Thesis Documentation

Version of the project

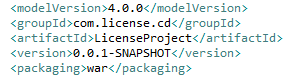
The project is created in Eclipse IDE for Enterprise Java Developers 2020-06 as a Maven Project, the version of Maven is 3.6.3. The Java application is deployed on an Apache Tomcat v9.0 application server. Java version is 8

Configuration of the project

For configuration of the project is necessary to edit and create some files as follows.

Setting up the pom.xml

The POM or Project Object Model is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project.



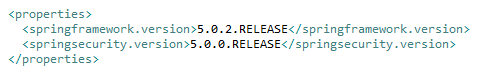
The modelVersion is set to 4.0.0 for Maven 2 and 3. Since I am using Maven 3.6.3 This is the value I must choose.

The groupId is set by the developer and represents a unique identifier for the project across all projects. The groupId should follow Java`s package name rules.

The artifactId represents the name of the archive.

I set the packaging to be WAR (web application archive), because this type of file can contain the web application that can be deployed on any servlet/jsp.

The next step is to set up the resources I need for this project. Since Spring Framework contains several projects, I should keep track of the version I am using. I created two properties, one for Spring Framework and the other one for Spring Security, as a good practice to keep the actual version in just one place.



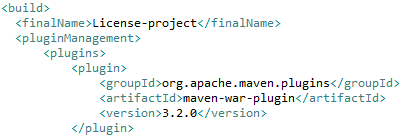
Furthermore, I created a dependencies section in which I imported from the Maven Central Repository all the libraries I need.



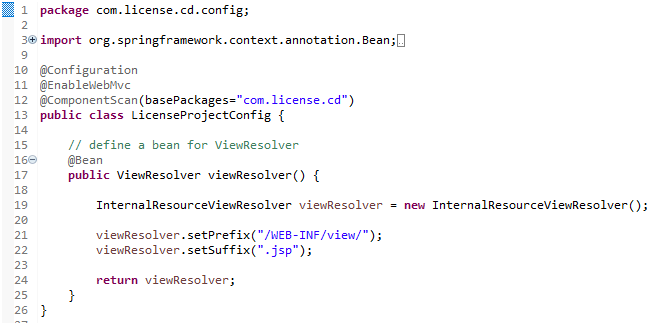
Beside Spring, I also imported resources like Hibernate, MySQL Connector, Web JARs etc.

The next step in configuration is to configure the Spring Container, which can be done in three ways. In this project I choose to use the Java Source Code approach. This method will provide the full configuration through Java code, getting rid of the xml files.

The first thing to do is to configure the web.xml file inside the pom.xml, because the Java Source Code approach will not allow a separate web.xml file.



Next, I created a new package which will contain all the configuration.

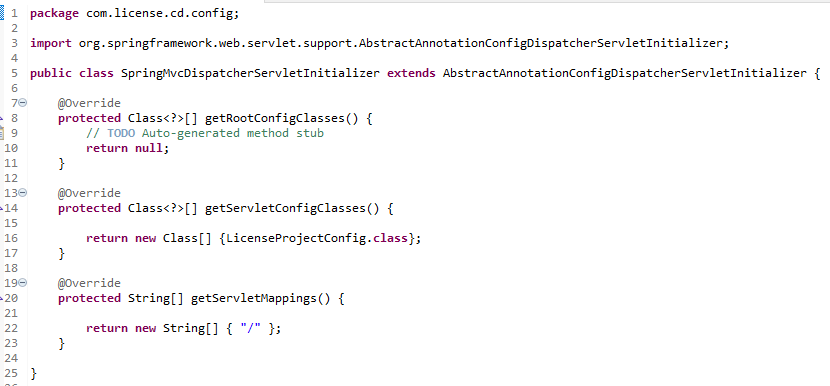


I use @Components annotation to explicitly say that this class is a configuration class

@EnableWebMvc it’s like <mvc:annotation-driven/> from xml. Is used for conversion, formatting and validation support.

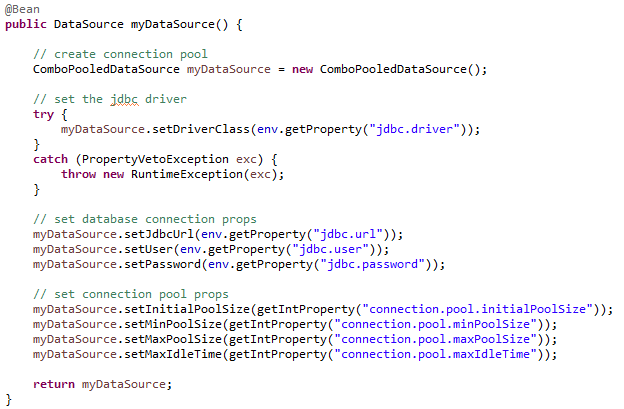
@ComponentScan define the base package in which components are searched.

Then I defined a bean for the ViewResolver. The ViewResolver has the role to create a path towards the JSP (Java Servlet Page) by combining a suffix with a prefix and the return statement from the Controller.



Next, I created another class for the Spring Dispatcher servlet Initializer. Here, for now, I just added the previous created file and the mapping.

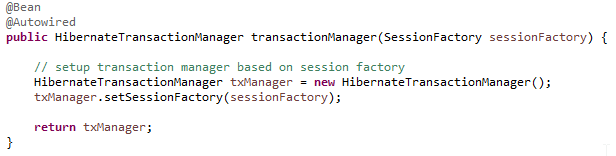
The next step is to configure the database connection.



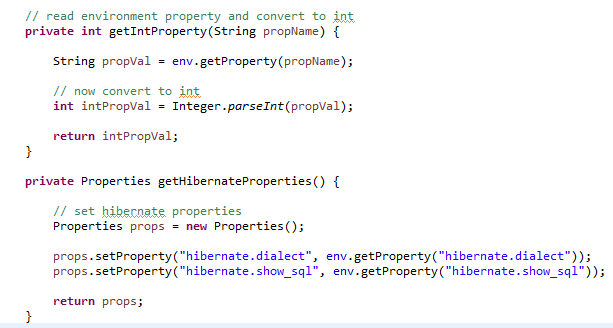
I created a new bean in which I defined the connection pool, I set the Java Database Connectivity driver, then I set database connection and connection pool properties.



I created another bean that will create the SessionFactory.

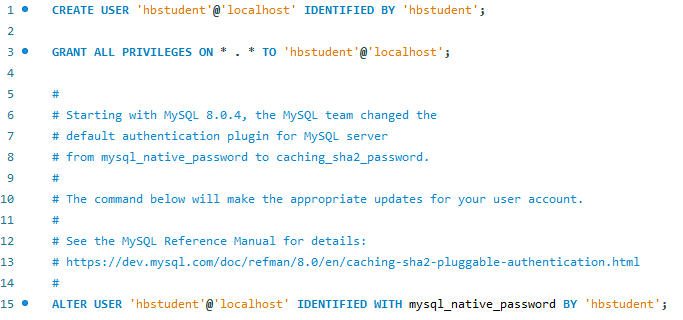


Another bean for the transaction manager. This method will @Autowired the session factory bean.

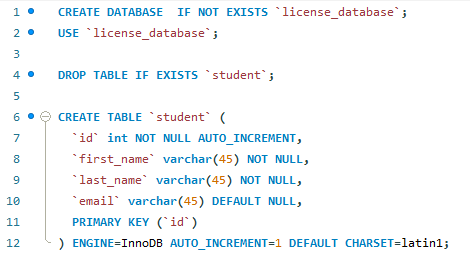


I also created two functions that will help in retrieving the necessary properties.

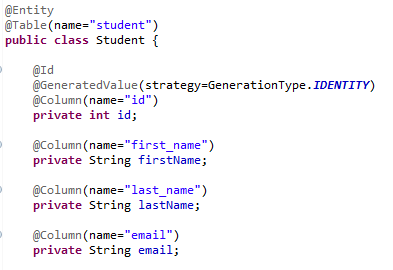
Next step is to create a new user that will be able to access the database.



Then we will create the database and the first table.



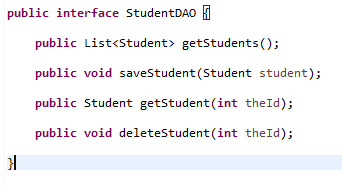
The next step is to create the java entity for this table



Developing the CRUD features

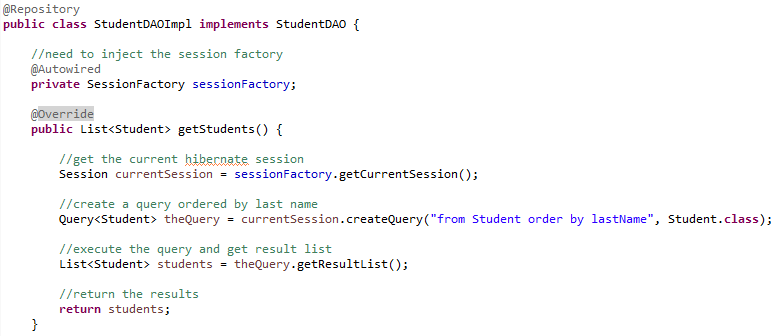
For this task I used DAO design pattern ( data access object ):

I created an interface

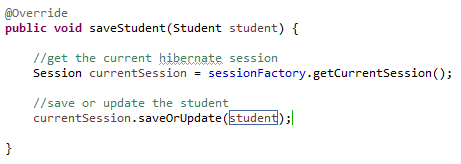


Then I created the implementation of this interface as follows:

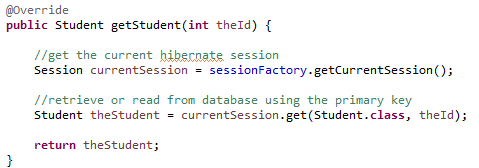
Implementation of getStudents() method



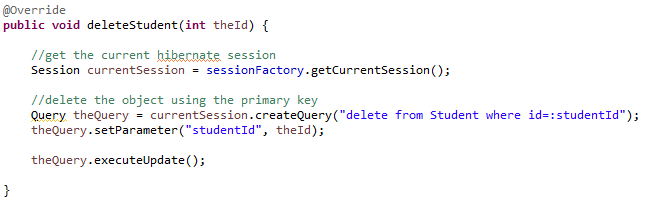
Implementation of saveStudent(Student student)



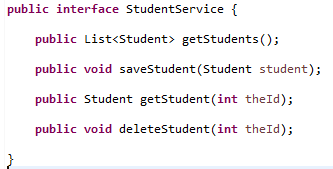
Implementation of getStudent(int theId)

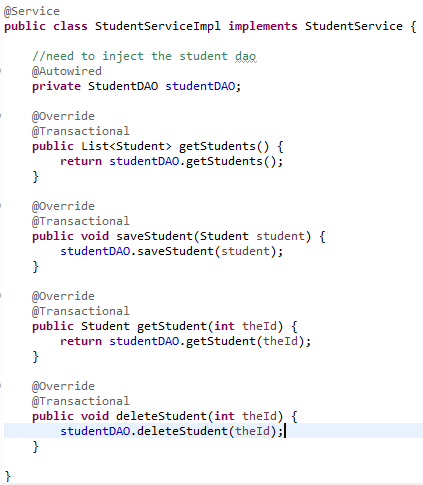


Implementation of deleteStudent(int theId)



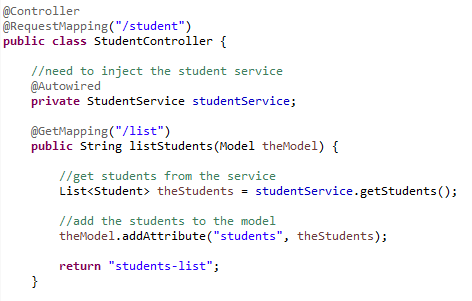
The next step is to create another layer, service class that will handle the DAO functionalities



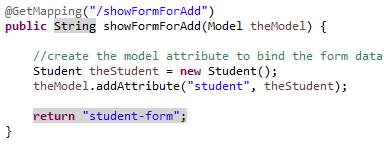


Spring MVC Implementation

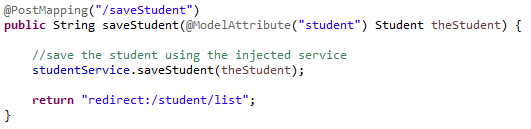
I created the controller that will handle the request for /student/list, will return the JSP name and will add into the model list of students, that will be displayed later into the JSP.



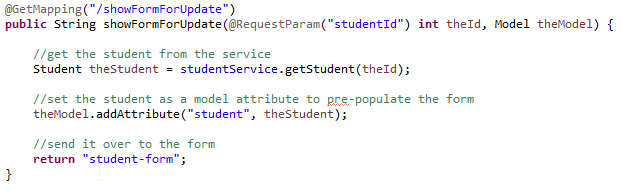
The request for /student/showFormForAdd



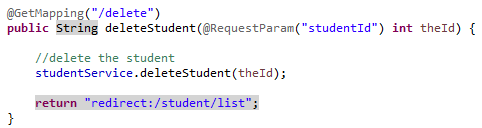
The request for saving the new created student



The request for showing the update form for a specific student



And the request for deleting a specific student

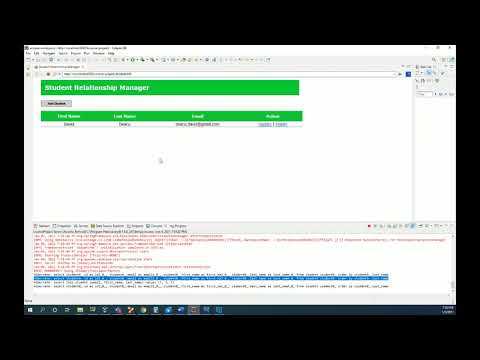


The next step is to design the views returned by the controller





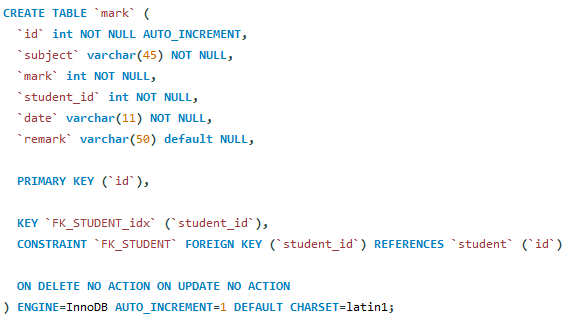
CRUD functionality demo: https://youtu.be/Pl6iTH-xYps

[](https://www.youtube.com/embed/Pl6iTH-xYps?feature=oembed)

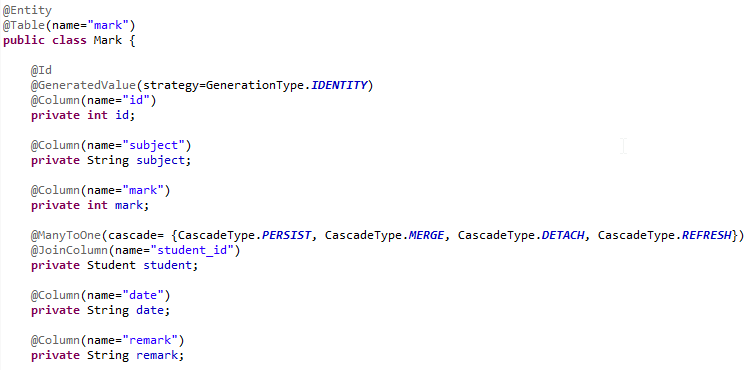
Developing One to Many relationships

Link between Student and Marks

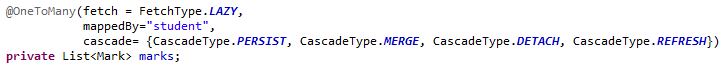
Firstly I created the “mark” table and I linked it with the “student” table through the “student\_id” column.



Then I created the Mark entity

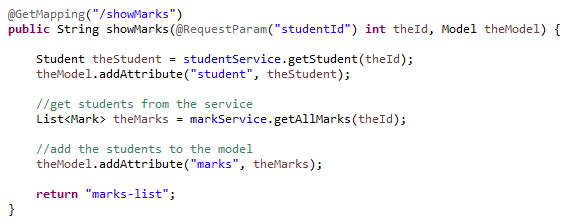


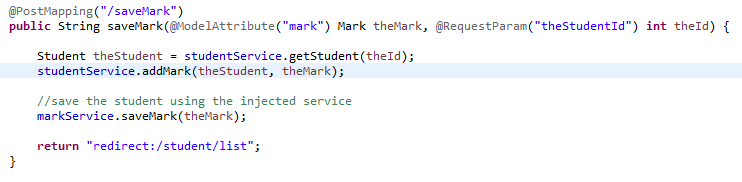
And I updated the Student entity by adding a new field, specifying the fetch type as LAZY (the data stored by this field will not be fetched when a Student instance will be requested. That results in an increased performance due to the fact that in the moment of displaying the list of Students, we will not fetch the Mark table for each student. The Mark instance will be fetched in the moment we request it by pressing on a specific student “Show Marks” link.



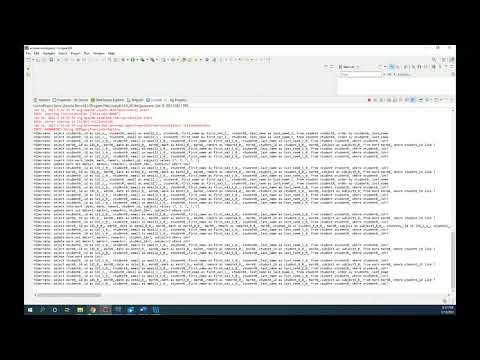
After this I implemented DAO and Service classes like the ones for Student entity.

Lastly updated the controller and I created the view page.





CRUD functionality demo for Student-Mark link: <https://youtu.be/ttGiocoYSXk>

[](https://www.youtube.com/embed/ttGiocoYSXk?feature=oembed)