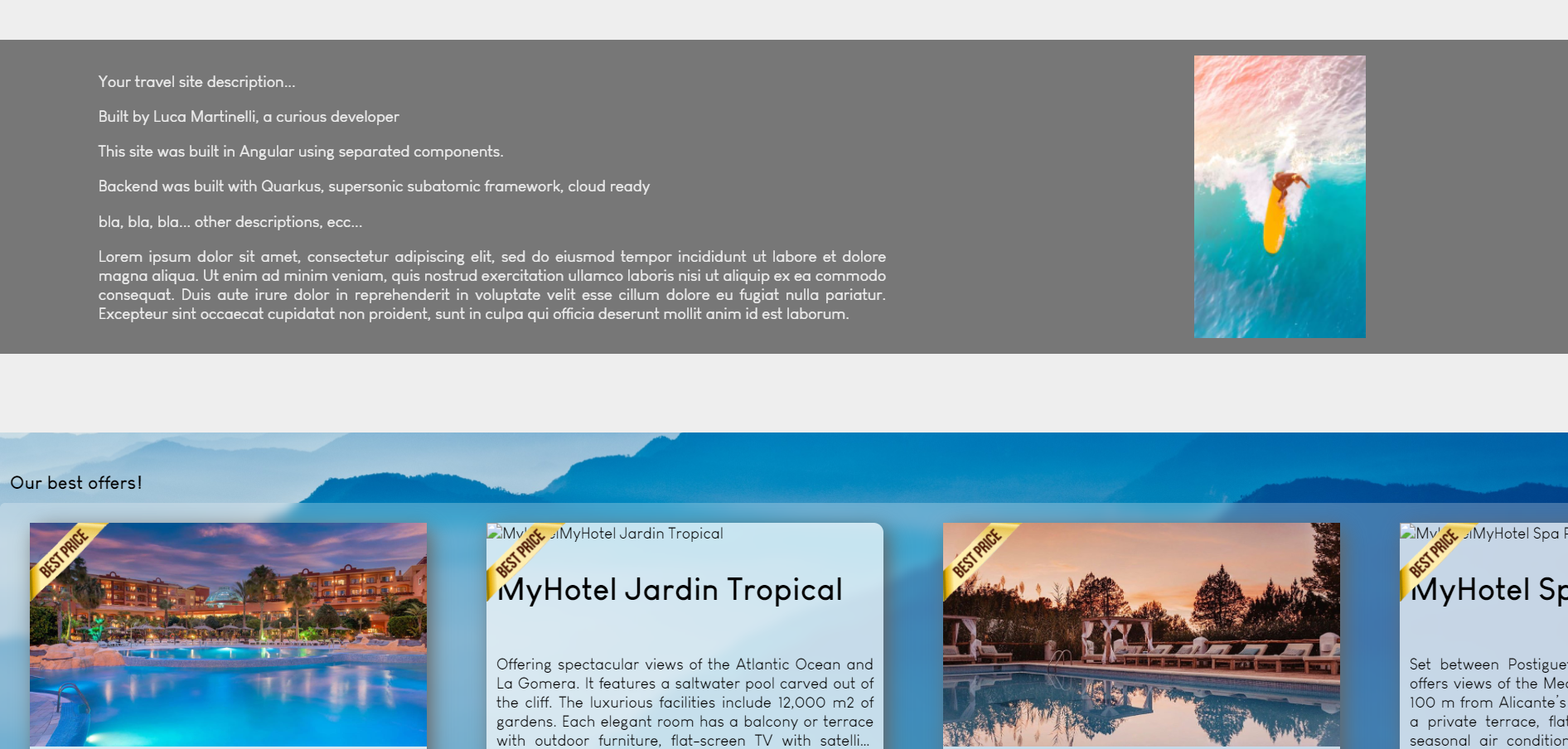
#### environment.prod.ts

Used on prod runtime with file built by *ng build* process , loaded automatically by Angular.

This class execute calls to BE layer and retrieve the endpoints value, additionally it will store in the singleton cache class.

## Module description

### Home Module





#### Use cases

This module contains the homepage of the travel site, it will show the offers in promotions and the showcase.

##### UC 1 – User not logged navigate

A simple homepage with offers and showcase will be opened. Top navigation bar shows the logo and the links for Login page and Registration page.

##### UC 2 – Logged user navigate

The home page will the same of the not logged users, but the navigation bar will shows the logged user’s avatar, login and registration links will be not visible.

#### Structure

##### Scripts

The scripts included in this module are:

* **local-build.sh** – build Angular code and start a developer Quarkus server (default port is 8080, as indicated in the application.properties)
* **openshift-build.sh** – this script compiles the Angular code and start an Openshift build/deploy with the local code, this works if you’re logged in with *OC* command line in the shell.
* **Mvnw / mvnw.bat** – Auto-generated by Maven, this is a wrapper for execute Maven without a physical installation on machine.

##### Folders

##### The following is the folder structure specification:

|  |  |
| --- | --- |
| 🌳 \home-quarkus-travel-simple-site | Root project |
| ├── 📁 src | Include source codes |
| | ├── 📁 main | Code for main logic |
| | | ├── 📁 angular | Angular part for FE layer |
| | | ├── 📁 docker | Include the Dockerfiles for Docker build |
| | | ├── 📁 java | Contains the Java classes with the implementations |
| | | └── 📁 resources | Configuration files, application.properties or config property files |
| | └── 📁 test | Contains test classes for Junit |
| | ├── 📁 java | Java code for Junit tests |
| | └── 📁 resources | Configuration used only in Junit tests |
| └── 📁 target | Binary built by Maven, this is a temp folder |

##### Application properties configuration

|  |  |  |
| --- | --- | --- |
| Property key | Default value | Description |
| quarkus.application.name | home | Name of the application (will be used as name on Openshift) |
| quarkus.application.version | 1.0.0 | Version (only internal) |
| %dev.quarkus.http.port | 8080 | Dev default port (Dev profile is indicated using “%dev” suffix) |
|  |  |  |
| quarkus.http.cors | true | Enable Server Cors for REST APIs |
| quarkus.http.cors.origins | \* | Define Cors Origin |
| quarkus.http.cors.headers | accept, authorization, content-type, x-requested-with | Define Cors headers |
| quarkus.http.cors.methods | GET, OPTIONS | REST Method list for Cors enabled |
|  |  |  |
| quarkus.log.file.format | %d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n | Log4j pattern prefix per-row in log file |
| %prod.quarkus.log.category."com.lucamartinelli.app.travelsite.home".level | INFO | Log level for production environment |
| %dev.quarkus.log.category."com.lucamartinelli.app.travelsite.home".level | DEBUG | Log level for developer envirnoment |
|  |  |  |
| %dev.quarkus.datasource.db-kind | mysql | (Dev) type of DB used |
| %dev.quarkus.datasource.username | travel\_user | (Dev) username used in DB |
| %dev.quarkus.datasource.password | 27BEYOBob7 | (Dev) password for DB – Auto generated |
| %dev.quarkus.datasource.jdbc.url | jdbc:mysql://localhost:3306/travel\_site | (Dev) database JDBC url |
| %dev.quarkus.datasource.jdbc.max-size | 5 | (Dev) Number of connection in the datasource |
|  |  |  |
| quarkus.datasource.db-kind | mysql | (Default) type of DB used – Used in production too |
| quarkus.datasource.username | \${datasource.username} | (Default) username used in DB |
| quarkus.datasource.password | \${datasource.password} | (Default) password for DB – Auto generated |
| quarkus.datasource.jdbc.url | jdbc:mysql://\${TRAVELSITE\_DB\_SERVICE\_HOST}:\${TRAVELSITE\_DB\_SERVICE\_PORT}/travel\_site | (Default) database JDBC URL |
| quarkus.datasource.jdbc.max-size | 15 | (Default) Number of connections in the datasource |
| quarkus.datasource.health.enabled | false | Check datasource connection (Openshift health checks) |
|  |  |  |
| quarkus.openshift.route.expose | true | Expose the application Openshift Routes |
| quarkus.openshift.name | home | Name used on the label/pods on Openshift |
| quarkus.kubernetes.deploy | true | Trigger the deploy on Openshift on module build |
| quarkus.container-image.build | true | Create the Docker Container image |
| quarkus.kubernetes-client.trust-certs | true | Trust un-secure certificates |
| quarkus.openshift.env.secrets | db-secrets | Include the secrets in application, managed by Openshift |
| quarkus.openshift.env.configmaps | endpoints-config | Add key/value configurations in application |
| quarkus.openshift.replicas | 2 | Number of pods in the deployment |
|  |  |  |
| quarkus.container-image.group | com.lucamartinelli.travelsite | Image group used in Docker image tag |
| quarkus.container-image.name | home | Image name used in Docker image tag |
| quarkus.container-image.tag | 1.0 | Image version used in Docker image tag |

An additional file with name **microprofile-config.properties** is included in the configurations; this contains the mode for the data access:

home.mode=DB

This means the data will be loaded from Database, you can use the value “*IN\_MEMORY*” in order to load the data from memory mocked code, very useful for testing in a virgin environment.

#### Java backend code

##### Exposed methods

* EndpointUrl.java, this class expose a simple way to get the URLs for navigate through services in the portal

|  |
| --- |
| 🏠 /endpoint *🡪 Root path*  ├── 👣 /home *🡪 Home endpoint path*  ├── 👣 /hotel *🡪 Hotel endpoint path*  ├── 👣 /login *🡪 Login endpoint path*  ├── 👣 /user-management *🡪 User management endpoint path*  └── 👣 /registration *🡪 Registration endpoint path* |

* Home.java, this class expose the microservices related to Home logic implementation

|  |
| --- |
| 🏠 /home *🡪 Root path*  ├── 👣 /showcase *🡪 Showcase endpoint path*  └── 👣 /offers *🡪 Offers endpoint path* |

##### DAO selector

The property “home.mode” define which DAO will be used for data loading, if no configuration value was provided, the default value is *IN\_MEMORY* and the data will be loaded from mocked hardcoded information. The database mode (*DB*) will load the information via Agroal datasource, the configuration must be provided with configmap/application.properties file.

##### Home – showcase

###### Input

No one is required, the method is a simple GET using for example:

<http://localhost:8080/home/showcase>

No authentication is required.

###### Output

The output will be a JSON (on XML if specified in the Accept header) that will contains these information:

* **Showcase (Object)**

|  |  |  |  |
| --- | --- | --- | --- |
| Field name | Type | Nullable | Description |
| products | Array of *ShowcaseProduct* | N | A list of Product Object, should be empty if no data was present |

* **ShowcaseProduct (Object)**

|  |  |  |  |
| --- | --- | --- | --- |
| Field name | Type | Nullable | Description |
| hotelID | String | N | Unique ID |
| title | String | Y | Name of the Hotel / Resort |
| location | String | Y | Geographical location, human readable |
| basicPrice | Number (Double) | Y | Basic price in Euro |
| description | String | Y | Description about the hotel |
| image | String | Y | Image path URL |

###### Main logic

The logic split the code on two branches, IN\_MEMORY and DB. In memory mode simply load mocked data, DB use SQL queries for retrieve data from DB, use additional fault tolerance policy, like fallback in case of error inside the method, and the retry policy to retry in case of temporary downtime.

Another implementation present in this module is a Liveness / Readiness health checks, the class *HealthHome* implements the ***HealthCheck*** interface by microprofile framework, this allow to expose a health service to indicate the application status.

##### Home – offers

###### Input

No one is required, the method is a simple GET using for example:

<http://localhost:8080/home/offers>

No authentication is required.

###### Output

The output will be a JSON (on XML if specified in the Accept header) that will contains these information:

* **Showcase (Object)**

|  |  |  |  |
| --- | --- | --- | --- |
| Field name | Type | Nullable | Description |
| products | Array of *OfferProduct* | N | A list of Product Object, should be empty if no data was present |

* **OfferProduct (Object)**

|  |  |  |  |
| --- | --- | --- | --- |
| Field name | Type | Nullable | Description |
| *hotelID* | String | N | Unique ID |
| *title* | String | Y | Name of the Hotel / Resort |
| *location* | String | Y | Geographical location, human readable |
| *basicPrice* | Number (Double) | Y | Basic price in Euro |
| *description* | String | Y | Description about the hotel |
| *image* | String | Y | Image path URL |
| offerPrice | Number (Double) | Y | Special offer price |

###### NOTE: In italic the inherit by ShowcaseProduct, the OfferProduct extends the ShowcaseProduct class.

###### Main logic

Ref. p. 8.3.1.3.3.3, same logic

#### Angular frontend code

Angular module is component separated in order to maintain the frontend layer smallest and readable. Take a look for each components.

##### Common utils classes

* **CookieManager**, create, delete and manage cookies
* **RestService**, an utils class to make REST call using angular common http library

##### App component

A standard and empty component, define only the title and the other components

##### Toolbar component

A simple navigation bar with user managing page (such login, registration and home page). The endpoint will recovered by backend REST method, the endpoints are loaded by utils class RestServiceEnvUrl. In dev environment these attributes are static, instead the production environment consumes the endpoints expose from backend.

Additional, the toolbar define a check for JWT token validation, the JWT token is stored in a cookie with name “user.jwt”, the toolbar execute a call to the login endpoint in order to validate the token, the endpoint is: *LoginHost* + "/validate". The method will return all information about the logged user (if any).

##### Home-head component

This component shows the first part of home page, the upper part is a random background image with the site title-

##### Home-description component

This part is a simple static textual description about the site and put a simple image on right side, not much logic here.

##### Showcase component

On start of this component, the code executes a REST call to home backend service and retrieve the showcase products, after that it will show them in multiple boxes with a presentation image.

##### Offers component

It is same of the showcase component, but it will show the offers and the container box is scrollable.

#### Building phase

##### Frontend

Stating from the Frontend, the build process required a

“*npm install*”

in order to add dependencies libraries, after that the

“*ng build --configuration production*”

will compile the angular typescript files and place the generated ones in the **/resources/META-INF/resources** folder. The configurations about Angular build are defined inside the *angular.json* file. These steps can be executed automatically with “openshift-build.sh” script.

The folder **/resources/META-INF/resources** will expose static web files and is managed by Quarkus server.

##### Backend

With the Angular file generated, the next step is building (and deploying eventually) the backend code. Maven can build the code with command

“*mvn clean package*”

this will generate a runnable JAR file, but is little useless for our scope, the real build and deploy is triggered by the command

“*mvn clean package -Dquarkus.kubernetes.deploy=true -Dquarkus.kubernetes.deployment-target=openshift*”

This will execute a Quarkus Openshift plugin that will build the code and start a deploy on Openshift cluster. You can add other parameters for other options, like: *-Dquarkus.openshift.expose=true* for expose the application to the Web local URL and access to the Browser, or use *-DskipTests* for skip some unit test.

And locally? If you do not have an Openshift you can run Quarkus directly using Maven command:

“*mvn quarkus:dev*”

With this command a Quarkus server will start on localhost at the port defined inside the application.properties file.

Docker image? If you want building a Docker image based on this application, you can run:

“*mvn package -Dquarkus.container-image.build=true*”

And you can configure the image details (like group, image-name and tag version) in the application.properties using these configurations:

|  |
| --- |
| quarkus.container-image.group=your.group.name  quarkus.container-image.name=image-name  quarkus.container-image.tag=version.number |

Note:

The parameters passed on command line with ‘-D’ are configurable on application.properties file (except for “-DskipTests”)

If you do not have a Maven installation on your machine you can execute the Maven wrapper “mvnw” included in each module.

### Login Module

#### Use cases

Simple login page, here the page asking to user the password and username.

##### UC 1 – User not pending, valid credentials

Already registered user inserts valid credentials, the user is not in pending status, the pending is defined in the DB table “user\_credentials”, inside the field “PENDING”.

System generate and sign a new JWT token and setting it on a browser cookie called “user.jwt”.

User will be redirected to homepage.

##### UC 2 – User in pending or not, wrong credentials

As UC 1, but the user inserted wrong credentials. The page shows a courtesy message indicate the wrong credentials.

No redirects will be done.

##### UC 3 – User in pending, valid credentials

System will show a dedicated page with the disclaimer that indicate a email confirmation required, the JWT will not be generated.

#### Structure

#### Java backend code

#### Angular frontend code

#### Building phase

### Static content server module

This module is a little different from others because this is made with NodeJS.

Static Content Server is a server that will take the static contents (like avatar images) and store it in a folder.

The endpoints are exposed via HTTPS certificate.

#### Technologies

The module uses NodeJS as core, the source code use Typescript as language. Server engine used is Express.

#### Use cases

The module exposes a secure endpoint for upload and store static contents, the security used is a mutual Certificate Authority.

#### Logic details

Starting from “main.ts”, the server load configuration and start at defined port, the “api.ts” define the endpoints and the implantation. The “properties-loader.ts” loads the information configured.

#### Configurations

##### Server arguments

Server required argument for startup server:

**--profile** 🡪 name of profile mode for server runtime (default: ‘prod’)

**--conf-file-path** 🡪 path to the config.json file (default: ‘conf/dev/config.json’)

##### config.json

This is the main config file, the configurations are (default vale can be found in dev config file):

**upload-path** 🡪 path where the uploaded file will be stored (default: "tmp/")

**server-port** 🡪 number of server port (default: “5000”),

**ca-path** 🡪 path to the Certificate Authority file (default: "conf/dev/ca-cert.pem”)

**cert-path** 🡪 path to the server certificate file (default: "conf/dev/server-cert.pem")

**key-path** 🡪 path to the server key certificate file (default: "conf/dev/server-key.pem")

An example of this file can be found inside the conf/dev or conf/prod folders

##### Generate certificate for TLS/CA mutual check

You can generate the certificate on your own following these commands (Openssl required)

* **Creating the Certificate Authority's Certificate and Keys**

Generate a private key for the CA:

|  |
| --- |
| openssl genrsa 2048 > ca-key.pem |

Generate the X509 certificate for the CA:

|  |
| --- |
| openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca-cert.pem |

* **Creating the Server's Certificate and Keys**

Generate the private key and certificate request:

|  |
| --- |
| openssl openssl req -newkey rsa:2048 -nodes -days 365000 -keyout server-key.pem -out server-req.pem |

Generate the X509 certificate for the server:

|  |
| --- |
| openssl openssl x509 -req -days 365000 -set\_serial 01 -in server-req.pem -out server-cert.pem -CA ca-cert.pem -CAkey ca-key.pem |

* **Creating the Client's Certificate and Keys**

Generate the private key and certificate request:

|  |
| --- |
| openssl openssl req -newkey rsa:2048 -nodes -days 365000 -keyout client-key.pem -out client-req.pem |

Generate the X509 certificate for the client:

|  |
| --- |
| openssl x509 -req -days 365000 -set\_serial 01 -in client-req.pem -out client-cert.pem -CA ca-cert.pem -CAkey ca-key.pem |

Ok, done now you generated server, client and CA certificates

#### Running

You can run the server locally using command (node/npm is required)

|  |
| --- |
| npm run dev |

For test the connection you can run curl command:

|  |
| --- |
| curl -XPOST -k --form 'avatar=@avatar.jpg' --form 'userid="id123"' --key dev/client-key.pem --cert dev/client-cert.pem https://localhost:5000/upload/image |

#### Installation

You can choose from manual or configured by YML file.

In both of cases you need to create a configMap and a ImageStream:

Login in in Openshift with console client (OC), create a config map using prod properties:

|  |
| --- |
| oc create configmap static-content-config.json --from-file=conf\prod\static-content-config.json |

Create an ImageStream:

|  |
| --- |
| oc create is static-content-travel-site |

##### Manual installation

Execute the command:

|  |
| --- |
| oc new-app node6~https://github.com/Quarkus-Travel-Site-Martins96/Static-Content-Travel-Site \  --as-deployment-config \  --name=static-content-travelsite \  --strategy=source \  -l app=travelsite |

And expose the service:

|  |
| --- |
| oc expose svc/static-content-travelsite |

##### Configured file for installation

You can use the YAML file for start the installation using more simplified commands, from a logged in CLI.

|  |
| --- |
| oc create -f build-config-openshift.yml |

Now you can start the build to create the image:

|  |
| --- |
| oc start-build static-content-travel-site |

Create new app from Image built:

|  |
| --- |
| oc new-app static-content-travel-site --name=static-content-travelsite |

Finally expose the service

|  |
| --- |
| oc expose svc/static-content-travelsite |

# Nonfunctional Aspects