# Chapter 3.

# MAIN APPLICATION AND USER PORTAL DESIGN

## Description of architecture in process of development.

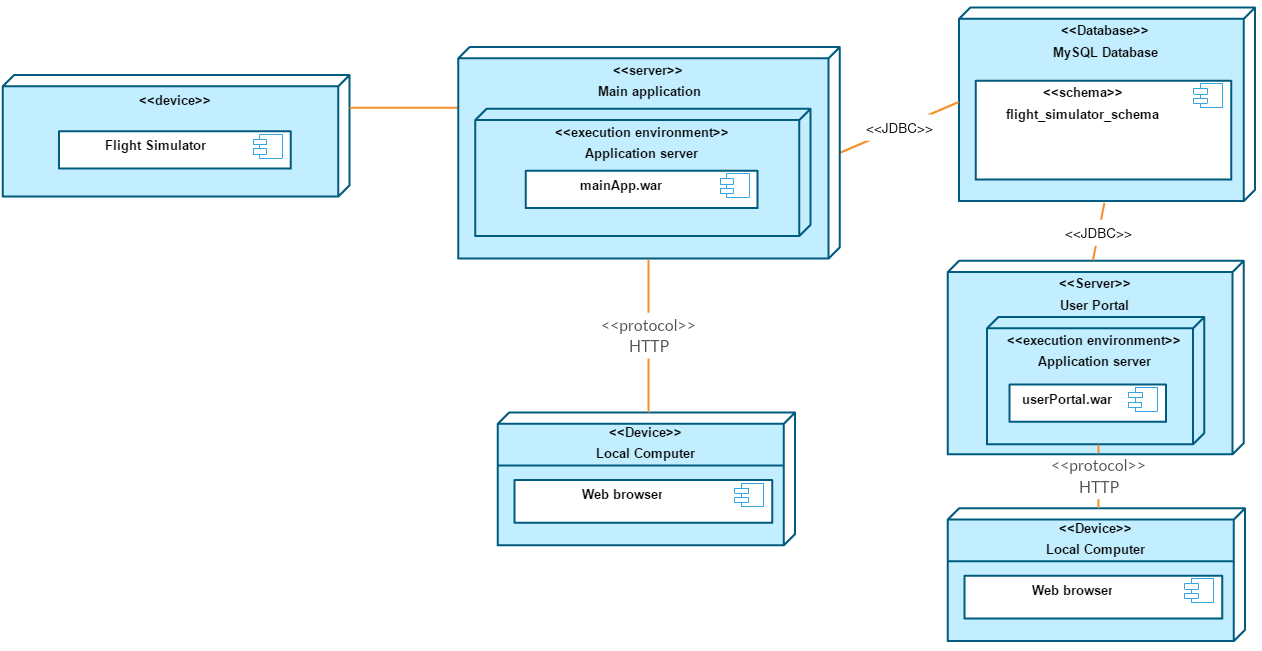


Fig.3.2. Deployment diagram of the system

## Database design.

After investigation were ensured, that database should store users’ accounts, plain models names, emergency situation, metrics for each emergency and records from flight simulation. Records have to have relative connection with plain model, user names and situations. Metrics have relation connection with situations. User can have many records. Record can have only one plain model and emergency in it. Emergency can have many metrics.

From the point of view user portal, we need to perform some security configurations on database connection. First of all we need to set up read only connection with creation specific type of user with limited rights. Moreover, all interactions between front-end and back-end will be limited. Only gathering data will be available.

An entity–relationship model (ER model) describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types.



Fig.3.3. ER model of database.

## Description of packages of main application.

A Java package organizes Java classes into namespaces, providing a unique namespace for each type it contains. Classes in the same package can access each other's package-private and protected members. Java packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually.n general, a package can contain the following kinds of types: classes, interfaces, enumerations, and annotation types.

A package allows a developer to group classes (and interfaces) together. These classes will all be related in some way – they might all have to do with a specific application or perform a specific set of tasks. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. In case of main application of system will be created Package will consist Users entity, Plain\_model entity, Metric entity, Records entity and one enum to store roles in String type.

Class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

### Entity package.

Entity – is a POJO-class, connected with database with help of annotation @Entity or with XML. This kind of class is required to have current requirements:

* 1. Must have empty constructor.
  2. Cannot be inner class, interface or enum.
  3. Cannot be final and contain final fields.
  4. Must contain @Id field.

Entity can:

* 1. Contain non-empty constructors.
  2. Inherit or be inherited.
  3. Contain other methods and implement interfaces.

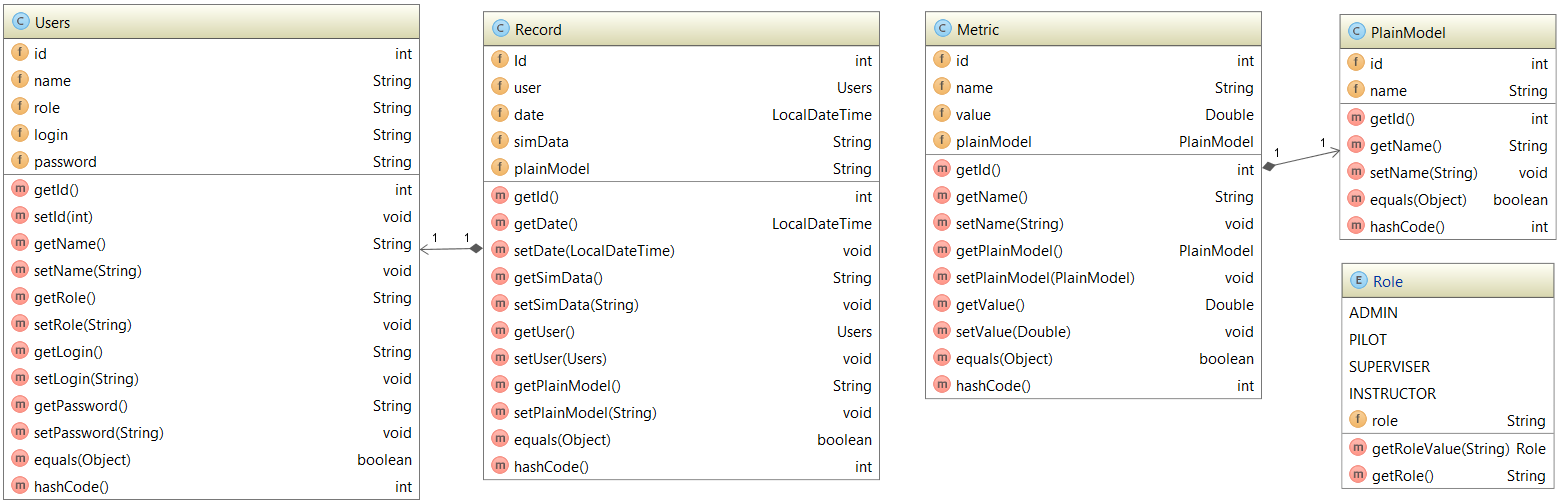
Entities can be related with each other with one-to-one, one-to-many and many-to-many relations (Fig.3.4.).

Fig.3.4. Class diagram for entity package.

For user portal user, records and plain models entities will be available (Fig.3.5.).

### “DAOImplementation” package.

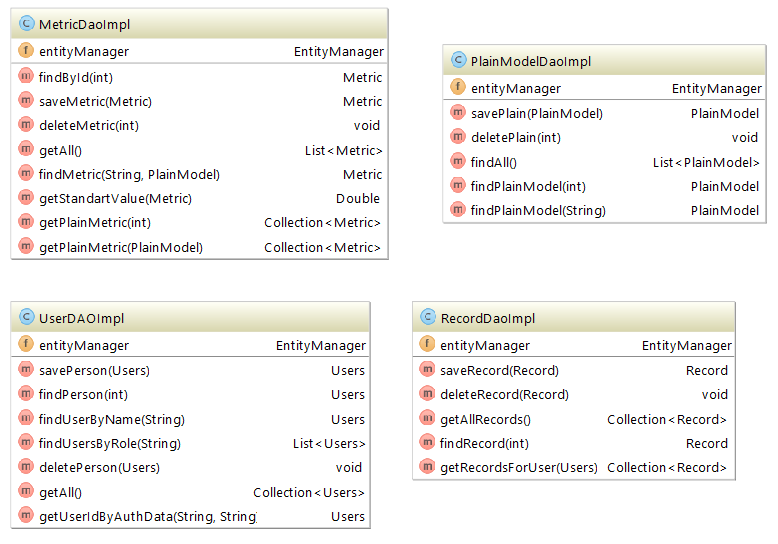
Fig.3.5. Class diagram for entity package in User portal.

Access to data varies depending on the source of the data. Access to persistent storage, such as to a database, varies greatly depending on the type of storage (relational databases, object-oriented databases, flat files, and so forth) and the vendor implementation.

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

* Data Access Object Interface - This interface defines the standard operations to be performed on a model object(s).
* Data Access Object concrete class - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
* Model Object or Value Object - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

Instead of commonly used Model object the Rest Server classes will use DAO implementation classes.

All functionality of DAO is described in functional requirements (Fig.3.6.). The needed methods involves functionality for adding, editing, removing users, managing records, plains and metrics. Nevertheless, it should involve configuration parameters for database connection that will ensure us that connection is protected.

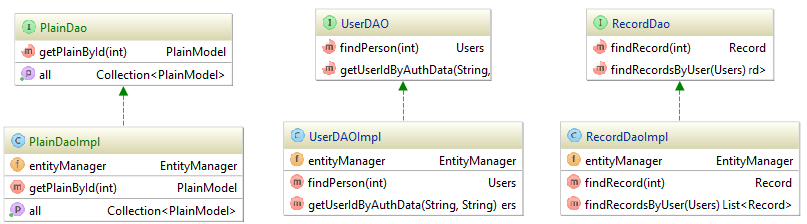
User portal DAO packages represent small functionality and methods for gathering data like: getAllPlainModel(), getUserByAuthData() and getRecordsByUser() (Fig.3.7.).

Fig.3.6. Class diagram for implementation classes of DAOs.

Fig.3.7. Class diagram for DAO implementation classes in User portal.

### “Rest” package.

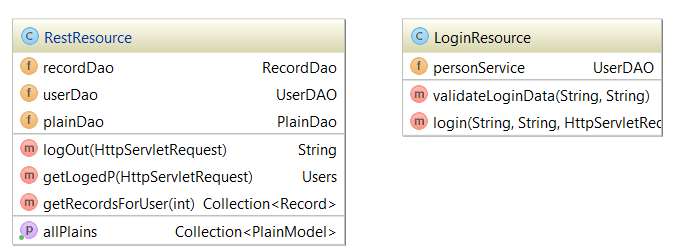
RESTful web services are built to work best on the Web (Fig.3.8,Fig.3.9.). Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

Fig.3.8. Rest package for User portal application.

Using REST services has really struck though during the last couple of years. They often act as a “public API” for third party solutions like mobile apps or as a “persistency layer” for client side web apps. Java is probably the most common platform for providing REST services, but several Java applications need to consume them as well. Moreover, it might even be that REST services consume other REST services for providing their data.

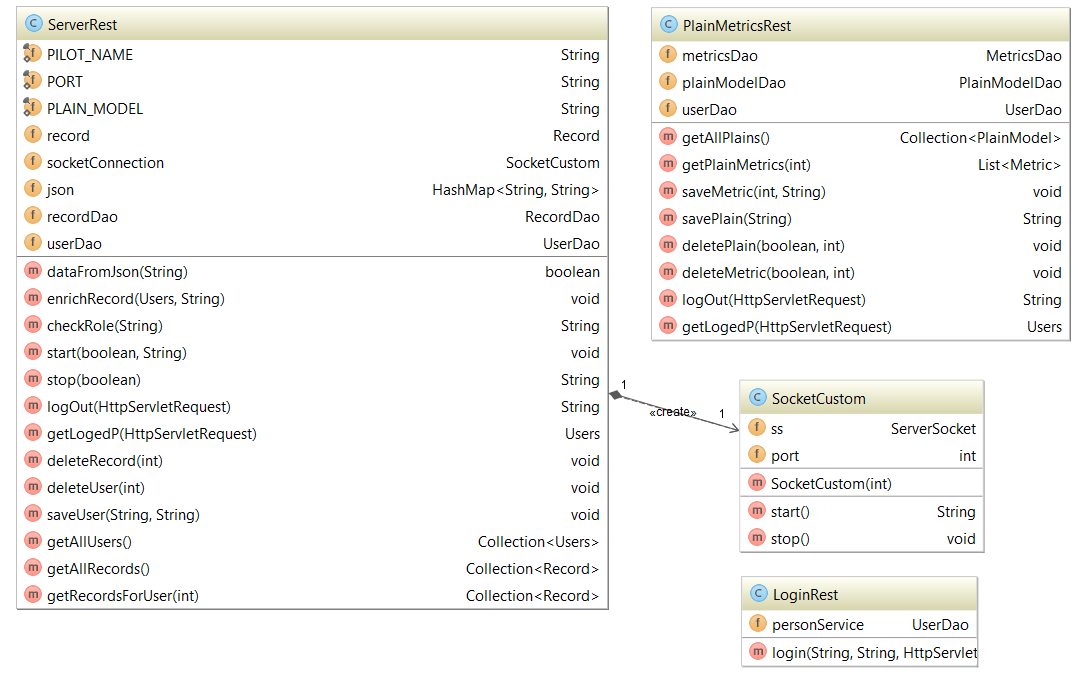
The following principles encourage RESTful applications to be simple, lightweight, and fast:

Fig.3.9. Class diagram for rest package with socket listener.

* Resource identification through URI: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients.
* Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE.
* Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.
* Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained.

Root resource classes are POJOs that are either annotated with @Path or have at least one method annotated with @Path or a request method designator, such as @GET, @PUT, @POST, or @DELETE. Resource methods are methods of a resource class annotated with a request method designator. This section explains how to use JAX-RS to annotate Java classes to create RESTful web services.

## 3.4 Graphical user interface design.

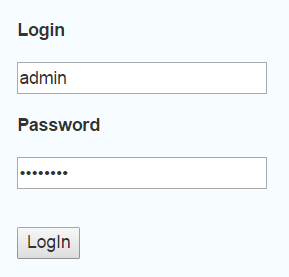
Initial page of main application is login page. With granted credentials user can sign up with different rights. The login page is represented on Fig.3.10.

Fig.3.10. Login page.

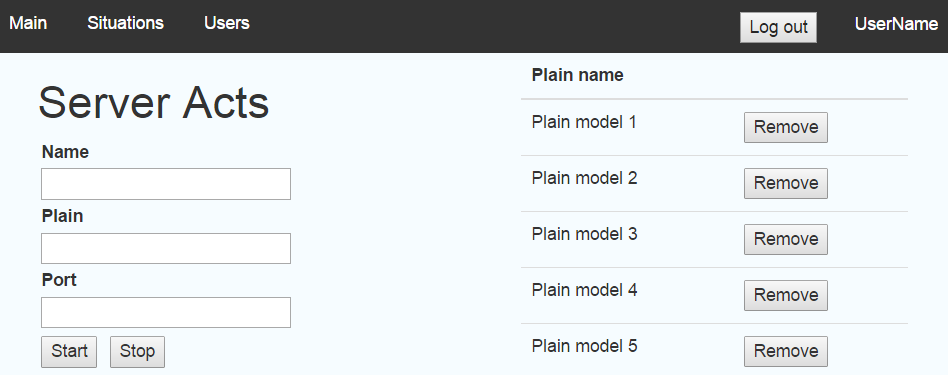
After that user will be relocated on main page with listening process initiation configuration and start-stop buttons. It is necessary to input name of pilot, plain model and port for which servler will be listening simulator data input. On the right side will be visualized list of plains that are represented in trainig center (Fig.3.11.). In angularjs if we want to bind data to tables and implement functionalities like sorting, paging and filtering it’s better to use ng-table module.

Fig.3.11. Main page.

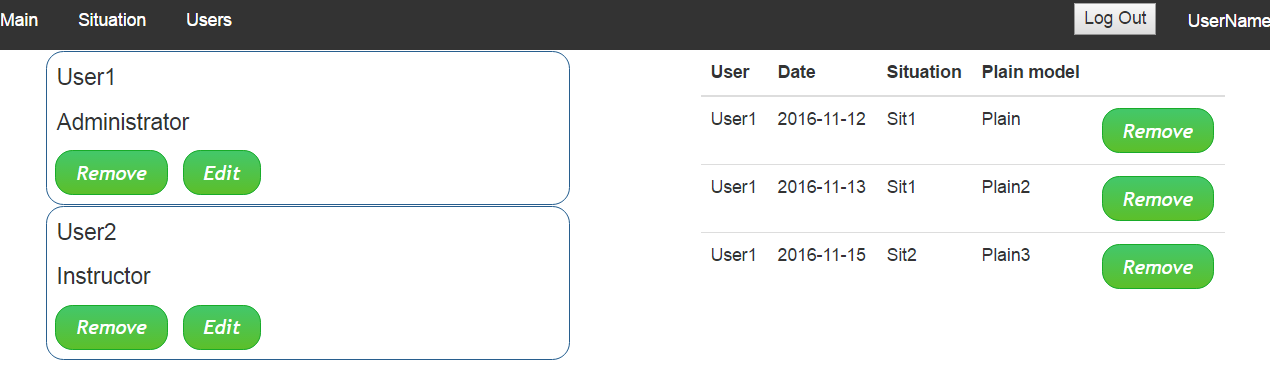
User can brows main application with menu on the top of page. Last item in menu is Users page. Clicking on User item the table with records for this peculiar user will be opened (Fig.3.12.).

Fig.3.12. Users page.

For dispaing tables will be used AngularJS factory ngTableBy using ng-table module in angularjs applications we can achieve functionalities like showing data in table format, sorting, filtering and paging without writing much code.

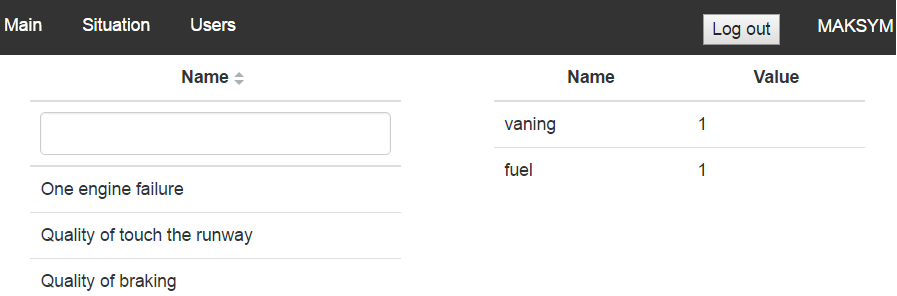
Next page is Situations. The list of emergency situations will be visualized in table view and if emergency item will be clicked the emergency metrics will be viewed alongside it, like was designed according to the requirements specification (Fig.3.13). Situation table provide us filtering these emergency situations by name. this option is not included into functionality of metric table because number of metrics ususaly not more then 6. It still is the normal number to visualize.

Fig.3.13. Plains page.

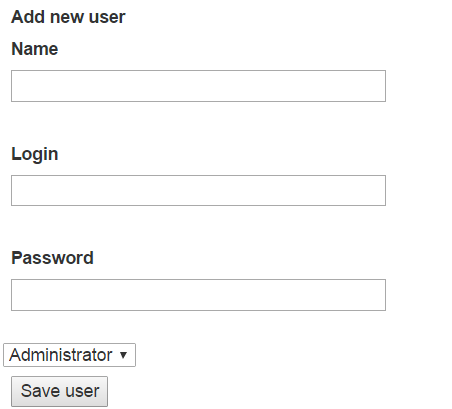
The form for adding new user should contain name, login, password fields and roles dropdown field (Fig.3.14.). Like it was described in requirements only administrator can use this functionality.

Fig.3.14. Add new user form.

The same template for adding emergency situation and metric for peculiar situation. Template for situation is not complicated due to it is necessary to add only one value into name of plain model field. But in case of metric it will be more complicated.

The idea is to bind peculiar metric to situation. That is why the form will be using gathered data from server in view of dropdown list, which will consist list of situation. Value of selection has to be “id”. The search can be proceed with name of situation model on the side of back-end, but it will make more issues then benefits due to mess that it can cause (Fig.3.15.).

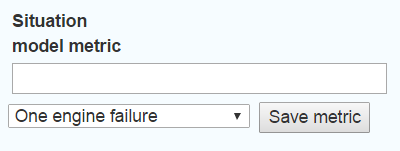


Fig.3.15.Form for adding plain metric

## Conclusion

In this chapter, the idea of architecture has been designed. All functional requirements have been described for the purposes of development. The design fully responds architectural needs that have been underpinned with implementation of described technologies and frameworks. The internal design of all the modules of the proposed architecture was clearly defined. User portal GUI will be similar but with cut functionality because the purposes differ a lot – it will just represent brief information.

In the result, the fully described architecture implementation with implemented design patterns, technologies and frameworks. System was divided into 6 main parts:

* Flight simulator;
* Main application server;
* User portal server;
* Database;
* Local station;
* External station.

Database as the milestone of data interaction has been represented. Dataflow of the entire system were described with all privileges for each kind of user with different roles. As the result were planned for implementation 4 tables: user, record, plain\_model and metric. Moreover, were described all relational interconnection between tables and described for creation entities in development process.

For correct implementation of design patterns were decided dividing application layers into 4 packages:

* Entities;
* DAO interfaces;
* DAO implementations;
* RESTful classes.