Report

Contents

1) Introduction:

This application walks through the process of creating a Spring MVC application for uploading and downloading a file from filesystem.

Optional bonus points covered in the application:

- Storing uploaded file in BLOB field of database.
- Added security.
- Added persistence.

2) Software Required:

- Apache Maven
- Tomcat/ Any application server – to deploy the project
- Eclipse/STS
- MySQL

3) How to run the Project:

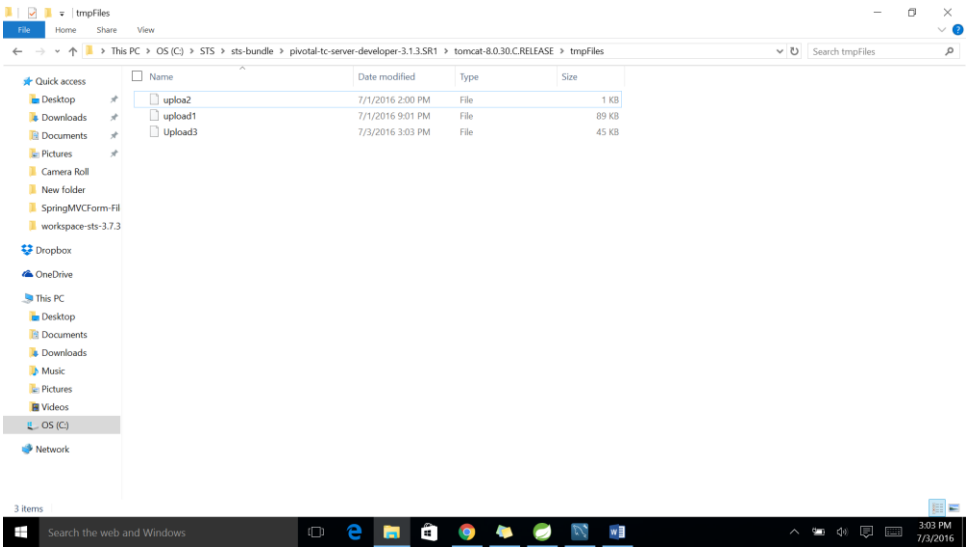
After you get the project into your workspace, open command prompt and get to the project directory then run the following commands after you ensure you have maven working:

mvn clean install

mvn eclipse:eclipse

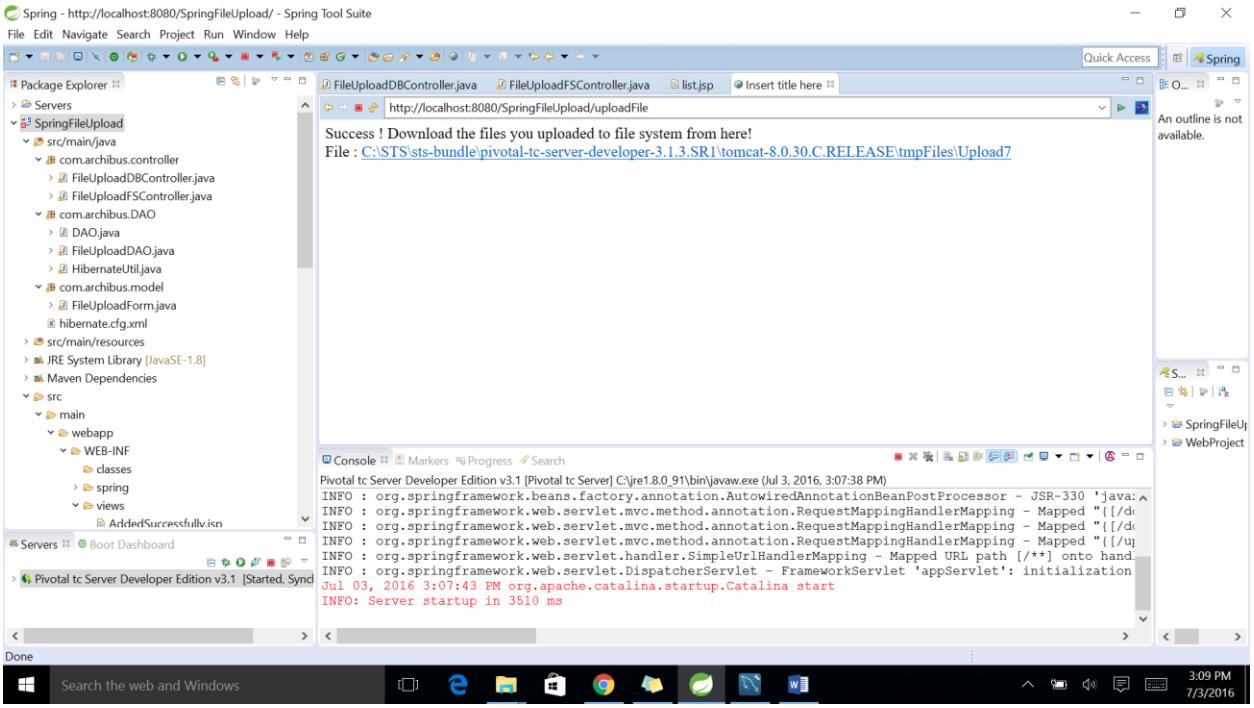
For storing the file to a file system:

Upload a file browsing through the file system section on the displayed page and type in a name for the file. If the name field is left blank, an error page is displayed. The location of the file stored is outputted on the console. It creates a folder by name tmpFiles to store the uploaded files.



Uploaded files are shown on the next page after clicking upload and the file is uploaded successfully.

User can download the file from here.



For uploading the file to database:

Now, go to your IDE and go to src/main/java hibernate.cfg.xml file and uncomment the following line to create a new schema for our hospital enterprise.

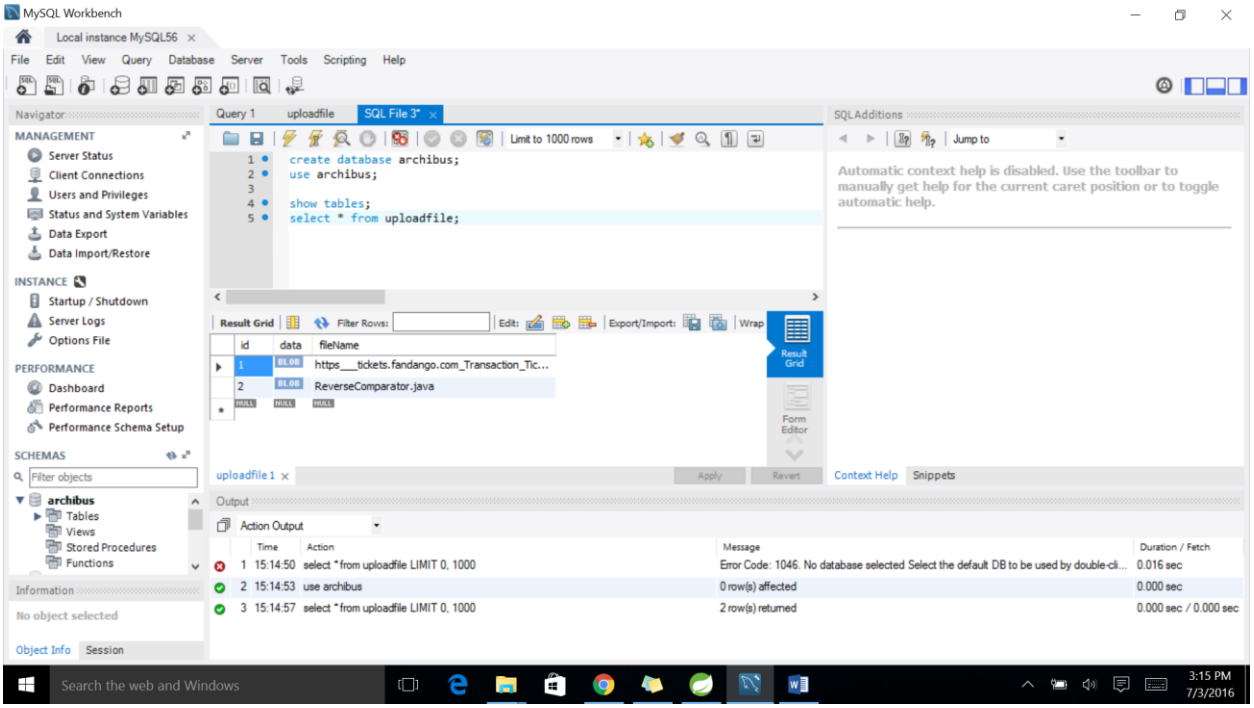
<property name="hibernate.hbm2ddl.auto">create</property>

Update your MySQL username and password

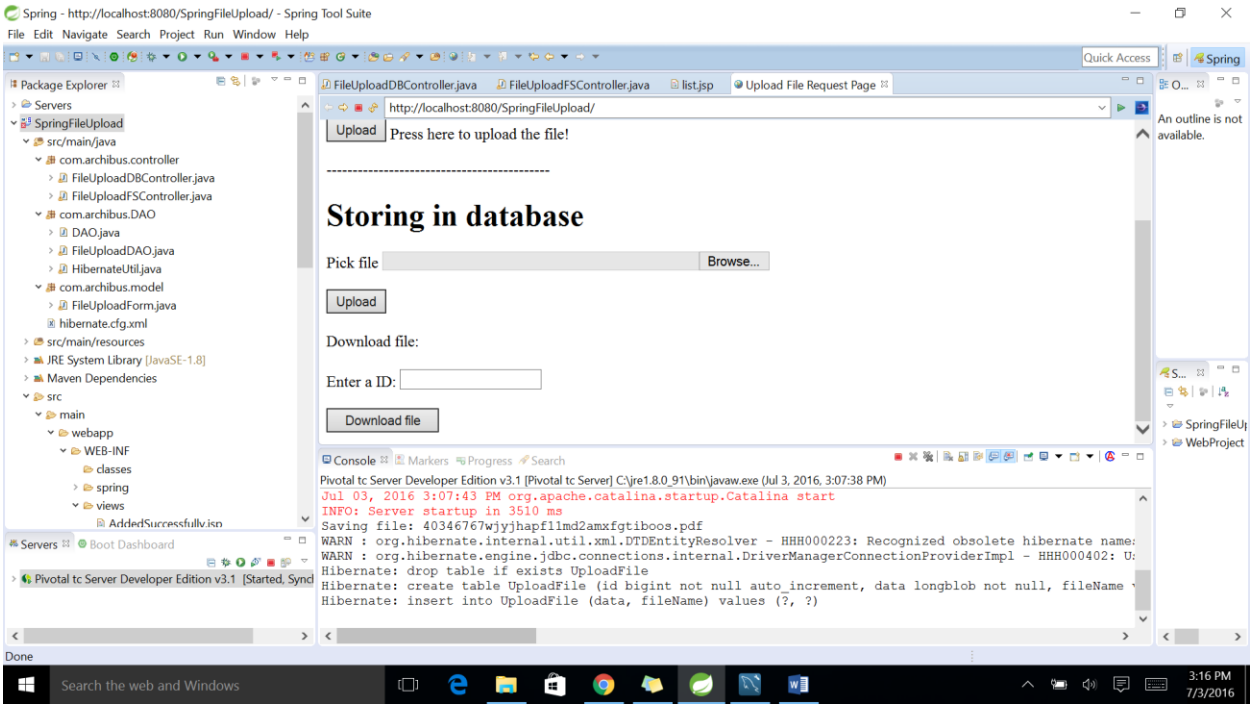
In MySql create a schema: ***CREATE SCHEMA archibus;***

This will create the schema when you run the project on server. It only creates for the first time now uncomment it.

After running it for the 1st time, go to MySQL and type **select * from uploadfile;** this command will display all the files uploaded to the database.



For storing the file in a database, upload the file through database section on the page displayed after running the project.



For downloading a file from database, enter an ID staring from 1. This ID is the primary key of the file in the database. If the user enters an ID which is in appropriate, an error page is displayed. Once you click on download button, the file to be downloaded is displayed.

4) Security:

To handle security for the application, I have implemented declarative security. This prevents unauthorized users from accessing sensitive data.

With declarative security, none of the individual servlets or JSP pages need any security-aware code. Instead, both of the major security aspects are handled by the server. To prevent unauthorized access, we use the Web application deployment descriptor (web.xml) to declare that certain URLs need protection. We also designate the authentication method that the server should use to identify users. At request time, the server automatically prompts users for usernames and passwords when they try to access restricted resources, automatically checks the results against a predefined set of usernames and passwords, and automatically keeps track of which users have previously been authenticated. This process is completely transparent to the servlets and JSP pages.

Setting Passwords with Tomcat: With this mechanism, Tomcat stores usernames, passwords, and roles in install_dir/ conf/tomcat-users.xml.

Following code must be added here:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<tomcat-users>

 <user name="vaishnavi" password="vaishnavi"

 roles="admin" />

</tomcat-users>
```

In *web.xml* following code must be added*:*

You use the login-config element in the deployment descriptor (web.xml) to control the authentication method. Although a few servers support nonstandard web.xml files (e.g., Tomcat has one in install_dir/conf that provides defaults for multiple Web applications), those files are entirely server specific.

```
<security-constraint>

<auth-constraint>

<role-name>admin</role-name>

<role-name>vaishnavi</role-name>

</auth-constraint>

</security-constraint>
```

5) Persistence:

Persistence is handled from the package com.archibus.DAO. I created the actual DAO class that will handle the persistence work. Database configuration is done through DAO class. HibernateUtil class configures a SessionFactory object. DAO class gets the SessionFactory instance from HibernteUtil and opens a session. FileUploadDAO extends DAO class and handles thread-safe opening and closing of a session and a transaction.

6) Technical Overview:

I have designed an Application using Spring MVC architecture and Hibernate.

I have used Declarative Security Mechanism to handle application's security. In order to validate the user inputs I have used Hibernate validator. The Application built is an Annotation Based Web-Application.

For an improved view, I have used JSPs.

Entities:

FileUploadForm: Contains three attributes. ID serves as the primary key for the table created. Filename is a string and an byte array for storing the file.