# Chapter 4.

# DEVELOPMENT, TESTING AND DEPLOYMENT.

## 4.1 Investigation of scope of product usage.

Creating several small systems that corresponds to cut functionality and has small amount of possibilities is cheap but very ineffective approach. Firstly because of closeness and restrictions of ones. It is necessary to spend small amount of resources for each sub-system and as result we will have several closed applications for working with which is necessary to adapt.

From other side is creation of one big system that will have all functionality and easiness of usage that corresponds to any needs that personnel, staff of training center and common user should use. However, the obstacle is in high price and resource-intensiveness. To develop and deploy such system it is necessary to hire big amount of specialists, from developers and testers to architects, which will implement this system on costly equipment.

Efficiency is main reason of dividing. To orient on a part of target audience and create product only for them, from point of view of efficiency is good approach. In case of flight simulators, it is necessary to have some data calculation application with data management, add web page with GUI, deploy on any computer with medium capabilities in local network, and deploy application.

Create web application that oriented on common a user, a pilot or a company that is interested in training with possibility of checking the results of training. Interconnect these two sub-systems with database data transitions and there will be fully functional system. Each part of it is holistic and working system by itself.

## 4.2 Development.

Development is complicated part of creation of product. In some cases it is necessary to create configuration basis for working on fundament of interconnected

tools for better performance and simplicity of further development. After this configuration, it is potentially to divide development of other parts of application into smaller tasks.

In our case, basis is configuration of application server, object-relational mapping tool and initialization of database with SQL scrips.

## 4.2.1 Configuration of Hibernate.

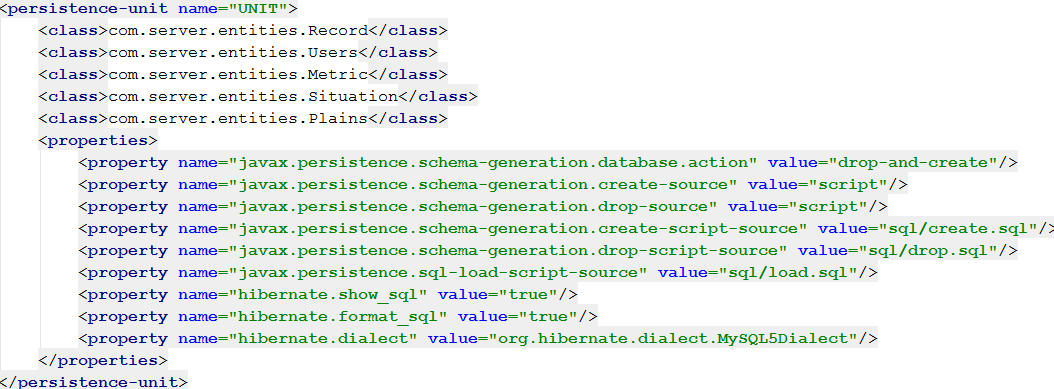
The beginning of development is configuration of object-relational mapping. For this purpose Hibernate ORM is one of the most effective and popular tools. For correct connection and configuration, it is necessary to create configuration file in XML format (Fig.4.1.).

Fig.4.1. percistance.xml file configuration code.

## 4.2.2 Initialization of database.

In case of main application, connection to database will be default, but it is needed to list entities and database drive. Moreover, if during deployment we need to fresh database, we have to set initial SQL queries.

These queries are for creation tables that corresponds to entities one to one and their relations are equal.

## CREATE TABLE IF NOT EXISTS USERS (

## ID INT NOT NULL AUTO\_INCREMENT,

## NAME VARCHAR(100),

## ROLE VARCHAR(15) NOT NULL,

## LOGIN VARCHAR(100) NOT NULL,

## PASSWORD VARCHAR(100) NOT NULL,

## PRIMARY KEY (ID)

## );

## CREATE TABLE IF NOT EXISTS SITUATION (

## ID INT NOT NULL AUTO\_INCREMENT,

## NAME VARCHAR(30),

## DESCRIPTION VARCHAR(500),

## PRIMARY KEY (ID)

## );

## CREATE TABLE IF NOT EXISTS METRICS (

## ID INT NOT NULL AUTO\_INCREMENT,

## NAME VARCHAR(20),

## VALUE DOUBLE,

## SITUATION\_ID INT,

## PRIMARY KEY (ID),

## FOREIGN KEY (SITUATION\_ID) REFERENCES SITUATION (ID)

## );

## CREATE TABLE IF NOT EXISTS PLAINS (

## ID INT NOT NULL AUTO\_INCREMENT,

## NAME VARCHAR(30) UNIQUE,

## PRIMARY KEY (ID)

## );

## CREATE TABLE IF NOT EXISTS RECORDS (

## ID INT NOT NULL AUTO\_INCREMENT,

## USER\_ID INT NOT NULL,

## DATE DATETIME,

## SITUATION\_ID INT NOT NULL,

## PLAIN\_ID INT NOT NULL,

## SIM\_DATA TEXT,

## PRIMARY KEY (ID),

## FOREIGN KEY (USER\_ID) REFERENCES USERS (ID),

## FOREIGN KEY (SITUATION\_ID) REFERENCES SITUATION (ID),

## FOREIGN KEY (PLAIN\_ID) REFERENCES PLAINS (ID)

## );

## 4.3. Configuration of application serve.

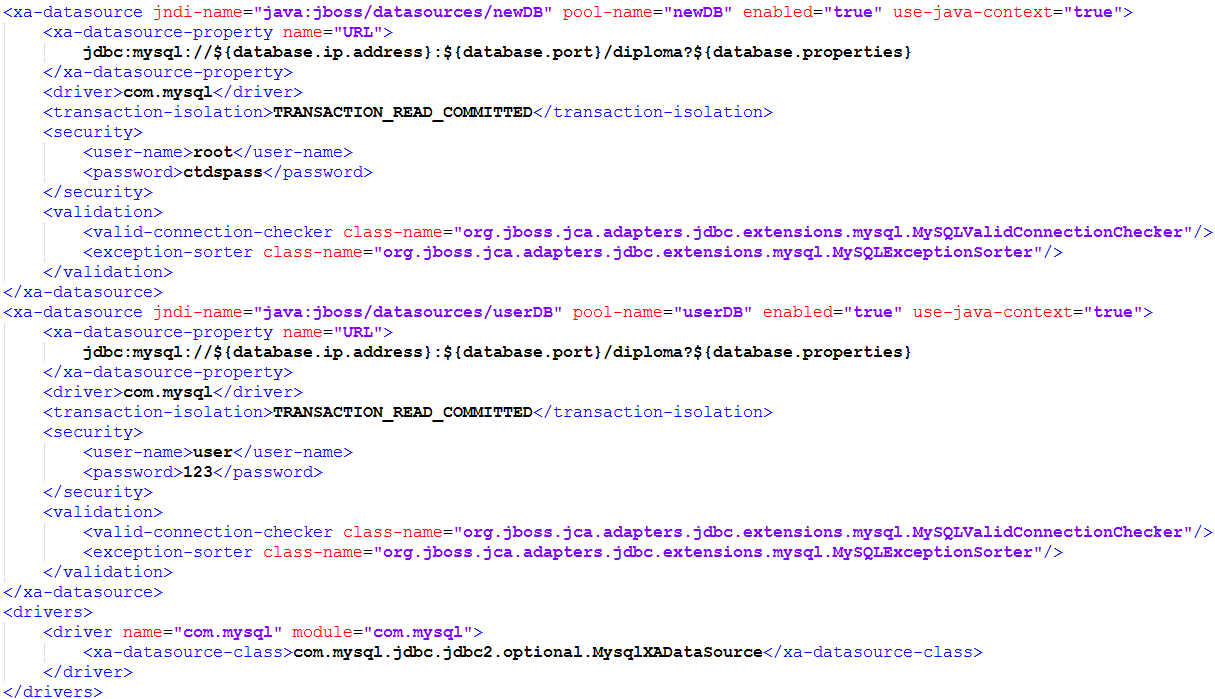
After these steps database will be ready to be connected to WildFly – application server that is needed to be configured to work with database (Fig.4.2.). After these procedures, configuration basis is ready.

Fig.4.2. XML configuration of datasource in application server.

## 4.4 Creation of entities.

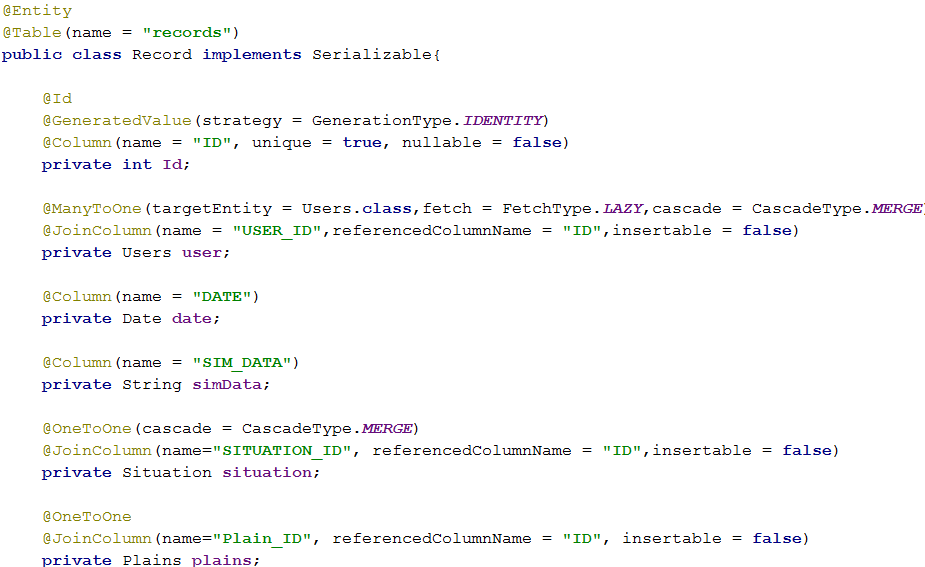
Dividing main application on layers demands creation of entities. As it was described in design part, for it is necessary to create 4 entities and one enum for roles. Hibernate JPA API will provide the main relative interconnection between entities. The most complicated from the side of relations is Record entity (Fig.4.3.). It is connected to Users with Many-to-One relation ,with Plains and Situations entities with One-to-One relation. JPA API provides functionality and configuration that is necessary for interconnection like fetch requesting and cascade persistence. For further representation of development process, working with Records is in most priority and is most complicated.

Fig.4.3. Record entity fields with configured types of transactions.

## 4.5. Data access objects creation and configuration.

DAO classes are simple interfaces that contain signature of methods that will be overrided in implementations, but it has to be assigned with Local annotation (Fig.4.4.). The Local annotation is applied to the session bean class or local business interface to designate a local interface of the bean. When used on the bean class, declares the local business interface(s) for a session bean. When used on an interface, designates that interface as a local business interface. In this case, no value element should be provided.

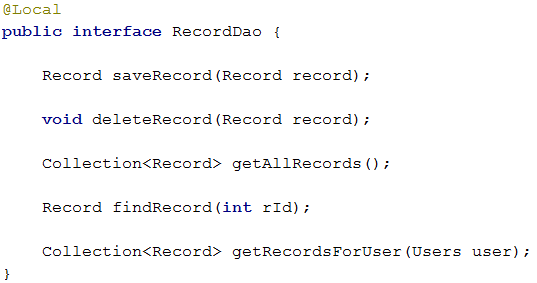
Right after we can create an implementation class for this interface. Up here it is necessary to annotate it with Stateless annotation and configure it like EJB. Enterprise JavaBeans (EJB) is one of several Java APIs for modular construction of enterprise software. EJB is a server-side software component that encapsulates business logic of an application. An EJB web container provides a runtime environment for web related software components, including computer security, Java servlet lifecycle management, transaction processing, and other web services. The EJB specification is a subset of the Java EE specification.

Fig.4.4. RecordsDao interface.

Stateless Session Beans are business objects that do not have state associated with them. However, access to a single bean instance is still limited to only one client at a time, concurrent access to the bean is prohibited. For example, a user of a website clicking on a "keep me informed of future updates" box may trigger a call to an asynchronous method of the session bean to add the user to a list in the company's database (this call is asynchronous because the user does not need to wait to be informed of its success or failure).

EJBs are deployed in an EJB container, typically within an application server. The specification describes how an EJB interacts with its container and how client code interacts with the container/EJB combination.

Moreover, for connection to object-relational mapping it is necessary to create instance of EntityManger with @PersistenceContext and define persistence unit (Fig.4.5.).

Fig.4.5. Configuration of DAO implementation.

## 4.6. RESTful class creation.

After creation and correct configuration DAO implementations, it is possible to create and configure Restful class. For this using annotations for configuration request path is necessary and calling EJB with correct JNDI name must be configured too.

Fig.4.6. RESTful class for processing requests from front end.

Consumes and Produces annotations set type of requests and responses. Currently it is set in type of JSON. Further method for processing requests must be created and configured too using JAX-RS annotations. JAX-RS core APIs enable developers to rapidly build Web applications in Java that are characteristic of the best designed parts of the Web. The API brings in support for designing and implementing Web resources and application that follow principles of REST (Representational State Transfer) architectural style to the Java Platform. In JAX-RS, Java POJOs can be exposed as RESTful Web resources independent of the underlying technology using a high level easy-to-use declarative annotation-based API. Table 4.1 lists some of the Java programming annotations that are defined by JAX-RS, with a brief description of how each is used.

Table 4.1.

User JAX-RS annotations

|  |  |
| --- | --- |
| Annotation | Description |
| @Path | The @Path annotation’s value is a relative URI path indicating where the Java class will be hosted: for example, /helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: /helloworld/{username}. |
| @GET | The @GET annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @POST | The @POST annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @PUT | The @PUT annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |
| @DELETE | The @DELETE annotation is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding. |

User JAX-RS annotations

|  |  |
| --- | --- |
| Annotation | Description |
| @PathParam | The @PathParam annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation. |
| @Consumes | The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client. |
| @Produces | The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, "text/plain". |

Below will be represented 3 used methods with JAX-RS annotations (Fig.4.7.). First one processes POST request that consumes data in JSON view. PathParam annotation looks in parameter that has been passed in URL. In this case it is user’s name. Further goes POST data that is compressed in JSON. Data selection from JSON is processed with JSON reader. Then entity object is being created, setting data and persisting it to data to database.

Second method processes GET request. Same as previous, it gets data from URL. This time it should be user’s id. With user’s id all records of peculiar user will be loaded and passed to front-end wrapped in Collection. This collection will be rearranged in JSON on front-end side.

Third method process simple GET request for getting all data. In this case it is just getting all users in collection. Further it will be rearranged in JSON like previous one.

## For calculation of flight simulation data will be used Socket. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

Fig.4.7. Methods with JAX-RS annotations for RESTful request interchanging.

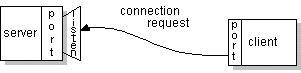
On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

Fig.4.8. Client’s connection request.

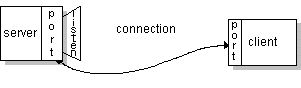
If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

Fig.4.9. The connection is made.

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets. Realized usage of socket is represented in Fig.4.10.

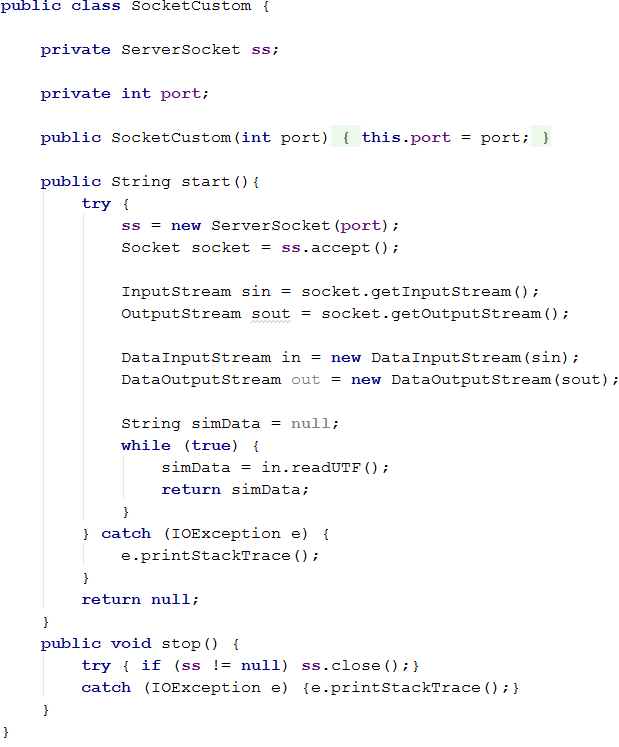
A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Fig.4.10. Custom Socket listener.

## 4.7 Front-end development.

AngularJS is a JavaScript framework, which is choose for calculations on front-end side. Interactions with back-end part carried with ngResource factory.

Object containing default options used when creating $resource instances.

The default values satisfy a wide range of usecases, but you may choose to overwrite any of them to further customize your instances. The available properties are:

1. stripTrailingSlashes – {boolean} – If true, then the trailing slashes from any calculated URL will be stripped. (Defaults to true.)
2. cancellable – {boolean} – If true, the request made by a "non-instance" call will be cancelled (if not already completed) by calling $cancelRequest() on the call's return value. For more details, see $resource. This can be overwritten per resource class or action. (Defaults to false.)
3. actions - {Object.<Object>} - A hash with default actions declarations. Actions are high-level methods corresponding to RESTful actions/methods on resources. An action may specify what HTTP method to use, what URL to hit, if the return value will be a single object or a collection (array) of objects etc. For more details, see $resource. The actions can also be enhanced or overwritten per resource class.

The default actions are:

* get: {method: 'GET'},
* save: {method: 'POST'},
* query: {method: 'GET', isArray: true},
* remove: {method: 'DELETE'},
* delete: {method: 'DELETE'}

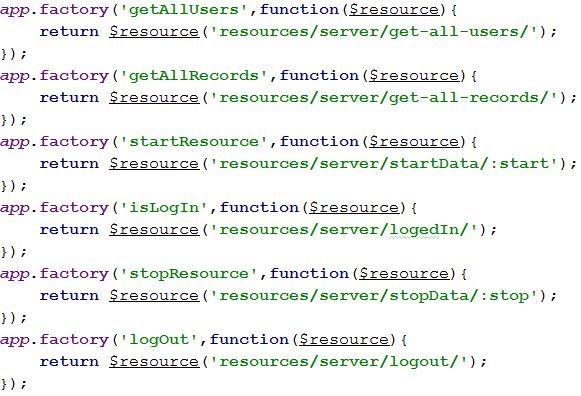
The example of front-end will be server.js file, which process data for records, users and send requests for starting flight simulation session reading. Further is represented list or $resource factories that are used for composing and assignation of request URL with data (Fig.4.11.).

Fig.4.11. List of $resource factories.

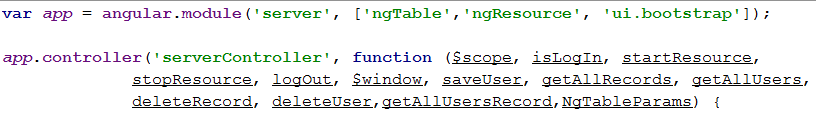
Thing that is necessary for AngulaJS file is assigning name of application, connect additional libraries, for example ngTable. Next step is to create a controller, name it and connect needed factories (Fig.4.12.).

Fig.4.12. Example of initialization AngularJS application.

In Angular, a Controller is defined by a JavaScript constructor function that is used to augment the Angular Scope.

When a Controller is attached to the DOM via the ng-controller directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function. A new child scope will be created and made available as an injectable parameter to the Controller's constructor function as $scope.

If the controller has been attached using the controller as syntax then the controller instance will be assigned to a property on the new scope.

For representing functions in AngularJS will be choose same functions for which where described RESTful methods.

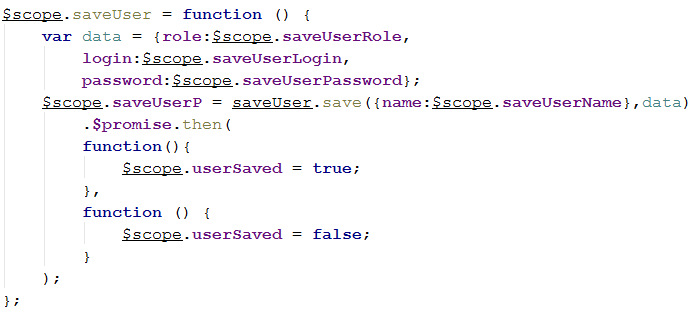
First one is saving user (Fig.4.13.). Firstly data is set with values from DOM objects using ng-model in HTML code so variables that are in $scope can be set values that user inputs. Next is calling $resource factory named “saveUser” and sending POST request with user’s name as URL parameter and JSON as POST data. $promise function works asynchronously and checking if there any respond from server. If respond is succeed then first function proceed, if not – second one.

Fig.4.13. SaveUser function.

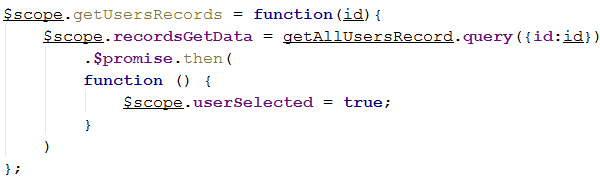
The second is getUsersRecords. As it was described in back-end part, this function consumes id of user, wraps it in URL parameter and sent to server. Similar to previous function, in has response validation.

Fig.4.13. GetUserRecords function.

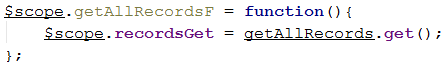
The third function is getAllRecords. Similar to back-end functionality (Fig4.14.).

Fig.4.14. GetAllRecords function.

As the result of getAllRecords we have filled table with two records that are in database. Moreover, logging in part performs with success and in corner the name of user is visualized (Fig.4.15.).

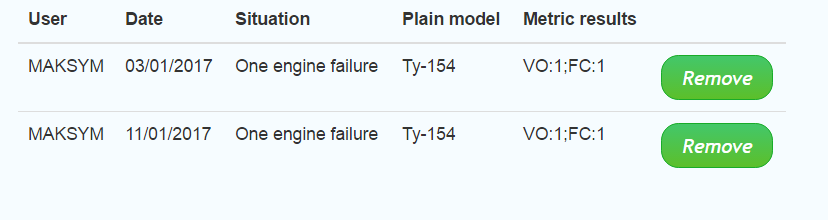


Fig.4.15. Final version of Main page in main application.

## 4.8 Unit testing.

For testing method was chosen unit testing for checking the correct interconnection between database and application. Junit is test framework for unit testing and configuration for unit testing dependency injections will perform Spring Framework XML configuration.

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. JUnit is linked as a JAR at compile-time; the framework resides under package junit.framework for JUnit 3.8 and earlier, and under package org.junit for JUnit 4 and later.

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform.

Central to the Spring Framework is its inversion of control (IoC) container, which provides a consistent means of configuring and managing Java objects using reflection. The container is responsible for managing object lifecycles of specific objects: creating these objects, calling their initialization methods, and configuring these objects by wiring them together.

Objects created by the container are also called managed objects or beans. The container can be configured by loading XML files or detecting specific Java annotations on configuration classes. These data sources contain the bean definitions that provide the information required to create the beans.

Spring configuration in XML view is a mapping for dependency injections like:

<bean id="pmDAO" class="com.server.daoimpl.PlainModelDaoImpl"/>.

Also for correctness of data transactions, it is necessary to configure database connection.

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">

<property name="entityManagerFactory" ref="entityManagerFactory" />

</bean>

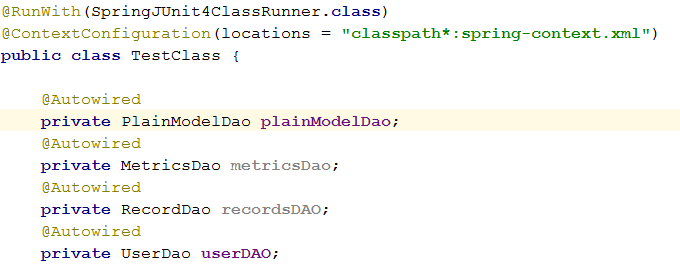
The next is the creation and configuration of test class. Firstly configure run class with which test will be ran. Then path to Spring configuration XML file. Autowire annotation wires mapped DAOs to needed interfaces like it was configured for dependency injection.

Fig.4.16. Unit test class configuration.

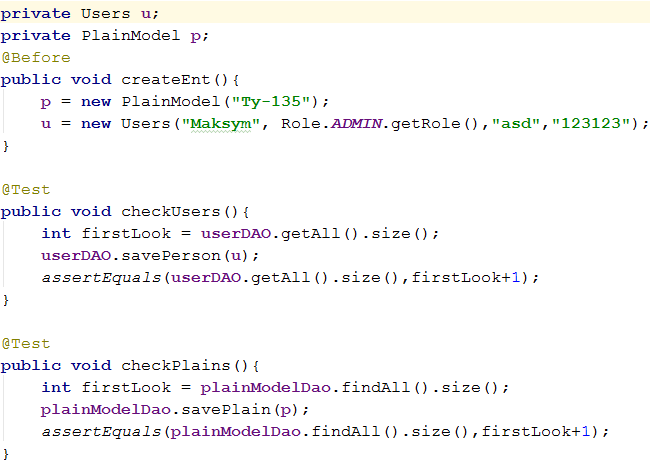
Next is to create test method with Test annotation and Before annotation for preparing objects for testing before executing test method (Fig.4.17.).

Fig.4.17. Example of unit test methods.

After running tests JUnit gives the result that all tests are passed(Fig.4.18.).

Fig.4.18. Result of unit testing.

## Conclusion.

The result of this chapter is fully functional system with implemented list of functional requirements, described technologies, tools and frameworks. The development implemented usage of architectural, design patterns, layered design.

The entities configuration has been proceed within configuration of an application server and initialization database with scripts. Described process of configuration with given examples and described tools, which been used with their conceptual purposes.

Data access object has been created with necessary configuration implementation for peculiar needs. Described types of beans and containers that wrap beans. Configuration of DAOs for peculiar needs has been described. EJBs have been implemented for usage in transactions and calls.

RESTful services have been created and described. JAX-RS has been implemented. Configured for working with requests and responses from front-end. Basics of EJB configuration and URL path parameterization were represented. Processing of REST requests were described both. Three typical functions were represented and fully described. Moreover, custom socket listener implementation has been represented.

Unit testing of database and application interconnection has been occurred. Spring framework configuration and inversion of control basics have been declared. JUnit framework was characterized and described with further configuration in test class.

AngularJS was limned as a powerful tool with number of factories that makes interaction between front-end and back-end easier. Three function have been analyzed and described with call REST requests and data transitions.